



devonfw guide

Updated at 2021-03-04 16:17:38 UTC

Table of Contents

I: Getting Started	1
1. Introduction	2
1.1. What is devonfw?.....	2
1.2. Why should I use devonfw?.....	5
1.3. devonfw-ide Download and Setup.....	7
2. Guides	8
2.1. Build Your First devonfw Application.....	8
2.2. Explore Our devonfw Sample Application.....	9
3. Further Information	12
3.1. Repository Overview.....	12
3.2. Links to our Community.....	13
3.3. Contributing	13
3.4. Code of Conduct	13
II: devonfw-ide	14
4. Introduction	15
4.1. Features	15
4.2. Setup	18
5. Usage.....	20
5.1. Roles	20
5.2. Configuration	23
5.3. Variables	25
5.4. Devon CLI.....	28
5.5. Structure	44
6. Advanced Features	52
6.1. Cross-Plattform Tooling	52
6.2. Windows Tooling	52
6.3. MacOS Tooling	56
6.4. Linux Tooling	62
6.5. Lombok	62
7. Support.....	64
7.1. Migration from oasp4j-ide	64
7.2. License	66
III: devonfw for Java (devon4j)	138
8. Architecture	139
8.1. Key Principles	139
8.2. Architecture Principles	139
8.3. Application Architecture	140
8.4. Components	144

9. Coding Conventions	146
9.1. Naming	146
9.2. Packages	146
9.3. Architecture Mapping	148
9.4. Code Tasks	149
9.5. Code-Documentation	150
9.6. Code-Style	151
10. Project structure	156
10.1. Make jar not war	156
10.2. Package Structure	156
11. Layers	157
11.1. Client Layer	157
11.2. Service Layer	159
11.3. Service-Versioning	161
11.4. Logic Layer	166
11.5. Data-Access Layer	173
11.6. Batch Layer	174
12. Guides	181
12.1. Dependency Injection	181
12.2. Configuration	184
12.3. Java Persistence API	193
12.4. Auditing	216
12.5. Transaction Handling	218
12.6. SQL	219
12.7. Database Migration	222
12.8. Logging	224
12.9. Security	229
12.10. Access-Control	232
12.11. Access Control Schema	241
12.12. JSON Web Token(JWT)	249
12.13. Cross-site request forgery (CSRF)	252
12.14. Validation	255
12.15. Aspect Oriented Programming (AOP)	258
12.16. Exception Handling	261
12.17. Internationalization	265
12.18. XML	267
12.19. JSON	268
12.20. REST	272
12.21. SOAP	282
12.22. Service Client	284
12.23. Testing	292

12.24. Transfer-Objects	304
12.25. Bean-Mapping	306
12.26. Datatypes	308
12.27. CORS support	311
12.28. BLOB support	313
12.29. Java Development Kit	314
12.30. Microservices in devonfw	318
12.31. Application Performance Management	320
12.32. Caching	321
12.33. Feature-Toggles	322
12.34. Accessibility	328
12.35. JEE	329
12.36. Messaging Services	331
12.37. Messaging	343
12.38. Monitoring	344
12.39. Full Text Search	348
13. Tutorials	349
13.1. Creating a new application	349
IV: devon4ng	352
14. Architecture	353
14.1. Architecture	353
14.2. Meta Architecture	354
15. Layers	362
15.1. Components Layer	362
15.2. Services Layer	365
16. Guides	370
16.1. Package Managers	370
16.2. Package Managers Workflow	372
16.3. Yarn 2	382
17. Angular	384
17.1. Accessibility	384
17.2. Angular Elements	390
17.3. Angular Lazy loading	403
17.4. Angular Library	414
17.5. Angular Material Theming	418
17.6. Angular Progressive Web App	428
17.7. APP_INITIALIZER	437
17.8. Component Decomposition	447
17.9. Consuming REST services	452
17.10. Error Handler in angular	458
17.11. File Structure	462

17.12. Internationalization	465
17.13. Routing	469
17.14. Testing	476
17.15. Update Angular CLI	484
17.16. Upgrade devon4ng Angular and Ionic/Angular applications	485
17.17. Working with Angular CLI	488
18. Ionic	492
18.1. Ionic 5 Getting started	492
18.2. Basic project set up	492
18.3. Ionic to android	493
18.4. Ionic Progressive Web App	501
19. Layouts	508
19.1. Angular Material Layout	508
20. NgRx	521
20.1. Introduction to NgRx	521
20.2. State, Selection and Reducers	523
20.3. Side effects with NgRx/Effects	532
20.4. Simplifying CRUD with NgRx/Entity	536
21. Cookbook	539
21.1. Abstract Class Store	539
21.2. Angular Mock Service	549
21.3. Testing e2e with Cypress	553
21.4. Angular ESLint support	564
V: devon4net	567
22. Architecture basics	568
22.1. Introduction	568
23. User guide	577
23.1. devon4net Guide	577
24. How To section	599
24.1. Introduction	599
24.2. How to	599
25. Cobigen guide	617
25.1. devon4net Cobigen Guide	617
26. Coding conventions	621
26.1. Code conventions	621
27. Environment	626
27.1. Environment	626
28. Packages	628
28.1. Packages	628
28.2. Required software	652
29. Templates	653

29.1. Templates	653
30. Samples	655
30.1. Samples	655
VI: devon4node	665
31. devon4node Architecture	666
31.1. HTTP layer	666
31.2. devon4node layers	667
31.3. devon4node application structure	667
32. Layers	672
32.1. Controller Layer	672
32.2. Service Layer	673
32.3. Data Access Layer	674
33. Guides	675
33.1. Key Principles	675
33.2. Code Generation	676
33.3. Coding Conventions	679
33.4. Dependency Injection	682
33.5. Configuration Module	685
33.6. Auth JWT module	687
33.7. Swagger	688
33.8. TypeORM	689
33.9. Serializer	690
33.10. Validation	691
33.11. Logger	692
33.12. Mailer Module	693
33.13. Importing your ESLint reports into SonarQube	698
34. devon4node applications	699
34.1. devon4node Samples	699
34.2. Create the employee sample step by step	700
VII: Choosing your Database	712
35. Database	713
35.1. RDBMS	713
35.2. NoSQL	713
36. SAP HANA	716
36.1. Driver	716
36.2. Developer Usage	716
36.3. Pooling	716
36.4. Fuzzy Search	717
37. Oracle RDBMS	718
37.1. XE	718
37.2. Driver	719

37.3. Pooling	719
37.4. Messaging	720
37.5. General Notes on the use of Oracle products	721
37.6. Fix for TNS-Listener issues	721
38. MS-SQL-Server	723
38.1. Driver	723
39. PostgreSQL	724
39.1. Driver	724
40. MariaDB	725
40.1. Driver	725
41. Database	726
41.1. RDBMS	726
41.2. NoSQL	726
42. Cassandra	729
42.1. Attention	729
42.2. Driver	729
42.3. Spring-Data	729
43. neo4j	730
43.1. Attention	730
43.2. Driver	730
43.3. Spring-Data	730
44. MongoDB	731
44.1. Attention	731
44.2. Driver	731
45. CouchDB	732
45.1. Attention	732
45.2. Driver	732
46. Redis	733
46.1. Attention	733
46.2. Driver	733
47. OrientDB	734
47.1. Attention	734
47.2. Driver	734
47.3. Administration	734
48. Blazegraph	735
48.1. Attention	735
48.2. Driver	735
49. HBase	736
49.1. Attention	736
49.2. Driver	736
50. RavenDB	737

50.1. Attention	737
50.2. Driver	737
VIII: devonfw shop floor	738
51. What is devonfw shop floor?	739
52. How to use it	740
52.1. Prerequisites - Provisioning environment	740
52.2. Step 1 - Configuration and services integration	740
52.3. Step 2 - Create the project	740
52.4. Step 3 - Deployment	741
52.5. Step 4 - Monitoring	741
53. Provisioning environments	742
53.1. Production Line provisioning environment	742
53.2. dsf4docker provisioning environment	742
54. Configuration and services integration	745
54.1. Nexus Configuration	745
54.2. SonarQube Configuration	752
55. Create project	755
55.1. Create and integrate git repository	755
55.2. start new devonfw project	755
55.3. cicd configuration	756
56. Deployment environments	769
56.1. OpenShift	769
57. Monitoring	785
57.1. Build monitor view	785
58. Annexes	788
58.1. BitBucket	788
58.2. Selenium Basic Grid	800
58.3. Mirabaud Experience	807
58.4. Azure DevOps	835
58.5. OKD (OpenShift Origin)	849
58.6. OKD (<i>OpenShift Origin</i>)	850
58.7. ISTIO Guide	864
IX: cicdgen	874
59. CICDGEN	875
59.1. What is angular schematics?	875
59.2. cicdgen CLI	875
59.3. cicdgen Schematics	875
59.4. Usage example	875
60. cicdgen CLI	876
60.1. CICDGEN CLI	876
60.2. cicdgen usage example	877

61. cicdgen Schematics	888
61.1. CICDGEN SCHEMATICS	888
61.2. devon4j schematic	891
61.3. devon4ng schematic	895
61.4. devon4net schematic	899
61.5. devon4node schematic	903
X: Production Line Templates	908
62. How to add a Template to your PL instance	909
63. devonfw Technologies Templates	911
63.1. How to add a Template to your PL instance	911
63.2. devon4j Template for Production Line	912
63.3. devon4ng Template for Production Line	916
63.4. devon4node Template for Production Line	920
63.5. From existing devonfw Template for Production Line	923
64. Utility Templates	927
64.1. Initialize Instance Template for Production Line	927
64.2. Install SonarQube Plugin	934
64.3. Docker Configuration	935
64.4. Docker Configuration	938
65. MrChecker	942
65.1. MrChecker under ProductionLine	942
66. Samples	947
66.1. devon4j My-Thai-Star Sample Application Template for Production Line	947
67. Troubleshooting	954
67.1. Troubleshootibng	954
XI: CobiGen — Code-based incremental Generator	955
68. Document Description	956
68.1. Guide to the Reader	957
68.2. CobiGen - Code-based incremental Generator	957
68.3. General use cases	959
69. CobiGen	963
69.1. Configuration	963
69.2. Plug-ins	973
70. CobiGen CLI	994
70.1. Cobigen Command line Interface generation	994
71. Maven Build Integration	997
71.1. Maven Build Integration	997
72. Eclipse Integration	1002
72.1. Installation	1002
72.2. Usage	1004
72.3. Logging	1011

73. How to	1012
73.1. Angular 8 Client Generation	1012
73.2. Ionic client generation	1018
73.3. Implementing a new Plug-in	1022
73.4. Introduction to CobiGen external plug-ins	1027
73.5. How to update CobiGen	1044
73.6. CobiGen Release creation	1047
73.7. End to End POC Code generation using OpenAPI	1047
73.8. End to End POC Code generation using Entity class	1076
73.9. Enable Composite Primary Keys in Entity	1104
74. Template Development	1107
XII: MrChecker - devonfw testing tool	1108
75. Who Is MrChecker	1109
75.1. Who is MrChecker?	1109
75.2. Where does MrChecker apply?	1109
75.3. MrChecker's specification:	1109
75.4. Benefits	1109
75.5. Test stages	1110
76. Test Framework Modules	1111
76.1. Core Test Module	1111
76.2. Selenium Module	1157
76.3. Selenium Structure	1159
76.4. Framework Features	1165
76.5. Web API Module	1176
76.6. Security Module	1203
76.7. Database Module	1204
76.8. Mobile Test Module	1205
76.9. DevOps Test Module	1211
77. MrChecker download	1233
77.1. Windows	1233
77.2. Mac	1240
78. Tutorials	1244
78.1. Project organization	1244
78.2. Basic Tutorials	1252
78.3. E2E Tutorials	1355
79. Migration from JUnit4 to JUnit5	1365
79.1. Migration guide	1365
80. FAQ	1376
80.1. Common problems	1376
80.2. How to	1377
80.3. Installation problems	1380

XIII: MyThaiStar	1381
81. 1.My Thai Star – Agile Framework	1382
81.1. 1.1 Team Setup	1382
81.2. 1.2 Scrum events	1382
81.3. 1.3 Establish Product Backlog	1383
82. 2.My Thai Star – Agile Diary	1385
82.1. 24.03.2017 Sprint 1 Planning	1385
82.2. 27.04.2017 Sprint 1 Review	1385
82.3. 03.05.2017 Sprint 2 Planning	1385
82.4. 01.06.2017 Sprint 2 Review	1386
83. User Stories	1387
83.1. Epic: Invite friends	1387
83.2. Epic: Digital Menu	1389
83.3. Epic: User Profile	1391
83.4. Epic: Rate by twitter	1391
83.5. Epic: Waiter Cockpit	1392
84. Technical design	1393
84.1. Data Model	1393
84.2. Server Side	1394
84.3. Client Side	1419
85. Security	1427
85.1. Two-Factor Authentication	1427
86. Testing	1431
86.1. Server Side	1431
86.2. Client Side	1433
86.3. End to end	1438
87. UI design	1440
87.1. Style guide	1440
87.2. Low and high fidelity wireframes	1440
88. CI/CD	1441
88.1. My Thai Star in Production Line	1441
88.2. Deployment	1455
88.3. MyThaiStar on Native Kubernetes as a Service (nKaaS)	1466
XIV: devonfw dashboard	1468
89. Landing page	1469
89.1. Landing page	1469
90. Home	1471
90.1. Home page	1471
91. Projects	1476
91.1. Introduction to project management in the dashboard	1476
91.2. Projects	1476

91.3. How to create a project	1478
92. Repositories	1481
92.1. Repositories	1481
93. Wiki	1483
93.1. Wiki page	1483
94. Settings	1484
94.1. Settings	1484
XV: Solicitor User Guide	1486
95. Introduction	1487
95.1. Licensing of Solicitor	1488
96. Architecture	1489
96.1. Data Model	1489
97. Usage	1494
97.1. Executing Solicitor	1494
97.2. Project Configuration File	1494
97.3. Starting a new project	1499
97.4. Exporting the Builtin Configuration	1499
97.5. Configuration of Technical Properties	1500
98. Reading License Information with Readers	1501
98.1. Maven	1501
98.2. CSV	1501
98.3. NPM	1502
98.4. Gradle (Windows)	1503
98.5. Gradle (Android)	1504
99. Working with Decision Tables	1506
99.1. Extended comparison syntax	1507
100. Standard Business Rules	1508
100.1. Phase 1: Determining assigned Licenses	1508
100.2. Phase 2: Selecting applicable Licenses	1509
100.3. Phase 3: Legal evaluation	1510
100.4. Amending the builtin decision tables with own rules	1511
101. Reporting / Creating output documents	1512
101.1. SQL transformation and filtering	1512
101.2. Determining difference to previously stored model	1513
101.3. Sample SQL statement	1514
101.4. Writers	1514
102. Resolving of License URLs	1516
102.1. Encoding of URLs	1516
103. Feature Deprecation	1517
103.1. List of Deprecated Features	1517
Appendix A: Default Base Configuration	1518

Appendix B: Built in Default Properties	1519
Appendix C: Extending Solicitor	1520
C.1. Format of the extension file	1520
C.2. Activating the Extension	1520
Appendix D: Release Notes	1521
XVI: Contributing	1522
104. OSS Compliance	1523
104.1. Preface	1523
104.2. Obligations when using OSS	1523
104.3. Automate OSS handling	1524
104.4. Configure the Mojo Maven License Plugin	1524
104.5. Retrieve licenses list	1527
104.6. Create an OSS inventory	1527
104.7. Create a THIRD PARTY file	1527
104.8. Download and package OSS SourceCode	1528
104.9. Handle OSS within CI-process	1529
XVII: Release Notes	1530
105. devonfw Release notes 2020.12	1531
105.1. Introduction	1531
105.2. devonfw IDE	1531
105.3. devon4j	1532
105.4. devon4node	1533
105.5. CobiGen	1534
105.6. Sonar devon4j plugin	1535
105.7. devon4net	1535
105.8. dashboard (beta version)	1536
105.9. Solicitor	1536
105.10. MrChecker	1537
105.11. Trainings/tutorials	1537
106. devonfw Release notes 2020.08	1538
106.1. Introduction	1538
106.2. devonfw IDE	1538
106.3. devon4j	1539
106.4. devon4ng	1540
106.5. devon4node	1540
106.6. CobiGen	1540
106.7. Sonar devon4j plugin	1541
106.8. My Thai Star with Microservices and ISTIO Service Mesh Implementation	1541
107. devonfw Release notes 2020.04	1543
107.1. Introduction	1543
107.2. devonfw IDE	1544

107.3. devon4j	1545
107.4. devon4ng	1546
107.5. devon4net	1546
107.6. devon4node	1547
107.7. CobiGen	1547
107.8. devonfw-shop-floor	1548
107.9. Sonar devon4j plugin	1549
107.10. My Thai Star	1550
108. devonfw Release notes 3.2 "Homer"	1551
108.1. Introduction	1551
108.2. Changes and new features	1552
109. devonfw Release notes 3.1 "Goku"	1559
109.1. Introduction	1559
109.2. Changes and new features	1560
110. devonfw Release notes 3.0 "Fry"	1567
110.1. Introduction	1567
110.2. Changes and new features	1568
111. devonfw Release notes 2.4 "EVE"	1576
111.1. Introduction	1576
111.2. Changes and new features	1576
112. devonfw Release notes 2.3 "Dash"	1583
112.1. Release: improving & strengthening the Platform	1583
112.2. An industrialized platform for the ADcenter	1583
112.3. Changes and new features	1584
113. devonfw Release notes 2.2 "Courage"	1590
113.1. Production Line Integration	1590
113.2. OASP4js 2.0	1590
113.3. A new OASP Portal	1590
113.4. New Cobigen	1590
113.5. MyThaiStar: New Restaurant Example, reference implementation & Methodology showcase	1591
113.6. The new OASP Tutorial	1591
113.7. OASP4j 2.4.0	1592
113.8. Microservices Netflix	1592
113.9. devonfw distribution based on Eclipse OOMPH	1593
113.10. Visual Studio Code / Atom	1593
113.11. More I18N options	1593
113.12. Spring Integration as devonfw Module	1593
113.13. devonfw Harvest contributions	1593
113.14. More Deployment options to JEE Application Servers and Docker/CloudFoundry ..	1594
113.15. Devcon on Linux	1594

113.16. New OASP Incubators	1594
114. Release notes devonfw 2.1.1 "Balu"	1595
114.1. Version 2.1.2: OASP4J updates & some new features	1595
114.2. Version 2.1.1 Updates, fixes & some new features	1596
114.3. Version 2.1 New features, improvements and updates	1597

Part I: Getting Started

1. Introduction

1.1. What is devonfw?



Welcome to the **devonfw** platform. This is a product of the CSD (Custom Solution Development) industrialization effort to establish a standardized platform for custom software development within Capgemini APPS2. This platform is aimed at engagements, in which clients don't specify the use of a predefined technology stack. In these cases we can offer a proven alternative as a result of our experience as a group.

devonfw is a development platform aiming for the standardization of processes and the boosting of productivity. It provides an architecture blueprint for server and client applications, alongside a set of tools to deliver a fully functional, *out-of-the-box* development environment.



The **devonfw name** is a registered trademark of **Capgemini**, but the software and documentation included in **devonfw** are fully open source. Please refer to our [OSS Compliance](#) section for more information.

1.1.1. Building Blocks of the Platform



devonfw uses a state-of-the-art, open source, core reference architecture for the server (these days considered a commodity in the IT-industry) and on top of that an ever increasing number of high-value assets, which are developed by Capgemini.

1.1.2. The devonfw Technology Stack

devonfw is fully open source and consists of the following technology stacks:

Back-End Solutions

For server applications, *devonfw* includes the following solutions:

- [devon4j](#): Server implementation based on Java, [Spring](#) and [Spring Boot](#).
 - [devon4net](#): Server implementation based on [.NET](#).
 - [devon4node](#): Server implementation based on [NestJS](#).

Front-End solutions

For client applications, `devonfw` includes two solutions based on *TypeScript*, *JavaScript*, *C#* and *.NET*:

- [devon4ng](#): Frontend implementation based on [Angular](#) and a hybrid mobile implementation based on [Ionic](#).
 - [devon4X](#): Mobile implementation based on [Xamarin](#).

1.1.3. Custom Tools

devonfw-ide

The `devonfw-ide` is *not* one monolithic program that is installed with a traditional executable; rather it's a collection of scripts which are invoked via command line to automate several, repetitive development tasks. These scripts then interact with other tools, frameworks, and third-party IDEs to streamline the development workflow.

```
..... /////
..... dd ..... ffffff
..... ddd ..... ffffff
..... ddd ..... ff
..... dddddd eeeeeee vvv ..... vvv oooo ..... nnnnnn ffffffff ww
..... dddddd eeeeeeee vvv ..... vvv oooooooo nnnnnnnnnn ffffffff ww w w
..... ddd ddd eeee eee vvv ..... vvv oooo oooo nnn nnn ff ww wwwww www
..... ddd ddd eeeeeeeee vvv ..... vvv ooo ooo nnn nnn ff ww wwwwwww www
..... ddd ddd eeeeeeeee vvvvvv ooo ooo nnn nnn ff wwwwwww www
..... dddddd eeeeeeee vvv ..... oooooooo nnn nnn ff ww ww
..... dddd eeeee v v oooo nnn nnn ff w w

Current version of devon-scripts is

usage: devon [command [args]]
Setup devonfw IDE environment and optionally lauch commands.

Commands:
build
cicdgen
eclipse
gradle
help
ide
intellij
java
jenkins
mvn
ng
node
npm
release
vscode
yarn

For further details use:
devon help <command>
```

The advantage of this approach is, that you can have as many instances of the *devonfw-ide* on your

machine as you need — for different projects with different tools, tool versions and configurations. No need for a physical installation and no tweaking of your operating system required!

Instances of the *devonfw-ide* do not interfere with each other, nor with other installed software. The package size of the *devonfw-ide* is initially very small, the setup is simple, and the included software is portable.

IDEs

It supports the following IDEs:

- [Eclipse](#)
- [Visual Studio Code](#)
- [IntelliJ IDEA](#)

Platforms

It supports the following platforms:

- [Java](#) (see also [devon4j](#))
- [Node.js](#) (see also [devon4node](#))
- [Angular](#) (see also [devon4ng](#))
- [C#](#) (see also [devon4net](#))

Build-Systems

It supports the following build-systems:

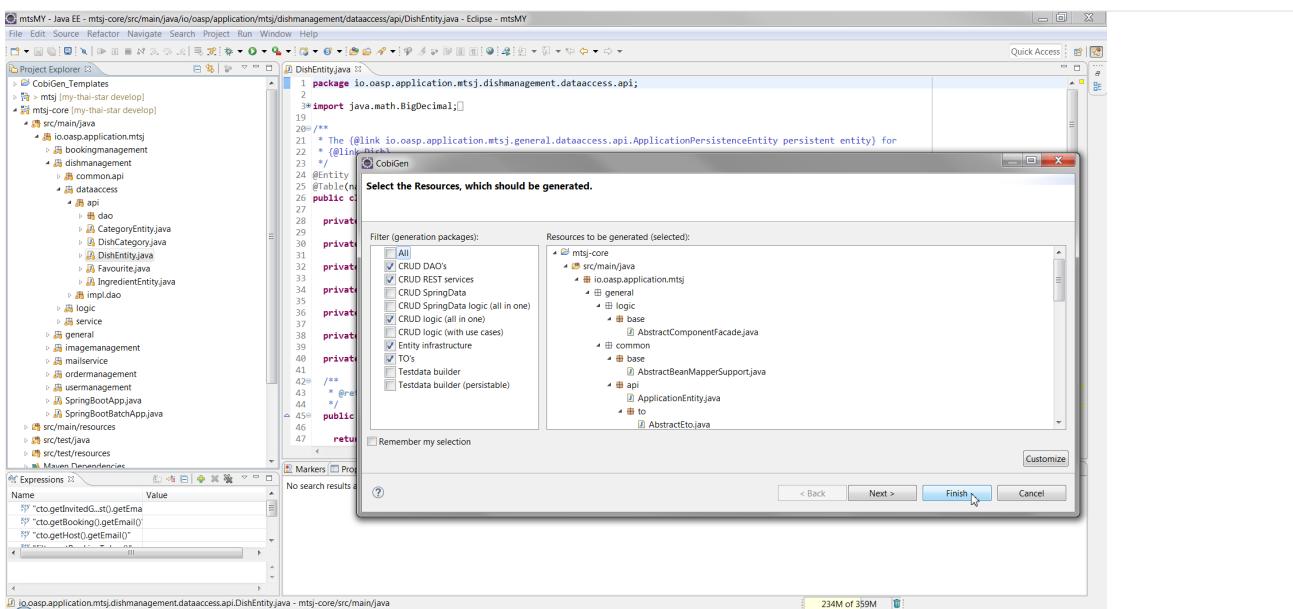
- [Maven](#)
- [NPM](#)
- [Gradle](#)



Other IDEs, platforms, or tools can easily be integrated as [commandlets](#).

CobiGen

[CobiGen](#) is a code generator included in the *devonfw-ide*, that allows users to generate the project structure and large parts of the application component code. This saves a lot of time, which is usually wasted on repetitive engineering tasks and/or writing boilerplate code.



Following the same philosophy as the devonfw-ide, *CobiGen* bundles a new command line interface (CLI), that enables the generation of code using only a few commands. This approach also allows us to decouple CobiGen from *Eclipse* and use it alongside *VS Code* or *IntelliJ IDEA*.

1.2. Why should I use devonfw?

devonfw aims to provide a framework for the development of web applications based on the Java EE programming model. It uses the Spring framework as its Java EE default implementation.

1.2.1. Objectives

Standardization

We don't want to keep reinventing the wheel for thousands of projects, for hundreds of customers, across dozens of countries. For this reason, we aim to rationalize, harmonize and standardize the development assets for software projects and industrialize the software development process.

Industrialization of Innovative Technologies & “Digital”

devonfw's goal is to standardize & industrialize. But this applies not only to large volume, “traditional” custom software development projects. devonfw also aims to offer a standardized platform which contains a range of state-of-the-art methodologies and technology stacks. devonfw supports agile development by small teams utilizing the latest technologies for projects related to Mobile, IoT and the Cloud.

Deliver & Improve Business Value



Efficiency

- Up to 20% reduction in time to market, with faster delivery due to automation and reuse.
- Up to 25% less implementation efforts due to code generation and reuse.
- Flat pyramid and rightshore, ready for junior developers.

Quality

- State-of-the-art architecture and design.
- Lower cost on maintenance and warranty.
- Technical debt reduction by reuse.
- Risk reduction due to continuous improvement of individual assets.
- Standardized, automated quality checks.

Agility

- Focus on business functionality, not on technicalities.
- Shorter release cycles.
- DevOps by design — Infrastructure as Code.
- Continuous Delivery pipeline.
- On- and off-premise flexibility.
- PoCs and prototypes in days not months.

1.2.2. Features

Everything in a Single ZIP

The devonfw distributions is packaged in a ZIP file that includes all the custom tools, software and configurations.

Having all the dependencies self-contained in the distribution's ZIP file, users don't need to install or configure anything. Just extracting the ZIP content is enough to have a fully functional *devonfw*.

devonfw — The Package

The devonfw platform provides:

- Implementation blueprints for a modern cloud-ready server and a choice on JS-Client technologies (either open source Angular or a very rich and impressive solution based on commercial Sencha UI).
- Quality documentation and step-by-step quick start guides.
- Highly integrated and packaged development environment based around Eclipse and Jenkins. You will be ready to start implementing your first customer-specific use case in 2h time.
- Iterative eclipse-based code-generator that understands "Java" and works on higher architectural concepts than Java-classes.
- An example application as a reference implementation.
- Support through a large community + industrialization services (Standard Platform as a Service) available in the iProd service catalog.

1.3. devonfw-ide Download and Setup

Please refer to our [devonfw-ide Setup](#) section.

2. Guides

Our goal is to provide a smooth starting experience to all users of *devonfw*, no matter how experienced they are or what their stakeholder role is. To achieve this, we provide a list of recommended guides here:

For Students and Junior Engineers:

- [Explore the devonfw sample application.](#)
- [Build your first application with **devon4j/devon4ng**.](#)
- [Create an application with **devon4node**.](#)
- [Create an application with **devon4net**.](#)

For Senior Engineers and Architects:

- [Explore the devonfw sample application.](#)

For Team Leaders and Product Ambassadors:

- [License management for OSS projects.](#)

2.1. Build Your First devonfw Application

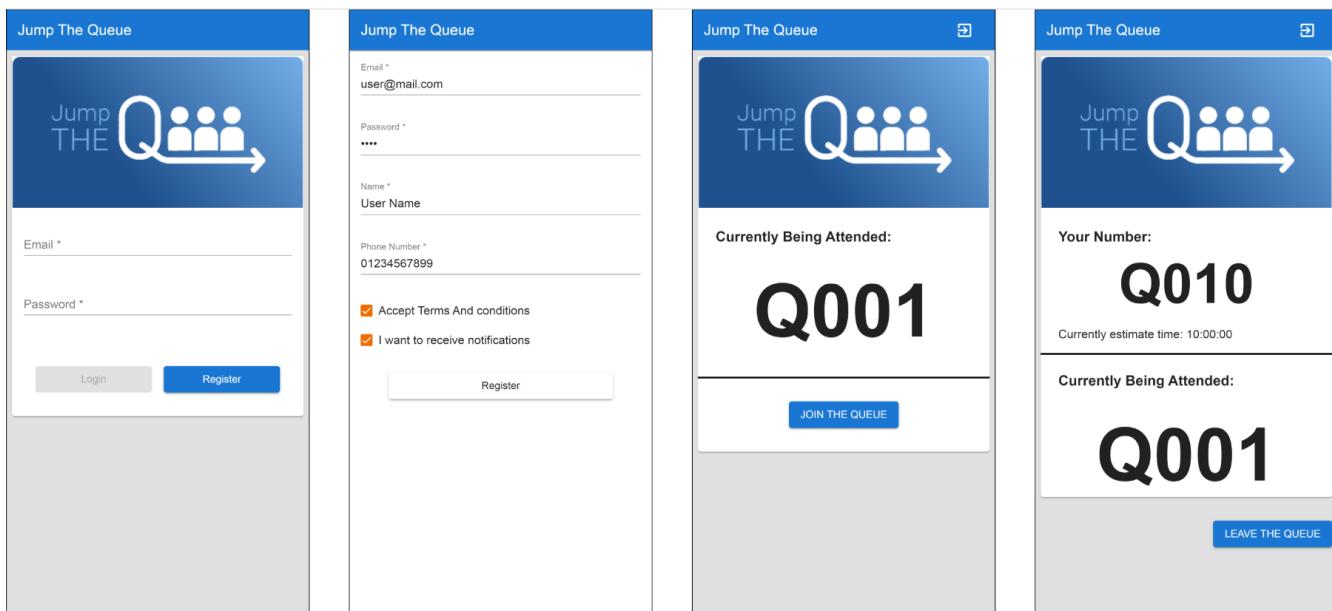
[JumpTheQueue](#) is a small application based on the devonfw framework, which you can create yourself by following our simple step-by-step tutorial. By doing so, you will learn about the app development workflow and gain insight into the design of a professional business information system. Please visit the [JumpTheQueue](#) wiki and start working through the tutorial [HERE](#).



The tutorial assumes you have successfully set up the [devonfw-ide](#) previously.

You can also clone the project and explore the finished source code via:

```
git clone https://github.com/devonfw/jump-the-queue.git
```



Another way to check out the JumpTheQueue-Application is to try our interactive katacoda scenario where you set up the application step by step.

[JumpTheQueue Katacoda Scenario](#)

2.2. Explore Our devonfw Sample Application

[MyThaiStar](#) is a complex sample app, that demonstrates the full capabilities of our framework. On this page we will describe how to download and launch the app on your system, so you can test the various functionalities it offers and explore its code.

You can also check out the interactive katacoda scenario for setting up and trying out the MyThaiStar-Application.

[MyThaiStar Katacoda Scenario](#)



We assume you have successfully set up the [devonfw-ide](#) previously.

1. In the root directory of a `devonfw-ide` directory, right click and select "**Open Devon CMD shell here**" from the Windows Explorer context menu. Then navigate to the main workspace and checkout the MyThaiStar Git repository like this:

```
cd workspaces/main
git clone https://github.com/devonfw/my-thai-star.git
```

2. Perform: `cd my-thai-star`
3. Execute: `devon eclipse ws-up`
4. Execute: `devon eclipse create-script`
5. Go to the root folder of the distribution and run `eclipse-main.bat`
6. In Eclipse navigate to `File > Import > Maven > Existing Maven Projects`, then import the cloned

project from your workspace by clicking the "Browse" button and selecting [/workspaces/my-thai-star/java/mtsji/](#).

- Run the backend by right-clicking `SpringBootApp.java` and selecting **Run as > Java Application** in the context menu. The backend will start up and create log entries in the Eclipse Console tab.



- Return to your command shell and perform: `cd angular`
- Execute: `npm install`
- Execute: `ng serve`
- Once started, the frontend will be available at localhost:4200/restaurant. Login with the username and password `waiter` and take a look at the various functionalities provided by MyThaiStar.

You should now take a look at both the front- and backend code and familiarize yourself with its structure and concepts, since most devonfw projects follow this exemplary implementation. Please visit the *architecture overview* pages of [devon4ng](#) and [devon4j](#) to learn more about the internal

workings of front- and backend.

3. Further Information

3.1. Repository Overview

The GitHub repositories within the [devonfw organization](#) contain the source code and documentation for official devonfw projects.



An overview of the [devonfw](#) organization repositories.

The most relevant repositories here are the individual devonfw technology stacks:

- [devon4j](#) (for Java)
- [devon4ng](#) (for Angular)
- [devon4net](#) (for .NET)
- [devon4node](#) (for Node.js)
- [devon4x](#) (for Xamarin)

Our framework also delivers a number of tools and plug-ins that aim to accelerate and streamline the development process, for example:

- [devonfw-ide](#) (for environment/workspace setup)
- [CobiGen](#) (for automated code generation)
- [devon4j-sonar-plugin](#) (for architecture validation)
- [MrChecker](#) (for E2E test automation)

We also provide educational material and reference implementations to aid new users and drive the adoption of our framework, for example:

- [JumpTheQueue](#) (simple, step-by-step app creation tutorial)
- [MyThaiStar](#) (well documented, complex reference application)
- [devonfw-shop-floor](#) (tools and tutorials for CI/CD efforts)

-
- [accelerated-solution-design](#) (work-methodology for faster app design)
 - and many more ...

Projects in early development and prototypes are located in the [devonfw forge](#) repository. They usually remain there until they are ready for broader release or use in production.

3.2. Links to our Community

We strive to foster an active, diverse and dynamic community around devonfw and are relying on modern collaboration tools to do so. Please note that some resources listed here might only be accessible to members or partners of Capgemini.

3.2.1. Microsoft Teams

The devonfw public channel is accessible to everyone who has a Microsoft Teams account. You can find the latest discussions on ongoing development topics here, as well as new commits and pull requests to our repos.

Join us to stay in the loop, and feel free to post your questions regarding devonfw here.



3.2.2. Yammer

Our corporate Yammer channel is accessible to Capgemini employees and members. If you are looking for information or feedback on current and planned projects regarding devonfw, we recommend you ask around here first.



3.2.3. E-Mail

You can reach our dedicated iCSD Support Team via e-mail at:

icsddevonfwsupport.apps2@capgemini.com

3.3. Contributing

Please refer to our [Contributing](#) section.

3.4. Code of Conduct

Please refer to our [Code of Conduct](#) section.

Part II: devonfw-ide

4. Introduction

devonfw provides a solution to building applications which combine best-in-class frameworks and libraries as well as industry proven practices and code conventions. It massively speeds up development, reduces risks and helps deliver better results.

This document contains the instructions for the tool `devonfw-ide` to set up and maintain your development tools including your favorite IDE (integrated development environment).

4.1. Features

Every developer needs great tools to work efficiently. Setting up these tools manually can be tedious and error-prone. Furthermore, some projects may require different versions and configurations of such tools. Especially configurations like code-formatters should be consistent within a project to avoid diff-wars.

The `devonfw-ide` will solve these issues. Here are the features you will find through `devonfw-ide`:

- **Efficient**

Set up your IDE within minutes tailored for the requirements of your project.

- **Automated**

Automate the setup and update, avoid manual steps and mistakes.

- **Simple**

KISS (Keep It Small and Simple), no native installers that globally mess your OS or tool-integrations that break with every release. Instead, use templates and simple shell scripts.

- **Configurable**

You can change the `configuration` depending on your needs. Furthermore, the `settings` contain configuration templates for the different tools (see `configurator`).

- **Maintainable**

For your project you should copy these `settings` to an own `git` repository that can be maintained and updated to manage the tool configurations during the project lifecycle. If you use github or gitlab every developer can easily suggest changes and improvements to these `settings` via pull/merge requests, which is easier to manage with big teams.

- **Customizable**

Do you need an additional tool you had never heard of before? Put it in the `software` folder of the `structure`. The `devon CLI` will then automatically add it to your `PATH` variable.

Further you can create your own `commandlet` for your additional tool. For closed-source tools you can create your own archive and distribute it to your team members as long as you care about the terms and licenses of these tools.

- **Multi-platform**

It works on all major platforms: Windows, Mac and Linux.

- **Multi-tenancy**

You can have several instances of the `devonfw-ide` "installed" on your machine for different projects with different tools, tool versions and configurations. You won't need to set up any physical installation nor changing your operating system. "Installations" of `devonfw-ide` do not

interfere with each other nor with other installed software.

- **Multiple Workspaces**

It supports working with different [workspaces](#) on different branches. You can create and update new workspaces with a few clicks. You can see the workspace name in the title-bar of your IDE so you do not get confused and work on the right branch.

- **Free**

The [devonfw-ide](#) is free just like everything from [devonfw](#). See [LICENSE](#) for details.

4.1.1. IDEs

We support the following IDEs:

- [Eclipse](#)
- [Visual Studio Code](#)
- [IntelliJ](#)

4.1.2. Platforms

We support the following platforms:

- [java](#) (see also [devon4j](#))
- [C#](#) (see [devon4net](#))
- [node.js](#), [angular](#) and [ionic](#) (see [devon4ng](#))

4.1.3. Build-Systems

We support the following build-systems:

- [mvn](#) (maven)
- [npm](#)
- [gradle](#)

However, also other IDEs, platforms, or tools can be easily integrated as [commandlet](#).

4.1.4. Motivation

TL;DR? Lets talk to developers a correct language. Here are some examples with [devonfw-ide](#):

```
[/]$ devon
You are not inside a devonfw-ide installation: /
[/$ cd /projects/devonfw
[devonfw]$ mvn
zsh: command not found: mvn
[devonfw]$ devon
devonfw-ide environment variables have been set for /projects/devonfw in workspace
main
[devonfw]$ mvn -v
Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5ffb20918da4f719f3; 2018-10-
24T20:41:47+02:00)
Maven home: /projects/devonfw/software/maven
Java version: 1.8.0_191, vendor: Oracle Corporation, runtime:
/projects/devonfw/software/java
Default locale: en_DE, platform encoding: UTF-8
OS name: "mac os x", version: "10.14.3", arch: "x86_64", family: "mac"
[devonfw]$ cd /projects/ide-test/workspaces/test/my-project
[my-project]$ devon
devonfw-ide environment variables have been set for /projects/ide-test in workspace
main
[my-project]$ mvn -v
Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5ffb20918da4f719f3; 2018-10-
24T20:41:47+02:00)
Maven home: /projects/ide-test/software/maven
Java version: 11.0.2, vendor: Oracle Corporation, runtime: /projects/ide-
test/software/jdk/Contents/Home
Default locale: en_DE, platform encoding: UTF-8
OS name: "mac os x", version: "10.14.3", arch: "x86_64", family: "mac"
[ide-test]$ devon eclipse
launching Eclipse for workspace test...
[my-project]$ devon build
[INFO] Scanning for projects...
...
[INFO] BUILD SUCCESS
```

This was just a very simple demo of what **devonfw-ide** can do. For further details have a look at our [CLI documentation](#).

Now you might ask:

- But I use Windows/Linux/MacOS//... - it works on all platforms!
- But how about Windows CMD or Power-Shell? - it works!
- But what if I use cygwin or git-bash on windows? - it works!
- But I love to use ConEmu or Commander - it works with full integration!
- How about MacOS Terminal or iTerm2? - it works with full integration!
- But I use zsh - it works!
- ...? - it works!

Wow! So let's get started with [download & setup](#).

4.2. Setup

4.2.1. Prerequisites

We try to make it as simple as possible for you. However, there are some minimal prerequisites:

- You need to have a tool to extract `*.tar.gz` files (`tar` and `gzip`). On Windows lower Windows 10 (1803) use [7-zip](#). On all other platforms this comes out of the box.
- You need to have `git` and `curl` installed.
 - On Windows you only need to download and install [git for windows](#). This also ships with `bash` and `curl`.
 - On Linux you might need to install the above tools in case they are not present (e.g. `sudo apt-get install git curl` or `sudo yum install git-core curl`)
 - On MacOS you only need to download and install [git for mac](#).

4.2.2. Download

The latest release of `devonfw-ide` can be downloaded from [here](#) (You can find all releases in [maven central](#)).

4.2.3. Install

Create a central folder like `C:\projects` or `/projects`. Inside this folder, create a sub-folder for your new project such as `my-project` and extract the contents of the downloaded archive (`devonfw-ide-scripts-*.*.tar.gz`) and send them to this new folder. Run the command `setup` in this folder (on windows double clicking on `setup.bat`). That's all. To get started read the [usage](#).

4.2.4. Uninstall

To "uninstall" your `devonfw-ide` you only need to call the following command:

```
devon ide uninstall
```

Then you can delete the `devonfw-ide` top-level folder(s) (`#{DEVON_IDE_HOME}`).

The `devonfw-ide` is designed to be **non-invasive** to your operating system and computer. Therefore it is not "installed" on your system in a classical way. Instead you just create a folder and extract the [downloaded](#) archive to it. You only have to install regularly in advance some specific prerequisites like `git`. All the other softwares remain locally in your `devonfw-ide` folder. However, there are the following excuses (what is reverted by `devon ide uninstall`):

- The `devon` command is copied to your home directory (`~/.devon/devon`)
- The `devon` alias is added to your shell config (`~/.bashrc` and `~/.zshrc`, search for `alias devon="source ~/.devon/devon"`).

- On Windows the `devon.bat` command is copied to your home directory (`%USERPROFILE%\scripts\devon.bat`)
- On Windows this `%USERPROFILE%\scripts` directory is added to the `PATH` of your user.
- The `devonfw-ide` will download a third party software to your `~/Downloads` folder to reduce redundant storage.

5. Usage

5.1. Roles

This section is designed to explain `devonfw-ide` depending on each role Everybody should read and follow the usage for `developer`. In case you want to administrate `devonfw-ide` settings for your project, you should also read the usage for the `admin`.

5.1.1. Developer

As a developer you are supported to `setup` your IDE automated and fast while you can have a nice cup of coffee (after you provided `settings-URL` and accepted the license). You only need the settings URL from your `devonfw-ide admin`. Experienced developers can directly call `setup <>settings-URL><`. Otherwise if you just call `setup` (e.g. by double-clicking it), you can enter it when you are prompted for `Settings URL` (using copy&paste to avoid typos).

Update

To update your IDE (if instructed by your `admin`), you only need to run the following command:

```
devon ide update
```

Please note that windows is using file-locking what can have ugly side-effects. To be safe, you should have your IDE tools shut down before invoking the above update command. E.g. if a tool needs to be updated, the old installation folder will be moved to a backup and the new version is installed on top. If there are windows file locks in place this can fail and mess up things. You can still delete the according installation from your `software` folder and rerun `devon ide update` if you ran into this error.

Working with multiple workspaces

If you are working on different branches in parallel you typically want to use multiple workspaces.

1. Go to the `workspaces` folder in your `${DEVON_IDE_HOME}` and create a new folder with the name of your choice (e.g. `release2.1`).
2. Check out (`git clone ...`) the according projects and branch into that workspace folder.
3. Open a shell in that new workspace folder (`cd` to it) and according to your IDE run e.g. `eclipse`, `vscode`, or `intellij` to create your workspace and launch the IDE. You can also add the parameter `create-script` to the IDE `commandlet` in order to create a launch-script for your IDE.

You can have multiple instances of `eclipse` running for each workspace in parallel. To distinguish these instances you will find the workspace name in the title of `eclipse`.

5.1.2. Admin

You can easily customize and `configure devonfw-ide` for the requirements of your project. In order

to do so, you need to create your own project-specific settings git repository and provide the URL to all developers for the [setup](#). With tools such as gitlab, bitbucket or github every developer can easily propose changes and improvements. However, we suggest that one team member is responsible to ensure that everything stays consistent and works. We will call this person the [devonfw-ide admin](#) of your project.

The following are the suggested step-by-step instructions how an [ide-admin](#) should prepare [devonfw-ide](#) for his new project:

1. Fork [ide-settings](#) to a git repository specific for your project (e.g. a new project in the [gitlab](#) of your [production-line](#) instance). In case you are using github, all you need to do is use the [Fork](#) button. In other cases simply create a new and empty git repository and clone this to your machine. Then add the default ide-settings as origin, fetch and pull from it:

```
git remote add upstream https://github.com/devonfw/ide-settings.git  
git fetch upstream  
git pull upstream master  
git push
```

Now you should have a full fork as a copy of the [settings](#) git repo with all its history that is ready for upstream merges.

2. Study the [structure](#) of this git repository to understand where to find which configuration.
3. Study the [configuration](#) and understand that general settings can be tweaked in the toplevel [devon.properties](#) file of your settings git repository.
4. Configure the tools and their versions for your project:

```
DEVON_IDE_TOOLS=(java mvn eclipse)  
ECLIPSE_VERSION=2020-06  
# use e.g. 8u242b08 for Java 8  
#JAVA_VERSION=8u242b08  
JAVA_VERSION=11.0.5_10  
MAVEN_VERSION=3.6.2
```

This way you will take over control of the tools and their versions for every developer in your project team and ensure that things get reproducible.

5. In case you need a proprietary or unsupported tool, you can study [how to include custom tools](#).
6. In case you have very restrictive policies about downloading tools from the internet, you can create and configure a [software repository](#) for your project or company.
7. Some of the tools (especially the actual IDEs) allow extensions via plugins. You can customize them to your needs for [eclipse](#), [VS code](#), or [intelliJ](#).
8. In your [settings](#) git repository you will find a [projects](#) folder. Here you will find configurations files for every git project relevant for your actual project. Feel free to create new projects for your needs and delete the [devonfw](#) specific default projects. The [projects](#) documentation will

explain you how to do this.

9. For every IDE you will also find an according folder in your `settings` git repository. Here are the individual configuration settings for that IDE. You can change them by directly editing the according configuration files directly with a text-editor in your `settings` git repository. However, this is a really complex way and will take you a lot of time to find the right file and property to tweak for your actual need. Instead we suggest to study [how to customize IDE specific settings](#).
10. You may also create new sub-folders in your `settings` git repository and put individual things according to your needs. E.g. you could add scripts for `greasemonkey` or `tampermonkey`, as well as scripts for your database or whatever may be useful and worth to share in your team. However, to share and maintain knowledge we recommend to use a wiki instead.
11. You may want to customize the [Eclipse spellchecker dictionary](#) for your project and your language.

All described in the above steps (except the first one) can be used to manage and update the configuration during the project lifecycle. However, when you have done changes especially in a larger project, please consider the following best-practices to avoid that a large teams gets blocked by a non-functional IDE:

- Commit your changes to a feature-branch.
- First test the changes yourself.
- If all works as expected, pick a pilot user of the team to test the changes from the feature branch (go to `settings` folder, `git fetch`, `git checkout -t origin/feature/<>name</>`, `devon ide update`).
- Only after that works well for a couple of days, inform the entire team to update.

Announce changes to your team

In order to roll out the perfectly configured `devonfw-ide` to your project initially or when new members join, you only have to provide the [Settings URL](#) to the [developers](#) of your team.

After you changed and tested your `settings` git repository (main branch), you only need to announce this to your [developers](#) (e.g. via email or some communication tool) so that they will can `devon ide update` and automatically get up-to-date with the latest changes (see [update](#)).

In case you want to go to a new version of `devonfw-ide` itself, [developers](#) have to call `devon ide update scripts`.

5.2. Configuration

The `devonfw-ide` aims to be highly configurable and flexible. The configuration of the `devon` command and environment variables takes place via `devon.properties` files. The following list shows these configuration files in the order they are loaded so files can override variables from files above in the list:

1. build in defaults (for `JAVA_VERSION`, `ECLIPSE_PLUGINS`, etc.)
2. `~/devon.properties` - user specific global defaults (on windows in `%USERPROFILE%/devon.properties`)
3. `scripts/devon.properties` - defaults provided by `devonfw-ide`. Never directly modify this file!
4. `devon.properties` - vendor variables for custom distributions of `devonfw-ide-scripts`, may e.g. tweak `SETTINGS_PATH` or predefine `SETTINGS_URL`.
5. `settings/devon.properties` (`${SETTINGS_PATH}/devon.properties`) - project specific configurations from `settings`.
6. `workspaces/${WORKSPACE}/devon.properties` - optional workspace specific configurations (especially helpful in projects using docker).
7. `conf/devon.properties` - user specific configurations (e.g. `M2_REPO=~/m2/repository`). During setup this file is created by copying a template from `${SETTINGS_PATH}/devon/conf/devon.properties`.
8. `settings/projects/*.properties`- properties to configure project checkout and import

5.2.1. devon.properties

The `devon.properties` files allow to define environment variables in a simple and OS independent way:

- `# comments begin with a hash sign (#) and are ignored`
- `variable_name=variable_value` with space etc.
- `variable_name=${predefined_variable}/folder_name`

variable values can refer to other variables that are already defined, which will be resolved to their value. You have to used `${...}` syntax to make it work on all platforms (never use `%...%`, `$...`, or `$(...)` syntax in `devon.properties` files).

- `export exported_variable=this value will be exported in bash, in windows CMD the export prefix is ignored`
- `variable_name=`

this will unset the specified variable

- `variable_name=~/some/path/and.file`

tilde is resolved to your personal home directory on any OS including windows.

- `array_variable=(value1 value2 value3)`

This will only work properly in `bash` worlds but as no arrays are used in `CMD` world of `devonfw-`

ide it does not hurt on windows.

- Please never surround values with quotes (`var="value"`)
- This format is similar to Java `*.properties` but does not support advanced features as unicode literals, multi-lined values, etc.

In order to know what to configure, have a look at the available [variables](#).

Please only tweak configurations that you need to change and take according responsibility. There is a price to pay for flexibility, which means you have to be careful what you do.

Further, you can configure `maven` via `conf/settings.xml`. To configure your IDE such as `eclipse` or `vscode` you can tweak the [settings](#).

5.3. Variables

The `devonfw-ide` defines a set of standard variables to your environment for [configuration](#) via `variables[.bat]` files. These environment variables are described by the following table. Those variables printed **bold** are also exported in your shell (except for windows CMD that does not have such concept). Variables with the value `-` are not set by default but may be set via [configuration](#) to override defaults. Please note that we are trying to minimize any potential side-effect from `devonfw-ide` to the outside world by reducing the number of variables and only exporting those that are required.

Table 1. Variables of devonfw-ide

Variable	Value	Meaning
<code>DEVON_IDE_HOME</code>	e.g. <code>/projects/my-project</code>	The top level directory of your devonfw-ide structure .
<code>PATH</code>	<code>\$PATH:\$DEVON_IDE_HOME/software/java:...</code>	Your system path is adjusted by devon command .
<code>DEVON_HOME_DIR</code>	<code>~</code>	The platform independent home directory of the current user. In some edge-cases (e.g. in cygwin) this differs from <code>~</code> to ensure a central home directory for the user on a single machine in any context or environment.
<code>DEVON_IDE_TOOLS</code>	<code>java mvn eclipse vscode node npm ng ionic</code>	List of tools that should be installed and upgraded by default for your current IDE.
<code>DEVON_IDE_CUSTOM_TOOLS</code>	<code>-</code>	List of custom tools that should be installed additionally. See software for further details.
<code>DEVON_OLD_PATH</code>	<code>...</code>	A "backup" of <code>PATH</code> before it was extended by devon to allow recovering it. Internal variable that should never be set or tweaked.
<code>WORKSPACE</code>	<code>main</code>	The workspace you are currently in. Defaults to <code>main</code> if you are not inside a workspace . Never touch this variable in any <code>variables</code> file.
<code>WORKSPACE_PATH</code>	<code>\$DEVON_IDE_HOME/workspaces/\$WORKSPACE</code>	Absolute path to current workspace . Never touch this variable in any <code>variables</code> file.
<code>JAVA_HOME</code>	<code>\$DEVON_IDE_HOME/software/java</code>	Path to JDK

Variable	Value	Meaning
SETTINGS_PATH	\$DEVON_IDE_HOME/settings	Path to your settings . To keep oasp4j-ide legacy behaviour set this to \$DEVON_IDE_HOME/workspaces/main/development/settings.
M2_REPO	\$DEVON_IDE_HOME/conf/.m2/repository	Path to your local maven repository. For projects without high security demands, you may change this to the maven default ~/.m2/repository and share your repository amongst multiple projects.
MAVEN_HOME	\$DEVON_IDE_HOME/software/maven	Path to Maven
MAVEN_OPTS	-Xmx512m -Duser.home=\$DEVON_IDE_HOME/conf	Maven options
DEVON_SOFTWARE_REPOSITORY	-	Project specific or custom software-repository .
DEVON_SOFTWARE_PATH	-	Globally shared user-specific local software installation location.
ECLIPSE_VMARGS	-Xms128M -Xmx768M -XX:MaxPermSize=256M	JVM options for Eclipse
deprecated: ECLIPSE_PLUGINS	-	Array with "feature groups" and "update site URLs" to customize required eclipse plugins . Deprecated - see Eclipse plugins .
<>TOOL>_VERSION	-	The version of the tool <>TOOL> to install and use (e.g. ECLIPSE_VERSION or MAVEN_VERSION).
<>TOOL>_BUILD_OPTS	e.g. clean install	The arguments provided to the build-tool <>TOOL> in order to run a build.
<>TOOL>_RELEASE_OPTS	e.g. clean deploy -Dchangelog=-Pdeploy	The arguments provided to the build-tool <>TOOL> in order to perform a release build.

Variable	Value	Meaning
DEVON_IDE_TRACE		If value is not an empty string, the <code>devonfw-ide</code> scripts will trace each script line executed. For bash two lines output: before and again after expansion. ATTENTION: This is not a regular variable working via <code>devon.properties</code> . Instead manually do <code>export DEVON_IDE_TRACE=true</code> in bash or <code>DEVON_IDE_TRACE=true</code> in windows CMD before running a devon command to get a trace log that you can provide to experts in order to trace down a bug and see what went wrong.

5.4. Devon CLI

The `devonfw-ide` is shipped with a central command `devon`. The `setup` will automatically register this command so it is available in any shell on your system. This page describes the Command Line Interface (CLI) of this command.

5.4.1. Devon

Without any argument the `devon` command will determine your `DEVON_IDE_HOME` and setup your `environment variables` automatically. In case you are not inside of a `devonfw-ide` folder the command will echo a message and do nothing.

```
[/]$ devon
You are not inside a devon IDE installation: /
[/$ cd /projects/my-project/workspaces/test/my-git-repo
[my-git-repo]$ devon
devonfw-ide has environment variables have been set for /projects/my-project in
workspace main
[my-git-repo]$ echo $DEVON_IDE_HOME
/projects/devon
[my-git-repo]$ echo $JAVA_HOME
/projects/my-project/software/java
```

5.4.2. Commandlets

The `devon` command supports a pluggable set of *commandlets*. Such commandlet is provided as first argument to the devon command and may take additional arguments:

```
devon <>commandlet<> [<>arg<>]*
```

Technically, a commandlet is a bash script located in `$DEVON_IDE_HOME/scripts/command`. So if you want to integrate another tool with `devonfw-ide` we are awaiting your pull-request.

The following commandlets are currently available:

- [build](#)
- [cobigen](#)
- [eclipse](#)
- [gradle](#)
- [help](#)
- [ide](#)
- [intellij](#)
- [ionic](#)
- [java](#)
- [jenkins](#)

- [mvn](#)
- [ng](#)
- [node](#)
- [npm](#)
- [release](#)
- [sonar](#)
- [vscode](#)
- [yarn](#)

5.4.3. build

The `build` commandlet is an abstraction of build systems like [maven](#), [gradle](#), [yarn](#), [npm](#), etc. It will auto-detect your build-system (via existence of files like `pom.xml`, `package.json`, etc.). According to this detection it will simply delegate to the according commandlet of the specific build system. If that build-system is not yet available it will be downloaded and installed automatically.

So `devon build` allows users to build any project without bothering about the build-system. Further specific build options can be configured per project. This makes `devon build` a universal part of every *definition of done*. Before pushing your changes, please always run the following command to verify the build:

```
devon build
```

You may also supply additional arguments as `devon build <>args</>`. This will simply delegate these arguments to the detected build command (e.g. call `mvn <>args</>`).

5.4.4. eclipse

The `eclipse` commandlet allows to install, configure, and launch the [Eclipse IDE](#). To launch Eclipse for your current workspace and devonfw-ide installation simply run: `devon eclipse`

You may also supply additional arguments as `devon eclipse <>args</>`. These are explained by the following table:

Table 2. Usage of `devon eclipse`

Argument(s)	Meaning
<code>--all</code>	if provided as first arg then to command will be invoked for each workspace
<code>setup</code>	setup Eclipse (install or update)
<code>add-plugin <>id</> [<>url</>]</code>	install an additional plugin
<code>run</code>	launch Eclipse (default if no argument is given)
<code>start</code>	same as <code>run</code>
<code>ws-up[<>date</>]</code>	update workspace

Argument(s)	Meaning
ws-re[verse]	reverse merge changes from workspace into settings
ws-reverse-add	reverse merge adding new properties
create-script	create launch script for this IDE, your current workspace and your OS

There are variables that can be used for Eclipse. These are explained by the following table:

Table 3. Variables of devonfw-ide for Eclipse

Variable	Default-Value	Meaning
ECLIPSE_VERSION	2020-12	The version of the tool Eclipse to install and use.
ECLIPSE_EDITION_TYPE	java	The edition of the tool Eclipse to install and use. You can choose between 'java' for standard edition or 'jee' for enterprise edition.

plugins

To be productive with Eclipse you need plugins. Of course `devonfw-ide` can automate this for you: In your `settings` git repository create a folder `eclipse/plugins` (click on this link to see more examples and see which plugins come by default). Here you can create a properties file for each plugin. This is an example `tmterminal.properties`:

```
plugin_url=http://download.eclipse.org/tm/terminal/marketplace
plugin_id=org.eclipse.tm.terminal.feature.feature.group,org.eclipse.tm.terminal.view.feature.feature.group,org.eclipse.tm.terminal.control.feature.feature.group,org.eclipse.tm.terminal.connector.ssh.feature.feature.group,org.eclipse.tm.terminal.connector.telnet.feature.feature.group
plugin_active=true
```

The variables are defined as follows:

- `plugin_url` defines the URL of the Eclipse update site of the plugin
- `plugin_id` defines the feature group ID(s) to install. To install multiple features/plugins provide a comma-separated list of IDs. If you want to customize `devonfw-ide` with new plugins you can first install them manually and then go to `About Eclipse > Installation Details` then you can filter for your newly installed plugin and find the values in the `Id` column. Copy & paste them from here to make up your own custom config.
- `plugin_active` is an optional parameter. If it is `true` (default) the plugin will be installed automatically during the project `setup` for all developers in your team. Otherwise, developers can still install the plugin manually via `devon eclipse add-plugin <>plugin-name<>` from the

config file [settings/eclipse/plugins/<>plugin-name>.properties](#). See the [settings/eclipse/plugins](#) folder for possible values of «plugin-name».

In general you should try to stick with the configuration pre-defined by your project. But some plugins may be considered as personal flavor and are typically not predefined by the project config. This e.g. applies for devstyle that allows a real dark mode for eclipse and tunes the theming and layout of Eclipse in general. Such plugins should be shipped with your [settings](#) as described above with `plugin_active=false` allowing you to easily install it manually.

As the maintainer of the [settings](#) for your project you should avoid to ship too many plugins that may waste resources but are not used by every developer. By configuring additional plugins with `plugin_active=false` you can give your developers the freedom to install some additional plugins easily.

legacy plugin config

For downward compatibility we still support the deprecated legacy configuration if the folder [settings/eclipse/plugins](#) does not exist: The project [configuration](#) typically defines the plugins that will be installed via [ECLIPSE_PLUGINS variable](#). Otherwise defaults from this [eclipse commandlet](#) will apply. Be aware that this comes at your own risk and sometimes plugins can conflict and break your IDE.

Here is an example how a project can configure the plugins in its [devon.properties](#) inside the [settings](#):

```
ECLIPSE_PLUGINS=("AnyEditTools.feature.group"
"https://raw.githubusercontent.com/iloveeclipse/plugins/latest/"
"com.ess.regexutil.feature.group" "http://regex-util.sourceforge.net/update/")
```

For the above listed plugins you can also use the short form:

```
ECLIPSE_PLUGINS=("anyedit" "" "regexutil" "")
```

Of course you may also mix plugin IDs with fully qualified plugins.

dictionary

Eclipse already comes with a build-in spellchecker. This is very helpful when writing comments. The default settings of [devonfw-ide](#) ship with a project specific [dictionary file](#) and according configurations to enable spellchecking and configuring this dictionary. When typing JavaDoc, inline comments or other texts the spellchecker will underline unknown words in red. If your cursor is located at such a word you can hit [\[Ctrl\]\[1\]](#) to get a context menu with additional options. There you can either choose similar correct words to correct a typo or you may even add the word (maybe a new business term) to your local dictionary.

```

    /**
     * @return the name of the property. By convention it should
     *         be a letter followed by alpha-numeric characters. The na
     *         dot character (.) is not allowed.
     */
    @Override
    String getName();
  
```

Change to 'alpha'

 Disable spell checking

 Ignore 'apha' during the current session

 Add 'apha' to dictionary

In the latter case, you should commit the changes to your [settings](#) so that it will be available to your entire team. For further details about committing changes to the settings please consult the [admin usage](#).

non-english dictionary

In case your project has to write documentation or text in languages other than English, you might want to prefill your project dictionary for that language. Here we collect a list of such dictionaries that you can download and merge into your project dictionary:

- German: <https://sourceforge.net/projects/germandict/> (has to be converted to UTF-8 e.g. with [Notepad++](#) via [Encoding > Convert to UTF-8](#))

5.4.5. gradle

The `gradle` commandlet allows to install, configure, and launch `gradle`. It is similar to `gradle-wrapper`. So calling `devon gradle <<args>>` is more or less the same as calling `gradle <<args>>` but with the benefit that the version of gradle preferred by your project is used (and will be installed if not yet available).

The arguments (`devon gradle <<args>>`) are explained by the following table:

Table 4. Usage of devon gradle

Argument(s)	Meaning
<code>setup</code>	setup gradle (install and verify), configurable via <code>GRADLE_VERSION</code>
<code><<args>></code>	run gradle with the given arguments (<code><<args>></code>)

5.4.6. help

The `help` commandlet provides help for the [CLI](#).

Table 5. Usage of devon help

Argument(s)	Meaning
	Print general help
<code><<command>></code>	Print help for the commandlet <code><<command>></code> .

Please note that `devon help <<command>>` will do the same as `devon <<command>> help`.

5.4.7. ide

The `ide` commandlet manages your `devonfw-ide`. You need to supply additional arguments as `devon ide <args>`. These are explained by the following table:

Table 6. Usage of `devon ide`

Argument(s)	Meaning
<code>setup [<SETTINGS_URL>]</code>	setup devonfw-ide (cloning the settings from the given URL)
<code>update [<package>]</code>	update devonfw-ide
<code>update scripts [to <version>]</code>	update devonfw-ide
<code>uninstall</code>	uninstall devonfw-ide (if you want remote it entirely from your system)

setup

Run `devon ide setup` to initially setup your `devonfw-ide`. It is recommended to run the `setup` script in the top-level directory (`$DEVON_IDE_HOME`). However, in case you want to skip some system specific integrations, you may also run this command directly instead. The setup only needs to be called once after a new `devonfw-ide` instance has been created. It will follow this process:

- `install` the `devon` command on your system (if not already installed).
- clone the `settings` (you may provide a git URL directly as argument or you will be prompted for it).
- install all required `software` from `DEVON_IDE_TOOLS` variable (if not already installed).
- configure all these tools
- create IDE launch scripts
- perform OS specific system integration such as WindowsExplorer integration (only done from `setup` script and not from `devon ide setup`)

update

Run `devon ide update` to update your `devonfw-ide`. This will check for updates and `install` them automatically. The optional extra argument (`<package>`) behaves as follows:

- `scripts`: check if a new version of `devonfw-ide-scripts` is available. If so it will be downloaded and installed. As Windows is using file-locks, it is tricky to update a script while it is executed. Therefore, we update the `scripts` folder as an async background task and have to abort further processing at this point on windows as a workaround.
- `settings`: update the `settings` (`git pull`).
- `software`: update the `software` (e.g. if versions have changed via `scripts` or `settings` update).
- `projects`: update the `projects` (checkout and import repositories into workspace/IDEs).
- `all`: do all the above sequentially.

- none: `settings` and `software` are updated by default if no extra argument is given. This is the regular usage for project developers. Only perform an update of `scripts` when you are requested to do so by your technical lead. Bigger projects especially need to test updates before rolling them out to the entire team. If developers always updated the latest release of the `scripts` which is released globally, some project functionality would break causing problems and extra efforts in the teams.

In order to update to a specific version of `scripts` an explicit version can be specified after the additional `to` argument:

```
devon ide update scripts to 3.1.99
```

The above example will update to the exact version `3.1.99` no matter if this is an upgrade or a downgrade of your current installed version. If you just use `devon ide update scripts` then the latest available version will be installed. In larger teams it is recommended to communicate exact version updates to avoid that a new release can interfere and break anything. Therefore, some pilot user will test a new version for the entire team and, only after a successful test, they will communicate to the team to update to that exact version by providing the complete command as in the above example.

uninstall

We hope you love `devonfw-ide`. However, if you don't and want to get rid of it entirely and completely remove all integrations, you can use this command:

```
devon ide uninstall
```

This will remove `devonfw-ide` from all central places of your OS (user home directory such as `scripts`, `.devon`, `.bashrc`, as well as windows registry, etc.). However, it will not remove your current installations (or shared `software` folder). So after running this `uninstall`, simply remove your `DEVON_IDE_HOME` directory of all `devonfw-ide` installations and potential shared `software` folder. You may also want to clean up your `~/Downloads` directory from files downloaded by `devonfw-ide`. We do not automate this as deleting a directory is a very simple manual step and we do not want to take responsibility for severe data loss if your workspaces contained valuable work.

5.4.8. intelliJ

The `intelliJ` commandlet allows to install, configure, and launch `IntelliJ`. To launch IntelliJ for your current workspace and `devonfw-ide` installation, simply run: `devon intelliJ`

You may also supply additional arguments as `devon intelliJ <args>`. These are explained by the following table:

Table 7. Usage of `devon intelliJ`

Argument(s)	Meaning
--all	if provided as first arg then to command will be invoked for each workspace
setup	setup IntelliJ (install or update)
run	launch IntelliJ (default if no argument is given)
start	same as run
ws-up[date]	update workspace
ws-re[verse]	reverse merge changes from workspace into settings
ws-reverse-add	reverse merge adding new properties
create-script	create launch script for this IDE, your current workspace and your OS

There are variables that can be used for IntelliJ. These are explained by the following table:

Table 8. Variables of devonfw-ide for intelliJ

Variable	Default-Value	Meaning
INTELLIJ_VERSION	latest_tested_version	The version of the tool IntelliJ to install and use.
INTELLIJ_EDITION_TYPE	C	The edition of the tool IntelliJ to install and use. The value C mean Community edition and the value U mean Ultimate edition. The Ultimate edition requires a license. The user has to buy the license separately and it is not part of devonfw-ide. The devonfw-ide only supports download and installation.

5.4.9. ionic

The `ionic` commandlet allows to install, configure, and launch `ionic` (`ionic-cli`). Calling `devon ionic <args>` is more or less the same as calling `ionic <args>` but with some advanced features and ensuring that `ionic` is properly set up for your project.

The arguments (`devon ionic <args>`) are explained by the following table:

Table 9. Usage of devon ionic

Argument(s)	Meaning
setup	setup yarn (install and verify), <code>configurable</code> via <code>YARN_VERSION</code>

Argument(s)	Meaning
<code>create</code>	Create a new devon4ng ionic project .
<code>cicd <>args></code>	generate cicd files for the current devon4ng project
<code><>args></code>	run ionic with the given arguments (<code><>args></code>)

5.4.10. java

The `java` commandlet allows to install and setup [Java](#). Also it supports [devon4j](#). The arguments (`devon java <>args>`) are explained by the following table:

Table 10. Usage of `devon java`

Argument(s)	Meaning
<code>setup</code>	setup OpenJDK (install or update and verify), configurable via <code>JAVA_VERSION</code> (e.g. <code>8u242b08</code> or <code>11.0.6_10</code>)
<code>create <>args></code>	create a new Java project based on devon4j application template . If a single argument is provided, this is the package name and is automatically split into groupId and artifactId. Use <code>-DdbType=<>db></code> to choose the database (hana, oracle, mssql, postgresql, mariadb, mysql, h2, hsqldb). Any option starting with dash is passed as is."
<code>migrate [from <>version>] [single]</code>	migrate a devon4j project to the latest version. If for some reasons the current devonfw version (e.g. oasp4j:2.6.0) can not be auto-detected you may provide it manually after the 'from' argument. Also the 'single' option allows to migrate only to the next available version."
<code>cicd <>args></code>	generate cicd files for the current devon4java project

create

Examples for create a new devon4j application:

```
devon java create com.example.domain.myapp
```

Will create an app with package `com.example.domain.myapp`, groupId `com.example.domain`, artifactId `myapp`, version `1.0.0-SNAPSHOT`, and h2 database.

```
devon java create -Dversion=0.0.1-alpha1 com.example.domain.myapp
```

Will create an app with package `com.example.domain.myapp`, groupId `com.example.domain`, artifactId `myapp`, version `0.0.1-alpha1`, and h2 database.

```
devon java create com.example.domain.myapp com.example.group
```

Will create an app with package `com.example.domain.myapp`, groupId `com.example.group`, artifactId `myapp`, version `1.0.0-SNAPSHOT`, and h2 database.

```
devon java create com.example.domain.myapp com.example.group demo-app
```

Will create an app with package `com.example.domain.myapp`, groupId `com.example.group`, artifactId `demo-app`, version `1.0.0-SNAPSHOT`, and h2 database.

```
devon java create com.example.domain.myapp -DartifactId=demo-app -DdbType=hana
```

Will create an app with package `com.example.domain.myapp`, groupId `com.example.group`, artifactId `demo-app`, version `1.0.0-SNAPSHOT`, and SAP hana database.

```
devon java create com.example.domain.myapp -DdbType=oracle -Dversion=0.0.1  
com.example.group -Dbatch=batch
```

Will create an app with package `com.example.domain.myapp`, groupId `com.example.group`, artifactId `myapp`, version `0.0.1`, oracle database, and with a batch module.

migrate

Example for migrating a devon4j application:

```
devon java migrate
```

Will migrate current devon4j application to the latest version available.

5.4.11. jenkins

The `jenkins` commandlet allows to install, configure, and launch [Jenkins](#).

Table 11. Usage of devon jenkins

Argument(s)	Meaning
<code>setup</code>	Setup Jenkins (install and verify)

Argument(s)	Meaning
start	Start your local Jenkins server
stop	Stop your local Jenkins server
add	Add current project as CI job to your local Jenkins

5.4.12. mvn

The `mvn` commandlet allows to install, configure, and launch `maven`. It is similar to `maven-wrapper` and `mdub`. So calling `devon mvn <>args>` is more or less the same as calling `mvn <>args>` but with the benefit that the version of maven preferred by your project is used (and will be installed if not yet available).

The arguments (`devon mvn <>args>`) are explained by the following table:

Table 12. Usage of `devon mvn`

Argument(s)	Meaning
	run default build, <code>configurable</code> via <code>MVN_BUILD_OPTS</code>
setup	setup Maven (install and verify), <code>configurable</code> via <code>MAVEN_VERSION</code>
get-version	Print the version of your current project. Will consolidate the version for multi-module projects ignoring <code>dev[-SNAPSHOT]</code> versions and fail on mixed versions.
set-version <nv> [<cv>]	Set the version of your current project to <code><>nv></code> (assuming your current version is <code><>cv></code>).
check-no-snapshots	Check if no <code><>version>-SNAPSHOT</code> dependencies are used.
check-top-level-project	Check if you are running on a top-level project or fail if in a module or no maven project at all.
release	Start a clean deploy release build, <code>configurable</code> via <code>MVN_RELEASE_OPTS</code>
<>args>	run maven with the given arguments (<code><>args></code>)

5.4.13. ng

The `ng` commandlet allows to install, configure, and launch `ng` (angular-cli). Calling `devon ng <>args>` is more or less the same as calling `ng <>args>` but with some advanced features and ensuring that `ng` is properly set up for your project.

The arguments (`devon ng <>args>`) are explained by the following table:

Table 13. Usage of `devon ng`

Argument(s)	Meaning
setup	setup yarn (install and verify), configurable via NG_VERSION
create	Create a new devon4ng project.
cicd <>args>	generate cicd files for the current devon4ng project
<>args>	run ng with the given arguments (<>args>)

5.4.14. node

The `node` commandlet allows to install and setup [node.js](#). The arguments (`devon node <>args>`) are explained by the following table:

Table 14. Usage of `devon node`

Argument(s)	Meaning
setup	setup node.js (install and verify), configurable via NODE_VERSION
create <>name> [<>args>]	create a new devon4node application (same as devon4node new)
generate <>s> [<>args>]	generate devon4node components using the schematic <>s> (same as devon4node generate)
db <>c> [<>args>]	execute a TypeORM command <>c> (same as devon4node db)
cicd <>args>	generate cicd files for the current devon4node project
<>args>	call NodeJS with the specified arguments

5.4.15. npm

The `npm` commandlet allows to install, configure, and launch [npm](#). Calling `devon npm <>args>` is more or less the same as calling `npm <>args>` but with the benefit that the version of npm preferred by your project is used (and will be installed if not yet available).

The arguments (`devon npm <>args>`) are explained by the following table:

Table 15. Usage of `devon npm`

Argument(s)	Meaning
	run default build, configurable via NPM_BUILD_OPTS
setup	setup NPM (install and verify), configurable via NPM_VERSION
get-version	print the version of your current project

Argument(s)	Meaning
<code>set-version <>nv<> [<>cv<>]</code>	set the version of your current project to <code><>nv<></code> (assuming your current version is <code><>cv<></code>)
<code>check-top-level-project</code>	check if you are running on a top-level project or fail if in a module or no NPM project at all
<code>release</code>	Start a clean deploy release build, configurable via <code>NPM_RELEASE_OPTS</code>
<code><>args<></code>	run NPM with the given arguments (<code><>args<></code>)

5.4.16. release

Create a release in a standardized way including the following steps:

- verify the current project (no local changes, etc.)
- warn if `<>version<>-SNAPSHOT` dependencies are used
- determine `<>version<>` (if currently `<>version<>-SNAPSHOT`) and print out release information.
- ask user for confirmation
- bump release to `<>version<>` in build configuration (e.g. `pom.xml` files)
- commit the change
- create annotated tag for your release as `release/<>version<>`
- invoke deployment on build-system
- set next version as `(<>version<>+1)-SNAPSHOT` in build configuration (e.g. `pom.xml` files)
- commit the change
- push your changes

Table 16. Usage of `devon java`

Argument(s)	Meaning
<code>...</code>	any optional argument will directly be passed to the actual command to build the deployment

Build-Tools

This `release` commandlet utilizes the `build` commandlet to support multiple build-tools such as `maven`, `gradle`, or `npm`. Each of those commandlets should respect the variable `<>RELEASE_OPTS` to customize the parameters for the release build.

So e.g. if a `pom.xml` is detected, maven will be used. In this example the variable `MVN_RELEASE_OPTS` is used that defaults to `clean deploy -Dchangelog= -Pdeploy`. If you provide a specific argument this will be passed additionally. So if you invoke the command `devon release -P myProfile`, the above step `invoke deployment on build-system` would technically call this:

```
mvn clean deploy -Dchangelog= -Pdeploy -P myProfile
```

Please also note that it is very tricky to determine and modify the version of a project in a fully generic way. Even though we try our best to support different scenarios, we can not ensure this is working for edge-cases. Therefore, we strongly encourage to follow best practices such as [ci-friendly maven](#). Further, sticking to the defaults and follow the devonfw standard to name the profile for custom goals in deployment simply `deploy` is recommended.

5.4.17. sonar

The `sonar` commandlet allows to install, configure, and launch [SonarQube](#).

Table 17. Usage of devon sonar

Argument(s)	Meaning
<code>setup</code>	Setup SonarQube (install and verify)
<code>start</code>	Start your local SonarQube server
<code>stop</code>	Stop your local SonarQube server
<code>analyze</code>	Analyze current project with SonarQube

5.4.18. vscode

The `vscode` commandlet allows to install, configure, and launch [Visual Studio Code](#). To launch VSCode for your current workspace and `devonfw-ide` installation, simply run: `devon vscode`

You may also supply additional arguments as `devon vscode <args>`. These are explained by the following table:

Table 18. Usage of devon vscode

Argument(s)	Meaning
<code>--all</code>	if provided as first arg then to command will be invoked for each workspace
<code>setup</code>	setup VSCode (install or update)
<code>run</code>	launch VSCode (default if no argument is given)
<code>start</code>	same as <code>run</code>
<code>ws-up[date]</code>	update workspace
<code>ws-re[verse]</code>	reverse merge changes from workspace into settings
<code>ws-reverse-add</code>	reverse merge adding new properties
<code>create-script</code>	create launch script for this IDE, your current workspace and your OS

plugins

To be productive with VS Code you need plugins (called **extensions** in VS Code). Of course **devonfw-ide** can automate this for your: In your **settings** git repository create a folder **vscode/plugins** (click this link to see more examples and see which plugins come by default). Here you can create a properties file for each plugin. This is an example **devonfw-extension-pack.properties**:

```
plugin_id=devonfw.devonfw-extension-pack
plugin_active=true
```

The variables are defined as following:

- **plugin_id** defines the unique ID of the plugin to install. If you want to customize **devonfw-ide** with new plugins click on **Extensions** at the bottom of the left navigation icon bar in VS code. Then use the search to find the plugin of your choice. If you click on it the plugin ID is displayed in grey beside the official title at the top of the plugin details page. Copy & paste the ID from here to make up your own custom config.
- **plugin_active** is an optional parameter. If it is **true** (default) the plugin will be installed automatically during the project **setup** for all developers in your team. Otherwise developers can still install the plugin manually via **devon vscode add-plugin <>plugin-name<>** from the config file **settings/vscode/plugins/<>plugin-name<>.properties**. See the **settings/vscode/plugins** folder for possible values of «**plugin-name**».

In general you should try to stick with the configuration pre-defined by your project. But some plugins may be considered as personal flavor and are typically not predefined by the project config. Such plugins should be shipped with your **settings** as described above with **plugin_active=false** allowing you to easily install it manually. Surely, you can easily add plugins via the UI of VS code. However, be aware that some plugins may collect sensitive data or could introduce other vulnerabilities. So consider the governance of your project and talk to your technical lead before installing additional plugins that are not pre-defined in your **settings**.

As maintainer of the **settings** for your project you should avoid to ship too many plugins that may waste resources but are not used by every developer. By configuring additional plugins with **plugin_active=false** you can give your developers the freedom to install some additional plugins easily.

cleaning plugins on update

If you want to strictly manage the plugins for **VS code** in your project, you can create or edit the file **settings/vscode/plugins** in your **settings** and add this variable:

```
clean_plugins_on_update=true
```

This will wipe all plugins when an update of **VS code** is performed (e.g. via **devon ide update**) and reinstall all configured plugins. While this gives you more control over the governance of the plugins and allows to remove a plugin later during project lifecycle. However, this will delete all manually installed plugins automatically without asking.

5.4.19. `yarn`

The `yarn` commandlet allows to install, configure, and launch `npm`. Calling `devon yarn <args>` is more or less the same as calling `yarn <args>` but with the benefit that the version of npm preferred by your project is used (and will be installed if not yet available).

The arguments (`devon yarn <args>`) are explained by the following table:

Table 19. Usage of `devon yarn`

Argument(s)	Meaning
	run default build, <code>configurable</code> via <code>YARN_BUILD_OPTS</code>
<code>setup</code>	setup yarn (install and verify), <code>configurable</code> via <code>YARN_VERSION</code>
<code>get-version</code>	print the version of your current project
<code>set-version <nv> [<cv>]</code>	set the version of your current project to <code><nv></code> (assuming your current version is <code><cv></code>)
<code>check-top-level-project</code>	check if you are running on a top-level project or fail if in a module or no NPM project at all
<code>release</code>	start a clean deploy release build, <code>configurable</code> via <code>YARN_RELEASE_OPTS</code>
<code><args></code>	run yarn with the given arguments (<code><args></code>)

5.5. Structure

The directory layout of your `devonfw-ide` will look like this:

Listing 1. File structure of your devonfw-ide

```
/ projects (or C:\Projects, etc.)
└── / my-project ($DEVON_IDE_HOME)
    ├── / conf
    ├── / log
    ├── / scripts
    ├── / settings
    ├── / software
    ├── / system
    ├── / updates
    ├── / workspaces
    ├── setup
    ├── setup.bat
    └── devon-ide-doc.pdf
```

The elements of the above structure are described in the individual sections. As they are hyperlinks you can simply click on them to get more details.

5.5.1. conf

This folder contains configurations for your IDE:

Listing 2. File structure of the conf folder

```
/ conf
├── / .m2
│   ├── / repository
│   │   ├── / ant
│   │   ├── / ...
│   │   └── / zw
│   ├── settings-security.xml
│   └── settings.xml
├── / .sonar
├── / ...
└── variables
```

The `.m2` folder is used for configurations of `maven`. It contains the local `repository` folder used as cache for artifacts downloaded and installed by maven (see also [maven repositories](#)). Further, there are two configuration files for maven:

- `settings.xml` initialized from a template from your `devonfw-ide` [settings](#). You may customize this to your needs (configuring HTTP proxies, credentials, or other user-specific settings).
- `settings-security.xml` is auto-generated for you by `devonfw-ide` with a random password. This should make it easier for `devonfw-ide` users to use [password encryption](#) and never add

passwords in plain text for better security.

Finally, there is a file **variables** for the user-specific **configuration** of **devonfw-ide**.

5.5.2. log

The log directory is used to store log files e.g. for the **IDE configurator**. You may look here for debug information if something goes wrong.

5.5.3. scripts

This directory is the heart of the **devonfw-ide** and contains the required **scripts**.

Listing 3. File structure of the conf folder

```
/scripts
├── / command
│   ├── build
│   ├── eclipse
│   ├── gradle
│   ├── help
│   ├── ide
│   ├── intellij
│   ├── ionic
│   ├── java
│   ├── jenkins
│   ├── mvn
│   ├── ng
│   ├── node
│   ├── npm
│   ├── project
│   ├── release
│   ├── sonar
│   ├── vscode
│   └── yarn
└── devon
    ├── devon.bat
    ├── environment-project
    ├── environment-project.bat
    ├── functions
    └── devon.properties
```

The **command** folder contains the **commandlets**. The **devon** script is the key **command line interface** for **devonfw-ide**. There is also **devon.bat** that can be used in CMD or PowerShell. As the **devon CLI** can be used as a global command on your computer from any directory and gets **installed** centrally, it aims to be stable, minimal, and lightweight. The key logic to set up the environment variables is therefore in a separate script **environment-project** and its Windows variant **environment-project.bat** inside this **scripts** folder. The file **functions** contains a collection of reusable bash functions. These are sourced and used by the **commandlets**. Finally the **devon.properties** file contains defaults for the general **configuration** of **devonfw-ide**.

5.5.4. settings

The `devonfw-ide` requires `settings` with configuration templates for the arbitrary tools.

To get an initial set of these settings we provide the default `ide-settings` as an initial package. These are also released so you can download the latest stable or any history version at [maven central](#).

To test `devonfw-ide` or for very small projects you can also use these the latest default settings (just hit return when `setup` is asking for the [Settings URL](#)). However, for collaborative projects we strongly encourage you to distribute and maintain the settings via a dedicated and project specific `git` repository. This gives you the freedom to control and manage the tools with their versions and configurations during the project lifecycle. Therefore simply follow the [admin usage guide](#).

Structure

The settings folder (see `SETTINGS_PATH`) has to follow this file structure:

Listing 4. File structure of settings

```
/settings
  └── / devon
      ├── / conf
      │   ├── / .m2
      │   │   └── settings.xml
      │   ├── / npm
      │   │   └── .npmrc
      │   └── devon.properties
  └── / eclipse
      ├── / workspace
      │   ├── / setup
      │   └── / update
      └── lifecycle-mapping-metadata.xml
      └── project.dictionary
  └── ...
  └── / sonarqube
      └── / profiles
          ├── Devon-C#.xml
          ├── ...
          └── Devon-XML.xml
  └── / vscode
      └── / workspace
          ├── / setup
          └── / update
      └── devon.properties
```

As you can see, the `settings` folder contains sub-folders for tools of the IDE. So the `devon` folder contains `devon.properties` files for the `configuration` of your environment. Further, for the IDEs such as `eclipse` or `vscode`, the according folders contain the templates to manage the workspace via our `configurator`.

Configuration Philosophy

Different tools and configuration files require a different handling:

- Where suitable, we directly use these configurations from your `settings` (e.g. for `eclipse/lifecycle-mapping-metadata.xml`, or `eclipse/project.dictionary`).
- The `devon` folder in `settings` contains templates for configuration files. There are copied to the `devonfw-ide` installation during `setup` (if no such file already exists). In this way the `settings` repository can provide reasonable defaults but allows the user to take over control and customize to his personal needs (e.g. `.m2/settings.xml`).
- Other configurations need to be imported manually. To avoid manual steps and simplify use we try to automate as much as possible. This currently applies to `sonarqube` profiles but will be automated with `sonar-devon4j-plugin` in the future.
- For tools with complex configuration structures like `eclipse`, `intellij`, or `vscode` we provide a smart mechanism via our `configurator`.

Customize Settings

You can easily customize these settings for the requirements of your project. We suggest that one team member is responsible to ensure that everything stays consistent and works.

You may also create new sub-folders in `settings` and put individual items according to your needs. E.g. you could add scripts for `greasemonkey` or `tampermonkey`, as well as scripts for your database or whatever may be useful and worth to share in your team. However, to share and maintain knowledge we recommend to use a wiki.

5.5.5. software

The `software` folder contains the third party tools for your IDE such as `maven`, `npm`, `java`, etc. With respect to the `licensing terms` you may create a custom archive containing a `devonfw-ide` together with the required software. However, to be platform independent and allow lightweight updates, the `devonfw-ide` is capable to download and `install` the software automatically for you.

Repository

By default, software is downloaded via the internet from public download URLs of the according tools. However, some projects may need specific tools or tool versions that are not publicly available. In such case, they can create their own software repository (e.g. in a VPN) and `configure` the base URL of it via `DEVON_SOFTWARE_REPOSITORY` variable. Then, `devonfw-ide` will download all software from this repository only instead of the default public download URLs. This repository (URL) should be accessible within your network via HTTPS (or HTTP) and without any authentication. The repository needs to have the following structure:

```
${DEVON_SOFTWARE_REPOSITORY}/<>tool<>/<>version<>/<>tool<>-<>version<>[-<>os<>].tgz
```

So for every tool `<>tool<>` (`java`, `maven`, `vscode`, `eclipse`, etc.) you need to provide a folder in your repository. Within this folder for every supported version `<>version<>` you need a subfolder. This

subfolder needs to contain the tool in that version for every operating system <>os>> ([windows](#), [linux](#), or [mac](#) - omitted if platform independent, e.g. for [maven](#)).

Shared

By default, each installation of [devonfw-ide](#) has its own physical installations of the required tools in the desired versions stored in its local [software](#) folder. While this is great for isolation of [devonfw-ide](#) installations and to prevent side-effects, it can cause a huge waste of disc resources in case you are having many installations of [devonfw-ide](#). If you are a power-user of [devonfw-ide](#) with more than ten or even up to hundreds of installations on your machine, you might love to share installations of a software tool in a particular version between multiple [devonfw-ide](#) installations.



If you use this power-feature you are taking responsibility for side-effects and should not expect support. Also if you are using Windows please read [Symlinks in Windows](#) and make your mind if you really want to do so. You might also use this [hint](#) and maintain it manually without enabling the following feature.

In order to do so, you only need to [configure](#) the variable `DEVON_SOFTWARE_PATH` in your `~/devon.properties` pointing to an existing directory on your disc (e.g. `/projects/software` or `C:\projects\software`). Then [devonfw-ide](#) will install required software into `${DEVON_SOFTWARE_PATH}/${{software_name}}/${{software_version}}` as needed and create a symbolic link to it in `${DEVON_IDE_HOME}/software/${{software_name}}`.

As a benefit, another [devonfw-ide](#) installation will use the same software with the same version can re-use the existing installation and only needs to create the symbolic link. No more waste of having many identical JDK installations on your disc.

As a drawback, you need to be aware that specific tools may be "manipulated" after installation. The most common case is that a tool allows to install plugins or extensions such as all IDEs do. Such "manipulations" will cause side-effects between the different [devonfw-ide](#) installations sharing the same version of that tool. While this can also be a benefit it may also cause trouble. If you have a sensitive project that should not be affected by such side-effects, you may again override the `DEVON_SOFTWARE_PATH` variable to the empty value in your `${DEVON_IDE_HOME}/conf/devon.properties` of that sensitive installation:

```
DEVON_SOFTWARE_PATH=
```

This will disable this feature particularly for that specific sensitive [devonfw-ide](#) installation but let you use it for all other ones.

Custom

In some cases, a project might need a (proprietary) tool(s) that (are) not supported by [devonfw-ide](#). A very simple solution is to get a release of [devonfw-ide](#) and add the tool(s) to the software folder and then distribute this modified release to your team. However, this has several drawbacks as you then have a fork of [devonfw-ide](#) all will lose your tool(s) when updating to a new release.

As a solution for this need, [devonfw-ide](#) lets you configure custom tools via the

DEVON_IDE_CUSTOM_TOOLS variable. It can be defined in `devon.properties` of your `settings` git repository as an array of the custom tools you need to add. Each entry applies:

- It needs to have the form `<tool>:<version>[:all][:<repository-url>]`
- The first entry must have the `<repository-url>` included which is used as default
- Further entries will inherit this default if omitted
- This URL is used in the same way as described above for a software `repository`.
- The `DEVON_SOFTWARE_REPOSITORY` variable is ignored by this feature.
- The optional infix `:all` is used to indicate that the tool is platform independent. Otherwise, an OS specific infix is appended to the URL file to download for your platform (`windows`, `linux`, or `mac`).

As an example, we define it in `${DEVON_IDE_HOME}/settings/devon.properties`:

```
DEVON_IDE_CUSTOM_TOOLS=(jboss-eap:7.1.4.GA:all:https://host.tld/projects/my-project  
firefox:70.0.1)
```

This will download and extract the following content to your `software` folder:

- `https://host.tld/projects/my-project/jboss-eap/7.1.4.GA/jboss-eap-7.1.4.GA.tgz`
- `https://host.tld/projects/my-project/firefox/70.0.1/firefox-70.0.1-windows.tgz`

Please note that if you are not using windows, the `-windows` suffix will be `-mac` or `-linux`.

5.5.6. system

The `system` folder contains documentation and solutions for operation system specific `integration`. Please have a look to get the maximum out of `devonfw-ide` and become a very efficient power user.

5.5.7. updates

The `updates` folder is used for temporary data. This includes:

- extracted archives for installation and updates
- backups of old content on updates to prevent data loss

If all works fine you may clean this folder to save some kilo- or mega-bytes. Otherwise, you can ignore it unless you are looking for a backup after a failed or unplanned upgrade.

5.5.8. workspaces

The `workspaces` folder contains folders for your active work. There is a workspace folder `main` dedicated for your primary work. You may do all your work inside the `main` workspace. Also, you are free to create any number of additional workspace folders named as you like (e.g. `test`, `release`, `testing`, `my-sub-project`, etc.). Using multiple workspaces is especially relevant for Eclipse as each workspace has its own Eclipse runtime instance and configuration.

Within the workspace folder (e.g. `workspaces/main`) you are again free to create sub-folders for (sub-)projects according to your needs. We assume that in most cases you clone git repositories here. The following structure shows an example layout for devonfw:

Listing 5. File structure of workspaces

```
/ workspaces
└── / main
    ├── / .metadata
    ├── / ide
    ├── / devon4j
    └── / my-thai-star
└── / stable
    ├── / .metadata
    ├── / ide
    └── / devon4j
```

In the `main` workspace you may find the cloned forks for regular work (in the example e.g. `devon4j`) as a base to create pull-requests while in the `stable` workspace there is a clone of `devon4j` from the official `devon4j`. However, this is just an example. Some people like to create separate workspaces for development and maintenance branches with git. Other people just switch between those via `git checkout`.

5.5.9. Project import

The `devonfw-ide` supports to automatically check out and import required projects into your IDE during `setup`. To configure this you put a `.properties` file for each desired project into the `projects` sub-folder in your `settings`. Each `.properties` file describes one "project" which you would like to check out and (potentially) import:

```
path=myproject
workingsets=Set1,Set2
workspace=example
git.url=http://github.com/someorg/someproject
git.branch=develop
build.path=
build.cmd=mvn -DskipTests=true -Darchetype.test.skip=true clean install
eclipse=import
active=true
```

Table 20. Variables of project import

Variable	Value	Meaning
<code>path</code>	e.g. <code>myproject</code> , will clone into <code> \${WORKSPACE_PATH} / myproject</code>	(required) Path into which the projects is cloned. This path is relative to the workspace.

Variable	Value	Meaning
<code>working_sets</code>	e.g. <code>ws1,ws2</code>	(optional) This will create working sets (in eclipse). Each module (eclipse project) of this project will be part of all these working sets. Working sets will be automatically created if necessary.
<code>workspace</code>	<code>main</code>	Workspace to use for checkout and import. Default is <code>main</code> .
<code>git.url</code>	e.g. <code>http://github.com/someorg/someproject</code>	(required) Git URL to use for cloning the project.
<code>git.branch</code>	e.g. <code>develop</code>	(optional) Git branch to checkout. Git default branch is default.
<code>build.path</code>	e.g. <code>.</code> (default)	(optional) The directory inside <code>path</code> where to trigger an initial build after clone or pull (if <code>build.cmd</code> is set). For a regular project use <code>.</code> to build top-level project.
<code>build.cmd</code>	e.g. <code>build</code> or <code>mvn -DskipTests=true -Darchetype.test.skip=true clean install</code>	(optional) The <code>devonfw</code> command to invoke to build the project after clone or pull. If omitted no build is triggered.
<code>eclipse</code>	e.g. <code>import</code>	(optional) Desired action for eclipse IDE. If you put <code>import</code> here all modules (eclipse projects) in the current project will be imported into eclipse. If you leave this out or put any other value for this parameter, no change in eclipse is done.
<code>active</code>	<code>true</code>	(optional) If set to <code>false</code> the project is skipped during the <code>setup</code> .

Please note that the `.properties` file is parsed via shell and not via java. So be careful with "advanced" features `.properties` files normally support.

6. Advanced Features

6.1. Cross-Platform Tooling

6.1.1. Git Client

If you are looking for a git client that works cross-platform we recommend to use [Fork](#).

6.1.2. Draw Diagrams

To draw diagrams for your project or for blueprints in devonfw, we recommend the following cross-platform tools:

- [draw.io](#) is a powerful generic vector painting program (similar to visio). You can get a free open-source edition for your desktop from [here](#).
- [ObjectAid](#) is a nice and easy to use eclipse plugin that you can use to quickly create UML diagrams from existing code. While class-diagrams are supported for free, you need to buy a license if you want to use the other diagram types.
- [PlantUML](#) is a great tool that can render UML diagrams from simple markup that can be easily managed in [git](#) or other version-control systems together with your code. Its simplicity allows branching and merging unlike other greedy binary UML data-formats.

6.1.3. Browser Plugins

There are tons of helpful browser plugins out there and it might be a matter of personal taste what you like to have installed. However, as we are heavily using github we want to promote [octotree](#). In case you also work with ZenHub you might want to install the [Zenhub Browser Extension](#).

6.2. Windows Tooling

6.2.1. Integration into Windows-Explorer

After you have [set up](#) your [devonfw-ide](#) on a windows machine, you already have windows-explorer integration out-of-the-box. Just right-click on the folder you would like to open in a terminal and choose from the context menu:

- [Git Bash](#)
- [Open devonfw CMD shell here](#)
- [Open devonfw PowerShell here](#)
- [Open devonfw Cygwin Bash Here](#) (only if cygwin was installed during setup)

6.2.2. Tabs everywhere

Many people got used to *tabs* that have been introduced by all major browsers:



Figure 1. Tabs in Firefox

This nice feature can be added to many other tools.

Tabs for Windows Explorer

If you want to have tabs for windows explorer simply install [Clover](#)

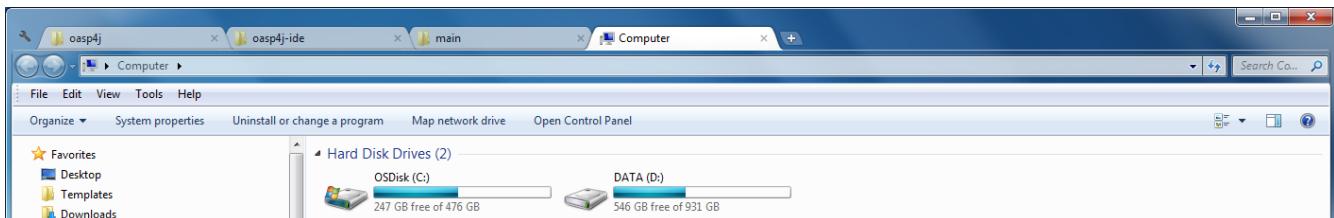


Figure 2. Tabs in Windows Explorer

Tabs for SSH

If you want to have tabs for your SSH client [Putty](#) (or even better [Kitty](#) that comes with [WinSCP](#) integration) you simply install [SuperPutty](#) BTW: Windows 10 has already an SSH client included.



Figure 3. Tabs for SSH

Tabs for CMD

If you want to have tabs for your windows command-line you simply install [ConEmu](#). Here you can also add other shells like Putty. Also you should have a look at the new [Windows Terminal](#) which also supports tabs.

```

[INFO] Reactor Summary:
[INFO]
[INFO] mmm-util ..... SUCCESS [ 0.880 s]
[INFO] mmm-util-bom ..... SUCCESS [ 0.050 s]
[INFO] mmm-util-modules ..... SUCCESS [ 0.033 s]
[INFO] mmm-util-test ..... SUCCESS [ 5.319 s]
[INFO] mmm-util-core ..... SUCCESS [ 19.194 s]
[INFO] mmm-util-pojo ..... SUCCESS [ 8.262 s]
[INFO] mmm-util-entity ..... SUCCESS [ 3.506 s]
[INFO] mmm-util-validation ..... SUCCESS [ 5.722 s]
[INFO] mmm-util-io ..... SUCCESS [ 19.249 s]
[INFO] mmm-util-cli ..... SUCCESS [ 5.017 s]
[INFO] mmm-util-context ..... SUCCESS [ 4.570 s]
[INFO] mmm-util-event ..... SUCCESS [ 2.844 s]
[INFO] mmm-util-search ..... SUCCESS [ 1.421 s]
[INFO] mmm-util-version ..... SUCCESS [ 4.263 s]
[INFO] mmm-util-json ..... SUCCESS [ 2.369 s]
[INFO] mmm-util-data ..... SUCCESS [ 0.210 s]
[INFO] mmm-util-gwt ..... SUCCESS [ 13.438 s]
[INFO] mmm-util-http ..... SUCCESS [ 3.912 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:40 min
[INFO] Finished at: 2017-11-16T17:11:15+01:00
[INFO] Final Memory: 31M/243M
[INFO] -----

```

D:\Projekte\mmm\workspaces\main\util>
cmd.exe* [64]:9444 <161206[64] 3/4 [+ NUM PRI: 129x29 (38,1000) 25V 904 100% /

Figure 4. Tabs for CMD

See [integration](#) to make ConEmu work flawless with [devonfw-ide](#).

6.2.3. Windows Helpers

Handle passwords

Do you want complex passwords that differ for each account for security? Do you only want to remember a single password for simplicity? Do you want to have both? Then, you need to install [KeePass](#) right now.

Real text editor

A real developer needs a real text editor and not windows build in [notepad](#). The most common choice is [Notepad++](#).

Real compression tool

Do you need to deal with ZIP files, TGZ, dpkg, etc.? Just install [7zip](#) and forget about windows build-in ZIP support (that is buggy with long file paths, etc.).

Smarter clipboard

Do you want to paste something from the clipboard but meanwhile you had to copy something else? Just, one of the many things you can easily do with [ditto](#).

Sysinternals Tools

A real developer will quickly notice that windows build in tools to analyze processes, network connections, autostarts, etc. are quite poor. So, what you really would like is the [Sysinternals-Suite](#). You can make process-explorer your [default task manager](#). Use autoruns to prevent nasty background things to be started automatically. Use tcpview to figure out which process is blocking

port 8080, etc.

Cope with file locks

Did you ever fail to delete a file or directory that was locked by some process and you did not even know which one it was? Then you might love [IoBit Unlocker](#). See also [this article](#).

Create symbolic links

Are you used to symbolic and hard links in Linux? Do you have to work with Windows? Would you also like to have such links in Windows? Why not? Windows [supports real links](#) (not shortcuts like in other cases). If you even want to have it integrated in windows explorer you might want to install [linkshellextension](#). However, you might want to disable [SmartMove](#) in the [configuration](#) if you face strange performance issues when moving folders.

Linux

Install [Cygwin](#) and get your bash in windows with ssh-agent, awk, sed, tar, and all the tools you love (or hate). Windows 10 has already a Linux as an installable feature included: WSL and from Version 2004 on WSL2, which is a native Linux Kernel running on Windows (an a light weight VM).

X11

Do you want to connect via SSH and need to open an X11 app from the server? Do you want to see the GUI on your windows desktop? No problem: Install [VcXsrv](#).

Keyboard Freak

Are you a keyboard shortcut person? Do you want to have shortcuts for things like « and » ? Then you should try [AutoHotKey](#). For the example (« and ») you can simply use this script to get started:

```
^<::Send {U+00AB}  
^+<::Send {U+00BB}
```

First, just press [\[ctrl\]\[<\]](#) and [\[ctrl\]\[>\]](#) ([\[ctrl\]\[shift\]\[<\]](#)). Next, create shortcuts to launch your IDE, to open your favorite tool, etc. If you like a GUI to easily configure the scripts, that comes with a lot of extensions preinstalled, you should have a look at [Ac'tive Aid](#).

Paint anywhere on your desktop

Do you collaborate sharing your screen, and want to mark a spot on top of what you see? Use [Epic Pen](#) to do just that.

Analyze graphs

Do you need to visualise complex graph structures? Convert them to [Trivial Graph Format \(.tgf\)](#), a run [yEd](#) to get an interactive visualization of your graph.

Up your screen capture game

Capture any part of your screen with a single click, directly upload to dropbox, or run a svn commit all in one go with [Greenshot](#). Another screen capture tool where you can easily manage and edit your screenshots and also do screen recordings with is [Screenpresso](#).

Fast Search in Windows

[Everything](#) is a desktop search utility for Windows that can rapidly find files and folders by name.

6.3. MacOS Tooling

6.3.1. Finder

If you want to open a terminal from a folder in [Finder](#) and automatically get your environment set properly for [devonfw-ide](#) you will find the perfect solution here.



So after installing (see below) the integration(s) provided here, you can easily open a terminal ready for your [devonfw-ide](#):

- right click ([\[control\]](#) + click) on file or folder in [Finder](#)
- Expand the [Quick-Actions](#) sub-menu

- Click on the desired action (e.g. [Open devonfw-Terminal here](#))
- Verify that your environment is properly initialized by invoking:

```
mvn -v
```

To get this feature for MacOS Terminal.app open [Finder](#) and run the workflow [system/mac/terminal/Open_deonfw-Terminal_here.workflow](#) (in `${DEVON_IDE_HOME}`). For [iTerm2.app](#) (that can be installed from [App Store](#)) do the same with [system/mac/iterm/Open_deonfw-iTerm_here.workflow](#).

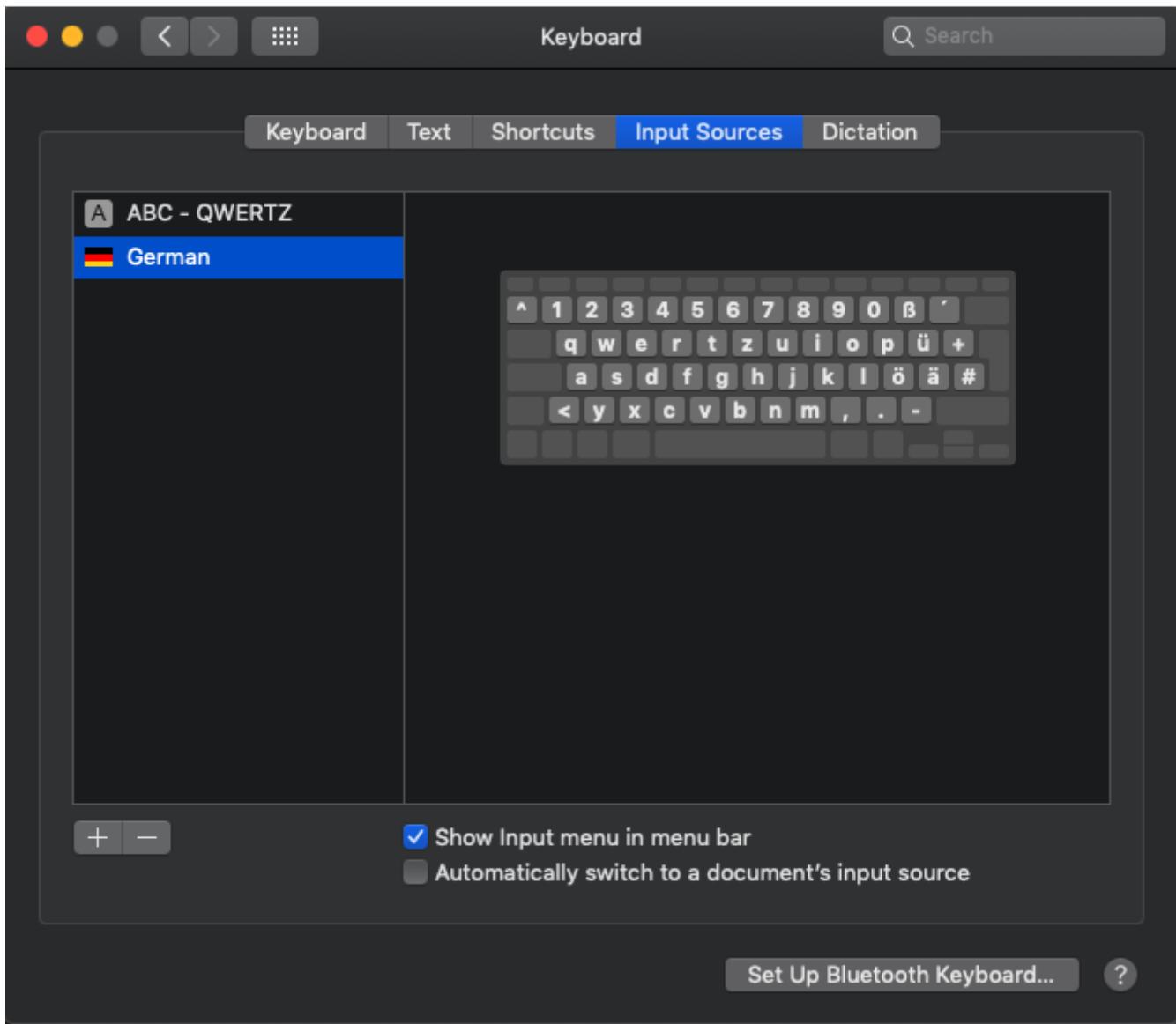
6.3.2. Keyboard

Keyboard support is not an integration however, some users coming from other platforms may struggle with the way MacOS deals with (external non-apple) keyboards. So to make it short: if you are happy with your keyboard and shortcuts, you can skip all the following. Otherwise, if you think that pressing keys like [Home](#), [End](#), etc. should just work as expected or pressing [Alt Gr](#) should allow you to type the special characters as printed on your German keyboard then here you will find a solution to your problems! To get all automated you can just run the script [system/mac/keyboard/install-mac-keyboard-support.sh](#) (in `${DEVON_IDE_HOME}`). If you would like to understand what is going on, you want to customize the keyboard settings to your needs, or you want a keyboard layout other than German ISO, please read on.

Keyboard Layouts

Keyboard layouts allow a fine-grained mapping of each key on your keyboard to its resulting input character or behaviour. They are MacOS native features and do not need to have software running as a background service to make the keyboard mapping work (see Karabiner section below as an alternative). They are provided as so called [bundle](#) (white lego brick icon). Like a MacOS app this is a folder containing a [Contents](#) folder with a specific sub-folder structure. In the [Resources](#) subfolder [*.keylayout](#) files are placed and define the exact mapping for the keyboard. As an example we provide a [Keyboard Layouts](#) folder containing a [bundle](#) for a German keyboard mapping.

To install keyboard layouts simply doubleclick the [bundle](#) or copy it to [~/Library/Keyboard Layouts](#). To actually use them go to [System Preferences](#) and select [Keyboard](#). Then, select the tab [Input Sources](#). With the [+](#) button you can add a keyboard layout for your daily usage with your mac. Please note that the keyboard layout shipped with [devonfw-ide](#) is called [German-ISO](#) and can be found in the [Others](#) section at the end of the list. It can be used as an example or template, if you want to create your own layout.



When you have multiple mappings in place, on the top menu bar you will find a little icon next to the current time that allows you to switch between the keyboard layouts, which is very handy when you switch from your native MacBook keyboard to an external USB keyboard or vice versa. Even for a pure MacOS geek this can be helpful in case a friend coming from Windows/Linux is supposed to type something on the Mac in a pair-programming session.

In our German keyboard mapping example you can use the keys like `Alt Gr`, etc. to type special characters as you would expect and as printed on your keyboard. To make `Pos1`, `End`, etc. work properly accross all apps please read on to the next section(s).

In case you would like to create your own keyboard layout you can of course edit the `*.keylayout` files in a text editor. However, to make this much more comfortable, you can use the graphical editor tool [Ukelele](#). Besides, the app itself, the Ukelele `dmg` file, also contains a [Documentation](#) and a [Resources](#) folder. The latter contains many keyboard layouts that you can use as a starting point.

Key Bindings

Still, various keyboard shortcuts might not work as expected for you. Therefore, we provide you with an advanced configuration in the folder `system/mac/keyboard/KeyBindings` that you can copy to your `~/Library` folder:

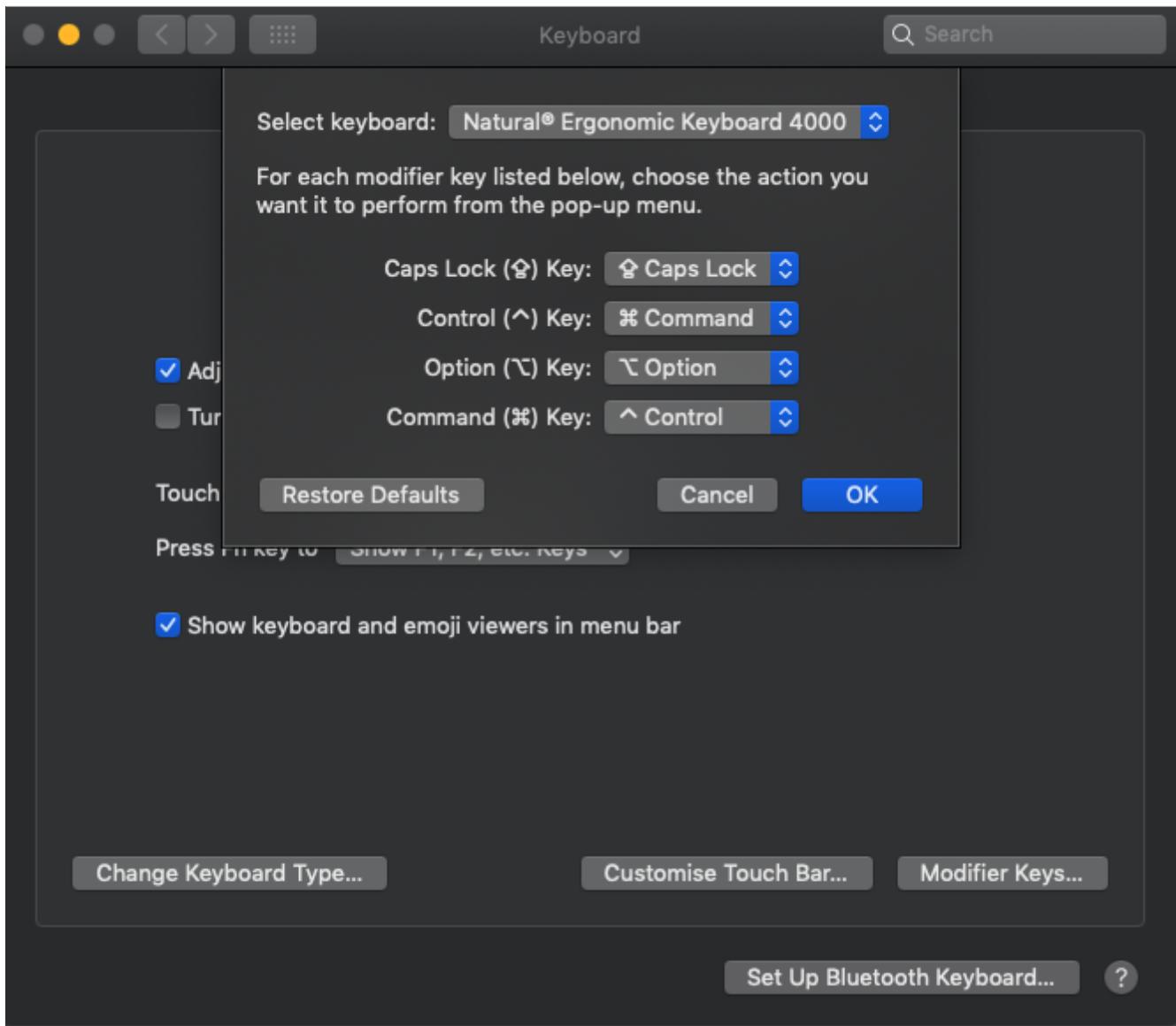
```
cd system/mac/keyboard/  
cp -r KeyBindings ~/Library
```

To make the changes work you need to log out and log in again or you can reboot. After that, your **Home** (**Pos1**) and **End** buttons should work as expected including with selection via **Shift** and/or **Command**. Also, you can use **Command** together with the left or right arrow key to move between words and combined it with **Shift** for selection. As an example, for further customization you can press **Command + <** to type the unicode character «.

However, still some apps listen to keyboard events on a lower level and come with their own keyboard mappings. In these apps you might still experience unexpected behaviour. Solutions can be found in the following sub-sections.

Switch Control and Command

If you are used to windows or linux and get easily confused by the apple keyboard behaviour you might want to switch the **Control** and the **Option** key. Open **System Preferences** and select **Keyboard**. Then, in the first tab, click on the button **Modifier Keys...**. For every keyboard you can customize the behaviour of your modifier keys and therefore switch **Control** and **Option** as illustrated in the screenshot:



Programmers now should also disable that **Control + Space** is opening **Spotlight Search** as otherwise this shortcut can not be redefined in other apps like common IDEs.



Eclipse

In Eclipse, move and select by word as described above does not work. Even worse, the most important shortcut does not work: **Control + Space** for code completion (content assist). You can manually redefine the key bindings in **Preferences** under **General > Keys**. However, with multiple IDE installations and workspaces this will quickly get tedious. Therefore, you can **Export** and **Import** specific **Preferences** such as **Keys Preferences** to/from a ***.epf** (Eclipse PreFerences) file. We have done all this for you so you can just import the file located in **system/mac/keyboard/Eclipse/eclipse-mac-keybindings.epf** into your Eclipse. Happy coding.

Karabiner

If you want more dynamics and do not worry about an app that has to run in the background to make your keyboard work as you like (no relevant performance overhead), you can try **Karabiner Elements**. This is a powerful tool to remap your keyboard shortcuts. In the UI you can only directly create and edit **Simple Modifications** that are too limited for most use-cases. However, using **Complex Modifications** you can do a lot of magic to customize the keyboard behaviour to your personal needs. A key with any combination of modifiers can be mapped to any key with arbitrary modifiers. This can also be bound to conditions based on the frontmost application or the keyboard model. These complex modifications are configured as ***.json** files. We have included a set with useful

rules for external keyboards, programmer shortcuts, etc. If you have Karabiner installed, you only need to copy the contents of the `karabiner` folder located in this directory to your `~/.config` folder:

```
cd system/mac/keyboard/
cp karabiner/assets/complex_modifications/*.json
~/.config/karabiner/assets/complex_modifications/
```

Now, if you open the `Complex Modifications` in the `Karabiner` app, you can click on the `+ Add rule` button and will see these mappings in the pop up. Select the rules you want to add (e.g. add all) and you are done. Unlike other solutions, you can quickly tweak your keyboard without the need to log out and restart apps, which gives faster trial and error turnarounds. Further, if you want to tweak your own configs, Karabiner comes with a secondary app called `Karabiner-EventViewer` that shows you the names of the keys, modifiers, and apps for the events you are triggering. This is very helpful to get the config right.

6.4. Linux Tooling

There is nothing in this section so far. If you are a Linux user, please share your experience and provide your valuable hints.

6.5. Lombok

Even though not officially recommended by `devon4j` some projects want to use `lombok` in their project. As this requires some tweaks for IDEs we do support you with this guide in case you want to use it.

6.5.1. Lombok in Eclipse

For eclipse there is a plugin to activate `lombok support in eclipse`. We have this already preconfigured for you in our default `settings`. So for manual installation after `setup`, you can get it via this command:

```
devon eclipse add-plugin lombok
```

However, to avoid manual extra effort for lombok based projects you only need to activate this plugin in your project specific `settings` in `lombok.properties for eclipse` (replace `false` with `true` for `plugin_active`).

6.5.2. Lombok for VS-Code

For VisualStudio Code there is an extension to activate `lombok support in VS-Code`. We have this already preconfigured for you in our default `settings`. So for manual installation after `setup`, you can get it via this command:

```
devon vscode add-plugin lombok
```

However, to avoid manual extra effort for lombok based projects you only need to activate this plugin in your project specific [settings](#) in [lombok.properties for vscode](#) (replace `false` with `true` for `plugin_active`).

6.5.3. Lombok for IntelliJ

For IntelliJ there is a plugin to activate [lombok support in IntelliJ](#). Currently we have not yet configured or automated this in [devonfw-ide](#). Please contribute to change this. See issues [#453](#) and [#491](#).

7. Support

7.1. Migration from oasp4j-ide

The `devonfw-ide` is a completely new and innovative solution for managing the local development environment that has been created from scratch. Releases of `OASP` as well as releases of `devonfw` until version 3.1.x are based on the old `oasp4j-ide` that is now considered deprecated. As `devonfw-ide` is a complete redesign this will have some impact for the users. This section should help and assist so you do not get lost.

7.1.1. Get familiar with devonfw-ide

First of all you should roughly get familiar with the new `devonfw-ide`. The key features and changes are:

- platform-agnostic (supports Windows, Mac, and Linux in a single distribution)
- small core (reduced the download package from ~2 gigabyte to ~2 megabyte)
- fast and easy updates (built in update support)
- minimum number of scripts (removed tons of end-user scripts making things much simpler)
- fully automated setup (run `setup` script and you are ready - even for advanced features that had to be configured manually before)
- single command for everything (entire CLI available via new `devon` command)

For all the details you should study the documentation starting from the [beginning](#).

7.1.2. Migration of existing oasp4j-ide installation

- extract new `devonfw-ide-scripts` on top of your existing installation
- run `setup`
- done

If you get errors:

- ask your technical lead to fix the `settings` git repo for `devonfw-ide` or offer him to do it for you.
- you need to merge the `devon` folder into your settings
- you need to merge the `devon.properties` into your settings
- you should check your `variables[-customized][.bat]` and merge required customizations into the proper `configuration`

7.1.3. Hints for users after migration

Getting used to all the new commands might be tedious when starting after a migration.

Table 21. Comparison of commands

oasp4j-ide command	devonfw-ide command	Comment
<code>create-or-update-workspace</code>	<code>devon eclipse ws-update</code>	actually not needed anymore as workspace is updated automatically when IDE is launched. To launch your IDE simply run <code>devon eclipse</code> , <code>devon intellij</code> , or <code>devon vscode</code> . If you like to get launch scripts for your IDE e.g. Eclipse just call <code>devon eclipse --all create-script</code> .
<code>create-or-update-workspace <>workspace><</code>	<code>cd <>workspace>&& devon eclipse ws-update</code>	
<code>update-all-workspaces</code>	<code>devon eclipse --all ws-update</code>	
<code>create-or-update-workspace-vs</code>	<code>devon vscode ws-update</code>	
<code>devcon workspace create <>workspace><</code>	Simply create the <code><>workspace></code> directory (e.g. <code>cd workspaces && mkdir examples</code>)	
<code>scripts/update-eclipse-workspace-settings</code>	<code>devon eclipse ws-reverse</code>	To add new properties (old option <code>--new</code>) use <code>devon eclipse ws-reverse-add</code>
<code>devcon project build</code> <code>devcon devon4j build</code> <code>devcon devon4ng build</code>	<code>devon build</code>	
<code>devcon devon4j create</code>	<code>devon java create</code>	
<code>devcon devon4ng create</code>	<code>devon ng create</code>	
<code>devcon system *</code> <code>devcon dist *</code>	<code>setup</code> or <code>devon ide setup</code>	
<code>console.bat</code>	-	Simply open terminal in selected folder. On Windows right-click folder in windows-explorer and select <code>open devonfw CMD here</code> .
<code>devcon help</code>	<code>devon help</code>	
<code>devcon doc</code>	Read the documentation from devonfw.com	

7.2. License

The product [devonfw-ide](#) is licensed under the following terms.

Binaries of this product have been made available to you by [devonfw](#) under the [Apache Public License 2.0](#).

The documentation of this product is licensed under the terms of the [Creative Commons License \(Attribution-NoDerivatives 4.0 International\)](#).

All of the source code to this product is available under licenses which are both free and open source.

More specifically, most of the source code is available under the Apache Public License 2.0. The remainder of the software which is not under the Apache license is available under one of a variety of other free and open source licenses. Those that require reproduction of the license text in the distribution are given below. (Note: your copy of this product may not contain code covered by one or more of the licenses listed here, depending on the exact product and version you choose.)

The following table shows the components that may be used. The column **inclusion** indicates the way the component is included:

- **directly included** means the component is directly contained in the download package of [devonfw-ide](#) we provide
- **default setup** means the component is not initially included but will be downloaded during the **setup** by default
- **optional** means the component is neither initially included nor downloaded by default, but only gets downloaded and installed if explicitly triggered by you when invoking additional commands or if explicitly configured by your project.

Table 22. Third party components

Component	Inclusion	License
devonfw-ide	Directly included	ASL 2.0
JSON-P API	Directly included	EPL 2.0
JSON-P Implementation	Directly included	EPL 2.0
OpenJDK / AdoptOpenJDK (Java)	Default Setup	GPLv2
Maven	Default Setup	ASL 2.0
VS Code	Default Setup	MIT (Terms)
extension-pack-vscode	Default Setup	ASL 2.0
Eclipse	Default Setup	EPL 2.0
CobiGen	Default Setup	ASL 2.0
TM Terminal	Default Setup	EPL 2.0 (see here)
AnyEdit	Default Setup	EPL 1.0

Component	Inclusion	License
EclipseCS	Default Setup	LGPL 2.1
SpotBugs Eclipse plugin	Default Setup	LGPL 2.1
EclEmma	Default Setup	EPL 1.0
StartExplorer	Default Setup	WTFPL 2
regex tester	Default Setup	GPL 2.0 (see here)
eclipse-templatevariables	Default Setup	ASL 2.0
Node.js	Default Setup	License
NPM	Default Setup	Artistic License 2.0 (Terms)
Angular CLI (ng)	Default Setup	MIT
Groovy	Default Setup	ASL 2.0
Apache Ant	Default Setup	ASL 2.0
Gradle	Optional	ASL 2.0
Jenkins	Optional	MIT
SonarQube (Community Edition)	Optional	LGPL 3.0
SonarLint	Optional	LGPL 3+
cicdgen	Optional	ASL 2.0
devon4j	Optional	ASL 2.0
devon4ng	Optional	ASL 2.0
devon4node	Optional	ASL 2.0

7.2.1. Apache Software License - Version 2.0

Apache License
Version 2.0, January 2004
<http://www.apache.org/licenses/>

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition,

"control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable

copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided

that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. **Submission of Contributions.** Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. **Trademarks.** This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. **Disclaimer of Warranty.** Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. **Limitation of Liability.** In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf

of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

7.2.2. Eclipse Public License - Version 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

- a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and
- b) in the case of each subsequent Contributor:
 - i) changes to the Program, and
 - ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

- a) it complies with the terms and conditions of this Agreement; and
- b) its license agreement:
 - i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;
 - ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;
 - iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and
 - iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

- a) it must be made available under this Agreement; and
- b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such

Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s),

then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.

7.2.3. Eclipse Public License - Version 2.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial content Distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are Distributed by that particular Contributor. A Contribution "originates" from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include changes or additions to the

Program that are not Modified Works.

"Contributor" means any person or entity that Distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions Distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement or any Secondary License (as applicable), including Contributors.

"Derivative Works" shall mean any work, whether in Source Code or other form, that is based on (or derived from) the Program and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship.

"Modified Works" shall mean any work in Source Code or other form that results from an addition to, deletion from, or modification of the contents of the Program, including, for purposes of clarity any new file in Source Code form that contains any contents of the Program. Modified Works shall not include works that contain only declarations, interfaces, types, classes, structures, or files of the Program solely in each case in order to link to, bind by name, or subclass the Program or Modified Works thereof.

"Distribute" means the acts of a) distributing or b) making available in any manner that enables the transfer of a copy.

"Source Code" means the form of a Program preferred for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Secondary License" means either the GNU General Public License, Version 2.0, or any later versions of that license, including any exceptions or additional permissions as identified by the initial Contributor.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, Distribute and sublicense the Contribution of such Contributor, if any, and such Derivative Works.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in Source Code or other form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its

Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to Distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

e) Notwithstanding the terms of any Secondary License, no Contributor makes additional grants to any Recipient (other than those set forth in this Agreement) as a result of such Recipient's receipt of the Program under the terms of a Secondary License (if permitted under the terms of Section 3).

3. REQUIREMENTS

3.1 If a Contributor Distributes the Program in any form, then:

a) the Program must also be made available as Source Code, in accordance with section 3.2, and the Contributor must accompany the Program with a statement that the Source Code for the Program is available under this Agreement, and informs Recipients how to obtain it in a reasonable manner on or through a medium customarily used for software exchange; and

b) the Contributor may Distribute the Program under a license different than this Agreement, provided that such license:

i) effectively disclaims on behalf of all other Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all other Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) does not attempt to limit or alter the recipients' rights in the Source Code under section 3.2; and

iv) requires any subsequent distribution of the Program by any party to be under a license that satisfies the requirements of this section 3.

3.2 When the Program is Distributed as Source Code:

a) it must be made available under this Agreement, or if the Program (i) is combined with other material in a separate file or files made available under a Secondary License, and (ii) the initial Contributor attached to the Source Code the notice described in Exhibit A of this Agreement, then the Program may be made available under the terms of such Secondary Licenses, and

b) a copy of this Agreement must be included with each copy of the Program.

3.3 Contributors may not remove or alter any copyright, patent, trademark, attribution notices, disclaimers of warranty, or limitations of liability ('notices') contained

within the Program from any copy of the Program which they Distribute, provided that Contributors may add their own appropriate notices.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, AND TO THE EXTENT PERMITTED BY APPLICABLE LAW, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement, including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, AND TO THE EXTENT PERMITTED BY APPLICABLE LAW, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH

DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be Distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to Distribute the Program (including its Contributions) under the new version.

Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved. Nothing in this Agreement is intended to be enforceable by any entity that is not a Contributor or Recipient. No third-party beneficiary rights are created under this Agreement.

Exhibit A – Form of Secondary Licenses Notice

"This Source Code may also be made available under the following Secondary Licenses when the conditions for such availability set forth in the Eclipse Public License, v. 2.0 are satisfied: {name license(s), version(s), and exceptions or additional permissions here}."

Simply including a copy of this Agreement, including this Exhibit A is not sufficient to license the Source Code under Secondary Licenses.

If it is not possible or desirable to put the notice in a particular file, then You may include the notice in a location (such as a LICENSE file in a relevant directory) where a recipient would be likely to look for such a notice.

You may add additional accurate notices of copyright ownership.

7.2.4. MIT License

Copyright <YEAR> <COPYRIGHT HOLDER>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7.2.5. Artistic License - Version 2.0

Copyright (c) 2000-2006, The Perl Foundation.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

This license establishes the terms under which a given free software Package may be copied, modified, distributed, and/or redistributed. The intent is that the Copyright Holder maintains some artistic control over the development of that Package while still keeping the Package available as open source and free software.

You are always permitted to make arrangements wholly outside of this license directly with the Copyright Holder of a given Package. If the terms of this license do not permit the full use that you propose to make of the Package, you should contact the Copyright Holder and seek a different licensing arrangement.

Definitions

"Copyright Holder" means the individual(s) or organization(s) named in the copyright

notice for the entire Package.

"Contributor" means any party that has contributed code or other material to the Package, in accordance with the Copyright Holder's procedures.

"You" and "your" means any person who would like to copy, distribute, or modify the Package.

"Package" means the collection of files distributed by the Copyright Holder, and derivatives of that collection and/or of those files. A given Package may consist of either the Standard Version, or a Modified Version.

"Distribute" means providing a copy of the Package or making it accessible to anyone else, or in the case of a company or organization, to others outside of your company or organization.

"Distributor Fee" means any fee that you charge for Distributing this Package or providing support for this Package to another party. It does not mean licensing fees.

"Standard Version" refers to the Package if it has not been modified, or has been modified only in ways explicitly requested by the Copyright Holder.

"Modified Version" means the Package, if it has been changed, and such changes were not explicitly requested by the Copyright Holder.

"Original License" means this Artistic License as Distributed with the Standard Version of the Package, in its current version or as it may be modified by The Perl Foundation in the future.

"Source" form means the source code, documentation source, and configuration files for the Package.

"Compiled" form means the compiled bytecode, object code, binary, or any other form resulting from mechanical transformation or translation of the Source form.

Permission for Use and Modification Without Distribution

(1) You are permitted to use the Standard Version and create and use Modified Versions for any purpose without restriction, provided that you do not Distribute the Modified Version.

Permissions for Redistribution of the Standard Version

(2) You may Distribute verbatim copies of the Source form of the Standard Version of this Package in any medium without restriction, either gratis or for a Distributor Fee, provided that you duplicate all of the original copyright notices and associated disclaimers. At your discretion, such verbatim copies may or may not include a Compiled form of the Package.

(3) You may apply any bug fixes, portability changes, and other modifications made available from the Copyright Holder. The resulting Package will still be considered the Standard Version, and as such will be subject to the Original License.

Distribution of Modified Versions of the Package as Source

(4) You may Distribute your Modified Version as Source (either gratis or for a Distributor Fee, and with or without a Compiled form of the Modified Version) provided that you clearly document how it differs from the Standard Version, including, but not limited to, documenting any non-standard features, executables, or modules, and provided that you do at least ONE of the following:

- (a) make the Modified Version available to the Copyright Holder of the Standard Version, under the Original License, so that the Copyright Holder may include your modifications in the Standard Version.
- (b) ensure that installation of your Modified Version does not prevent the user installing or running the Standard Version. In addition, the Modified Version must bear a name that is different from the name of the Standard Version.
- (c) allow anyone who receives a copy of the Modified Version to make the Source form of the Modified Version available to others under
 - (i) the Original License or
 - (ii) a license that permits the licensee to freely copy, modify and redistribute the Modified Version using the same licensing terms that apply to the copy that the licensee received, and requires that the Source form of the Modified Version, and of any works derived from it, be made freely available in that license fees are prohibited but Distributor Fees are allowed.

Distribution of Compiled Forms of the Standard Version or Modified Versions without the Source

(5) You may Distribute Compiled forms of the Standard Version without the Source, provided that you include complete instructions on how to get the Source of the Standard Version. Such instructions must be valid at the time of your distribution. If these instructions, at any time while you are carrying out such distribution, become invalid, you must provide new instructions on demand or cease further distribution. If you provide valid instructions or cease distribution within thirty days after you become aware that the instructions are invalid, then you do not forfeit any of your rights under this license.

(6) You may Distribute a Modified Version in Compiled form without the Source, provided that you comply with Section 4 with respect to the Source of the Modified Version.

Aggregating or Linking the Package

(7) You may aggregate the Package (either the Standard Version or Modified Version) with other packages and Distribute the resulting aggregation provided that you do not charge a licensing fee for the Package. Distributor Fees are permitted, and licensing fees for other components in the aggregation are permitted. The terms of this license apply to the use and Distribution of the Standard or Modified Versions as included in the aggregation.

(8) You are permitted to link Modified and Standard Versions with other works, to embed the Package in a larger work of your own, or to build stand-alone binary or bytecode versions of applications that include the Package, and Distribute the result without restriction, provided the result does not expose a direct interface to the Package.

Items That are Not Considered Part of a Modified Version

(9) Works (including, but not limited to, modules and scripts) that merely extend or make use of the Package, do not, by themselves, cause the Package to be a Modified Version. In addition, such works are not considered parts of the Package itself, and are not subject to the terms of this license.

General Provisions

(10) Any use, modification, and distribution of the Standard or Modified Versions is governed by this Artistic License. By using, modifying or distributing the Package, you accept this license. Do not use, modify, or distribute the Package, if you do not accept this license.

(11) If your Modified Version has been derived from a Modified Version made by someone other than you, you are nevertheless required to ensure that your Modified Version complies with the requirements of this license.

(12) This license does not grant you the right to use any trademark, service mark, tradename, or logo of the Copyright Holder.

(13) This license includes the non-exclusive, worldwide, free-of-charge patent license to make, have made, use, offer to sell, sell, import and otherwise transfer the Package with respect to any patent claims licensable by the Copyright Holder that are necessarily infringed by the Package. If you institute patent litigation (including a cross-claim or counterclaim) against any party alleging that the Package constitutes direct or contributory patent infringement, then this Artistic License to you shall terminate on the date that such litigation is filed.

(14) Disclaimer of Warranty: THE PACKAGE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS "AS IS' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES. THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT ARE DISCLAIMED TO THE EXTENT PERMITTED BY YOUR LOCAL LAW. UNLESS REQUIRED BY LAW, NO COPYRIGHT HOLDER OR CONTRIBUTOR WILL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING IN ANY WAY OUT OF THE USE OF THE PACKAGE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

7.2.6. Creative Commons License - Attribution-NoDerivatives 4.0 International

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution-NoDerivatives 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

Adapted Material means material subject to Copyright and Similar Rights that is

derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

Copyright and Similar Rights means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

Effective Technological Measures means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

Exceptions and Limitations means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

Licensed Material means the artistic or literary work, database, or other material to which the Licensor applied this Public License.

Licensed Rights means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

Licensor means the individual(s) or entity(ies) granting rights under this Public License.

Share means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

Sui Generis Database Rights means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

You means the individual or entity exercising the Licensed Rights under this Public License. Your has a corresponding meaning.

Section 2 – Scope.

License grant.

Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

reproduce and Share the Licensed Material, in whole or in part; and
produce and reproduce, but not Share, Adapted Material.

Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

Term. The term of this Public License is specified in Section 6(a).

Media and formats; technical modifications allowed. The Licensor authorizes

You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

Downstream recipients.

Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.

No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

Other rights.

Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

Patent and trademark rights are not licensed under this Public License.

To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

Attribution.

If You Share the Licensed Material, You must:

retain the following if it is supplied by the Licensor with the Licensed Material:

identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

a copyright notice;

a notice that refers to this Public License;

a notice that refers to the disclaimer of warranties;

a URI or hyperlink to the Licensed Material to the extent reasonably

practicable;

indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

For the avoidance of doubt, You do not have permission under this Public License to Share Adapted Material.

You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database, provided You do not Share Adapted Material;

if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

The disclaimer of warranties and limitation of liability provided above shall be

interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or
upon express reinstatement by the Licenser.

For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.

To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

7.2.7. GNU LESSER GENERAL PUBLIC LICENSE - Version 2.1

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts
as the successor of the GNU Library Public License, version 2, hence
the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called

"this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) The modified work must itself be a software library.
- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
- d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a

"work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with

the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy,

distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose

distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the library's name and an idea of what it does.

Copyright (C) year name of author

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public

License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

signature of Ty Coon, 1 April 1990
Ty Coon, President of Vice

7.2.8. GNU LESSER GENERAL PUBLIC LICENSE - Version 3

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <<https://fsf.org/>>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.

"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or

source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- d) Do one of the following:
 - 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section

6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

7.2.9. GNU GENERAL PUBLIC LICENSE - Version 2

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part

regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this

License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have

the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the program's name and an idea of what it does.
Copyright (C) yyyy name of author

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type 'show w'. This is free software, and you are welcome
to redistribute it under certain conditions; type 'show c'
for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright
interest in the program 'Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice

7.2.10. DO WHAT THE FUCK YOU WANT TO PUBLIC LICENSE - Version 2

DO WHAT THE FUCK YOU WANT TO PUBLIC LICENSE
Version 2, December 2004

Copyright (C) 2004 Sam Hocevar

14 rue de Plaisance, 75014 Paris, France

Everyone is permitted to copy and distribute verbatim or modified copies of this license document, and changing it is allowed as long as the name is changed.

DO WHAT THE FUCK YOU WANT TO PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. You just DO WHAT THE FUCK YOU WANT TO.

7.2.11. License of Node.js

Node.js is licensed for use as follows:

"""

Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

"""

This license applies to parts of Node.js originating from the <https://github.com/joyent/node> repository:

"""

Copyright Joyent, Inc. and other Node contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or

sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

"""

The Node.js license applies to all parts of Node.js that are not externally maintained libraries.

The externally maintained libraries used by Node.js are:

- Acorn, located at deps/acorn, is licensed as follows:

"""

Copyright (C) 2012-2018 by various contributors (see AUTHORS)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

"""

- Acorn plugins, located at deps/acorn-plugins, is licensed as follows:

"""

Copyright (C) 2017-2018 by Adrian Heine

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

"""

- c-ares, located at deps/cares, is licensed as follows:

"""

Copyright (c) 2007 - 2018, Daniel Stenberg with many contributors, see AUTHORS file.

Copyright 1998 by the Massachusetts Institute of Technology.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

"""

- ICU, located at deps/icu-small, is licensed as follows:

"""

COPYRIGHT AND PERMISSION NOTICE (ICU 58 and later)

Copyright © 1991-2019 Unicode, Inc. All rights reserved.

Distributed under the Terms of Use in <https://www.unicode.org/copyright.html>.

Permission is hereby granted, free of charge, to any person obtaining a copy of the Unicode data files and any associated documentation (the "Data Files") or Unicode software and any associated documentation (the "Software") to deal in the Data Files or Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Data Files or Software, and to permit persons to whom the Data Files or Software are furnished to do so, provided that either
(a) this copyright and permission notice appear with all copies of the Data Files or Software, or
(b) this copyright and permission notice appear in associated Documentation.

THE DATA FILES AND SOFTWARE ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS.

IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THE DATA FILES OR SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in these Data Files or Software without prior written authorization of the copyright holder.

Third-Party Software Licenses

This section contains third-party software notices and/or additional terms for licensed third-party software components included within ICU libraries.

1. ICU License - ICU 1.8.1 to ICU 57.1

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995-2016 International Business Machines Corporation and others
All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

All trademarks and registered trademarks mentioned herein are the property of their respective owners.

2. Chinese/Japanese Word Break Dictionary Data (cjdict.txt)

```
#      The Google Chrome software developed by Google is licensed under
# the BSD license. Other software included in this distribution is
# provided under other licenses, as set forth below.
#
# The BSD License
# http://opensource.org/licenses/bsd-license.php
# Copyright (C) 2006-2008, Google Inc.
#
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are met:
#
# Redistributions of source code must retain the above copyright notice,
# this list of conditions and the following disclaimer.
# Redistributions in binary form must reproduce the above
# copyright notice, this list of conditions and the following
# disclaimer in the documentation and/or other materials provided with
# the distribution.
# Neither the name of Google Inc. nor the names of its
# contributors may be used to endorse or promote products derived from
# this software without specific prior written permission.
#
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
# CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
# INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
# MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
# DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
# LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
# CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
# SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
# BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
# LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
# NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
# SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
#
# The word list in cjdict.txt are generated by combining three word lists
# listed below with further processing for compound word breaking. The
```

```
# frequency is generated with an iterative training against Google web
# corpora.
#
# * Libtabe (Chinese)
#   - https://sourceforge.net/project/?group_id=1519
#   - Its license terms and conditions are shown below.
#
# * IPADIC (Japanese)
#   - http://chasen.aist-nara.ac.jp/chasen/distribution.html
#   - Its license terms and conditions are shown below.
#
# -----COPYING.libtabe ---- BEGIN-----
#
# /*
#  * Copyright (c) 1999 TaBE Project.
#  * Copyright (c) 1999 Pai-Hsiang Hsiao.
#  * All rights reserved.
#  *
#  * Redistribution and use in source and binary forms, with or without
#  * modification, are permitted provided that the following conditions
#  * are met:
#  *
#  * . Redistributions of source code must retain the above copyright
#  * notice, this list of conditions and the following disclaimer.
#  * . Redistributions in binary form must reproduce the above copyright
#  * notice, this list of conditions and the following disclaimer in
#  * the documentation and/or other materials provided with the
#  * distribution.
#  * . Neither the name of the TaBE Project nor the names of its
#  * contributors may be used to endorse or promote products derived
#  * from this software without specific prior written permission.
#  *
#  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
#  * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
#  * REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
#  * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
#  * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
#  * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
#  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
#  * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
#  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
#  * OF THE POSSIBILITY OF SUCH DAMAGE.
# */
#
# /*
#  * Copyright (c) 1999 Computer Systems and Communication Lab,
#  * Institute of Information Science, Academia
#  * Sinica. All rights reserved.
#  *
```

```
# * Redistribution and use in source and binary forms, with or without
# * modification, are permitted provided that the following conditions
# * are met:
# *
# * . Redistributions of source code must retain the above copyright
# * notice, this list of conditions and the following disclaimer.
# * . Redistributions in binary form must reproduce the above copyright
# * notice, this list of conditions and the following disclaimer in
# * the documentation and/or other materials provided with the
# * distribution.
# * . Neither the name of the Computer Systems and Communication Lab
# * nor the names of its contributors may be used to endorse or
# * promote products derived from this software without specific
# * prior written permission.
# *
# * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# * FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
# * REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
# * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
# * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
# * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
# * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
# * OF THE POSSIBILITY OF SUCH DAMAGE.
# */
#
#
# Copyright 1996 Chih-Hao Tsai @ Beckman Institute,
# University of Illinois
# c-tsai4@uiuc.edu http://casper.beckman.uiuc.edu/~c-tsai4
#
# -----COPYING.libtabe----END-----
#
#
# -----COPYING.ipadic----BEGIN-----
#
# Copyright 2000, 2001, 2002, 2003 Nara Institute of Science
# and Technology. All Rights Reserved.
#
# Use, reproduction, and distribution of this software is permitted.
# Any copy of this software, whether in its original form or modified,
# must include both the above copyright notice and the following
# paragraphs.
#
# Nara Institute of Science and Technology (NAIST),
# the copyright holders, disclaims all warranties with regard to this
# software, including all implied warranties of merchantability and
# fitness, in no event shall NAIST be liable for
# any special, indirect or consequential damages or any damages
```

```
# whatsoever resulting from loss of use, data or profits, whether in an
# action of contract, negligence or other tortious action, arising out
# of or in connection with the use or performance of this software.
#
# A large portion of the dictionary entries
# originate from ICOT Free Software. The following conditions for ICOT
# Free Software applies to the current dictionary as well.
#
# Each User may also freely distribute the Program, whether in its
# original form or modified, to any third party or parties, PROVIDED
# that the provisions of Section 3 ("NO WARRANTY") will ALWAYS appear
# on, or be attached to, the Program, which is distributed substantially
# in the same form as set out herein and that such intended
# distribution, if actually made, will neither violate or otherwise
# contravene any of the laws and regulations of the countries having
# jurisdiction over the User or the intended distribution itself.
#
# NO WARRANTY
#
# The program was produced on an experimental basis in the course of the
# research and development conducted during the project and is provided
# to users as so produced on an experimental basis. Accordingly, the
# program is provided without any warranty whatsoever, whether express,
# implied, statutory or otherwise. The term "warranty" used herein
# includes, but is not limited to, any warranty of the quality,
# performance, merchantability and fitness for a particular purpose of
# the program and the nonexistence of any infringement or violation of
# any right of any third party.
#
# Each user of the program will agree and understand, and be deemed to
# have agreed and understood, that there is no warranty whatsoever for
# the program and, accordingly, the entire risk arising from or
# otherwise connected with the program is assumed by the user.
#
# Therefore, neither ICOT, the copyright holder, or any other
# organization that participated in or was otherwise related to the
# development of the program and their respective officials, directors,
# officers and other employees shall be held liable for any and all
# damages, including, without limitation, general, special, incidental
# and consequential damages, arising out of or otherwise in connection
# with the use or inability to use the program or any product, material
# or result produced or otherwise obtained by using the program,
# regardless of whether they have been advised of, or otherwise had
# knowledge of, the possibility of such damages at any time during the
# project or thereafter. Each user will be deemed to have agreed to the
# foregoing by his or her commencement of use of the program. The term
# "use" as used herein includes, but is not limited to, the use,
# modification, copying and distribution of the program and the
# production of secondary products from the program.
#
# In the case where the program, whether in its original form or
```

```
# modified, was distributed or delivered to or received by a user from
# any person, organization or entity other than ICOT, unless it makes or
# grants independently of ICOT any specific warranty to the user in
# writing, such person, organization or entity, will also be exempted
# from and not be held liable to the user for any such damages as noted
# above as far as the program is concerned.
#
# -----COPYING.ipadic----END-----
```

3. Lao Word Break Dictionary Data (laodict.txt)

```
# Copyright (c) 2013 International Business Machines Corporation
# and others. All Rights Reserved.
#
# Project: http://code.google.com/p/lao-dictionary/
# Dictionary: http://lao-dictionary.googlecode.com/git/Lao-Dictionary.txt
# License: http://lao-dictionary.googlecode.com/git/Lao-Dictionary-LICENSE.txt
#           (copied below)
#
# This file is derived from the above dictionary, with slight
# modifications.
#
# -----
# Copyright (C) 2013 Brian Eugene Wilson, Robert Martin Campbell.
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification,
# are permitted provided that the following conditions are met:
#
#
# Redistributions of source code must retain the above copyright notice, this
# list of conditions and the following disclaimer. Redistributions in
# binary form must reproduce the above copyright notice, this list of
# conditions and the following disclaimer in the documentation and/or
# other materials provided with the distribution.
#
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
# COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
# INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
# (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
# SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
# STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
# OF THE POSSIBILITY OF SUCH DAMAGE.
#
# -----
```

4. Burmese Word Break Dictionary Data (burmesedict.txt)

```

# Copyright (c) 2014 International Business Machines Corporation
# and others. All Rights Reserved.
#
# This list is part of a project hosted at:
#   github.com/kanyawtech/myanmar-karen-word-lists
#
# -----
# Copyright (c) 2013, LeRoy Benjamin Sharon
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions
# are met: Redistributions of source code must retain the above
# copyright notice, this list of conditions and the following
# disclaimer. Redistributions in binary form must reproduce the
# above copyright notice, this list of conditions and the following
# disclaimer in the documentation and/or other materials provided
# with the distribution.
#
# Neither the name Myanmar Karen Word Lists, nor the names of its
# contributors may be used to endorse or promote products derived
# from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
# CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
# INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
# MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
# DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS
# BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
# EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED
# TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
# DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
# ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
# TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
# THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
# SUCH DAMAGE.
# -----

```

5. Time Zone Database

ICU uses the public domain data and code derived from Time Zone Database for its time zone support. The ownership of the TZ database is explained in BCP 175: Procedure for Maintaining the Time Zone Database section 7.

```

# 7. Database Ownership
#
# The TZ database itself is not an IETF Contribution or an IETF
# document. Rather it is a pre-existing and regularly updated work

```

```
# that is in the public domain, and is intended to remain in the
# public domain. Therefore, BCPs 78 [RFC5378] and 79 [RFC3979] do
# not apply to the TZ Database or contributions that individuals make
# to it. Should any claims be made and substantiated against the TZ
# Database, the organization that is providing the IANA
# Considerations defined in this RFC, under the memorandum of
# understanding with the IETF, currently ICANN, may act in accordance
# with all competent court orders. No ownership claims will be made
# by ICANN or the IETF Trust on the database or the code. Any person
# making a contribution to the database or code waives all rights to
# future claims in that contribution or in the TZ Database.
```

6. Google double-conversion

Copyright 2006-2011, the V8 project authors. All rights reserved.
 Redistribution and use in source and binary forms, with or without
 modification, are permitted provided that the following conditions are
 met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

"""

- libuv, located at deps/uv, is licensed as follows:

"""

libuv is licensed for use as follows:

====

Copyright (c) 2015-present libuv project contributors.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to

deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

====

This license applies to parts of libuv originating from the <https://github.com/joyent/libuv> repository:

====

Copyright Joyent, Inc. and other Node contributors. All rights reserved. Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

====

This license applies to all parts of libuv that are not externally maintained libraries.

The externally maintained libraries used by libuv are:

- tree.h (from FreeBSD), copyright Niels Provos. Two clause BSD license.
- inet_pton and inet_ntop implementations, contained in src/inet.c, are

copyright the Internet Systems Consortium, Inc., and licensed under the ISC license.

- stdint-msvc2008.h (from msinttypes), copyright Alexander Chemeris. Three clause BSD license.
 - pthread-fixes.c, copyright Google Inc. and Sony Mobile Communications AB. Three clause BSD license.
 - android-ifaddrs.h, android-ifaddrs.c, copyright Berkeley Software Design Inc, Kenneth MacKay and Emergya (Cloud4all, FP7/2007-2013, grant agreement n° 289016). Three clause BSD license.
- """

- llhttp, located at deps/llhttp, is licensed as follows:

"""

This software is licensed under the MIT License.

Copyright Fedor Indutny, 2018.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

"""

- OpenSSL, located at deps/openssl, is licensed as follows:

"""

Copyright (c) 1998-2019 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. All advertising materials mentioning features or use of this software must display the following acknowledgment:

"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)"

4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.

5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.

6. Redistributions of any form whatsoever must retain the following acknowledgment:

"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

=====

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

""

- Punycode.js, located at lib/punycode.js, is licensed as follows:

""

Copyright Mathias Bynens <<https://mathiasbynens.be/>>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish,

distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

"""

- V8, located at deps/v8, is licensed as follows:

"""

This license applies to all parts of V8 that are not externally maintained libraries. The externally maintained libraries used by V8 are:

- PCRE test suite, located in test/mjsunit/third_party/regexp-pcre/regexp-pcre.js. This is based on the test suite from PCRE-7.3, which is copyrighted by the University of Cambridge and Google, Inc. The copyright notice and license are embedded in regexp-pcre.js.
- Layout tests, located in test/mjsunit/third_party/object-keys. These are based on layout tests from webkit.org which are copyrighted by Apple Computer, Inc. and released under a 3-clause BSD license.
- Strongtalk assembler, the basis of the files assembler-arm-inl.h, assembler-arm.cc, assembler-arm.h, assembler-ia32-inl.h, assembler-ia32.cc, assembler-ia32.h, assembler-x64-inl.h, assembler-x64.cc, assembler-x64.h, assembler-mips-inl.h, assembler-mips.cc, assembler-mips.h, assembler.cc and assembler.h. This code is copyrighted by Sun Microsystems Inc. and released under a 3-clause BSD license.
- Valgrind client API header, located at src/third_party/valgrind/valgrind.h. This is released under the BSD license.
- The Wasm C/C++ API headers, located at third_party/wasm-api/wasm.{h, hh}. This is released under the Apache license. The API's upstream prototype implementation also formed the basis of V8's implementation in src/wasm/c-api.cc.

These libraries have their own licenses; we recommend you read them, as their terms may differ from the terms below.

Further license information can be found in LICENSE files located in sub-directories.

Copyright 2014, the V8 project authors. All rights reserved.
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

"""

- SipHash, located at deps/v8/src/third_party/siphash, is licensed as follows:

"""

SipHash reference C implementation

Copyright (c) 2016 Jean-Philippe Aumasson <jeanphilippe.aumasson@gmail.com>

To the extent possible under law, the author(s) have dedicated all copyright and related and neighboring rights to this software to the public domain worldwide. This software is distributed without any warranty.

"""

- zlib, located at deps/zlib, is licensed as follows:

"""

zlib.h -- interface of the 'zlib' general purpose compression library
version 1.2.11, January 15th, 2017

Copyright (C) 1995-2017 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages

arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly
jloup@gzip.org

Mark Adler
madler@alumni.caltech.edu

"""

- npm, located at deps/npm, is licensed as follows:

"""

The npm application
Copyright (c) npm, Inc. and Contributors
Licensed on the terms of The Artistic License 2.0

Node package dependencies of the npm application
Copyright (c) their respective copyright owners
Licensed on their respective license terms

The npm public registry at <https://registry.npmjs.org>
and the npm website at <https://www.npmjs.com>
Operated by npm, Inc.
Use governed by terms published on <https://www.npmjs.com>

"Node.js"
Trademark Joyent, Inc., <https://joyent.com>
Neither npm nor npm, Inc. are affiliated with Joyent, Inc.

The Node.js application
Project of Node Foundation, <https://nodejs.org>

The npm Logo
Copyright (c) Mathias Pettersson and Brian Hammond

"Gubblebum Blocky" typeface
Copyright (c) Tjarda Koster, <https://jelloween.deviantart.com>
Used with permission

The Artistic License 2.0

Copyright (c) 2000-2006, The Perl Foundation.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

This license establishes the terms under which a given free software Package may be copied, modified, distributed, and/or redistributed. The intent is that the Copyright Holder maintains some artistic control over the development of that Package while still keeping the Package available as open source and free software.

You are always permitted to make arrangements wholly outside of this license directly with the Copyright Holder of a given Package. If the terms of this license do not permit the full use that you propose to make of the Package, you should contact the Copyright Holder and seek a different licensing arrangement.

Definitions

"Copyright Holder" means the individual(s) or organization(s) named in the copyright notice for the entire Package.

"Contributor" means any party that has contributed code or other material to the Package, in accordance with the Copyright Holder's procedures.

"You" and "your" means any person who would like to copy, distribute, or modify the Package.

"Package" means the collection of files distributed by the Copyright Holder, and derivatives of that collection and/or of those files. A given Package may consist of either the Standard Version, or a Modified Version.

"Distribute" means providing a copy of the Package or making it accessible to anyone else, or in the case of a company or organization, to others outside of your company or organization.

"Distributor Fee" means any fee that you charge for Distributing this Package or providing support for this Package to another party. It does not mean licensing fees.

"Standard Version" refers to the Package if it has not been modified, or has been modified only in ways explicitly requested by the Copyright Holder.

"Modified Version" means the Package, if it has been changed, and such changes were not explicitly requested by the Copyright Holder.

"Original License" means this Artistic License as Distributed with the Standard Version of the Package, in its current version or as it may be modified by The Perl Foundation in the future.

"Source" form means the source code, documentation source, and configuration files for the Package.

"Compiled" form means the compiled bytecode, object code, binary, or any other form resulting from mechanical transformation or translation of the Source form.

Permission for Use and Modification Without Distribution

(1) You are permitted to use the Standard Version and create and use Modified Versions for any purpose without restriction, provided that you do not Distribute the Modified Version.

Permissions for Redistribution of the Standard Version

(2) You may Distribute verbatim copies of the Source form of the Standard Version of this Package in any medium without restriction, either gratis or for a Distributor Fee, provided that you duplicate all of the original copyright notices and associated disclaimers. At your discretion, such verbatim copies may or may not include a Compiled form of the Package.

(3) You may apply any bug fixes, portability changes, and other modifications made available from the Copyright Holder. The resulting Package will still be considered the Standard Version, and as such will be subject to the Original License.

Distribution of Modified Versions of the Package as Source

(4) You may Distribute your Modified Version as Source (either gratis or for a Distributor Fee, and with or without a Compiled form of the Modified Version) provided that you clearly document how it differs from the Standard Version, including, but not limited to, documenting any non-standard features, executables, or modules, and provided that you do at least ONE of the following:

(a) make the Modified Version available to the Copyright Holder of the Standard Version, under the Original License, so that the Copyright Holder may include your modifications in the Standard Version.

(b) ensure that installation of your Modified Version does not prevent the user installing or running the Standard Version. In addition, the Modified Version must bear a name that is different from the name of the Standard Version.

(c) allow anyone who receives a copy of the Modified Version to make the Source form of the Modified Version available to others under

- (i) the Original License or
- (ii) a license that permits the licensee to freely copy, modify and redistribute the Modified Version using the same licensing terms that apply to the copy that the licensee received, and requires that the Source form of the Modified Version, and of any works derived from it, be made freely available in that license fees are prohibited but Distributor Fees are allowed.

Distribution of Compiled Forms of the Standard Version or Modified Versions without the Source

(5) You may Distribute Compiled forms of the Standard Version without the Source, provided that you include complete instructions on how to get the Source of the Standard Version. Such instructions must be valid at the time of your distribution. If these instructions, at any time while you are carrying out such distribution, become invalid, you must provide new instructions on demand or cease further distribution. If you provide valid instructions or cease distribution within thirty days after you become aware that the instructions are invalid, then you do not forfeit any of your rights under this license.

(6) You may Distribute a Modified Version in Compiled form without the Source, provided that you comply with Section 4 with respect to the Source of the Modified Version.

Aggregating or Linking the Package

(7) You may aggregate the Package (either the Standard Version or Modified Version) with other packages and Distribute the resulting aggregation provided that you do not charge a licensing fee for the Package. Distributor Fees are permitted, and licensing fees for other components in the aggregation are permitted. The terms of this license apply to the use and Distribution of the Standard or Modified Versions as included in the aggregation.

(8) You are permitted to link Modified and Standard Versions with other works, to embed the Package in a larger work of your own, or to build stand-alone binary or bytecode versions of applications that include the Package, and Distribute the result without restriction, provided the result does not expose a direct interface to the Package.

Items That are Not Considered Part of a Modified Version

(9) Works (including, but not limited to, modules and scripts) that merely extend or make use of the Package, do not, by themselves, cause

the Package to be a Modified Version. In addition, such works are not considered parts of the Package itself, and are not subject to the terms of this license.

General Provisions

(10) Any use, modification, and distribution of the Standard or Modified Versions is governed by this Artistic License. By using, modifying or distributing the Package, you accept this license. Do not use, modify, or distribute the Package, if you do not accept this license.

(11) If your Modified Version has been derived from a Modified Version made by someone other than you, you are nevertheless required to ensure that your Modified Version complies with the requirements of this license.

(12) This license does not grant you the right to use any trademark, service mark, tradename, or logo of the Copyright Holder.

(13) This license includes the non-exclusive, worldwide, free-of-charge patent license to make, have made, use, offer to sell, sell, import and otherwise transfer the Package with respect to any patent claims licensable by the Copyright Holder that are necessarily infringed by the Package. If you institute patent litigation (including a cross-claim or counterclaim) against any party alleging that the Package constitutes direct or contributory patent infringement, then this Artistic License to you shall terminate on the date that such litigation is filed.

(14) Disclaimer of Warranty:

THE PACKAGE IS PROVIDED BY THE COPYRIGHT HOLDER AND CONTRIBUTORS "AS IS' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES. THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT ARE DISCLAIMED TO THE EXTENT PERMITTED BY YOUR LOCAL LAW. UNLESS REQUIRED BY LAW, NO COPYRIGHT HOLDER OR CONTRIBUTOR WILL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING IN ANY WAY OUT OF THE USE OF THE PACKAGE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

"""

- GYP, located at tools/gyp, is licensed as follows:

"""

Copyright (c) 2009 Google Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

"""

- inspector_protocol, located at tools/inspector_protocol, is licensed as follows:

"""

```
// Copyright 2016 The Chromium Authors. All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions are
// met:
//
//   * Redistributions of source code must retain the above copyright
//     notice, this list of conditions and the following disclaimer.
//   * Redistributions in binary form must reproduce the above
//     copyright notice, this list of conditions and the following disclaimer
//     in the documentation and/or other materials provided with the
//     distribution.
//   * Neither the name of Google Inc. nor the names of its
//     contributors may be used to endorse or promote products derived from
//     this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
// OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
// SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
// LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
// DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
// THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
```

// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

"""

- jinja2, located at tools/inspector_protocol/jinja2, is licensed as follows:

"""

Copyright (c) 2009 by the Jinja Team, see AUTHORS for more details.

Some rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * The names of the contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

"""

- markupsafe, located at tools/inspector_protocol/markupsafe, is licensed as follows:

"""

Copyright (c) 2010 by Armin Ronacher and contributors. See AUTHORS for more details.

Some rights reserved.

Redistribution and use in source and binary forms of the software as well as documentation, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * The names of the contributors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE AND DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

"""

- cpplint.py, located at tools/cpplint.py, is licensed as follows:

"""

Copyright (c) 2009 Google Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT

(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

"""

- ESLint, located at tools/node_modules/eslint, is licensed as follows:

"""

Copyright JS Foundation and other contributors, <https://js.foundation>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

"""

- babel-eslint, located at tools/node_modules/babel-eslint, is licensed as follows:

"""

Copyright (c) 2014-2016 Sebastian McKenzie <sebmck@gmail.com>

MIT License

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

- gtest, located at test/cctest/gtest, is licensed as follows:

Copyright 2008, Google Inc.
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- ngnhttp2, located at deps/ngnhttp2, is licensed as follows:

The MIT License

Copyright (c) 2012, 2014, 2015, 2016 Tatsuhiro Tsujikawa
Copyright (c) 2012, 2014, 2015, 2016 ngnhttp2 contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

"""

- node-inspect, located at deps/node-inspect, is licensed as follows:

"""

Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

"""

- large_pages, located at src/large_pages, is licensed as follows:

"""

Copyright (C) 2018 Intel Corporation

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES

OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
OR OTHER DEALINGS IN THE SOFTWARE.

"""

- caja, located at lib/internal/freeze_intrinsics.js, is licensed as follows:

"""

Adapted from SES/Caja - Copyright (C) 2011 Google Inc.
Copyright (C) 2018 Agoric

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

"""

- brotli, located at deps/brotli, is licensed as follows:

"""

Copyright (c) 2009, 2010, 2013-2016 by the Brotli Authors.

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.

"""

- HdrHistogram, located at deps/histogram, is licensed as follows:

"""

The code in this repository was written by Gil Tene, Michael Barker,
and Matt Warren, and released to the public domain, as explained at
<http://creativecommons.org/publicdomain/zero/1.0/>

For users of this code who wish to consume it under the "BSD" license rather than under the public domain or CC0 contribution text mentioned above, the code found under this directory is *also* provided under the following license (commonly referred to as the BSD 2-Clause License). This license does not detract from the above stated release of the code into the public domain, and simply represents an additional license granted by the Author.

** Beginning of "BSD 2-Clause License" text. **

Copyright (c) 2012, 2013, 2014 Gil Tene
Copyright (c) 2014 Michael Barker
Copyright (c) 2014 Matt Warren
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

"""

- node-heapdump, located at `src/heap_utils.cc`, is licensed as follows:

"""

ISC License

Copyright (c) 2012, Ben Noordhuis <info@bnoordhuis.nl>

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES

WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

==== src/compat.h src/compat-inl.h ===

ISC License

Copyright (c) 2014, StrongLoop Inc.

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

"""

- rimraf, located at lib/internal/fs/rimraf.js, is licensed as follows:

"""

The ISC License

Copyright (c) Isaac Z. Schlueter and Contributors

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

"""

- uvwasi, located at deps/uvwasi, is licensed as follows:

"""

MIT License

Copyright (c) 2019 Colin Ihrig and Contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

"""

7.2.12. MICROSOFT SOFTWARE LICENSE TERMS

MICROSOFT VISUAL STUDIO CODE

These license terms are an agreement between you and Microsoft Corporation (or based on where you live, one of its affiliates). They apply to the software named above. The terms also apply to any Microsoft services or updates for the software, except to the extent those have different terms.

IF YOU COMPLY WITH THESE LICENSE TERMS, YOU HAVE THE RIGHTS BELOW.

1. INSTALLATION AND USE RIGHTS.

a. General. You may use any number of copies of the software to develop and test your applications, including deployment within your internal corporate network.

b. Demo use. The uses permitted above include use of the software in demonstrating your applications.

c. Third Party Components. The software may include third party components with separate legal notices or governed by other agreements, as may be described in the `ThirdPartyNotices` file accompanying the software.

d. Extensions. The software gives you the option to download other Microsoft and third party software packages from our extension marketplace or package managers. Those packages are under their own licenses, and not this agreement. Microsoft does not distribute, license or provide any warranties for any of the third party packages. By accessing or using our extension marketplace, you agree to the extension marketplace terms located at <https://aka.ms/vsmarketplace-ToU>.

2. DATA.

a. Data Collection. The software may collect information about you and your use of the software, and send that to Microsoft. Microsoft may use this information to provide services and improve our products and services. You may opt-out of many of these scenarios, but not all, as described in the product documentation located at https://code.visualstudio.com/docs/supporting/faq#_how-to-disable-telemetry-reporting.

There may also be some features in the software that may enable you and Microsoft to collect data from users of your applications. If you use these features, you must comply with applicable law, including providing appropriate notices to users of your applications together with Microsoft's privacy statement. Our privacy statement is located at <https://go.microsoft.com/fwlink/?LinkId=824704>. You can learn more about data collection and use in the help documentation and our privacy statement. Your use of the software operates as your consent to these practices.

c. Processing of Personal Data. To the extent Microsoft is a processor or subprocessor of personal data in connection with the software, Microsoft makes the commitments in the European Union General Data Protection Regulation Terms of the Online Services Terms to all customers effective May 25, 2018, at <https://go.microsoft.com/fwlink/?linkid=9840733>.

3. UPDATES. The software may periodically check for updates and download and install them for you. You may obtain updates only from Microsoft or authorized sources. Microsoft may need to update your system to provide you with updates. You agree to receive these automatic updates without any additional notice. Updates may not include or support all existing software features, services, or peripheral devices. If you do not want automatic updates, you may turn them off by following the instructions in the documentation at <https://go.microsoft.com/fwlink/?LinkId=616397>.

4. FEEDBACK. If you give feedback about the software to Microsoft, you give to Microsoft, without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You will not give feedback that is subject to a license that requires Microsoft to license its software or documentation to third parties because we include your feedback in them. These rights survive this agreement.

5. SCOPE OF LICENSE. This license applies to the Visual Studio Code product. Source code for Visual Studio Code is available at <https://github.com/Microsoft/vscode> under the MIT license agreement. The software is licensed, not sold. This agreement only gives you some rights to use the software. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the software only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the software that only allow you to use it in certain ways. You may not

reverse engineer, decompile or disassemble the software, or otherwise attempt to derive the source code for the software except and solely to the extent required by third party licensing terms governing use of certain open source components that may be included in the software;

remove, minimize, block or modify any notices of Microsoft or its suppliers in the software;

use the software in any way that is against the law;

share, publish, rent or lease the software, or provide the software as a stand-alone offering for others to use.

6. SUPPORT SERVICES. Because this software is "as is," we may not provide support services for it.

7. ENTIRE AGREEMENT. This agreement, and the terms for supplements, updates, Internet-based services and support services that you use, are the entire agreement for the software and support services.

8. EXPORT RESTRICTIONS. You must comply with all domestic and international export laws and regulations that apply to the software, which include restrictions on destinations, end-users, and end use. For further information on export restrictions, see <https://www.microsoft.com/exporting>.

9. APPLICABLE LAW. If you acquired the software in the United States, Washington

law applies to interpretation of and claims for breach of this agreement, and the laws of the state where you live apply to all other claims. If you acquired the software in any other country, its laws apply.

10. CONSUMER RIGHTS; REGIONAL VARIATIONS. This agreement describes certain legal rights. You may have other rights, including consumer rights, under the laws of your state or country. Separate and apart from your relationship with Microsoft, you may also have rights with respect to the party from which you acquired the software. This agreement does not change those other rights if the laws of your state or country do not permit it to do so. For example, if you acquired the software in one of the below regions, or mandatory country law applies, then the following provisions apply to you:

a. Australia. You have statutory guarantees under the Australian Consumer Law and nothing in this agreement is intended to affect those rights.

b. Canada. If you acquired this software in Canada, you may stop receiving updates by turning off the automatic update feature, disconnecting your device from the Internet (if and when you re-connect to the Internet, however, the software will resume checking for and installing updates), or uninstalling the software. The product documentation, if any, may also specify how to turn off updates for your specific device or software.

c. Germany and Austria.

Warranty. The properly licensed software will perform substantially as described in any Microsoft materials that accompany the software. However, Microsoft gives no contractual guarantee in relation to the licensed software.

Limitation of Liability. In case of intentional conduct, gross negligence, claims based on the Product Liability Act, as well as, in case of death or personal or physical injury, Microsoft is liable according to the statutory law.

Subject to the foregoing clause (ii), Microsoft will only be liable for slight negligence if Microsoft is in breach of such material contractual obligations, the fulfillment of which facilitate the due performance of this agreement, the breach of which would endanger the purpose of this agreement and the compliance with which a party may constantly trust in (so-called "cardinal obligations"). In other cases of slight negligence, Microsoft will not be liable for slight negligence.

11. DISCLAIMER OF WARRANTY. The software is licensed "as-is." You bear the risk of using it. Microsoft gives no express warranties, guarantees or conditions. To the extent permitted under your local laws, Microsoft excludes the implied warranties of merchantability, fitness for a particular purpose and non-infringement.

12. LIMITATION ON AND EXCLUSION OF DAMAGES. You can recover from Microsoft and its suppliers only direct damages up to U.S. \$5.00. You cannot recover any other damages, including consequential, lost profits, special, indirect or incidental damages.

This limitation applies to (a) anything related to the software, services, content (including code) on third party Internet sites, or third party applications; and (b) claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your state or country may not allow the exclusion or limitation of incidental, consequential or other damages.

Part III: devonfw for Java (devon4j)

The devonfw community dev-SNAPSHOT, 2021-03-04_16.11.10

devonfw provides a solution to building applications which combine best-in-class frameworks and libraries as well as industry proven practices and code conventions. It massively speeds up development, reduces risks and helps you to deliver better results.

The following sections contain the complete compendium of [devon4j](#), the Java stack of devonfw. You can also read the latest version of this documentation online in the [devon4j wiki](#) or at [devon4j on devonfw.com](#).

8. Architecture

There are many different views that are summarized by the term *architecture*. First, we will introduce the [key principles](#) and [architecture principles](#) of devonfw. Then, we will go into details of the [the architecture of an application](#).

8.1. Key Principles

For devonfw we follow these fundamental key principles for all decisions about architecture, design, or choosing standards, libraries, and frameworks:

- **KISS**
Keep it small and simple
- **Open**
Commitment to open standards and solutions (no required dependencies to commercial or vendor-specific standards or solutions)
- **Patterns**
We concentrate on providing patterns, best-practices and examples rather than writing framework code.
- **Solid**
We pick solutions that are established and have been proven to be solid and robust in real-live (business) projects.

8.2. Architecture Principles

Additionally we define the following principles that our architecture is based on:

- **Component Oriented Design**
We follow a strictly component oriented design to address the following sub-principles:
 - [Separation of Concerns](#)
 - [Reusability](#) and avoiding [redundant code](#)
 - [Information Hiding](#) via component API and its exchangeable implementation treated as secret.
 - *Design by Contract* for self-contained, descriptive, and stable component APIs.
 - [Layering](#) as well as separation of business logic from technical code for better maintenance.
 - *Data Sovereignty* (and *high cohesion with low coupling*) says that a component is responsible for its data and changes to this data shall only happen via the component. Otherwise, maintenance problems will arise to ensure that data remains consistent. Therefore, interfaces of a component that may be used by other components are designed *call-by-value* and not *call-by-reference*.
- **Homogeneity**
Solve similar problems in similar ways and establish a uniform [code-style](#).

As an architect you should be prepared for the future by reading the [TechnoVision](#).

8.3. Application Architecture

For the architecture of an application we distinguish the following views:

- The [Business Architecture](#) describes an application from the business perspective. It divides the application into business components and with full abstraction of technical aspects.
- The [Technical Architecture](#) describes an application from the technical implementation perspective. It divides the application into technical layers and defines which technical products and frameworks are used to support these layers.
- The Infrastructure Architecture describes an application from the operational infrastructure perspective. It defines the nodes used to run the application including clustering, load-balancing and networking. This view is not explored further in this guide.

8.3.1. Business Architecture

The *business architecture* divides the application into *business components*. A business component has a well-defined responsibility that it encapsulates. All aspects related to that responsibility have to be implemented within that business component. Further, the business architecture defines the dependencies between the business components. These dependencies need to be free of cycles. A business component exports its functionality via well-defined interfaces as a self-contained API. A business component may use another business component via its API and compliant with the dependencies defined by the business architecture.

As the business domain and logic of an application can be totally different, the devonfw can not define a standardized business architecture. Depending on the business domain it has to be defined from scratch or from a domain reference architecture template. For very small systems it may be suitable to define just a single business component containing all the code.

8.3.2. Technical Architecture

The *technical architecture* divides the application into technical *layers* based on the [multilayered architecture](#). A layer is a unit of code with the same category such as a service or presentation logic. So, a layer is often supported by a technical framework. Each business component can therefore be split into *component parts* for each layer. However, a business component may not have component parts for every layer (e.g. only a presentation part that utilized logic from other components).

An overview of the technical reference architecture of the devonfw is given by [figure "Technical Reference Architecture"](#). It defines the following layers visualized as horizontal boxes:

- [client layer](#) for the front-end (GUI).
- [service layer](#) for the services used to expose functionality of the back-end to the client or other consumers.
- [batch layer](#) for exposing functionality in batch-processes (e.g. mass imports).
- [logic layer](#) for the business logic.

- **data-access layer** for the data access (esp. persistence).

Also, you can see the (business) components as vertical boxes (e.g. A and X) and how they are composed out of component parts each one assigned to one of the technical layers.

Further, there are technical components for cross-cutting aspects grouped by the gray box on the left. Here is a complete list:

- [Security](#)
- [Logging](#)
- [Monitoring](#)
- [Transaction-Handling](#)
- [Exception-Handling](#)
- [Internationalization](#)
- [Dependency-Injection](#)

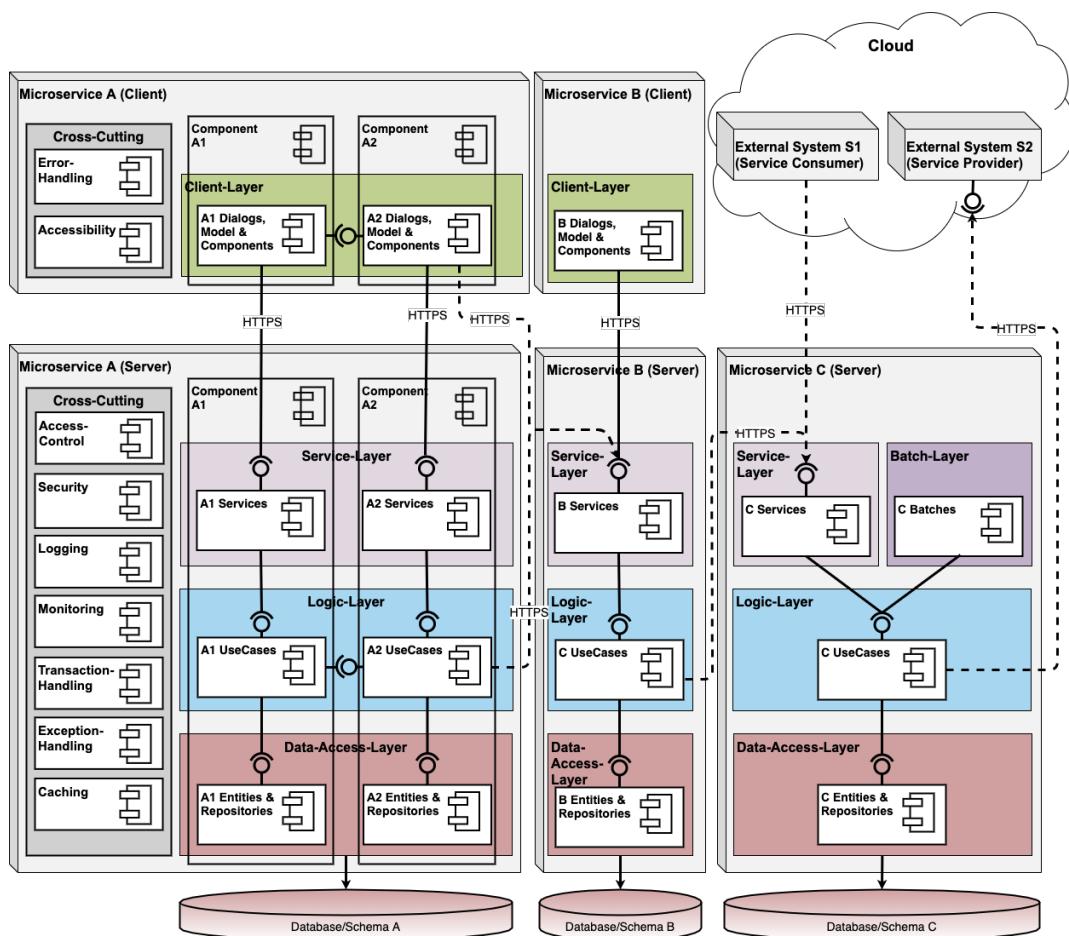


Figure 5. Technical Reference Architecture

Please click on the architecture image to open it as SVG and click on the layers and cross-cutting topics to open the according documentation guide.

We reflect this architecture in our code as described in our [coding conventions](#) allowing a traceability of business components, use-cases, layers, etc. into the code and giving developers a sound orientation within the project.

Further, the architecture diagram shows the allowed dependencies illustrated by the dark green connectors. Within a business component a component part can call the next component part on the layer directly below via a dependency on its API (vertical connectors). While this is natural and obvious, it is generally forbidden to have dependencies upwards the layers or to skip a layer by a direct dependency on a component part two or more layers below. The general dependencies allowed between business components are defined by the [business architecture](#). In our reference architecture diagram we assume that the business component [A1](#) is allowed to depend on component [A2](#). Therefore, a use-case within the logic component part of [A1](#) is allowed to call a use-case from [A2](#) via a dependency on the component API. The same applies for dialogs on the client layer. This is illustrated by the horizontal connectors. Please note that [persistence entities](#) are part of the API of the data-access component part so only the logic component part of the same business component may depend on them.

The technical architecture has to address non-functional requirements:

- **scalability**

is established by keeping state in the client and making the server state-less (except for login session). Via load-balancers new server nodes can be added to improve performance (horizontal scaling).

- **availability and reliability**

are addressed by clustering with redundant nodes avoiding any single-point-of failure. If one node fails the system is still available. Further, the software has to be robust so there are no dead-locks or other bad effects that can make the system unavailable or not reliable.

- **security**

is archived in the devonfw by the right templates and best-practices that avoid vulnerabilities. See [security guidelines](#) for further details.

- **performance**

is obtained by choosing the right products and proper configurations. While the actual implementation of the application matters for performance a proper design is important as it is the key to allow performance-optimizations (see e.g. [caching](#)).

Technology Stack

The technology stack of the devonfw is illustrated by the following table.

Table 23. Technology Stack of devonfw

Topic	Detail	Standard	Suggested implementation
runtime	language & VM	Java	Oracle JDK
runtime	servlet-container	JEE	tomcat
component management	dependency injection	JSR330 & JSR250	spring
configuration	framework	-	spring-boot
persistence	OR-mapper	JPA	hibernate
batch	framework	JSR352	spring-batch

Topic	Detail	Standard	Suggested implementation
service	SOAP services	JAX-WS	CXF
service	REST services	JAX-RS	CXF
logging	framework	slf4j	logback
validation	framework	beanvalidation/JSR303	hibernate-validator
security	Authentication & Authorization	JAAS	spring-security
monitoring	framework	JMX	spring
monitoring	HTTP Bridge	HTTP & JSON	jolokia
AOP	framework	dynamic proxies	spring AOP

8.4. Components

Following [separation-of-concerns](#) we divide an application into components using our [package-conventions](#) and [architecture-mapping](#). As described by the [architecture](#) each component is divided into these layers:

- [client-layer](#) with the dialogs to view and modify the component's data.
- [service-layer](#) with the services to access the component's data remotely.
- [logic-layer](#) with the [component-facade](#) providing the business-logic to manage the component's data.
- [dataaccess-layer](#) with the [entities](#) defining and the [repositories](#) (or [DAOs](#)) accessing the component's data.

Please note that only CRUD oriented components will have all four layers within the same component. Some types of applications may have completely different components for the client.

8.4.1. General Component

Cross-cutting aspects belong to the implicit component [general](#). It contains technical configurations and very general code that is not business specific. Such code shall not have any dependencies to other components and therefore business related code.

8.4.2. Business Component

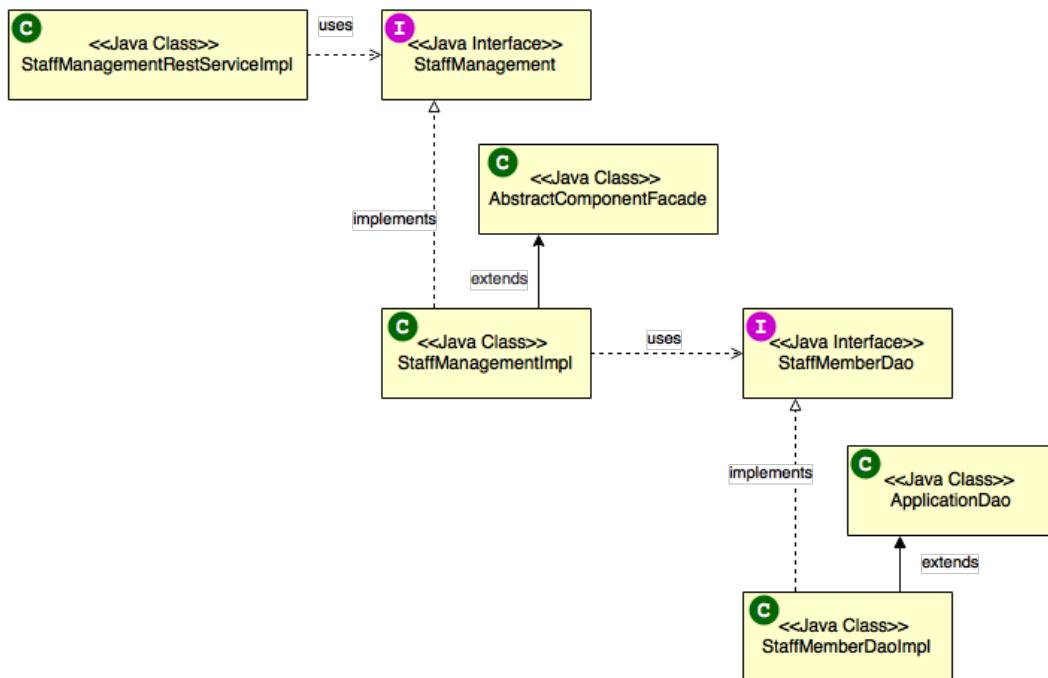
The [business-architecture](#) defines the business components with their allowed dependencies. A small application (microservice) may just have one component and no dependencies making it simple while the same architecture can scale up to large and complex applications (from bigger microservice up to modulith). Tailoring an business domain into applications and applications into components is a tricky task that needs the skills of an experienced architect. Also, the tailoring should follow the business and not split by technical reasons or only by size. Size is only an indicator but not a driver of tailoring. Whatever hypes like microservices are telling you, never get misled in this regard: If your system grows and reaches [MAX+1](#) lines of code, it is not the right motivation to split it into two microservices of [~MAX/2](#) lines of code - such approaches will waste huge amounts of money and lead to chaos.

8.4.3. App Component

Only in case you need cross-cutting code that aggregates another component you may introduce the component [app](#). It is allowed to depend on all other components but no other component may depend on it. With the modularity and flexibility of spring you typically do not need this. However, when you need to have a class that registers all services or component-facades using direct code dependencies, you can introduce this component.

8.4.4. Component Example

The following class diagram illustrates an example of the business component [Staffmanagement](#):

Service layer**Logic layer****Data access layer**

In this scheme, you can see the structure and flow from the [service-layer](#) (REST service call) via the [logic-layer](#) to the [dataaccess-layer](#) (and back).

9. Coding Conventions

The code should follow general conventions for Java (see [Oracle Naming Conventions](#), [Google Java Style](#), etc.). We consider this as common sense and provide configurations for [SonarQube](#) and related tools such as [Checkstyle](#) instead of repeating this here.

9.1. Naming

Besides general Java naming conventions, we follow the additional rules listed here explicitly:

- Always use short but speaking names (for types, methods, fields, parameters, variables, constants, etc.).
- For package segments and type names prefer singular forms (`CustomerEntity` instead of `CustomersEntity`). Only use plural forms when there is no singular or it is really semantically required (e.g. for a container that contains multiple of such objects).
- Avoid having duplicate type names. The name of a class, interface, enum or annotation should be unique within your project unless this is intentionally desired in a special and reasonable situation.
- Avoid artificial naming constructs such as prefixes (`I*`) or suffixes (`*IF`) for interfaces.
- Use CamelCase even for abbreviations (`XmlUtil` instead of `XMLUtil`)
- Avoid property/field names where the second character is upper-case at all (e.g. 'aBc'). See [#1095](#) for details.
- Names of Generics should be easy to understand. Where suitable follow the common rule `E=Element, T=Type, K=Key, V=Value` but feel free to use longer names for more specific cases such as `ID, DTO` or `ENTITY`. The capitalized naming helps to distinguish a generic type from a regular class.

9.2. Packages

Java Packages are the most important element to structure your code. We use a strict packaging convention to map technical layers and business components (slices) to the code (See [technical architecture](#) for further details). By using the same names in documentation and code we create a strong link that gives orientation and makes it easy to find from business requirements, specifications or story tickets into the code and back.

For an devon4j based application we use the following Java-Package schema:

```
<<rootpackage>>.<<application>>.<<component>>.<<layer>>.<<scope>>[.<<detail>>]*
```

E.g. in our example application we find the Spring Data repositories for the `ordermanagement` component in the package `com.devonfw.application.mtsj.ordermanagement.dataaccess.api.repo`

Table 24. Segments of package schema

Segment	Description	Example
«rootpackage»	Is the basic Java Package namespace of the organization or IT project owning the code following common Java Package conventions. Consists of multiple segments corresponding to the Internet domain of the organization.	com.devonfw.application.mtsj
«application»	The name of the application build in this project.	devonfw
«component»	The (business) component the code belongs to. It is defined by the business architecture and uses terms from the business domain. Use the implicit component general for code not belonging to a specific component (foundation code).	salesmanagement
«layer»	The name of the technical layer (See technical architecture) which is one of the predefined layers (dataaccess , logic , service , batch , gui , client) or common for code not assigned to a technical layer (datatypes, cross-cutting concerns).	dataaccess
«scope»	The scope which is one of api (official API to be used by other layers or components), base (basic code to be reused by other implementations) and impl (implementation that should never be imported from outside)	api
«detail»	Here you are free to further divide your code into sub-components and other concerns according to the size of your component part.	dao

Please note that for library modules where we use `com.devonfw.module` as «`basepackage`» and the name of the module as «`component`». E.g. the API of our `beanmapping` module can be found in the package `com.devonfw.module.beanmapping.common.api`.

9.3. Architecture Mapping

We combine the above naming and packaging conventions to map the entire architecture to the code. This also allows tools such as [CobiGen](#) or [sonar-devon4j-plugin](#) to "understand" the code. Also this helps developers going from one devon4j project to the next one to quickly understand the code-base. If every developer knows where to find what, the project gets more efficient. A long time ago maven standardized the project structure with `src/main/java`, etc. and turned chaos into structure. With devonfw we experienced the same for the codebase (what is inside `src/main/java`).

Listing 6. Architecture mapped to code

```
<<rootpackage>>.<<application>>
  .<<component>>
    .common
      .api[.<<detail>>]
        .datatype
          .<<Datatype>>
          .<<BusinessObject>>
        .impl[.<<detail>>]
          .<<Aspect>>ConfigProperties
          .<<Datatype>>JsonSerializer
          .<<Datatype>>JsonDeserializer
    .dataaccess
      .api[.<<detail>>]
        .repo
          .<<BusinessObject>>Repository
        .dao (alternative to repo)
          .<<BusinessObject>>Dao (alternative to Repository)
          .<<BusinessObject>>Entity
        .impl[.<<detail>>]
          .dao (alternative to repo)
            .<<BusinessObject>>DaoImpl (alternative to Repository)
          .<<Datatype>>AttributeConverter
    .logic
      .api
        .[<<detail>>.]to
          .<<MyCustom>><<To
          .<<DataStructure>>Embeddable
          .<<BusinessObject>>Eto
          .<<BusinessObject>><<Subset>>Cto
        .[<<detail>>.]usecase
          .UcFind<<BusinessObject>>
          .UcManage<<BusinessObject>>
          .Uc<<Operation>><<BusinessObject>>
        .<<Component>>
      .base
        .[<<detail>>.]usecase
          .Abstract<<BusinessObject>>Uc
      .impl
        .[<<detail>>.]usecase
```



9.4. Code Tasks

Code spots that need some rework can be marked with the following tasks tags. These are already properly pre-configured in your development environment for auto completion and to view tasks you are responsible for. It is important to keep the number of code tasks low. Therefore, every member of the team should be responsible for the overall code quality. So if you change a piece of code and hit a code task that you can resolve in a reliable way, please do this as part of your change.

and remove the according tag.

9.4.1. TODO

Used to mark a piece of code that is not yet complete (typically because it can not be completed due to a dependency on something that is not ready).

```
// TODO <><author></> <><description></>
```

A TODO tag is added by the author of the code who is also responsible for completing this task.

9.4.2. FIXME

```
// FIXME <><author></> <><description></>
```

A FIXME tag is added by the author of the code or someone who found a bug he can not fix right now. The «author» who added the FIXME is also responsible for completing this task. This is very similar to a TODO but with a higher priority. FIXME tags indicate problems that should be resolved before a release is completed while TODO tags might have to stay for a longer time.

9.4.3. REVIEW

```
// REVIEW <><responsible></> (<><reviewer></>) <><description></>
```

A REVIEW tag is added by a reviewer during a code review. Here the original author of the code is responsible to resolve the REVIEW tag and the reviewer is assigning this task to him. This is important for feedback and learning and has to be aligned with a review "process" where people talk to each other and get into discussion. In smaller or local teams a peer-review is preferable but this does not scale for large or even distributed teams.

9.5. Code-Documentation

As a general goal, the code should be easy to read and understand. Besides, clear naming the documentation is important. We follow these rules:

- APIs (especially component interfaces) are properly documented with JavaDoc.
- JavaDoc shall provide actual value - we do not write JavaDoc to satisfy tools such as checkstyle but to express information not already available in the signature.
- We make use of `{@link}` tags in JavaDoc to make it more expressive.
- JavaDoc of APIs describes how to use the type or method and not how the implementation internally works.
- To document implementation details, we use code comments (e.g. `// we have to flush explicitly to ensure version is up-to-date`). This is only needed for complex logic.

- Avoid the pointless `{@inheritDoc}` as since Java 1.5 there is the `@Override` annotation for overridden methods and your JavaDoc is inherited automatically even without any JavaDoc comment at all.

9.6. Code-Style

This section gives you best practices to write better code and avoid pitfalls and mistakes.

9.6.1. BLOBs

Avoid using `byte[]` for BLOBs as this will load them entirely into your memory. This will cause performance issues or out of memory errors. Instead, use streams when dealing with BLOBs. For further details see [BLOB support](#).

9.6.2. Closing Resources

Resources such as streams (`InputStream`, `OutputStream`, `Reader`, `Writer`) or transactions need to be handled properly. Therefore, it is important to follow these rules:

- Each resource has to be closed properly, otherwise you will get out of file handles, TX sessions, memory leaks or the like
- Where possible avoid to deal with such resources manually. That is why we are recommending `@Transactional` for transactions in devonfw (see [Transaction Handling](#)).
- In case you have to deal with resources manually (e.g. binary streams) ensure to close them properly. See the example below for details.

Closing streams and other such resources is error prone. Have a look at the following example:

```
try {
    InputStream in = new FileInputStream(file);
    readData(in);
    in.close();
} catch (IOException e) {
    throw new IllegalStateException("Failed to read data.", e);
}
```

The code above is wrong as in case of an `IOException` the `InputStream` is not properly closed. In a server application such mistakes can cause severe errors that typically will only occur in production. As such resources implement the `AutoCloseable` interface you can use the `try-with-resource` syntax to write correct code. The following code shows a correct version of the example:

```
try (InputStream in = new FileInputStream(file)) {
    readData(in);
} catch (IOException e) {
    throw new IllegalStateException("Failed to read data.", e);
}
```

9.6.3. Catching and handling Exceptions

When catching exceptions always ensure the following:

- Never call `printStackTrace()` method on an exception
- Either log or wrap and re-throw the entire caught exception. Be aware that the cause(s) of an exception is very valuable information. If you loose such information by improper exception-handling you may be unable to properly analyse production problems what can cause severe issues.
 - If you wrap and re-throw an exception ensure that the caught exception is passed as cause to the newly created and thrown exception.
 - If you log an exception ensure that the entire exception is passed as argument to the logger (and not only the result of `getMessage()` or `toString()` on the exception).
- See [exception handling](#)

9.6.4. Lambdas and Streams

With Java8 you have cool new features like lambdas and monads like (`Stream`, `CompletableFuture`, `Optional`, etc.). However, these new features can also be misused or led to code that is hard to read or debug. To avoid pain, we give you the following best practices:

1. Learn how to use the new features properly before using. Developers are often keen on using cool new features. When you do your first experiments in your project code you will cause deep pain and might be ashamed afterwards. Please study the features properly. Even Java8 experts still write for loops to iterate over collections, so only use these features where it really makes sense.
2. Streams shall only be used in fluent API calls as a Stream can not be forked or reused.
3. Each stream has to have exactly one terminal operation.
4. Do not write multiple statements into lambda code:

```
collection.stream().map(x -> {
  Foo foo = doSomething(x);
  ...
  return foo;
}).collect(Collectors.toList());
```

This style makes the code hard to read and debug. Never do that! Instead, extract the lambda body to a private method with a meaningful name:

```
collection.stream().map(this::convertToFoo).collect(Collectors.toList());
```

5. Do not use `parallelStream()` in general code (that will run on server side) unless you know exactly what you are doing and what is going on under the hood. Some developers might think that using parallel streams is a good idea as it will make the code faster. However, if you want to

do performance optimizations talk to your technical lead (architect). Many features such as security and transactions will rely on contextual information that is associated with the current thread. Hence, using parallel streams will most probably cause serious bugs. Only use them for standalone (CLI) applications or for code that is just processing large amounts of data.

6. Do not perform operations on a sub-stream inside a lambda:

```
set.stream().flatMap(x -> x.getChildren().stream().filter(this::isSpecial)).  
collect(Collectors.toList()); // bad  
set.stream().flatMap(x -> x.getChildren().stream()).filter(this::isSpecial).  
collect(Collectors.toList()); // fine
```

7. Only use `collect` at the end of the stream:

```
set.stream().collect(Collectors.toList()).forEach(...); // bad  
set.stream().peek(...).collect(Collectors.toList()); // fine
```

8. Lambda parameters with Types inference

```
(a,b,c) -> a.toString() + Float.toString(b) + Arrays.toString(c) // fine  
(String a, Float b, Byte[] c) -> a.toString() + Float.toString(b) + Arrays.  
toString(c) // bad  
  
Collections.sort(personList, (p1, p2) -> p1.getSurName().compareTo(p2.getSurName()  
)); // fine  
Collections.sort(personList, (Person p1, Person p2) -> p1.getSurName().compareTo(  
p2.getSurName())); // bad
```

9. Avoid Return Braces and Statement

```
a -> a.toString(); // fine  
a -> { return a.toString(); } // bad
```

10. Avoid Parentheses with Single Parameter

```
a -> a.toString(); // fine  
(a) -> a.toString(); // bad
```

11. Avoid if/else inside foreach method. Use Filter method & comprehension

Bad

```
static public Iterator<String> TwitterHandles(Iterator<Author> authors, string
company) {
    final List result = new ArrayList<String> ();
    foreach (Author a : authors) {
        if (a.Company.equals(company)) {
            String handle = a.TwitterHandle;
            if (handle != null)
                result.Add(handle);
        }
    }
    return result;
}
```

Fine

```
public List<String> twitterHandles(List<Author> authors, String company) {
    return authors.stream()
        .filter(a -> null != a && a.getCompany().equals(company))
        .map(a -> a.getTwitterHandle())
        .collect(toList());
}
```

9.6.5. Optionals

With **Optional** you can wrap values to avoid a **NullPointerException** (NPE). However, it is not a good code-style to use **Optional** for every parameter or result to express that it may be null. For such case use **@Nullable** or even better instead annotate **@NotNull** where **null** is not acceptable.

However, **Optional** can be used to prevent NPEs in fluent calls (due to the lack of the elvis operator):

```
Long id;
id = fooCto.getBar().getBar().getId(); // may cause NPE
id = Optional.ofNullable(fooCto).map(FooCto::getBar).map(BarCto::getBar).map(BarEto::
:id).orElse(null); // null-safe
```

9.6.6. Encoding

Encoding (esp. Unicode with combining characters and surrogates) is a complex topic. Please study this topic if you have to deal with encodings and processing of special characters. For the basics follow these recommendations:

- When you have explicitly decided for an encoding always prefer Unicode (UTF-8 or better). This especially impacts your databases and has to be defined upfront as it typically can not be changed (easily) afterwards.
- Do not cast from **byte** to **char** (Unicode characters can be composed of multiple bytes, such cast

may only work for ASCII characters)

- Never convert the case of a String using the default locale (esp. when writing generic code like in devonfw). E.g. if you do "HI".toLowerCase() and your system locale is Turkish, then the output will be "hi" instead of "hi", which can lead to wrong assumptions and serious problems. If you want to do a "universal" case conversion always use explicitly an according western locale (e.g. `toLowerCase(Locale.US)`). Consider using a library (<https://github.com/m-m-m/util/blob/master/core/src/main/java/net/sf/mmm/util/lang/api/BasicHelper.java>) or create your own little static utility for that in your project.
- Write your code independent from the default encoding (system property `file.encoding`) - this will most likely differ in JUnit from production environment
 - Always provide an encoding when you create a `String` from `byte[]: new String(bytes, encoding)`
 - Always provide an encoding when you create a `Reader` or `Writer` : `new InputStreamReader(inStream, encoding)`

9.6.7. Prefer general API

Avoid unnecessary strong bindings:

- Do not bind your code to implementations such as `Vector` or `ArrayList` instead of `List`
- In APIs for input (=parameters) always consider to make little assumptions:
 - prefer `Collection` over `List` or `Set` where the difference does not matter (e.g. only use `Set` when you require uniqueness or highly efficient `contains`)
 - consider preferring `Collection<? extends Foo>` over `Collection<Foo>` when `Foo` is an interface or super-class

10. Project structure

The structure of a [devon4j](#) application is divided into the following modules:

- **api**: module containing the API of your application. The API contains the required artifacts to interact with your application via remote services. This can be [REST service interfaces](#), [transfer-objects](#) with their interfaces and [datatypes](#) but also [OpenAPI](#) or [gRPC](#) contracts.
- **core**: maven module containing the core of the application with service implementation, as well as entire [logic layer](#) and [dataaccess layer](#).
- **batch**: optional module for [batch layer](#)
- **server**: module that bundles the entire app (**core** with optional **batch**) typically as a bootified WAR file.

10.1. Make jar not war

First of all it is important to understand that the above defined structure aims to make modules like **api**, **core**, and **batch** reusable maven artifacts, that can be used as a regular maven dependency. On the other hand to build and deploy your application you want a final artifact that is containing all required 3rd party libraries. This artifact is not reusable as a maven dependency. That is exactly the purpose of the **server** module to build and package this final deployment artifact. By default we first build a regular **WAR** file with maven in your **server/target** directory (***-server-<>version>.war**) and in a second step create a bootified **WAR** out of this (***-server-bootified.war**). The bootified **WAR** file can then be started standalone (**java -jar <filename>.war**). However, it is also possible to deploy the same **WAR** file to a servlet container like [tomcat](#) or [jetty](#). As application servers and externally provided servlet containers are not recommendet anymore for various reasons (see [JEE](#)), you may also want to create a bootified **JAR** file instead. All you need to do in that case is to change the **packaging** in your **server/pom.xml** from **war** to **jar**.

10.2. Package Structure

The package structure of your code inside **src/main/java** (and **src/test/java**) of your modules is described in our coding conventions in the sections [packages](#) and [architecture-mapping](#).

11. Layers

11.1. Client Layer

There are various technical approaches to building GUI clients. The devonfw proposes rich clients that connect to the server via data-oriented services (e.g. using REST with JSON). In general, we have to distinguish among the following types of clients:

- web clients
- native desktop clients
- (native) mobile clients

Our main focus is on web-clients. In our sample application [my-thai-star](#) we offer a responsive web-client based on Angular following [devon4ng](#) that integrates seamlessly with the back ends of my-thai-star available for Java using devon4j as well as .NET/C# using [devon4net](#). For building angular clients read the separate [devon4ng guide](#).

11.1.1. JavaScript for Java Developers

In order to get started with client development as a Java developer we give you some hints to get started. Also if you are an experienced JavaScript developer and want to learn Java this can be helpful. First, you need to understand that the JavaScript ecosystem is as large as the Java ecosystem and developing a modern web client requires a lot of knowledge. The following table helps you as experienced developer to get an overview of the tools, configuration-files, and other related aspects from the new world to learn. Also it helps you to map concepts between the ecosystems. Please note that we list the tools recommended by devonfw here (and we know that there are alternatives not listed here such as gradle, grunt, bower, etc.).

Table 25. Aspects in JavaScript and Java ecosystem

Topic	Aspect	JavaScript	Java
Programming	Language	TypeScript (extends JavaScript)	Java
Runtime	VM	nodejs (or web-browser)	jvm
Dependency-Management	Tool	yarn (or npm)	maven
	Config	package.json	pom.xml
	Repository	npm repo	maven central (repo search)

Topic	Aspect	JavaScript	Java
Build-Management	Taskrunner	gulp	maven (or more comparable ant)
	Config	gulpfile.js (and gulp/*)	pom.xml (or build.xml)
	Clean cmd	gulp clean	mvn clean
	Build cmd	yarn install && gulp build:dist	mvn install (see lifecycle)
	Test cmd	gulp test	mvn test
Testing	Test-Tool	jasmine	junit
	Test-Framework	karma	junit / surefire
	Browser Testing	PhantomJS	Selenium
	Extensions	karma-*, PhantomJS for browser emulation	AssertJ,*Unit and spring-test, etc.)
Code Analysis	Code Coverage	karma-coverage (and remap-istanbul for TypeScript)	JaCoCo/EclEmma
Development	IDE	MS VS Code or IntelliJ	Eclipse or IntelliJ
	Framework	Angular (etc.)	Spring (etc.)

11.2. Service Layer

The service layer is responsible for exposing functionality made available by the [logical layer](#) to external consumers over a network via [technical protocols](#).

11.2.1. Types of Services

We distinguish between the following types of services:

- **External Services**

are used for communication between different companies, vendors, or partners.

- **Internal Services**

are used for communication between different applications in the same application landscape of the same vendor.

- **Back-end Services**

are internal services between Java back-end components typically with different release and deployment cycles (if not Java consider this as an external service).

- **JS-Client Services**

are internal services provided by the Java back-end for JavaScript clients (GUI).

- **Java-Client Services**

are internal services provided by the Java back-end for a native Java client (JavaFx, EclipseRcp, etc.).

The choices for technology and protocols will depend on the type of service. The following table gives a guideline for aspects according to the service types.

Table 26. Aspects according to service-type

Aspect	External Service	Back-end Service	JS-Client Service	Java-Client Service
Versioning	required	required	not required	not required
Interoperability	mandatory	not required	implicit	not required
Recommended Protocol	SOAP or REST	REST	REST+JSON	REST

11.2.2. Versioning

For services consumed by other applications we use versioning to prevent incompatibilities between applications when deploying updates. This is done by the following conventions:

- We define a version number and prefix it with [v](#) (e.g. [v1](#)).
- If we support previous versions we use that version numbers as part of the Java package defining the service API (e.g. [com.foo.application.component.service.api.v1](#))
- We use the version number as part of the service name in the remote URL (e.g. <https://application.foo.com/services/rest/component/v1/resource>)

- Whenever breaking changes are made to the API, create a separate version of the service and increment the version (e.g. v1 → v2). The implementations of the different versions of the service contain compatibility code and delegate to the same unversioned use-case of the logic layer whenever possible.
- For maintenance and simplicity, avoid keeping more than one previous version.

11.2.3. Interoperability

For services that are consumed by clients with different technology, *interoperability* is required. This is addressed by selecting the right protocol, following protocol-specific best practices and following our considerations especially *simplicity*.

11.2.4. Service Considerations

The term *service* is quite generic and therefore easily misunderstood. It is a unit exposing coherent functionality via a well-defined interface over a network. For the design of a service, we consider the following aspects:

- **self-contained**

The entire API of the service shall be self-contained and have no dependencies on other parts of the application (other services, implementations, etc.).

- **idempotence**

E.g. creation of the same master-data entity has no effect (no error)

- **loosely coupled**

Service consumers have minimum knowledge and dependencies on the service provider.

- **normalized**

Complete, no redundancy, minimal

- **coarse-grained**

Service provides rather large operations (save entire entity or set of entities rather than individual attributes)

- **atomic**

Process individual entities (for processing large sets of data, use a **batch** instead of a service)

- **simplicity**

Avoid polymorphism, RPC methods with unique name per signature and no overloading, avoid attachments (consider separate download service), etc.

11.2.5. Security

Your services are the major entry point to your application. Hence, security considerations are important here.

See [REST Security](#).

11.3. Service-Versioning

This guide describes the aspect and details about versioning of [services](#)

11.3.1. Motivation

Why versioning of services? First of all, you should only care about this topic if you really have to. Service versioning is complex and requires effort (time and budget). The best way to avoid this is to be smart in the first place when designing the service API. Further, if you are creating services where the only consumer is e.g. the web-client that you deploy together with the consumed services then you can change your service without the overhead to create new service versions and keeping old service versions for compatibility.

However, if the following indicators are given you typically need to do service versioning:

- Your service is part of a complex and distributed IT landscape
- Your service requires incompatible changes
- There are many consumers or there is at least one (relevant) consumer that can not be updated at the same time or is entirely out of control (unknown or totally different party/company)

What are incompatible changes?

- Almost any change when [SOAP](#) is used (as it changes the WSDL and breaks the contract). Therefore, we recommend to use [REST](#) instead. Then, only the following changes are critical.
 - A change where existing properties (attributes) have to change their name
 - A change where existing features (properties, operations, etc.) have to change their semantics (meaning)

What changes do not cause incompatibilities?

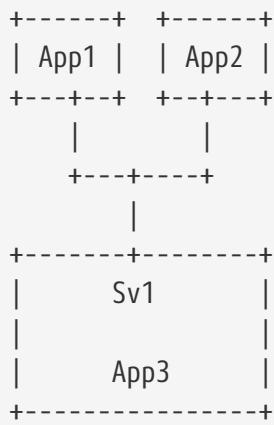
- Adding new service operations is entirely uncritical with [REST](#).
- Adding new properties is only a problem in the following cases:
 - Adding new mandatory properties to the input of a service is causing incompatibilities. This problem can be avoided by contract-design.
 - If a consumer is using a service to read data, modify it and then save it back via a service and a property is added to the data, then this property might be lost. This is not a problem with dynamic languages such as JavaScript/TypeScript but with strictly typed languages such as Java. In Java you will typically use structured typed transfer-objects (and not [Map<String, Object>](#)) so new properties that have been added but are not known to the consumer can not be mapped to the transfer-object and will be lost. When saving that transfer-object later the property will be gone. It might be impossible to determine the difference between a lost property and a property that was removed on purpose. This is a general problem that you need to be aware of and that you have to consider by your design in such situations.

Even if you hit an indicator for incompatible changes you can still think about adding a new service

operation instead of changing an existing one (and deprecating the old one). Be creative to simplify and avoid extra effort.

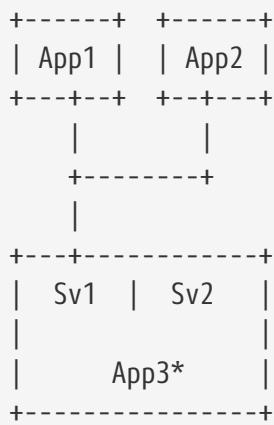
11.3.2. Procedure

The procedure when rolling out incompatible changes is illustrated by the following example:



So, here we see a simple example where **App3** provides a Service **S** in Version **v1** that is consumed both by **App1** and **App2**.

Now for some reason the service **S** has to be changed in an incompatible way to make it future-proof for demands. However, upgrading all 3 applications at the same time is not possible in this case for whatever reason. Therefore, service versioning is applied for the changes of **S**.



Now, **App3** has been upgraded and the new release was deployed. A new version **v2** of **S** has been added while **v1** is still kept for compatibility reasons and that version is still used by **App1** and **App2**.



Now, **App2** has been updated and deployed and it is using the new version **v2** of **S**.



Now, also **App1** has been updated and deployed and it is using the new version **v2** of **S**. The version **v1** of **S** is not used anymore. This can be verified via logging and monitoring.



Finally, version **v1** of the service **S** was removed from **App3** and the new release has been deployed.

11.3.3. Versioning Schema

In general anything can be used to differentiate versions of a service. Possibilities are:

- Code names (e.g. **Strawberry**, **Blueberry**, **Grapefruit**)
- Timestamps (**YYYYMMDD-HHmmSS**)
- Sequential version numbers (e.g. **v1**, **v2**, **v3**)
- Composed version numbers (e.g. **1.0.48-pre-alpha-3-20171231-235959-Strawberry**)

As we are following the KISS principle (see [key principles](#)) we propose to use sequential version numbers. These are short, clear, and easy while still allowing to see what version is after another one. Especially composed version numbers (even **1.1** vs. **2.0**) lead to decisions and discussions that easily waste more time than adding value. It is still very easy to maintain an Excel sheet or release-notes document that is explaining the changes for each version (**v1**, **v2**, **v3**) of a particular service.

We suggest to always add the version schema to the service URL to be prepared for service versioning even if service versioning is not (yet) actively used. For simplicity it is explicitly stated that you may even do incompatible changes to the current version (typically **v1**) of your service if you can update the according consumers within the same deployment.

11.3.4. Practice

So assuming you know that you have to do service versioning, the question is how to do it practically in the code. The approach for your devon4j project in case of code-first should be as described below:

- Determine which types in the code need to be changed. It is likely to be the API and implementation of the according service but it may also impact transfer objects and potentially even datatypes.
- Create new packages for all these concerned types containing the current version number (e.g. **v1**).
- Copy all these types to that new packages.
- Rename these copies so they carry the version number as suffix (e.g. **V1**).
- Increase the version of the service in the unversioned package (e.g. from **v1** to **v2**).
- Now you have two versions of the same service (e.g. **v1** and **v2**) but so far they behave exactly the same.
- You start with your actual changes and modify the original files that have been copied before.
- You will also ensure the links (import statements) of the copied types point to the copies with the version number
- This will cause incompatibilities (and compile errors) in the copied service. Therefore, you need to fix that service implementation to map from the old API to the new API and behavior. In some cases, this may be easy (e.g. mapping **x.y.z.v1.FooTo** to **x.y.z.FooTo** using [bean-mapping](#) with some custom mapping for the incompatible changes), in other cases this can get very complex. Be aware of this complexity from the start before you make your decision about service versioning.
- As far as possible this mapping should be done in the service-layer, not to pollute your business code in the core-layer with versioning-aspects. If there is no way to handle it in the service layer, e.g. you need some data from the persistence-layer, implement the "mapping" in the core-

layer then, but don't forget to remove this code, when removing the old service version.

- Finally, ensure that both the old service behaves as before as well as the new service works as planned.

Modularization

For modularization, we also follow the KISS principle (see [key principles](#)): we suggest to have one **api** module per application that will contain the most recent version of your service and get released with every release-version of the application. The compatibility code with the versioned packages will be added to the **core** module and therefore is not exposed via the **api** module (because it has already been exposed in the previous release of the app). This way, you can always determine for sure which version of a service is used by another application just by its maven dependencies.

The KISS approach with only a single module that may contain multiple services (e.g. one for each business component) will cause problems when you want to have mixed usages of service versions: You can not use an old version of one service and a new version of another service from the same APP as then you would need to have its API module twice as a dependency on different versions, which is not possible. However, to avoid complicated overhead we always suggest to follow this easy approach. Only if you come to the point that you really need this complexity you can still solve it (even afterwards by publishing another maven artefact). As we are all on our way to build more but smaller applications (SOA, microservices, etc.) we should always start simple and only add complexity when really needed.

The following example gives an idea of the structure:

```
/<<my-app>>
  └── /api
      └── /src/main/java/
          └── <<rootpackage>>/<<application>>/<<component>>
              ├── /common/api/to
              │   └── FooTo
              └── /service/api/rest
                  └── FooRestService
  └── /core
      └── /src/main/java/
          └── <<rootpackage>>/<<application>>/<<component>>
              ├── /common/api/to/v1
              │   └── FooToV1
              └── /service
                  ├── /api/rest/v1
                  │   └── FooRestServiceV1
                  └── impl/rest
                      └── /v1
                          └── FooRestServiceImplV1
                      └── FooRestServiceImpl
```

11.4. Logic Layer

The logic layer is the heart of the application and contains the main business logic. According to our [business architecture](#), we divide an application into [components](#). For each component, the logic layer defines a [component-facade](#). According to the complexity, you can further divide this into individual [use-cases](#). It is very important that you follow the links to understand the concept of component-facade and use-case in order to properly implement your business logic.

11.4.1. Responsibility

The logic layer is responsible to implement the business logic according to the specified functional demands and requirements. Therefore, it creates the actual value of the application. The following additional aspects are also included in its responsibility:

- [validation](#)
- [authorization](#)
- [transaction-handling](#) (in addition to [service layer](#)).

11.4.2. Security

The logic layer is the heart of the application. It is also responsible for authorization and hence security is important in this current case. Every method exposed in an interface needs to be annotated with an authorization check, stating what role(s) a caller must provide in order to be allowed to make the call. The authorization concept is described [here](#).

Direct Object References

A security threat are [Insecure Direct Object References](#). This simply gives you two options:

- avoid direct object references
- ensure that direct object references are secure

Especially when using REST, direct object references via technical IDs are common sense. This implies that you have a proper [authorization](#) in place. This is especially tricky when your authorization does not only rely on the type of the data and according to static permissions but also on the data itself. Vulnerabilities for this threat can easily happen by design flaws and inadvertence. Here is an example from our sample application:

We have a generic use-case to manage BLOBs. In the first place, it makes sense to write a generic REST service to load and save these BLOBs. However, the permission to read or even update such BLOB depends on the business object hosting the BLOB. Therefore, such a generic REST service would open the door for this OWASP A4 vulnerability. To solve this in a secure way, you need individual services for each hosting business object to manage the linked BLOB and have to check permissions based on the parent business object. In this example the ID of the BLOB would be the direct object reference and the ID of the business object (and a BLOB property indicator) would be the indirect object reference.

11.4.3. Component Facade

For each component of the application, the [logic layer](#) defines a component facade. This is an interface defining all business operations of the component. It carries the name of the component (`<>Component`) and has an implementation named `<>ComponentImpl` (see [implementation](#)).

API

The component facade interface defines the logic API of the component and has to be business oriented. This means that all parameters and return types of all methods from this API have to be business [transfer-objects](#), [datatypes](#) (`String`, `Integer`, `MyCustomerNumber`, etc.), or collections of these. The API may also only access objects of other business components listed in the (transitive) dependencies of the [business-architecture](#).

Here is an example how such an API may look like:

```
public interface Bookingmanagement {
    BookingEto findBooking(Long id);
    BookingCto findBookingCto(Long id);
    Page<BookingEto> findBookingEtos(BookingSearchCriteriaTo criteria);
    void approveBooking(BookingEto booking);
}
```

Implementation

The implementation of an interface from the [logic layer](#) (a component facade or a [use-case](#)) carries the name of that interface with the suffix `Impl` and is annotated with `@Named`. An implementation typically needs access to the persistent data. This is done by [injecting](#) the corresponding [repository](#) (or [DAO](#)). According to [data-sovereignty](#), only repositories of the same business component may be accessed directly. For accessing data from other components the implementation has to use the corresponding API of the logic layer (the component facade). Further, it shall not expose persistent entities from the [dataaccess layer](#) and has to map them to [transfer objects](#) using the [bean-mapper](#).

```

@Named
@Transactional
public class BookingmanagementImpl extends AbstractComponentFacade implements
Bookingmanagement {

    @Inject
    private BookingRepository bookingRepository;

    @Override
    public BookingEto findBooking(Long id) {

        LOG.debug("Get Booking with id {} from database.", id);
        BookingEntity entity = this.bookingRepository.findOne(id);
        return getBeanMapper().map(entity, BookingEto.class));
    }
}

```

As you can see, **entities** (`BookingEntity`) are mapped to corresponding **ETOs** (`BookingEto`). Further details about this can be found in [bean-mapping](#).

For complex applications, the component facade consisting of many different methods. For better maintainability in such case it is recommended to split it into separate **use-cases** that are then only aggregated by the component facade.

11.4.4. UseCase

A use-case is a small unit of the **logic layer** responsible for an operation on a particular **entity** (business object). It is defined by an interface (API) with its according implementation. Following our [architecture-mapping](#), use-cases are named `Uc<<Operation>><<BusinessObject>>[Impl]`. The prefix `Uc` stands for use-case and allows to easily find and identify them in your IDE. The `<<Operation>>` stands for a verb that is operated on the entity identified by `<<BusinessObject>>`. For **CRUD** we use the standard operations **Find** and **Manage** that can be generated by [CobiGen](#). This also separates read and write operations (e.g. if you want to do CQSR, or to configure read-only transactions for read operations).

Find

The `UcFind<<BusinessObject>>` defines all read operations to retrieve and search the `<<BusinessObject>>`. Here is an example:

```
public interface UcFindBooking {
    BookingEto findBooking(Long id);
    BookingCto findBookingCto(Long id);
    Page<BookingEto> findBookingEtos(BookingSearchCriteriaTo criteria);
    Page<BookingCto> findBookingCtos(BookingSearchCriteriaTo criteria);
}
```

Manage

The `UcManage<<BusinessObject>>` defines all CRUD write operations (create, update and delete) for the `<<BusinessObject>>`. Here is an example:

```
public interface UcManageBooking {
    BookingEto saveBooking(BookingEto booking);
    boolean deleteBooking(Long id);
}
```

Custom

Any other non CRUD operation `Uc<<Operation>><<BusinessObject>>` uses any other custom verb for `<<Operation>>`. Typically, such custom use-cases only define a single method. Here is an example:

```
public interface UcApproveBooking {
    void approveBooking(BookingEto booking);
}
```

Implementation

For the implementation of a use-case, the same rules that are described for the [component-facade implementation](#).

.

However, when following the use-case approach, your component facade simply changes to:

```
public interface Bookingmanagement extends UcFindBooking, UcManageBooking,
UcApproveBooking {
}
```

Where the implementation only delegates to the use-cases and gets entirely generated by CobiGen:

```
public class BookingmanagementImpl implements {
    @Inject
    private UcFindBooking ucFindBooking;

    @Inject
    private UcManageBooking ucManageBooking;

    @Inject
    private UcApproveBooking ucApproveBooking;

    @Override
    public BookingEto findBooking(Long id) {
        return this.ucFindBooking.findBooking(id);
    }

    @Override
    public Page<BookingEto> findBookingEtos(BookingSearchCriteriaTo criteria) {
        return this.ucFindBooking.findBookingEtos(criteria);
    }

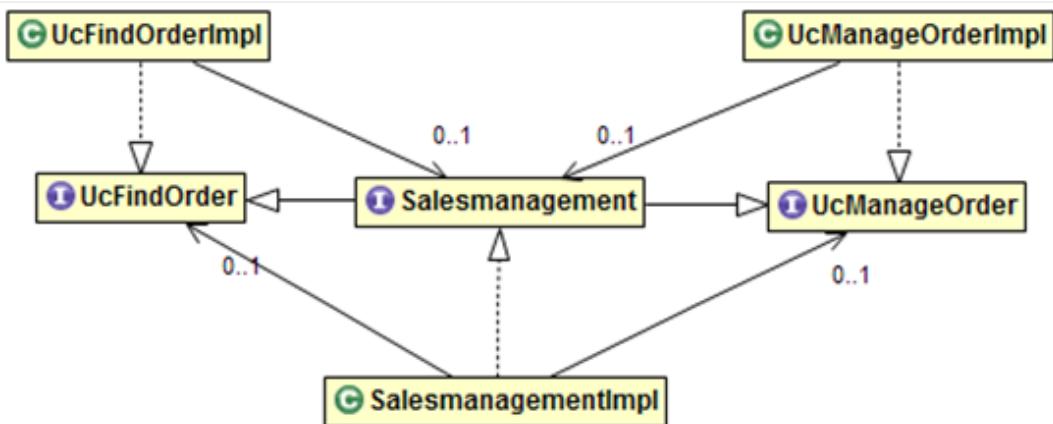
    @Override
    public BookingEto saveBooking(BookingEto booking) {
        return this.ucManageBooking.saveBooking(booking);
    }

    @Override
    public boolean deleteBooking(Long id) {
        return this.ucManageBooking.deleteBooking(booking);
    }

    @Override
    public void approveBooking(BookingEto booking) {
        this.ucApproveBooking.approveBooking(booking);
    }

    ...
}
```

This approach is also illustrated by the following UML diagram:



Internal use case

Sometimes, a component with multiple related entities and many use-cases needs to reuse business logic internally. Of course, this can be exposed as an official use-case API but this will imply using transfer-objects (ETOs) instead of entities. In some cases, this is undesired e.g. for better performance to prevent unnecessary mapping of entire collections of entities. In the first place, you should try to use abstract base implementations providing reusable methods the actual use-case implementations can inherit from. If your business logic is even more complex and you have multiple aspects of business logic to share and reuse but also run into multi-inheritance issues, you may also just create use-cases that have their interface located in the `impl` scope package right next to the implementation (or you may just skip the interface). In such a case, you may define methods that directly take or return entity objects. To avoid confusion with regular use-cases, we recommend to add the `Internal` suffix to the type name leading to `Uc<<Operation>><<BusinessObject>>Internal[Impl]`.

Injection issues

Technically, now you have two implementations of your use-case:

- the direct implementation of the use-case (`Uc*Impl`)
- the component facade implementation (`<<Component>>Impl`)

When injecting a use-case interface this could cause ambiguities. This is addressed as following:

- In the component facade implementation (`<<Component>>Impl`) spring is smart enough to resolve the ambiguity as it assumes that a spring bean never wants to inject itself (it can already be an access via `this`). Therefore, only the proper use-case implementation remains as a candidate and injection works as expected.
- In all other places, simply always inject the component facade interface instead of the use-case.

In case you might have the lucky occasion to hit this nice exception:

```
org.springframework.beans.factory.BeanCurrentlyInCreationException: Error creating bean with name 'uc...Impl': Bean with name 'uc...Impl' has been injected into other beans [...Impl] in its raw version as part of a circular reference, but has eventually been wrapped. This means that said other beans do not use the final version of the bean. This is often the result of over-eager type matching - consider using 'getBeanNamesOfType' with the 'allowEagerInit' flag turned off, for example.
```

To get rid of such an error you need to annotate your according implementation also with [@Lazy](#) in addition to [@Named](#).

11.5. Data-Access Layer

The data-access layer is responsible for all outgoing connections to access and process data. This is mainly about accessing data from a persistent data-store but also about invoking external services.

11.5.1. Database

You need to make your choice for a database. Options are documented [here](#).

The classical approach is to use a Relational Database Management System (RDMS). In such a case, we strongly recommend to follow our [JPA Guide](#). Some NoSQL databases are supported by [spring-data](#) so you can consider the [repository guide](#).

11.6. Batch Layer

We understand batch processing as a bulk-oriented, non-interactive, typically long running execution of tasks. For simplicity, we use the term "batch" or "batch job" for such tasks in the following documentation.

devonfw uses [Spring Batch](#) as a batch framework.

This guide explains how Spring Batch is used in devonfw applications. It focuses on aspects which are special to devonfw. If you want to learn about spring-batch you should adhere to springs references documentation.

There is an example of a simple batch implementation in the [my-thai-star batch module](#).

In this chapter, we will describe the overall architecture (especially concerning layering) and how to administer batches.

11.6.1. Layering

Batches are implemented in the batch layer. The batch layer is responsible for batch processes, whereas the business logic is implemented in the logic layer. Compared to the [service layer](#), you may understand the batch layer just as a different way of accessing the business logic. From a component point of view, each batch is implemented as a subcomponent in the corresponding business component. The business component is defined by the [business architecture](#).

Let's make an example for that. The sample application implements a batch for exporting ingredients. This ingredientExportJob belongs to the dishmanagement business component. So the ingredientExportJob is implemented in the following package:

```
<basepackage>.dishmanagement.batch.impl.*
```

Batches should invoke use cases in the logic layer for doing their work. Only "batch specific" technical aspects should be implemented in the batch layer.

Example: For a batch, which imports product data from a CSV file, this means that all code for actually reading and parsing the CSV input file is implemented in the batch layer. The batch calls the use case "create product" in the logic layer for actually creating the products for each line read from the CSV input file.

Directly accessing data access layer

In practice, it is not always appropriate to create use cases for every bit of work a batch should do. Instead, the data access layer can be used directly. An example for that is a typical batch for data retention which deletes out-of-time data. Often deleting, out-dated data is done by invoking a single SQL statement. It is appropriate to implement that SQL in a [Repository](#) or [DAO](#) method and call this method directly from the batch. But be careful: this pattern is a simplification which could lead to business logic cluttered in different layers, which reduces the maintainability of your application. It

is a typical design decision you have to make when designing your specific batches.

11.6.2. Project structure and packaging

Batches will be implemented in a separate Maven module to keep the application core free of batch dependencies. The batch module includes a dependency on the application core-module to allow the reuse of the use cases, DAOs etc. Additionally the batch module has dependencies on the required spring batch jars:

```
<dependencies>

    <dependency>
        <groupId>${project.groupId}</groupId>
        <artifactId>mtsj-core</artifactId>
        <version>dev-SNAPSHOT</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-batch</artifactId>
    </dependency>

</dependencies>
```

To allow an easy [start of the batches](#) from the command line it is advised to create a bootified jar for the batch module by adding the following to the [pom.xml](#) of the batch module:

```

<build>
  <resources>
    <resource>
      <directory>src/main/resources</directory>
      <filtering>true</filtering>
    </resource>
  </resources>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
        <excludes>
          <exclude>config/application.properties</exclude>
        </excludes>
      </configuration>
    </plugin>
    <!-- Create bootified jar for batch execution via command line.
         Your applications spring boot app is used as main-class.
    -->
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <configuration>
        <mainClass>com.devonfw.application.mtsj.SpringBootApp</mainClass>
        <classifier>bootified</classifier>
      </configuration>
      <executions>
        <execution>
          <goals>
            <goal>repackage</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

11.6.3. Implementation

Most of the details about implementation of batches is described in the [spring batch documentation](#). There is nothing special about implementing batches in devonfw. You will find an easy [example in my-thai-star](#).

11.6.4. Starting from command line

Devonfw advises to start batches via command line. This is most common to many ops teams and allows easy integration in existing [schedulers](#). In general batches are started with the following command:

```
java -jar <app>-batch-<version>-bootified.jar --spring.main.web-application-type=none
--spring.batch.job.enabled=true --spring.batch.job.names=<myJob> <params>
```

Parameter	Explanation
--spring.main.web-application-type=none	This disables the web app (e.g. Tomcat)
--spring.batch.job.names=<myJob>	This specifies the name of the job to run. If you leave this out ALL jobs will be executed. Which probably does not make too much sense.
<params>	(Optional) additional parameters which are passed to your job

This will launch your normal spring boot app, disables the web application part and runs the designated job via Spring Boots `org.springframework.boot.autoconfigure.batch.JobLauncherCommandLineRunner`.

Scheduling

In real world scheduling of batches is not as simple as it first might look like.

- Multiple batches have to be executed in order to achieve complex tasks. If one of those batches fails the further execution has to be stopped and operations should be notified for example.
- Input files or those created by batches have to be copied from one node to another.
- Scheduling batch executing could get complex easily (quarterly jobs, run job on first workday of a month, ...)

For devonfw we propose the batches themselves should not mess around with details of scheduling. Likewise your application should not do so. This complexity should be externalized to a dedicated batch administration service or scheduler. This service could be a complex product or a simple tool like cron. We propose [Rundeck](#) as an open source job scheduler.

This gives full control to operations to choose the solution which fits best into existing administration procedures.

11.6.5. Handling restarts

If you start a job with the same parameters set after a failed run (`BatchStatus.FAILED`) a restart will occur. In many cases your batch should then not reprocess all items it processed in the previous runs. For that you need some logic to start at the desired offset. There are different ways to implement such logic:

- Marking processed items in the database in a dedicated column
- Write all IDs of items to process in a separate table as an initialization step of your batch. You can then delete IDs of already processed items from that table during the batch execution.
- Storing restart information in spring's `ExecutionContext` (see below)

Using spring batch ExecutionContext for restarts

By implementing the `ItemStream` interface in your `ItemReader` or `ItemWriter` you may store information about the batch progress in the `ExecutionContext`. You will find an example for that in the CountJob in My Thai Star.

Additional hint: It is important that bean definition method of your `ItemReader/ItemWriter` return types implementing `ItemStream`(and not just `ItemReader` or `ItemWriter` alone). For that the `ItemStreamReader` and `ItemStreamWriter` interfaces are provided.

11.6.6. Exit codes

Your batches should create a meaningful exit code to allow reaction to batch errors e.g. in a `scheduler`. For that spring batch automatically registers an `org.springframework.boot.autoconfigure.batch.JobExecutionExitCodeGenerator`. To make this mechanism work your spring boot app main class as to populate this exit code to the JVM:

```
@SpringBootApplication
public class SpringBootApp {

    public static void main(String[] args) {
        if (Arrays.stream(args).anyMatch((String e) -> e.contains(
--spring.batch.job.names")))) {
            // if executing batch job, explicitly exit jvm to report error code from batch
            System.exit(SpringApplication.exit(SpringApplication.run(SpringBootApp.class,
args)));
        } else {
            // normal web application start
            SpringApplication.run(SpringBootApp.class, args);
        }
    }
}
```

11.6.7. Stop batches and manage batch status

Spring batch uses several database tables to store the status of batch executions. Each execution may have `different status`. You may use this mechanism to `gracefully stop batches`. Additionally in some edge cases (batch process crashed) the execution status may be in an undesired state. E.g. the state will be running, despite the process crashed sometime ago. For that cases you have to change the status of the execution in the database.

CLI-Tool

Devonfw provides a easy to use cli-tool to manage the executing status of your jobs. The tool is implemented in the devonfw module `devon4j-batch-tool`. It will provide a runnable jar, which may be used as follows:

List names of all previous executed jobs

```
java -D'spring.datasource.url=jdbc:h2:~/mts;AUTO_SERVER=TRUE' -jar devon4j-batch-tool.jar
```

```
jobs list
```

Stop job named 'countJob'

```
java -D'spring.datasource.url=jdbc:h2:~/mts;AUTO_SERVER=TRUE' -jar devon4j-batch-tool.jar
jobs stop countJob
```

Show help

```
java -D'spring.datasource.url=jdbc:h2:~/mts;AUTO_SERVER=TRUE' -jar devon4j-batch-tool.jar
```

As you can see each invocation includes the JDBC connection string to your database. This means that you have to make sure that the corresponding DB driver is in the classpath (the prepared jar only contains H2).

11.6.8. Authentication

Most business application incorporate authentication and authorization. Your spring boot application will implement some kind of security, e.g. integrated login with username+password or in many cases authentication via an existing IAM. For security reasons your batch should also implement an authentication mechanism and obey the authorization implemented in your application (e.g. via @RolesAllowed).

Since there are many different authentication mechanism we cannot provide an out-of-the-box solution in devonfw, but we describe a pattern how this can be implemented in devonfw batches.

We suggest to implement the authentication in a Spring Batch tasklet, which runs as the first step in your batch. This tasklet will do all of the work which is required to authenticate the batch. A simple example which authenticates the batch "locally" via username and password could be implemented like this:

```
@Named
public class SimpleAuthenticationTasklet implements Tasklet {

    @Override
    public RepeatStatus execute(StepContribution contribution, ChunkContext
chunkContext) throws Exception {

        String username = chunkContext.getStepContext().getStepExecution()
.getJobParameters().getString("username");
        String password = chunkContext.getStepContext().getStepExecution()
.getJobParameters().getString("password");
        Authentication authentication = new UsernamePasswordAuthenticationToken(username,
password);

        SecurityContextHolder.getContext().setAuthentication(authentication);
        return RepeatStatus.FINISHED;
    }

}
```

The username and password have to be supplied via two cli parameters `-username` and `-password`.

This implementation creates an "authenticated" **Authentication** and sets it in the Spring Security context. This is just for demonstration normally you should not provide passwords via command line. The actual authentication will be done automatically via Spring Security as in your "normal" application. If you have a more complex authentication mechanism in your application e.g. via OpenID connect just call this in the tasklet. Naturally you may read authentication parameters (e.g. secrets) from the command line or more securely from a configuration file.

In your Job Configuration set this tasklet as the first step:

```
@Configuration
@EnableBatchProcessing
public class BookingsExportBatchConfig {
    @Inject
    private JobBuilderFactory jobBuilderFactory;

    @Inject
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Job myBatchJob() {
        return this.jobBuilderFactory.get("myJob").start(myAuthenticationStep()).next(...).build();
    }

    @Bean
    public Step myAuthenticationStep() {
        return this.stepBuilderFactory.get("myAuthenticationStep").tasklet(
            myAuthenticatonTasklet()).build();
    }

    @Bean
    public Tasklet myAuthenticatonTasklet() {
        return new SimpleAuthenticationTasklet();
    }
}
```

11.6.9. Tipps & tricks

Identifying job parameters

Spring uses a jobs parameters to identify **job executions**. Parameters starting with "-" are not considered for identifying a job execution.

12. Guides

12.1. Dependency Injection

Dependency injection is one of the most important design patterns and is a key principle to a modular and component based architecture. The Java Standard for dependency injection is [javax.inject \(JSR330\)](#) that we use in combination with [JSR250](#).

There are many frameworks which support this standard including all recent Java EE application servers. We recommend to use [Spring](#) (also known as springframework) that we use in our example application. However, the modules we provide typically just rely on JSR330 and can be used with any compliant container.

12.1.1. Key Principles

A Bean in CDI (Contexts and Dependency-Injection) or Spring is typically part of a larger component and encapsulates some piece of logic that should in general be replaceable. As an example we can think of a Use-Case, Data-Access-Object (DAO), etc. As best practice we use the following principles:

- **Separation of API and implementation**

We create a self-contained API documented with JavaDoc. Then we create an implementation of this API that we annotate with [@Named](#). This implementation is treated as secret. Code from other components that wants to use the implementation shall only rely on the API. Therefore we use dependency injection via the interface with the [@Inject](#) annotation.

- **Stateless implementation**

By default implementations (CDI-Beans) shall always be stateless. If you store state information in member variables you can easily run into concurrency problems and nasty bugs. This is easy to avoid by using local variables and separate state classes for complex state-information. Try to avoid stateful CDI-Beans wherever possible. Only add state if you are fully aware of what you are doing and properly document this as a warning in your JavaDoc.

- **Usage of JSR330**

We use [javax.inject \(JSR330\)](#) and [JSR250](#) as a common standard that makes our code portable (works in any modern Java EE environment). However, we recommend to use the [springframework](#) as container. But we never use proprietary annotations such as [@Autowired](#) instead of standardized annotations like [@Inject](#). Generally we avoid proprietary annotations in business code ([common](#) and [logic layer](#)).

- **Simple Injection-Style**

In general you can choose between constructor, setter or field injection. For simplicity we recommend to do private field injection as it is very compact and easy to maintain. We believe that constructor injection is bad for maintenance especially in case of inheritance (if you change the dependencies you need to refactor all sub-classes). Private field injection and public setter injection are very similar but setter injection is much more verbose (often you are even forced to have javadoc for all public methods). If you are writing re-usable library code setter injection will make sense as it is more flexible. In a business application you typically do not need that and can save a lot of boiler-plate code if you use private field injection instead.

Nowadays you are using container infrastructure also for your tests (see [spring integration tests](#)) so there is no need to inject manually (what would require a public setter).

- **KISS**

To follow the KISS (keep it small and simple) principle we avoid advanced features (e.g. [AOP](#), non-singleton beans) and only use them where necessary.

12.1.2. Example Bean

Here you can see the implementation of an example bean using JSR330 and JSR250:

```
@Named
public class MyBeanImpl implements MyBean {
    @Inject
    private MyOtherBean myOtherBean;

    @PostConstruct
    public void init() {
        // initialization if required (otherwise omit this method)
    }

    @PreDestroy
    public void dispose() {
        // shutdown bean, free resources if required (otherwise omit this method)
    }
}
```

It depends on [MyOtherBean](#) that should be the interface of an other component that is injected into the field because of the [@Inject](#) annotation. To make this work there must be exactly one implementation of [MyOtherBean](#) in the container (in our case spring). In order to put a Bean into the container we use the [@Named](#) annotation so in our example we put [MyBeanImpl](#) into the container. Therefore it can be injected into all setters that take the interface [MyBean](#) as argument and are annotated with [@Inject](#).

In some situations you may have an Interface that defines a kind of "plugin" where you can have multiple implementations in your container and want to have all of them. Then you can request a list with all instances of that interface as in the following example:

```
@Inject
private List<MyConverter> converters;
```

Please note that when writing library code instead of annotating implementation with [@Named](#) it is better to provide [@Configuration](#) classes that choose the implementation via [@Bean](#) methods (see [@Bean documentation](#)). This way you can better "export" specific features instead of relying library users to do a component-scan to your library code and loose control on upgrades.

12.1.3. Bean configuration

Wiring and Bean configuration can be found in [configuration guide](#).

12.2. Configuration

An application needs to be configurable in order to allow internal setup (like CDI) but also to allow externalized configuration of a deployed package (e.g. integration into runtime environment). Using [Spring Boot](#) (must read: [Spring Boot reference](#)) we rely on a comprehensive configuration approach following a "convention over configuration" pattern. This guide adds on to this by detailed instructions and best-practices how to deal with configurations.

In general we distinguish the following kinds of configuration that are explained in the following sections:

- [Internal Application configuration](#) maintained by developers
- [Externalized Environment configuration](#) maintained by operators
- [Externalized Business configuration](#) maintained by business administrators

12.2.1. Internal Application Configuration

The application configuration contains all internal settings and wirings of the application (bean wiring, database mappings, etc.) and is maintained by the application developers at development time. There usually is a main configuration registered with main Spring Boot App, but differing configurations to support automated test of the application can be defined using profiles (not detailed in this guide).

Spring Boot Application

The devonfw recommends using [spring-boot](#) to build web applications. For a complete documentation see the [Spring Boot Reference Guide](#).

With `spring-boot` you provide a simple `main class` (also called `starter class`) like this:
`com.devonfw.mtsj.application`

```

@SpringBootApplication(exclude = { EndpointAutoConfiguration.class })
@EntityScan(basePackages = { "com.devonfw.mtsj.application" }, basePackageClasses = {
AdvancedRevisionEntity.class })
@EnableGlobalMethodSecurity(jsr250Enabled = true)
@ComponentScan(basePackages = { "com.devonfw.mtsj.application.general",
"com.devonfw.mtsj.application" })
public class SpringBootApp {

/**
 * Entry point for spring-boot based app
 *
 * @param args - arguments
 */
public static void main(String[] args) {

    SpringApplication.run(SpringBootApp.class, args);
}
}

```

In an devonfw application this main class is always located in the <basepackage> of the application package namespace (see [package-conventions](#)). This is because a spring boot application will automatically do a classpath scan for components (spring-beans) and entities in the package where the application main class is located including all sub-packages. You can use the [@ComponentScan](#) and [@EntityScan](#) annotations to customize this behaviour.

If you want to map spring configuration properties into your custom code please see [configuration mapping](#).

Standard beans configuration

For basic bean configuration we rely on spring boot using mainly configuration classes and only occasionally XML configuration files. Some key principle to understand Spring Boot auto-configuration features:

- Spring Boot auto-configuration attempts to automatically configure your Spring application based on the jar dependencies and annotated components found in your source code.
- Auto-configuration is non-invasive, at any point you can start to define your own configuration to replace specific parts of the auto-configuration by redefining your identically named bean (see also [exclude](#) attribute of [@SpringBootApplication](#) in example code above).

Beans are configured via annotations in your java code (see [dependency-injection](#)).

For technical configuration you will typically write additional spring config classes annotated with [@Configuration](#) that provide bean implementations via methods annotated with [@Bean](#). See [spring @Bean documentation](#) for further details. Like in XML you can also use [@Import](#) to make a [@Configuration](#) class include other configurations.

More specific configuration files (as required) reside in an adequately named subfolder of:

[src/main/resources/app](#)

BeanMapper Configuration

In case you are still using dozer, you will find further details in [bean-mapper configuration](#).

Security configuration

The abstract base class `BaseWebSecurityConfig` should be extended to configure web application security thoroughly. A basic and secure configuration is provided which can be overridden or extended by subclasses. Subclasses must use the `@Profile` annotation to further discriminate between beans used in production and testing scenarios. See the following example:

Listing 7. How to extend `BaseWebSecurityConfig` for Production and Test

```
@Configuration
@EnableWebSecurity
@Profile(SpringProfileConstants.JUNIT)
public class TestWebSecurityConfig extends BaseWebSecurityConfig {...}

@Configuration
@EnableWebSecurity
@Profile(SpringProfileConstants.NOT_JUNIT)
public class WebSecurityConfig extends BaseWebSecurityConfig {...}
```

See [WebSecurityConfig](#).

WebSocket configuration

A websocket endpoint is configured within the business package as a Spring configuration class. The annotation `@EnableWebSocketMessageBroker` makes Spring Boot registering this endpoint.

```
package your.path.to.the.websocket.config;
...
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig extends AbstractWebSocketMessageBrokerConfigurer {
...
```

Database Configuration

To choose database of your choice , set `spring.profiles.active=XXX` in [src/main/resources/config/application.properties](#). Also, one has to set all the active spring profiles in this `application.properties` and not in any of the other `application.properties`.

12.2.2. Externalized Configuration

Externalized configuration is a configuration that is provided separately to a deployment package and can be maintained undisturbed by re-deployments.

Environment Configuration

The environment configuration contains configuration parameters (typically port numbers, host names, passwords, logins, timeouts, certificates, etc.) specific for the different environments. These are under the control of the operators responsible for the application.

The environment configuration is maintained in `application.properties` files, defining various properties (see [common application properties](#) for a list of properties defined by the spring framework). These properties are explained in the corresponding configuration sections of the guides for each topic:

- [persistence configuration](#)
- [service configuration](#)
- [logging guide](#)

For a general understanding how spring-boot is loading and bootstrapping your `application.properties` see [spring-boot external configuration](#). The following properties files are used in every devonfw application:

- `src/main/resources/application.properties` providing a default configuration - bundled and deployed with the application package. It further acts as a template to derive a tailored minimal environment-specific configuration.
- `src/main/resources/config/application.properties` providing additional properties only used at development time (for all local deployment scenarios). This property file is excluded from all packaging.
- `src/test/resources/config/application.properties` providing additional properties only used for testing (JUnits based on [spring test](#)).

For other environments where the software gets deployed such as `test`, `acceptance` and `production` you need to provide a tailored copy of `application.properties`. The location depends on the deployment strategy:

- standalone run-able Spring Boot App using embedded tomcat: `config/application.properties` under the installation directory of the spring boot application.
- dedicated tomcat (one tomcat per app): `$CATALINA_BASE/lib/config/application.properties`
- tomcat serving a number of apps (requires expanding the wars): `$CATALINA_BASE/webapps/<app>/WEB-INF/classes/config`

In this `application.properties` you only define the minimum properties that are environment specific and inherit everything else from the bundled `src/main/resources/application.properties`. In any case, make very sure that the classloader will find the file.

Make sure your properties are thoroughly documented by providing a comment to each property. This inline documentation is most valuable for your operating department.

Business Configuration

Often applications do not need business configuration. In case they do it should typically be

editable by administrators via the GUI. The business configuration values should therefore be stored in the database in key/value pairs.

Therefore we suggest to create a dedicated table with (at least) the following columns:

- ID
- Property name
- Property type (Boolean, Integer, String)
- Property value
- Description

According to the entries in this table, an administrative GUI may show a generic form to modify business configuration. Boolean values should be shown as checkboxes, integer and string values as text fields. The values should be validated according to their type so an error is raised if you try to save a string in an integer property for example.

We recommend the following base layout for the hierarchical business configuration:

`component.[subcomponent].[subcomponent].propertyname`

12.2.3. Security

Often you need to have passwords (for databases, third-party services, etc.) as part of your configuration. These are typically environment specific (see above). However, with DevOps and continuous-deployment you might be tempted to commit such configurations into your version-control (e.g. [git](#)). Doing that with plain text passwords is a severe problem especially for production systems. Never do that! Instead we offer some suggestions how to deal with sensible configurations:

Password Encryption

A simple but reasonable approach is to configure the passwords encrypted with a master-password. The master-password should be a strong secret that is specific for each environment. It must never be committed to version-control. In order to support encrypted passwords in [spring-boot application.properties](#) all you need to do is to add [jasypt-spring-boot](#) as dependency in your [pom.xml](#)(please check for recent version):

```
<dependency>
  <groupId>com.github.ulisesbocchio</groupId>
  <artifactId>jasypt-spring-boot-starter</artifactId>
  <version>1.17</version>
</dependency>
```

This will smoothly integrate [jasypt](#) into your [spring-boot](#) application. Read this [HOWTO](#) to learn how to encrypt and decrypt passwords using jasypt. Here is a simple example output of an encrypted password (of course you have to use strong passwords instead of [secret](#) and [postgres](#) - this is only an example):

----ARGUMENTS-----

```
input: postgres
password: secret
```

----OUTPUT-----

```
jd5ZREpBqxuN9ok0IhnXabgw7V3EoG2p
```

The master-password can be configured on your target environment via the property `jasypt.encryptor.password`. As system properties given on the command-line are visible in the process list, we recommend to use an `config/application.yml` file only for this purpose (as we recommended to use `application.properties` for regular configs):

```
jasypt:
  encryptor:
    password: secret
```

(of course you will replace `secret` with a strong password). In case you happen to have multiple apps on the same machine, you can symlink the `application.yml` from a central place. Now you are able to put encrypted passwords into your `application.properties`

```
spring.datasource.password=ENC(jd5ZREpBqxuN9ok0IhnXabgw7V3EoG2p)
```

To prevent jasypt to throw an exception in dev or test scenarios simply put this in your local config (`src/main/config/application.properties` and same for `test`, see above for details):

```
jasypt.encryptor.password=none
```

Is this Security by Obscurity?

- Yes, from the point of view to protect the passwords on the target environment this is nothing but security by obscurity. If an attacker somehow got full access to the machine this will only cause him to spend some more time.
- No, if someone only gets the configuration file. So all your developers might have access to the version-control where the config is stored. Others might have access to the software releases that include this configs. But without the master-password that should only be known to specific operators none else can decrypt the password (except with brute-force what will take a very long time, see jasypt for details).

12.2.4. Mapping configuration to your code

If you are using `spring-boot` as suggested by `devon4j` your application can be configured by `application.properties` file as described in `configuration`. To get a single configuration option into your code for flexibility, you can use

```
@Value("${my.property.name}")
private String myConfigurableField;
```

Now, in your `application.properties` you can add the property:

```
my.property.name=my-property-value
```

You may even use `@Value("${my.property.name:my-default-value}")` to make the property optional.

Naming conventions for configuration properties

As a best practice your configuration properties should follow these naming conventions:

- build the property-name as a path of segments separated by the dot character (.)
- segments should get more specific from left to right
- a property-name should either be a leaf value or a tree node (prefix of other property-names) but never both! So never have something like `foo.bar=value` and `foo.bar.child=value2`.
- start with a segment namespace unique to your context or application
- a good example would be `<>myapp<>.billing.service.email.sender` for the sender address of billing service emails send by `<>myapp<>`.

Mapping advanced configuration

However, in many scenarios you will have features that require more than just one property. Injecting those via `@Value` is not leading to good code quality. Instead we create a class with the suffix `ConfigProperties` containing all configuration properties for our aspect that is annotated with `@ConfigurationProperties`:

```

@ConfigurationProperties(prefix = "myapp.billing.service")
public class BillingServiceConfigProperties {

    private final Email email = new Email();
    private final Smtplib smtp = new Smtplib();

    public Email getEmail() { return this.email; }
    public Email getSmtp() { return this.smtp; }

    public static class Email {

        private String sender;
        private String subject;

        public String getSender() { return this.sender; }
        public void setSender(String sender) { this.sender = sender; }
        public String getSubject() { return this.subject; }
        public void setSubject(String subject) { this.subject = subject; }
    }

    public static class Smtplib {

        private String host;
        private int port = 25;

        public String getHost() { return this.host; }
        public void setHost(String host) { this.host = host; }
        public int getPort() { return this.port; }
        public void setPort(int port) { this.port = port; }
    }
}

```

Of course this is just an example to demonstrate this feature of [spring-boot](#). In order to send emails you would typically use the existing [spring-email](#) feature. But as you can see this allows us to define and access our configuration in a very structured and comfortable way. The annotation `@ConfigurationProperties(prefix = "myapp.billing.service")` will automatically map spring configuration properties starting with `myapp.billing.service` via the according getters and setters into our `BillingServiceConfigProperties`. We can easily define defaults (e.g. `25` as default value for `myapp.billing.service.smtp.port`). Also `Email` or `Smtplib` could be top-level classes to be reused in multiple configurations. Of course you would also add helpful [JavaDoc](#) comments to the [getters](#) and classes to document your configuration options. Further to access this configuration, we can use standard [dependency-injection](#):

```

@Inject
private BillingServiceConfigProperties config;

```

For very generic cases you may also use `Map<String, String>` to map any kind of property in an

untyped way. An example for generic configuration from `devon4j` can be found in [ServiceConfigProperties](#).

For further details about this feature also consult [Guide to @ConfigurationProperties in Spring Boot](#).

Generate configuration metadata

You should further add this `dependency` to your module containing the `*ConfigProperties`:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

This will generate configuration metadata so projects using your code can benefit from autocompletion and getting your JavaDoc as tooltip when editing `application.properites` what makes this approach very powerful. For further details about this please read [A Guide to Spring Boot Configuration Metadata](#).

12.3. Java Persistence API

For mapping java objects to a relational database we use the [Java Persistence API \(JPA\)](#). As JPA implementation we recommend to use [hibernate](#). For general documentation about JPA and hibernate follow the links above as we will not replicate the documentation. Here you will only find guidelines and examples how we recommend to use it properly. The following examples show how to map the data of a database to an entity. As we use JPA we abstract from [SQL](#) here. However, you will still need a [DDL](#) script for your schema and during maintenance also [database migrations](#). Please follow our [SQL guide](#) for such artifacts.

12.3.1. Entity

Entities are part of the persistence layer and contain the actual data. They are POJOs (Plain Old Java Objects) on which the relational data of a database is mapped and vice versa. The mapping is configured via JPA annotations ([javax.persistence](#)). Usually an entity class corresponds to a table of a database and a property to a column of that table. A persistent entity instance then represents a row of the database table.

A Simple Entity

The following listing shows a simple example:

```
@Entity
@Table(name="TEXTMESSAGE")
public class MessageEntity extends ApplicationPersistenceEntity implements Message {

    private String text;

    public String getText() {
        return this.text;
    }

    public void setText(String text) {
        this.text = text;
    }
}
```

The `@Entity` annotation defines that instances of this class will be entities which can be stored in the database. The `@Table` annotation is optional and can be used to define the name of the corresponding table in the database. If it is not specified, the simple name of the entity class is used instead.

In order to specify how to map the attributes to columns we annotate the corresponding getter methods (technically also private field annotation is also possible but approaches can not be mixed). The `@Id` annotation specifies that a property should be used as [primary key](#). With the help of the `@Column` annotation it is possible to define the name of the column that an attribute is mapped to as well as other aspects such as [nullable](#) or [unique](#). If no column name is specified, the name of the property is used as default.

Note that every entity class needs a constructor with public or protected visibility that does not have any arguments. Moreover, neither the class nor its getters and setters may be final.

Entities should be simple POJOs and not contain business logic.

Entities and Datatypes

Standard datatypes like `Integer`, `BigDecimal`, `String`, etc. are mapped automatically by JPA. Custom [datatypes](#) are mapped as serialized `BLOB` by default what is typically undesired. In order to map atomic custom datatypes (implementations of `+SimpleDatatype`) we implement an [AttributeConverter](#). Here is a simple example:

```
@Converter(autoApply = true)
public class MoneyAttributeConverter implements AttributeConverter<Money, BigDecimal>
{
    public BigDecimal convertToDatabaseColumn(Money attribute) {
        return attribute.getValue();
    }

    public Money convertToEntityAttribute(BigDecimal dbData) {
        return new Money(dbData);
    }
}
```

The annotation `@Converter` is detected by the JPA vendor if the annotated class is in the packages to scan. Further, `autoApply = true` implies that the converter is automatically used for all properties of the handled datatype. Therefore all entities with properties of that datatype will automatically be mapped properly (in our example `Money` is mapped as `BigDecimal`).

In case you have a composite datatype that you need to map to multiple columns the JPA does not offer a real solution. As a workaround you can use a bean instead of a real datatype and declare it as `@Embeddable`. If you are using hibernate you can implement `CompositeUserType`. Via the `@TypeDef` annotation it can be registered to hibernate. If you want to annotate the `CompositeUserType` implementation itself you also need another annotation (e.g. `MappedSuperclass` though not technically correct) so it is found by the scan.

Enumerations

By default JPA maps Enums via their ordinal. Therefore the database will only contain the ordinals (0, 1, 2, etc.). So, inside the database you can not easily understand their meaning. Using `@Enumerated` with `EnumType.STRING` allows to map the enum values to their name (`Enum.name()`). Both approaches are fragile when it comes to code changes and refactoring (if you change the order of the enum values or rename them) after the application is deployed to production. If you want to avoid this and get a robust mapping you can define a dedicated string in each enum value for database representation that you keep untouched. Then you treat the enum just like any other [custom datatype](#).

BLOB

If binary or character large objects (BLOB/CLOB) should be used to store the value of an attribute, e.g. to store an icon, the `@Lob` annotation should be used as shown in the following listing:

```
@Lob
public byte[] getIcon() {
    return this.icon;
}
```



Using a byte array will cause problems if BLOBS get large because the entire BLOB is loaded into the RAM of the server and has to be processed by the garbage collector. For larger BLOBS the type `Blob` and `streaming` should be used.

```
public Blob getAttachment() {
    return this.attachment;
}
```

Date and Time

To store date and time related values, the temporal annotation can be used as shown in the listing below:

```
@Temporal(TemporalType.TIMESTAMP)
public java.util.Date getStart() {
    return start;
}
```

Until Java8 the java data type `java.util.Date` (or Jodatime) has to be used. `TemporalType` defines the granularity. In this case, a precision of nanoseconds is used. If this granularity is not wanted, `TemporalType.DATE` can be used instead, which only has a granularity of milliseconds. Mixing these two granularities can cause problems when comparing one value to another. This is why we **only** use `TemporalType.TIMESTAMP`.

QueryDSL and Custom Types

Using the Aliases API of QueryDSL might result in an `InvalidDataAccessApiUsageException` when using custom datatypes in entity properties. This can be circumvented in two steps:

1. Ensure you have the following maven dependencies in your project (`core` module) to support custom types via the Aliases API:

```
<dependency>
    <groupId>org.ow2.asm</groupId>
    <artifactId>asm</artifactId>
</dependency>
<dependency>
    <groupId>cglib</groupId>
    <artifactId>cglib</artifactId>
</dependency>
```

2. Make sure, that all your custom types used in entities provide a non-argument constructor with at least visibility level **protected**.

IdRef

IdRef can be used to reference the other entities in TOs

IdRef is just a wrapper for reference ID as Long. so instead of

```
private Long orderId;
```

you can write

```
private IdRef<Order> orderId;
```

When using IdRef make sure that the Junit tests contains the following

Junit test should be derived from DbTest (e.g. ComponentDbTest and SubsystemDbTest).

Primary Keys

We only use simple Long values as primary keys (IDs). By default it is auto generated (`@GeneratedValue(strategy=GenerationType.AUTO)`). This is already provided by the class `com.devonfw.<projectName>.general.dataaccess.api.AbstractPersistenceEntity` that you can extend. In case you have business oriented keys (often as `String`), you can define an additional property for it and declare it as unique (`@Column(unique=true)`). Be sure to include "AUTO_INCREMENT" in your sql table field ID to be able to persist data (or similar for other databases).

12.3.2. Relationships

n:1 and 1:1 Relationships

Entities often do not exist independently but are in some relation to each other. For example, for every period of time one of the StaffMember's of the restaurant example has worked, which is represented by the class `WorkingTime`, there is a relationship to this StaffMember.

The following listing shows how this can be modeled using JPA:

```

...
@Entity
public class WorkingTimeEntity {
    ...

    private StaffMemberEntity staffMember;

    @ManyToOne
    @JoinColumn(name="STAFFMEMBER")
    public StaffMemberEntity getStaffMember() {
        return this.staffMember;
    }

    public void setStaffMember(StaffMemberEntity staffMember) {
        this.staffMember = staffMember;
    }
}

```

To represent the relationship, an attribute of the type of the corresponding entity class that is referenced has been introduced. The relationship is a n:1 relationship, because every `WorkingTime` belongs to exactly one `StaffMember`, but a `StaffMember` usually worked more often than once.

This is why the `@ManyToOne` annotation is used here. For 1:1 relationships the `@OneToOne` annotation can be used which works basically the same way. To be able to save information about the relation in the database, an additional column in the corresponding table of `WorkingTime` is needed which contains the primary key of the referenced `StaffMember`. With the `name` element of the `@JoinColumn` annotation it is possible to specify the name of this column.

1:n and n:m Relationships

The relationship of the example listed above is currently an unidirectional one, as there is a getter method for retrieving the `StaffMember` from the `WorkingTime` object, but not vice versa.

To make it a bidirectional one, the following code has to be added to `StaffMember`:

```

private Set<WorkingTimeEntity> workingTimes;

@OneToMany(mappedBy="staffMember")
public Set<WorkingTimeEntity> getWorkingTimes() {
    return this.workingTimes;
}

public void setWorkingTimes(Set<WorkingTimeEntity> workingTimes) {
    this.workingTimes = workingTimes;
}

```

To make the relationship bidirectional, the tables in the database do not have to be changed. Instead the column that corresponds to the attribute `staffMember` in class `WorkingTime` is used, which

is specified by the `mappedBy` element of the `@OneToMany` annotation. Hibernate will search for corresponding `WorkingTime` objects automatically when a `StaffMember` is loaded.

The problem with bidirectional relationships is that if a `WorkingTime` object is added to the set or list `workingTimes` in `StaffMember`, this does not have any effect in the database unless the `staffMember` attribute of that `WorkingTime` object is set. That is why the devon4j advises not to use bidirectional relationships but to use queries instead. How to do this is shown [here](#). If a bidirectional relationship should be used nevertheless, appropriate add and remove methods must be used.

For 1:n and n:m relations, the devon4j demands that (unordered) Sets and no other collection types are used, as shown in the listing above. The only exception is whenever an ordering is really needed, (sorted) lists can be used.

For example, if `WorkingTime` objects should be sorted by their start time, this could be done like this:

```
private List<WorkingTimeEntity> workingTimes;

@OneToMany(mappedBy = "staffMember")
@OrderBy("startTime asc")
public List<WorkingTimeEntity> getWorkingTimes() {
    return this.workingTimes;
}

public void setWorkingTimes(List<WorkingTimeEntity> workingTimes) {
    this.workingTimes = workingTimes;
}
```

The value of the `@OrderBy` annotation consists of an attribute name of the class followed by `asc` (ascending) or `desc` (descending).

To store information about a n:m relationship, a separate table has to be used, as one column cannot store several values (at least if the database schema is in first normal form).

For example if one wanted to extend the example application so that all ingredients of one `FoodDrink` can be saved and to model the ingredients themselves as entities (e.g. to store additional information about them), this could be modeled as follows (extract of class `FoodDrink`):

```
private Set<IngredientEntity> ingredients;

@ManyToMany()
@JoinTable
public Set<IngredientEntity> getIngredients() {
    return this.ingredients;
}

public void setOrders(Set<IngredientEntity> ingredients) {
    this.ingredients = ingredients;
}
```

Information about the relation is stored in a table called `BILL_ORDER` that has to have two columns,

one for referencing the Bill, the other one for referencing the Order. Note that the `@JoinTable` annotation is not needed in this case because a separate table is the default solution here (same for n:m relations) unless there is a `mappedBy` element specified.

For 1:n relationships this solution has the disadvantage that more joins (in the database system) are needed to get a Bill with all the Orders it refers to. This might have a negative impact on performance so that the solution to store a reference to the Bill row/entity in the Order's table is probably the better solution in most cases.

Note that bidirectional n:m relationships are not allowed for applications based on devon4j. Instead a third entity has to be introduced, which "represents" the relationship (it has two n:1 relationships).

Eager vs. Lazy Loading

Using JPA it is possible to use either lazy or eager loading. Eager loading means that for entities retrieved from the database, other entities that are referenced by these entities are also retrieved, whereas lazy loading means that this is only done when they are actually needed, i.e. when the corresponding getter method is invoked.

Application based on devon4j are strongly advised to **always use lazy loading**. The JPA defaults are:

- `@OneToMany`: LAZY
- `@ManyToMany`: LAZY
- `@ManyToOne`: EAGER
- `@OneToOne`: EAGER

So at least for `@ManyToOne` and `@OneToOne` you always need to override the default by providing `fetch = FetchType.LAZY`.



Please read the [performance guide](#).

Cascading Relationships

For relations it is also possible to define whether operations are cascaded (like a recursion) to the related entity. By default, nothing is done in these situations. This can be changed by using the `cascade` property of the annotation that specifies the relation type (`@OneToOne`, `@ManyToOne`, `@OneToMany`, `@ManyToOne`). This property accepts a `CascadeType` that offers the following options:

- PERSIST (for `EntityManager.persist`, relevant to inserted transient entities into DB)
- REMOVE (for `EntityManager.remove` to delete entity from DB)
- MERGE (for `EntityManager.merge`)
- REFRESH (for `EntityManager.refresh`)
- DETACH (for `EntityManager.detach`)
- ALL (cascade all of the above operations)

See [here](#) for more information.

12.3.3. Embeddable

An embeddable Object is a way to group properties of an [entity](#) into a separate Java (child) object. Unlike with implement [relationships](#) the embeddable is not a separate entity and its properties are stored (embedded) in the same table together with the entity. This is helpful to structure and reuse groups of properties.

The following example shows an [Address](#) implemented as an embeddable class:

```
@Embeddable
public class AddressEmbeddable {

    private String street;
    private String number;
    private Integer zipCode;
    private String city;

    @Column(name="STREETNUMBER")
    public String getNumber() {
        return number;
    }

    public void setNumber(String number) {
        this.number = number;
    }

    ... // other getter and setter methods, equals, hashCode
}
```

As you can see an embeddable is similar to an entity class, but with an [@Embeddable](#) annotation instead of the [@Entity](#) annotation and without primary key or modification counter. An Embeddable does not exist on its own but in the context of an entity. As a simplification Embeddables do not require a separate interface and [ETO](#) as the [bean-mapper](#) will create a copy automatically when converting the owning entity to an ETO. However, in this case the embeddable becomes part of your [api](#) module that therefore needs a dependency on the [JPA](#).

In addition to that the methods [equals\(Object\)](#) and [hashCode\(\)](#) need to be implemented as this is required by Hibernate (it is not required for entities because they can be unambiguously identified by their primary key). For some hints on how to implement the [hashCode\(\)](#) method please have a look [here](#).

Using this [AddressEmbeddable](#) inside an entity class can be done like this:

```

private AddressEmbeddable address;

@Embedded
public AddressEmbeddable getAddress() {
    return this.address;
}

public void setAddress(AddressEmbeddable address) {
    this.address = address;
}
}

```

The `@Embedded` annotation needs to be used for embedded attributes. Note that if in all columns of the embeddable (here `Address`) are `null`, then the embeddable object itself is also `null` inside the entity. This has to be considered to avoid `NullPointerException`'s. Further this causes some issues with primitive types in embeddable classes that can be avoided by only using object types instead.

12.3.4. Inheritance

Just like normal java classes, `entity` classes can inherit from others. The only difference is that you need to specify how to map a class hierarchy to database tables. Generic abstract super-classes for entities can simply be annotated with `@MappedSuperclass`.

For all other cases the JPA offers the annotation `@Inheritance` with the property `strategy` talking an `InheritanceType` that has the following options:

- **SINGLE_TABLE**: This strategy uses a single table that contains all columns needed to store all entity-types of the entire inheritance hierarchy. If a column is not needed for an entity because of its type, there is a null value in this column. An additional column is introduced, which denotes the type of the entity (called `dtype`).
- **TABLE_PER_CLASS**: For each concrete entity class there is a table in the database that can store such an entity with all its attributes. An entity is only saved in the table corresponding to its most concrete type. To get all entities of a super type, joins are needed.
- **JOINED**: In this case there is a table for every entity class including abstract classes, which contains only the columns for the persistent properties of that particular class. Additionally there is a primary key column in every table. To get an entity of a class that is a subclass of another one, joins are needed.

Each of the three approaches has its advantages and drawbacks, which are discussed in detail [here](#). In most cases, the first one should be used, because it is usually the fastest way to do the mapping, as no joins are needed when retrieving, searching or persisting entities. Moreover it is rather simple and easy to understand. One major disadvantage is that the first approach could lead to a table with a lot of null values, which might have a negative impact on the database size.

The inheritance strategy has to be annotated to the top-most entity of the class hierarchy (where `@MappedSuperclass`'es are not considered) like in the following example:

```

@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
public abstract class MyParentEntity extends ApplicationPersistenceEntity implements
MyParent {
    ...
}

@Entity
public class MyChildEntity extends MyParentEntity implements MyChild {
    ...
}

@Entity
public class MyOtherEntity extends MyParentEntity implements MyChild {
    ...
}

```

As a best practice we advise you to avoid entity hierarchies at all where possible and otherwise to keep the hierarchy as small as possible. In order to just ensure reuse or establish a common API you can consider a shared interface, a `@MappedSuperclass` or an `@Embeddable` instead of an entity hierarchy.

12.3.5. Repositories and DAOs

For each entity a code unit is created that groups all database operations for that entity. We recommend to use [spring-data repositories](#) for that as it is most efficient for developers. As an alternative there is still the classic approach using [DAOs](#).

Concurrency Control

The concurrency control defines the way concurrent access to the same data of a database is handled. When several users (or threads of application servers) concurrently access a database, anomalies may happen, e.g. a transaction is able to see changes from another transaction although that one did, not yet commit these changes. Most of these anomalies are automatically prevented by the database system, depending on the *isolation level* (property `hibernate.connection.isolation` in the `jpa.xml`, see [here](#)).

Another anomaly is when two stakeholders concurrently access a record, do some changes and write them back to the database. The JPA addresses this with different locking strategies (see [here](#)).

As a best practice we are using optimistic locking for regular end-user [services](#) (OLTP) and pessimistic locking for [batches](#).

Optimistic Locking

The class `com.devonfw.module.jpa.persistence.api.AbstractPersistenceEntity` already provides optimistic locking via a `modificationCounter` with the `@Version` annotation. Therefore JPA takes care of optimistic locking for you. When entities are transferred to clients, modified and sent back for update you need to ensure the `modificationCounter` is part of the game. If you follow our guides

about [transfer-objects](#) and [services](#) this will also work out of the box. You only have to care about two things:

- How to deal with optimistic locking in [relationships](#)?

Assume an entity **A** contains a collection of **B** entities. Should there be a locking conflict if one user modifies an instance of **A** while another user in parallel modifies an instance of **B** that is contained in the other instance? To address this, take a look at [FeatureForceIncrementModificationCounter](#).

- What should happen in the UI if an [OptimisticLockException](#) occurred?

According to KISS our recommendation is that the user gets an error displayed that tells him to do his change again on the recent data. Try to design your system and the work processing in a way to keep such conflicts rare and you are fine.

Pessimistic Locking

For back-end [services](#) and especially for [batches](#) optimistic locking is not suitable. A human user shall not cause a large batch process to fail because he was editing the same entity. Therefore such use-cases use pessimistic locking what gives them a kind of priority over the human users. In your [DAO](#) implementation you can provide methods that do pessimistic locking via [EntityManager](#) operations that take a [LockModeType](#). Here is a simple example:

```
getEntityManager().lock(entity, LockModeType.READ);
```

When using the [lock\(Object, LockModeType\)](#) method with [LockModeType.READ](#), Hibernate will issue a [SELECT ... FOR UPDATE](#). This means that no one else can update the entity (see [here](#) for more information on the statement). If [LockModeType.WRITE](#) is specified, Hibernate issues a [SELECT ... FOR UPDATE NOWAIT](#) instead, which has the same meaning as the statement above, but if there is already a lock, the program will not wait for this lock to be released. Instead, an exception is raised. Use one of the types if you want to modify the entity later on, for read only access no lock is required.

As you might have noticed, the behavior of Hibernate deviates from what one would expect by looking at the [LockModeType](#) (especially [LockModeType.READ](#) should not cause a [SELECT ... FOR UPDATE](#) to be issued). The framework actually deviates from what is [specified](#) in the JPA for unknown reasons.

12.3.6. Database Auditing

See [auditing guide](#).

12.3.7. Testing Data-Access

For testing of Entities and Repositories or DAOs see [testing guide](#).

12.3.8. Principles

We strongly recommend these principles:

- Use the JPA where ever possible and use vendor (hibernate) specific features only for situations

when JPA does not provide a solution. In the latter case consider first if you really need the feature.

- Create your entities as simple POJOs and use JPA to annotate the getters in order to define the mapping.
- Keep your entities simple and avoid putting advanced logic into entity methods.

12.3.9. Database Configuration

The [configuration](#) for spring and hibernate is already provided by devonfw in our sample application and the application template. So you only need to worry about a few things to customize.

Database System and Access

Obviously you need to configure which type of database you want to use as well as the location and credentials to access it. The defaults are configured in [application-default.properties](#) that is bundled and deployed with the release of the software. It should therefore contain the properties as in the given example:

```
database.url=jdbc:postgresql://database.enterprise.com/app
database.user.login=appuser01
database.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
database.hibernate.hbm2ddl.auto=validate
```

The environment specific settings (especially passwords) are configured by the operators in [application.properties](#). For further details consult the [configuration guide](#). It can also override the default values. The relevant configuration properties can be seen by the following example for the development environment (located in [src/test/resources](#)):

```
database.url=jdbc:postgresql://localhost/app
database.user.password=*****
database.hibernate.hbm2ddl.auto=create
```

For further details about [database.hibernate.hbm2ddl.auto](#) please see [here](#). For production and acceptance environments we use the value [validate](#) that should be set as default. In case you want to use Oracle RDBMS you can find additional hints [here](#).

Database Migration

See [database migration](#).

Database Logging

Add the following properties to [application.properties](#) to enable logging of database queries for debugging purposes.

```
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.use_sql_comments=true
spring.jpa.properties.hibernate.format_sql=true
```

Pooling

You typically want to pool JDBC connections to boost performance by recycling previous connections. There are many libraries available to do connection pooling. We recommend to use [HikariCP](#). For Oracle RDBMS see [here](#).

12.3.10. Security

SQL-Injection

A common [security](#) threat is [SQL-injection](#). Never build queries with string concatenation or your code might be vulnerable as in the following example:

```
String query = "Select op from OrderPosition op where op.comment = " + userInput;
return getEntityManager().createQuery(query).getResultList();
```

Via the parameter `userInput` an attacker can inject SQL (JPQL) and execute arbitrary statements in the database causing extreme damage. In order to prevent such injections you have to strictly follow our rules for [queries](#): Use named queries for static queries and QueryDSL for dynamic queries. Please also consult the [SQL Injection Prevention Cheat Sheet](#).

Limited Permissions for Application

We suggest that you operate your application with a database user that has limited permissions so he can not modify the SQL schema (e.g. drop tables). For initializing the schema (DDL) or to do schema migrations use a separate user that is not used by the application itself.

12.3.11. Queries

The [Java Persistence API \(JPA\)](#) defines its own query language, the [java persistence query language \(JPQL\)](#) (see also [JPQL tutorial](#)), which is similar to SQL but operates on entities and their attributes instead of tables and columns.

The simplest CRUD-Queries (e.g. find an entity by its ID) are already build in the devonfw CRUD functionality (via [Repository](#) or [DAO](#)). For other cases you need to write your own query. We distinguish between *static* and *dynamic* queries. [Static queries](#) have a fixed JPQL query string that may only use parameters to customize the query at runtime. Instead, [dynamic queries](#) can change their clauses (`WHERE`, `ORDER BY`, `JOIN`, etc.) at runtime depending on the given search criteria.

Static Queries

E.g. to find all [DishEntries](#) (from MTS sample app) that have a price not exceeding a given `maxPrice` we write the following JPQL query:

```
SELECT dish FROM DishEntity dish WHERE dish.price <= :maxPrice
```

Here `dish` is used as alias (variable name) for our selected `DishEntity` (what refers to the simple name of the Java entity class). With `dish.price` we are referring to the Java property `price` (`getPrice()/setPrice(...)`) in `DishEntity`. A named variable provided from outside (the search criteria at runtime) is specified with a colon (`:`) as prefix. Here with `:maxPrice` we reference to a variable that needs to be set via `query.setParameter("maxPrice", maxPriceValue)`. JPQL also supports indexed parameters (`?`) but they are discouraged because they easily cause confusion and mistakes.

Using Queries to Avoid Bidirectional Relationships

With the usage of queries it is possible to avoid exposing relationships or modelling bidirectional relationships, which have some disadvantages (see [relationships](#)). This is especially desired for relationships between entities of different business components. So for example to get all `OrderLineEntities` for a specific `OrderEntity` without using the `orderLines` relation from `OrderEntity` the following query could be used:

```
SELECT line FROM OrderLineEntity line WHERE line.order.id = :orderId
```

Dynamic Queries

For dynamic queries we use [QueryDSL](#). It allows to implement queries in a powerful but readable and type-safe way (unlike Criteria API). If you already know JPQL you will quickly be able to read and write QueryDSL code. It feels like JPQL but implemented in Java instead of plain text.

Please be aware that code-generation can be painful especially with large teams. We therefore recommend to use QueryDSL without code-generation. Here is an example from our sample application:

```

public List<DishEntity> findOrders(DishSearchCriteriaTo criteria) {
    DishEntity dish = Alias.alias(DishEntity.class);
    JPAQuery<OrderEntity> query = newDslQuery(alias); // new
    JPAQuery<>(getEntityManager()).from(Alias.$(dish));
    Range<BigDecimal> priceRange = criteria.getPriceRange();
    if (priceRange != null) {
        BigDecimal min = priceRange.getMin();
        if (min != null) {
            query.where(Alias.$(order.getPrice()).ge(min));
        }
        BigDecimal max = priceRange.getMax();
        if (max != null) {
            query.where(Alias.$(order.getPrice()).le(max));
        }
    }
    String name = criteria.getName();
    if ((name != null) && (!name.isEmpty())) {
        // query.where(Alias.$(alias.getName()).eq(name));
        QueryUtil.get().whereString(query, Alias.$(alias.getName()), name, criteria
            .getNameOption());
    }
    return query.fetch();
}

```

Using Wildcards

For flexible queries it is often required to allow wildcards (especially in [dynamic queries](#)). While users intuitively expect glob syntax the SQL and JPQL standards work different. Therefore a mapping is required. devonfw provides this on a lower level by [LikePatternSyntax](#) and on a high level by [QueryUtil](#) (see [QueryHelper.newStringClause\(...\)](#)).

Pagination

devonfw provides pagination support. If you are using [spring-data repositories](#) you will get that directly from spring for static queries. Otherwise for dynamic or generally handwritten queries we provide this via [QueryUtil.findPaginated\(...\)](#):

```

boolean determineTotalHitCount = ....;
return QueryUtil.get().findPaginated(criteria.getPageable(), query,
determineTotalHitCount);

```

Pagination example

For the table entity we can make a search request by accessing the REST endpoint with pagination support like in the following examples:

```

POST mythaistar/services/rest/tablemanagement/v1/table/search
{
    "pagination": {
        "size": 2,
        "total": true
    }
}

//Response
{
    "pagination": {
        "size": 2,
        "page": 1,
        "total": 11
    },
    "result": [
        {
            "id": 101,
            "modificationCounter": 1,
            "revision": null,
            "waiterId": null,
            "number": 1,
            "state": "OCCUPIED"
        },
        {
            "id": 102,
            "modificationCounter": 1,
            "revision": null,
            "waiterId": null,
            "number": 2,
            "state": "FREE"
        }
    ]
}

```



As we are requesting with the `total` property set to `true` the server responds with the total count of rows for the query.

For retrieving a concrete page, we provide the `page` attribute with the desired value. Here we also left out the `total` property so the server doesn't incur on the effort to calculate it:

```

POST mythaistar/services/rest/tablemanagement/v1/table/search
{
    "pagination": {
        "size": 2,
        "page": 2
    }
}

//Response

{
    "pagination": {
        "size": 2,
        "page": 2,
        "total": null
    },
    "result": [
        {
            "id": 103,
            "modificationCounter": 1,
            "revision": null,
            "waiterId": null,
            "number": 3,
            "state": "FREE"
        },
        {
            "id": 104,
            "modificationCounter": 1,
            "revision": null,
            "waiterId": null,
            "number": 4,
            "state": "FREE"
        }
    ]
}

```

Query Meta-Parameters

Queries can have meta-parameters and that are provided via [SearchCriteriaTo](#). Besides paging (see above) we also get [timeout support](#).

Advanced Queries

Writing queries can sometimes get rather complex. The current examples given above only showed very simple basics. Within this topic a lot of advanced features need to be considered like:

- [Joins](#)
- [Constructor queries](#)
- [Order By \(Sorting\)](#)

- [Grouping](#)
- [Having](#)
- [Unions](#)
- [Sub-Queries](#)
- Aggregation functions like e.g. [count/avg/sum](#)
- [Distinct selections](#)
- SQL Hints (see e.g. [Oracle hints](#) or [SQL-Server hints](#)) - only when required for ultimate performance tuning

This list is just containing the most important aspects. As we can not cover all these topics here, they are linked to external documentation that can help and guide you.

12.3.12. Spring-Data

If you are using the Spring Framework and have no restrictions regarding that, we recommend to use [spring-data-jpa](#) via [devon4j-starter-spring-data-jpa](#) that brings advanced integration (esp. for QueryDSL).

Motivation

The benefits of spring-data are (for examples and explanations see next sections):

- All you need is one single repository interface for each entity. No need for a separate implementation or other code artifacts like XML descriptors, [NamedQuerries](#) class, etc.
- You have all information together in one place (the repository interface) that actually belong together (where as in the classic approach you have the static [queries](#) in an XML file, constants to them in [NamedQuerries](#) class and referencing usages in DAO implementation classes).
- Static [queries](#) are most simple to realize as you do not need to write any method body. This means you can develop faster.
- Support for paging is already build-in. Again for static [query](#) method there is nothing you have to do except using the paging objects in the signature.
- Still you have the freedom to write custom implementations via default methods within the repository interface (e.g. for dynamic queries).

Repository

For each entity [`<>Entity`](#) an interface is created with the name [`<>EntityRepository`](#) extending [`DefaultRepository`](#). Such repository is the analogy to a [Data-Access-Object \(DAO\)](#) used in the classic approach or when spring-data is not an option.

Example

The following example shows how to write such a repository:

```

public interface ExampleRepository extends DefaultRepository<ExampleEntity> {

    @Query("SELECT example FROM ExampleEntity example" //
        + " WHERE example.name = :name")
    List<ExampleEntity> findByName(@Param("name") String name);

    @Query("SELECT example FROM ExampleEntity example" //
        + " WHERE example.name = :name")
    Page<ExampleEntity> findByNamePaginated(@Param("name") String name, Pageable pageable);

    default Page<ExampleEntity> findByCriteria(ExampleSearchCriteriaTo criteria) {
        ExampleEntity alias = newDslAlias();
        JPAQuery<ExampleEntity> query = newDslQuery(alias);
        String name = criteria.getName();
        if ((name != null) && !name.isEmpty()) {
            QueryUtil.get().whereString(query, $(alias.getName()), name, criteria
                .getNameOption());
        }
        return QueryUtil.get().findPaginated(criteria.getPageable(), query, false);
    }

}

```

This `ExampleRepository` has the following features:

- CRUD support from spring-data (see [JavaDoc](#) for details).
- Support for [QueryDSL integration, paging and more](#) as well as [locking](#) via [GenericRepository](#)
- A static `query` method `findByName` to find all `ExampleEntity` instances from DB that have the given name. Please note the `@Param` annotation that links the method parameter with the variable inside the query (`:name`).
- The same with pagination support via `findByNamePaginated` method.
- A dynamic `query` method `findByCriteria` showing the QueryDSL and paging integration into spring-data provided by devon.

Further examples

You can also read the JUnit test-case `DefaultRepositoryTest` that is testing an example [FooRepository](#).

Auditing

In case you need [auditing](#), you only need to extend `DefaultRevisedRepository` instead of `DefaultRepository`. The auditing methods can be found in [GenericRevisedRepository](#).

Dependency

In case you want to switch to or add spring-data support to your devon application all you need is

this maven dependency:

```
<!-- Starter for consuming REST services -->
<dependency>
    <groupId>com.devonfw.java.starters</groupId>
    <artifactId>devon4j-starter-spring-data-jpa</artifactId>
</dependency>
```

Drawbacks

Spring-data also has some drawbacks:

- Some kind of magic behind the scenes that are not so easy to understand. So in case you want to extend all your repositories without providing the implementation via a default method in a parent repository interface you need to deep-dive into spring-data. We assume that you do not need that and hope what spring-data and devon already provides out-of-the-box is already sufficient.
- The spring-data magic also includes guessing the query from the method name. This is not easy to understand and especially to debug. Our suggestion is not to use this feature at all and either provide a `@Query` annotation or an implementation via default method.

12.3.13. Data Access Object

The *Data Access Objects* (DAOs) are part of the persistence layer. They are responsible for a specific `entity` and should be named `<<Entity>>Dao` and `<<Entity>>DaoImpl`. The DAO offers the so called CRUD-functionalities (create, retrieve, update, delete) for the corresponding entity. Additionally a DAO may offer advanced operations such as `query` or locking methods.

DAO Interface

For each DAO there is an interface named `<<Entity>>Dao` that defines the API. For CRUD support and common naming we derive it from the `ApplicationDao` interface that comes with the devon application template:

```
public interface MyEntityDao extends ApplicationDao<MyEntity> {
    List<MyEntity> findByCriteria(MyEntitySearchCriteria criteria);
}
```

All CRUD operations are inherited from `ApplicationDao` so you only have to declare the additional methods.

DAO Implementation

Implementing a DAO is quite simple. We create a class named `<<Entity>>DaoImpl` that extends `ApplicationDaoImpl` and implements your `<<Entity>>Dao` interface:

```

public class MyEntityDaoImpl extends ApplicationDaoImpl<MyEntity> implements
MyEntityDao {

    public List<MyEntity> findByCriteria(MyEntitySearchCriteria criteria) {
        TypedQuery<MyEntity> query = createQuery(criteria, getEntityManager());
        return query.getResultList();
    }
    ...
}

```

Again you only need to implement the additional non-CRUD methods that you have declared in your `<>Entity>Dao` interface. In the DAO implementation you can use the method `getEntityManager()` to access the `EntityManager` from the JPA. You will need the `EntityManager` to create and execute `queries`.

Static queries for DAO Implementation

All static `queries` are declared in the file `src\main\resources\META-INF\orm.xml`:

```

<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings version="1.0" xmlns="http://java.sun.com/xml/ns/persistence/orm"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
    http://java.sun.com/xml/ns/persistence/orm_1_0.xsd">
    <named-query name="find.dish.with.max.price">
        <query><![SELECT dish FROM DishEntity dish WHERE dish.price <= :maxPrice]>
    </query>
    </named-query>
    ...
</hibernate-mapping>

```

When your application is started, all these static queries will be created as prepared statements. This allows better performance and also ensures that you get errors for invalid JPQL queries when you start your app rather than later when the query is used.

To avoid redundant occurrences of the query name (`get.open.order.positions.for.order`) we define a constant for each named query:

```

public class NamedQueries {
    public static final String FIND_DISH_WITH_MAX_PRICE = "find.dish.with.max.price";
}

```

Note that changing the name of the java constant (`FIND_DISH_WITH_MAX_PRICE`) can be done easily with refactoring. Further you can trace where the query is used by searching the references of the constant.

The following listing shows how to use this query:

```
public List<DishEntity> findDishByMaxPrice(BigDecimal maxPrice) {
    Query query = getEntityManager().createNamedQuery(NamedQueries
        .FIND_DISH_WITH_MAX_PRICE);
    query.setParameter("maxPrice", maxPrice);
    return query.getResultList();
}
```

Via `EntityManager.createNamedQuery(String)` we create an instance of `Query` for our predefined static query. Next we use `setParameter(String, Object)` to provide a parameter (`maxPrice`) to the query. This has to be done for all parameters of the query.

Note that using the `createQuery(String)` method, which takes the entire query as string (that may already contain the parameter) is not allowed to avoid SQL injection vulnerabilities. When the method `getResultList()` is invoked, the query is executed and the result is delivered as `List`. As an alternative, there is a method called `getSingleResult()`, which returns the entity if the query returned exactly one and throws an exception otherwise.

12.3.14. JPA Performance

When using JPA the developer sometimes does not see or understand where and when statements to the database are triggered.

Establishing expectations Developers shouldn't expect to sprinkle magic pixie dust on POJOs in hopes they will become persistent.

— Dan Allen, <https://epdf.tips/seam-in-action.html>

So in case you do not understand what is going on under the hood of JPA, you will easily run into performance issues due to lazy loading and other effects.

N plus 1 Problem

The most prominent phenomena is call the [N+1 Problem](#). We use entities from our [MTS](#) demo app as an example to explain the problem. There is a `DishEntity` that has a `@ManyToMany` relation to `IngredientEntity`. Now we assume that we want to iterate all ingredients for a dish like this:

```
DishEntity dish = dao.findDishById(dishId);
BigDecimal priceWithAllExtras = dish.getPrice();
for (IngredientEntity ingredient : dish.getExtras()) {
    priceWithAllExtras = priceWithAllExtras.add(ingredient.getPrice());
}
```

Now `dish.getExtras()` is loaded lazy. Therefore the JPA vendor will provide a list with lazy initialized instances of `IngredientEntity` that only contain the ID of that entity. Now with every call of `ingredient.getPrice()` we technically trigger an SQL query statement to load the specific `IngredientEntity` by its ID from the database. Now `findDishById` caused 1 initial query statement and for any number `N` of ingredients we are causing an additional query statement. This makes a

total of **N+1** statements. As causing statements to the database is an expensive operation with a lot of overhead (creating connection, etc.) this ends in bad performance and is therefore a problem (the **N+1 Problem**).

Solving N plus 1 Problem

To solve the **N+1 Problem** you need to change your code to only trigger a single statement instead. This can be achieved in various ways. The most universal solution is to use **FETCH JOIN** in order to pre-load the nested **N** child entities into the first level cache of the JPA vendor implementation. This will behave very similar as if the **@ManyToMany** relation to **IngredientEntity** was having **FetchType.EAGER** but only for the specific query and not in general. Because changing **@ManyToMany** to **FetchType.EAGER** would cause bad performance for other usecases where only the dish but not its extra ingredients are needed. For this reason all relations, including **@OneToOne** should always be **FetchType.LAZY**. Back to our example we simply replace `dao.findDishById(dishId)` with `dao.findDishWithExtrasById(dishId)` that we implement by the following JPQL query:

```
SELECT dish FROM DishEntity dish
LEFT JOIN FETCH dish.extras
WHERE dish.id = :dishId
```

The rest of the code does not have to be changed but now `dish.getExtras()` will get the **IngredientEntity** from the first level cache where it was fetched by the initial query above.

Please note that if you only need the sum of the prices from the extras you can also create a query using an aggregator function:

```
SELECT sum(dish.extras.price) FROM DishEntity dish
```

As you can see you need to understand the concepts in order to get good performance.

There are many advanced topics such as creating database indexes or calculating statistics for the query optimizer to get the best performance. For such advanced topics we recommend to have a database expert in your team that cares about such things. However, understanding the **N+1 Problem** and its solutions is something that every Java developer in the team needs to understand.

12.4. Auditing

For database auditing we use [hibernate envers](#). If you want to use auditing ensure you have the following dependency in your pom.xml:

```
<dependency>
    <groupId>com.devonfw.java.modules</groupId>
    <artifactId>devon4j-jpa-envers</artifactId>
</dependency>
```

Make sure that entity manager also scans the package from the devon4j-jpa[-envers] module in order to work properly. And make sure that correct Repository Factory Bean Class is chosen.

```
@EntityScan(basePackages = { "<>my.base.package>" }, basePackageClasses = {
AdvancedRevisionEntity.class })
...
@EnableJpaRepositories(repositoryFactoryBeanClass =
GenericRevisedRepositoryFactoryBean.class)
...
public class SpringBootApp {
...
}
```

Now let your [Entity]Repository extend from DefaultRevisedRepository instead of DefaultRepository.

The repository now has a method getRevisionHistoryMetadata(id) and getRevisionHistoryMetadata(id, boolean lazy) available to get a list of revisions for a given entity and a method find(id, revision) to load a specific revision of an entity with the given ID or getLastRevisionHistoryMetadata(id) to load last revision. To enable auditing for a entity simply place the @Audited annotation to your entity and all entity classes it extends from.

```
@Entity(name = "Drink")
@Audited
public class DrinkEntity extends ProductEntity implements Drink {
...
```

When auditing is enabled for an entity an additional database table is used to store all changes to the entity table and a corresponding revision number. This table is called <ENTITY_NAME>_AUD per default. Another table called REVINFO is used to store all revisions. Make sure that these tables are available. They can be generated by hibernate with the following property (only for development environments).

```
database.hibernate.hbm2ddl.auto=create
```

Another possibility is to put them in your [database migration](#) scripts like so.

```
CREATE CACHED TABLE PUBLIC.REVINFO(
    id BIGINT NOT NULL generated by default as identity (start with 1),
    timestamp BIGINT NOT NULL,
    user VARCHAR(255)
);
...
CREATE CACHED TABLE PUBLIC.<TABLE_NAME>_AUD(
    <ALL_TABLE_ATTRIBUTES>,
    revtype TINYINT,
    rev BIGINT NOT NULL
);
```

12.5. Transaction Handling

Transactions are technically processed by the [data access layer](#). However, the transaction control has to be performed in upper layers. To avoid dependencies on persistence layer and technical code in upper layers, we use [AOP](#) to add transaction control via annotations as aspect.

We recommend using the `@Transactional` annotation (the JEE standard `javax.transaction.Transactional` rather than `org.springframework.transaction.annotation.Transactional`). We use this annotation in the [logic layer](#) to annotate business methods that participate in transactions (what typically applies to most up to all business components):

```
@Transactional  
public MyDataTo getData(MyCriteriaTo criteria) {  
    ...  
}
```

In case a [service operation](#) should invoke multiple use-cases, you would end up with multiple transactions what is undesired (what if the first TX succeeds and then the second TX fails?). Therefore you would then also annotate the service operation. This is not proposed as a pattern in any case as in some rare cases you need to handle constraint-violations from the database to create a specific business exception (with specified message). In such case you have to surround the transaction with a `try {} catch` statement what is not working if that method itself is `@Transactional`.

12.5.1. Batches

Transaction control for batches is a lot more complicated and is described in the [batch layer](#).

12.6. SQL

For general guides on dealing or avoiding SQL, preventing SQL-injection, etc. you should study [data-access layer](#).

12.6.1. Naming Conventions

Here we define naming conventions that you should follow whenever you write SQL files:

- All SQL-Keywords in UPPER CASE
- Table names in upper CamelCase (e.g. `RestaurantOrder`)
- Column names in CamelCase (e.g. `drinkState`)
- Indentation should be 2 spaces as suggested by devonfw for every format.

DDL

For DDLs follow these additional guidelines:

- ID column names without underscore (e.g. `tableId`)
- Define columns and constraints inline in the statement to create the table
- Indent column types so they all start in the same text column
- Constraints should be named explicitly (to get a reasonable hint error messages) with:
 - `PK_{table}` for primary key (name optional here as PK constraint are fundamental)
 - `FK_{table}_{property}` for foreign keys (`{table}` and `{property}` are both on the source where the foreign key is defined)
 - `UC_{table}_{property}[_{propertyN}]^*` for unique constraints
 - `CK_{table}_{check}` for check constraints (`{check}` describes the check, if it is defined on a single property it should start with the property).
- Databases have hard limitations for names (e.g. 30 characters). If you have to shorten names try to define common abbreviations in your project for according (business) terms. Especially do not just truncate the names at the limit.
- If possible add comments on table and columns to help DBAs understanding your schema. This is also honored by many tools (not only DBA-tools).

Here is a brief example of a DDL:

```

CREATE SEQUENCE HIBERNATE_SEQUENCE START WITH 1000000;

-- *** Table ***
CREATE TABLE Table (
    id BIGINT NOT NULL AUTO_INCREMENT,
    modificationCounter INTEGER NOT NULL,
    seatsNumber INTEGER NOT NULL,
    CONSTRAINT PK_Table PRIMARY KEY(id)
);

-- *** UserRole ***
CREATE TABLE UserRole (
    id BIGINT NOT NULL AUTO_INCREMENT,
    modificationCounter INTEGER NOT NULL,
    name VARCHAR (255),
    active BOOLEAN,
    CONSTRAINT PK_UserRole PRIMARY KEY(id)
-- *** User ***
CREATE TABLE User (
    id BIGINT NOT NULL AUTO_INCREMENT,
    modificationCounter INTEGER NOT NULL,
    username VARCHAR (255) NULL,
    password VARCHAR (255) NULL,
    email VARCHAR (120) NULL,
    idRole BIGINT NOT NULL,
    CONSTRAINT PK_User PRIMARY KEY(id),
    CONSTRAINT PK_User_idRole FOREIGN KEY(idRole) REFERENCES UserRole(id) NOCHECK
);
COMMENT ON TABLE User is 'The users of the restaurant site';
...

```

Data

For insert, update, delete, etc. of data SQL scripts should additionally follow these guidelines:

- Inserts always with the same order of columns in blocks for each table.
- Insert column values always starting with id, modificationCounter, [dtype,] ...
- List columns with fixed length values (boolean, number, enums, etc.) before columns with free text to support alignment of multiple insert statements
- Pro Tip: Get familiar with column mode of [notepad++](#) when editing large blocks of similar insert statements.

```
INSERT INTO UserRole(id, modificationCounter, name, active) VALUES (0, 1, 'Customer', true);
INSERT INTO UserRole(id, modificationCounter, name, active) VALUES (1, 1, 'Waiter', true);
INSERT INTO User(id, modificationCounter, username, password, email, idRole) VALUES (0, 1, 'user0', 'password', 'user0@mail.com', 0);
INSERT INTO User(id, modificationCounter, username, password, email, idRole) VALUES (1, 1, 'waiter', 'waiter', 'waiter@mail.com', 1);

INSERT INTO Table(id, modificationCounter, seatsNumber) VALUES (0, 1, 4);
INSERT INTO Table(id, modificationCounter, seatsNumber) VALUES (1, 1, 4);
INSERT INTO Table(id, modificationCounter, seatsNumber) VALUES (2, 1, 4);
INSERT INTO Table(id, modificationCounter, seatsNumber) VALUES (3, 1, 4);
INSERT INTO Table(id, modificationCounter, seatsNumber) VALUES (4, 1, 6);
INSERT INTO Table(id, modificationCounter, seatsNumber) VALUES (5, 1, 6);
INSERT INTO Table(id, modificationCounter, seatsNumber) VALUES (6, 1, 6);
INSERT INTO Table(id, modificationCounter, seatsNumber) VALUES (7, 1, 8);
INSERT INTO Table(id, modificationCounter, seatsNumber) VALUES (8, 1, 8);

...
```

See also [Database Migrations](#).

12.7. Database Migration

For database migrations we use [Flyway](#). As illustrated [here](#) database migrations have three advantages:

1. Recreate a database from scratch
2. Make it clear at all times what state a database is in
3. Migrate in a deterministic way from your current version of the database to a newer one

Flyway can be used standalone or can be integrated via its [API](#) to make sure the database migration takes place on startup.

12.7.1. Organizational Advice

A few considerations with respect to project organization will help to implement maintainable Flyway migrations.

At first, testing and production environments must be clearly and consistently distinguished. Use the following directory structure to achieve this distinction:

```
src/main/resources/db
src/test/resources/db
```

Although this structure introduces redundancies, the benefit outweighs this disadvantage. An even more fine-grained production directory structure which contains one sub folder per release should be implemented:

```
src/main/resources/db/migration/releases/X.Y/x.sql
```

Emphasizing that migration scripts below the current version must never be changed will aid the second advantage of migrations: it will always be clearly reproducible in which state the database currently is. Here, it is important to mention that, if test data is required, it must be managed separately from the migration data in the following directory:

```
src/test/resources/db/migration/
```

The **migration** directory is added to aid easy usage of Flyway defaults. Of course, test data should also be managed per release as like production data.

With regard to content, separation of concerns (SoC) is an important goal. SoC can be achieved by distinguishing and writing multiple scripts with respect to business components/use cases (or database tables in case of large volumes of master data ^[1]). Comprehensible file names aid this separation.

It is important to have clear responsibilities regarding the database, the persistence layer (JPA), and

migrations. Therefore a dedicated database expert should be in charge of any migrations performed or she should at least be informed before any change to any of the mentioned parts is applied.

12.7.2. Technical Configuration

Database migrations can be [SQL](#) based or [Java](#) based.

To enable auto migration on startup (not recommended for productive environment) set the following property in the [application.properties](#) file for an environment.

```
flyway.enabled=true  
flyway.clean-on-validation-error=false
```

For development environment it is helpful to set both properties to [true](#) in order to simplify development. For regular environments [flyway.clean-on-validation-error](#) should be [false](#).

If you want to use Flyway set the following property in any case to prevent Hibernate from doing changes on the database (pre-configured by default in devonfw):

```
spring.jpa.hibernate.ddl-auto=validate
```

The setting must be communicated to and coordinated with the customer and their needs. In acceptance testing the same configuration as for the production environment should be enabled.

Since migration scripts will also be versioned the end-of-line (EOL) style must be fixated according to [this issue](#). This is however solved in flyway 4.0+ and the latest devonfw release. Also, the version numbers of migration scripts should not consist of simple ascending integer numbers like V0001..., V0002..., ... This naming may lead to problems when merging branches. Instead the usage of timestamps as version numbers will help to avoid such problems.

12.7.3. Naming Conventions

Database migrations should follow this naming convention: V<version>_<description> (e.g.: V12345__Add_new_table.sql).

It is also possible to use Flyway for test data. To do so place your test data migrations in src/main/resources/db/testdata/ and set property

```
flyway.locations=classpath:db/migration/releases,classpath:db/migration/testdata
```

Then Flyway scans the additional location for migrations and applies all in the order specified by their version. If migrations V0001__... and V0002__... exist and a test data migration should be applied in between you can name it V0001_1__....

12.8. Logging

We use [SLF4J](#) as API for logging. The recommended implementation is [Logback](#) for which we provide additional value such as configuration templates and an appender that prevents log-forging and reformatting of stack-traces for operational optimizations.

12.8.1. Usage

Maven Integration

In the pom.xml of your application add this dependency (that also adds transitive dependencies to SLF4J and logback):

```
<dependency>
  <groupId>com.devonfw.java</groupId>
  <artifactId>devon4j-logging</artifactId>
</dependency>
```

Configuration

The configuration file is logback.xml and is to put in the directory src/main/resources of your main application. For details consult the [logback configuration manual](#). devon4j provides a production ready configuration [here](#). Simply copy this configuration into your application in order to benefit from the provided [operational](#) and [security](#) aspects. We do not include the configuration into the devon4j-logging module to give you the freedom of customizations (e.g. tune log levels for components and integrated products and libraries of your application).

The provided logback.xml is configured to use variables defined on the config/application.properties file. On our example, the log files path point to .. /logs/ in order to log to tomcat log directory when starting tomcat on the bin folder. Change it according to your custom needs.

Listing 8. config/application.properties

```
log.dir=../logs/
```

Logger Access

The general pattern for accessing loggers from your code is a static logger instance per class. We pre-configured the development environment so you can just type LOG and hit [ctrl][space] (and then [arrow up]) to insert the code pattern line into your class:

```
public class MyClass {
    private static final Logger LOG = LoggerFactory.getLogger(MyClass.class);
    ...
}
```

Please note that in this case we are not using injection pattern but use the convenient static alternative. This is already a common solution and also has performance benefits.

How to log

We use a common understanding of the log-levels as illustrated by the following table. This helps for better maintenance and operation of the systems by combining both views.

Table 27. Log-levels

Log-level	Description	Impact	Active Environments
FATAL	Only used for fatal errors that prevent the application to work at all (e.g. startup fails or shutdown/restart required)	Operator has to react immediately	all
ERROR	An abnormal error indicating that the processing failed due to technical problems.	Operator should check for known issue and otherwise inform development	all
WARNING	A situation where something worked not as expected. E.g. a business exception or user validation failure occurred.	No direct reaction required. Used for problem analysis.	all
INFO	Important information such as context, duration, success/failure of request or process	No direct reaction required. Used for analysis.	all
DEBUG	Development information that provides additional context for debugging problems.	No direct reaction required. Used for analysis.	development and testing
TRACE	Like DEBUG but exhaustive information and for code that is run very frequently. Will typically cause large log-files.	No direct reaction required. Used for problem analysis.	none (turned off by default)

Exceptions (with their stacktrace) should only be logged on FATAL or ERROR level. For business exceptions typically a WARNING including the message of the exception is sufficient.

12.8.2. Operations

Log Files

We always use the following log files:

- **Error Log:** Includes log entries to detect errors.
- **Info Log:** Used to analyze system status and to detect bottlenecks.
- **Debug Log:** Detailed information for error detection.

The log file name pattern is as follows:

```
<<LOGTYPE>>_log_<<HOST>>_<<APPLICATION>>_<<TIMESTAMP>>.log
```

Table 28. Segments of Logfilename

Element	Value	Description
«LOGTYPE»	info, error, debug	Type of log file
«HOST»	e.g. mywebserver01	Name of server, where logs are generated
«APPLICATION»	e.g. myapp	Name of application, which causes logs
«TIMESTAMP»	YYYY-MM-DD_HH00	date of log file

Example: error_log_mywebserver01_myapp_2013-09-16_0900.log

Error log from mywebserver01 at application myapp at 16th September 2013 9pm.

Output format

We use the following output format for all log entries to ensure that searching and filtering of log entries work consistent for all logfiles:

```
[D: <<timestamp>>] [P: <<priority>>] [C: <<NDC>>][T: <<thread>>][L: <<logger>>]-[M: <<message>>]
```

- **D:** Date (Timestamp in ISO8601 format e.g. 2013-09-05 16:40:36,464)
- **P:** Priority (the log level)
- **C:** **Correlation ID** (ID to identify users across multiple systems, needed when application is distributed)
- **T:** Thread (Name of thread)
- **L:** Logger name (use class name)
- **M:** Message (log message)

Example:

```
[D: 2013-09-05 16:40:36,464] [P: DEBUG] [C: 12345] [T: main] [L: my.package.MyClass]-  
[M: My message...]
```

12.8.3. Security

In order to prevent [log forging](#) attacks we provide a special appender for logback in devon4i-logging. If you use it (see [configuration](#)) you are safe from such attacks.

12.8.4. Correlation ID

In order to correlate separate HTTP requests to services belonging to the same user / session, we provide a servlet filter called [DiagnosticContextFilter](#). This filter takes a provided correlation ID from the HTTP header [X-Correlation-Id](#). If none was found, it will generate a new correlation id as [UUID](#). This correlation ID is added as MDC to the logger. Therefore, it will then be included to any log message of the current request (thread). Further concepts such as [service invocations](#) will pass this correlation ID to subsequent calls in the application landscape. Hence you can find all log messages related to an initial request simply via the correlation ID even in highly distributed systems.

12.8.5. Monitoring

In highly distributed systems (from clustering up to microservices) it might get tedious to search for problems and details in log files. Therefore, we recommend to setup a central log and analysis server for your application landscape. Then you feed the logs from all your applications (using [logstash](#)) into that central server that adds them to a search index to allow fast searches (using [elasticsearch](#)). This should be completed with a UI that allows dashboards and reports based on data aggregated from all the logs. This is addressed by [ELK](#) or [Graylog](#).

Adding custom values to JSON log

When you use a external system for gathering distributed logs, we strongly suggest that you use a JSON based log format (e.g. by using the provided [logback.xml](#), see above). In that case it might be useful to log customs field to the produced JSON. This is fully supported by slf4j together with logstash. The *trick* is to use the class [net.logstash.logback.argument.StructuredArguments](#) for adding the arguments to you log message, e.g.

```
import static net.logstash.logback.argument.StructuredArguments.v;  
  
...  
  
public void processMessage(MyMessage msg) {  
    LOG.info("Processing message with {}", v("msgId", msg.getId()));  
    ...  
}
```

This will produce something like:

```
{ "timestamp":"2018-11-06T18:36:37.638+00:00",
  "@version":"1",
  "message":"Processing message with msgId=MSG-999-000",
  "msgId":"MSG-999-000",
  "logger_name":"com.myapplication...Processor",
  "thread_name":"http-nio-8081-exec-6",
  "level":INFO,
  "level_value":20000,
  "appname":basic,}
```

12.9. Security

Security is today's most important cross-cutting concern of an application and an enterprise IT-landscape. We seriously care about security and give you detailed guides to prevent pitfalls, vulnerabilities, and other disasters. While many mistakes can be avoided by following our guidelines you still have to consider security and think about it in your design and implementation. The security guide will not only automatically prevent you from any harm, but will provide you hints and best practices already used in different software products.

An important aspect of security is proper authentication and authorization as described in [access-control](#). In the following we discuss about potential vulnerabilities and protection to prevent them.

12.9.1. Vulnerabilities and Protection

Independent from classical authentication and authorization mechanisms there are many common pitfalls that can lead to vulnerabilities and security issues in your application such as XSS, CSRF, SQL-injection, log-forging, etc. A good source of information about this is the [OWASP](#). We address these common threats individually in *security* sections of our technological guides as a concrete solution to prevent an attack typically depends on the according technology. The following table illustrates common threats and contains links to the solutions and protection-mechanisms provided by the devonfw:

Table 29. Security threats and protection-mechanisms

Threat	Protection	Link to details
A1 Injection	validate input, escape output, use proper frameworks	SQL Injection
A2 Broken Authentication	encrypt all channels, use a central identity management with strong password-policy	Authentication
A3 Sensitive Data Exposure	Use secured exception facade, design your data model accordingly	REST exception handling
A4 XML External Entities	Prefer JSON over XML, ensure FSP when parsing (external) XML	XML guide
A5 Broken Access Control	Ensure proper authorization for all use-cases, use <code>@DenyAll</code> as default to enforce	Access-control guide especially method authorization
A6 Security Misconfiguration	Use devon4j application template and guides to avoid	tutorial-newapp and sensitive configuration
A7 Cross-Site Scripting	prevent injection (see A1) for HTML, JavaScript and CSS and understand same-origin-policy	client-layer

Threat	Protection	Link to details
A8 Insecure Deserialization	Use simple and established serialization formats such as JSON, prevent generic deserialization (for polymorphic types)	JSON guide especially inheritance, XML guide
A9 Using Components with Known Vulnerabilities	subscribe to security newsletters, recheck products and their versions continuously, use devonfw dependency management	CVE newsletter and dependency check
A10 Insufficient Logging & Monitoring	Ensure to log all security related events (login, logout, errors), establish effective monitoring	Logging guide and monitoring guide
Insecure Direct Object References	Using direct object references (IDs) only with appropriate authorization	logic-layer
Cross-Site Request Forgery (CSRF)	secure mutable service operations with an explicit CSRF security token sent in HTTP header and verified on the server	service-layer security
Log-Forging	Escape newlines in log messages	logging security
Unvalidated Redirects and Forwards	Avoid using redirects and forwards, in case you need them do a security audit on the solution.	devonfw proposes to use rich-clients (SPA/RIA). We only use redirects for login in a safe way.

12.9.2. Tools

Dependency Check

To address the threat [Using Components with Known Vulnerabilities](#) we integrated [OWASP dependency check](#) into the devon4j maven build. If you build an devon4j application (sample or any app created from our [app-template](#)) you can activate dependency check with the [security](#) profile:

```
mvn clean install -P security
```

This does not run by default as it causes a huge overhead for the build performance. However, consider to build this in your CI at least nightly. After the dependency check is performed, you will find the results in [target/dependency-check-report.html](#) of each module. The report will also be

generated when the site is build (`mvn site`) even without the profile.

Penetration Testing

For penetration testing (testing for vulnerabilities) of your web application, we recommend the following tools:

- [ZAP](#) (OWASP Zed Attack Proxy Project)
- [sqlmap](#) (or [HQLmap](#))
- [nmap](#)
- See the marvelous presentation [Toolbox of a security professional](#) from [Christian Schneider](#).

12.10. Access-Control

Access-Control is a central and important aspect of [Security](#). It consists of two major aspects:

- [Authentication](#) (Who tries to access?)
- [Authorization](#) (Is the one accessing allowed to do what he wants to do?)

12.10.1. Authentication

Definition:

Authentication is the verification that somebody interacting with the system is the actual subject for whom he claims to be.

The one authenticated is properly called *subject* or *principal*. However, for simplicity we use the common term *user* even though it may not be a human (e.g. in case of a service call from an external system).

To prove his authenticity the user provides some secret called *credentials*. The most simple form of credentials is a password.



Please never implement your own authentication mechanism or credential store. You have to be aware of implicit demands such as salting and hashing credentials, password life-cycle with recovery, expiry, and renewal including email notification confirmation tokens, central password policies, etc. This is the domain of access managers and identity management systems. In a business context you will typically already find a system for this purpose that you have to integrate (e.g. via LDAP). Otherwise you should consider establishing such a system e.g. using [keycloak](#).

We use [spring-security](#) as a framework for authentication purposes.

Therefore you need to provide an implementation of [WebSecurityConfigurerAdapter](#):

```

@Configuration
@EnableWebSecurity
public class MyWebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Inject
    private UserDetailsService userDetailsService;
    ...

    public void configure(HttpSecurity http) throws Exception {
        http.userDetailsService(this.userDetailsService)
            .authorizeRequests().antMatchers("/public/**").permitAll()
            .anyRequest().authenticated().and()
            ...
    }
}

```

As you can see spring-security offers a fluent API for easy configuration. You can simply add invocations like `formLogin().loginPage("/public/login")` or `httpBasic().realmName("MyApp")`. Also `CSRF` protection can be configured by invoking `csrf()`. For further details see [spring Java-config for HTTP security](#).

Further, you need to provide an implementation of the `UserDetailsService` interface. A good starting point comes with our application template.

Preserve original request anchors after form login redirect

Spring Security will automatically redirect any unauthorized access to the defined login-page. After successful login, the user will be redirected to the original requested URL. The only pitfall is, that anchors in the request URL will not be transmitted to server and thus cannot be restored after successful login. Therefore the `devon4j-security` module provides the `RetainAnchorFilter`, which is able to inject javascript code to the source page and to the target page of any redirection. Using javascript this filter is able to retrieve the requested anchors and store them into a cookie. Heading the target URL this cookie will be used to restore the original anchors again.

To enable this mechanism you have to integrate the `RetainAnchorFilter` as follows: First, declare the filter with

- `storeUrlPattern`: an regular expression matching the URL, where anchors should be stored
- `restoreUrlPattern`: an regular expression matching the URL, where anchors should be restored
- `cookieName`: the name of the cookie to save the anchors in the intermediate time

You can easily configure this as code in your `WebSecurityConfig` as following:

```

RetainAnchorFilter filter = new RetainAnchorFilter();
filter.setStoreUrlPattern("http://[^/]+/[^\n]+/login.*");
filter.setRestoreUrlPattern("http://[^/]+/[^\n]+/.*");
filter.setCookieName("TARGETANCHOR");
http.addFilterBefore(filter, UsernamePasswordAuthenticationFilter.class);

```

Users vs. Systems

If we are talking about authentication we have to distinguish two forms of principals:

- human users
- autonomous systems

While e.g. a Kerberos/SPNEGO Single-Sign-On makes sense for human users it is pointless for authenticating autonomous systems. So always keep this in mind when you design your authentication mechanisms and separate access for human users from access for systems.

Using JWT

For authentication via JSON Web Token (JWT) see [JWT guide](#).

Mixed Authentication

In rare cases you might need to mix multiple authentication mechanisms (form based, basic-auth, SAMLv2, OAuth, etc.) within the same app (for different URLs). For KISS this should be avoided where possible. However, when needed, you can find a solution [here](#).

12.10.2. Authorization

Definition:

Authorization is the verification that an authenticated user is allowed to perform the operation he intends to invoke.

Clarification of terms

For clarification we also want to give a common understanding of related terms that have no unique definition and consistent usage in the wild.

Table 30. Security terms related to authorization

Term	Meaning and comment
Permission	A permission is an object that allows a principal to perform an operation in the system. This permission can be <i>granted</i> (give) or <i>revoked</i> (taken away). Sometimes people also use the term <i>right</i> what is actually wrong as a right (such as the right to be free) can not be revoked.
Group	We use the term group in this context for an object that contains permissions. A group may also contain other groups. Then the group represents the set of all recursively contained permissions.

Term	Meaning and comment
Role	<p>We consider a role as a specific form of group that also contains permissions. A role identifies a specific function of a principal. A user can act in a role.</p> <p>For simple scenarios a principal has a single role associated. In more complex situations a principal can have multiple roles but has only one active role at a time that he can choose out of his assigned roles. For KISS it is sometimes sufficient to avoid this by creating multiple accounts for the few users with multiple roles. Otherwise at least avoid switching roles at run-time in clients as this may cause problems with related states. Simply restart the client with the new role as parameter in case the user wants to switch his role.</p>
Access Control	Any permission, group, role, etc., which declares a control for access management.

Suggestions on the access model

For the access model we give the following suggestions:

- Each Access Control (permission, group, role, ...) is uniquely identified by a human readable string.
- We create a unique permission for each use-case.
- We define groups that combine permissions to typical and useful sets for the users.
- We define roles as specific groups as required by our business demands.
- We allow to associate users with a list of Access Controls.
- For authorization of an implemented use case we determine the required permission. Furthermore, we determine the current user and verify that the required permission is contained in the tree spanned by all his associated Access Controls. If the user does not have the permission we throw a security exception and thus abort the operation and transaction.
- We avoid negative permissions, that is a user has no permission by default and only those granted to him explicitly give him additional permission for specific things. Permissions granted can not be reduced by other permissions.
- Technically we consider permissions as a secret of the application. Administrators shall not fiddle with individual permissions but grant them via groups. So the access management provides a list of strings identifying the Access Controls of a user. The individual application itself contains these Access Controls in a structured way, whereas each group forms a permission tree.

Naming conventions

As stated above each Access Control is uniquely identified by a human readable string. This string should follow the naming convention:

```
<<app-id>>.<<local-name>>
```

For Access Control Permissions the «local-name» again follows the convention:

```
<<verb>><<object>>
```

The segments are defined by the following table:

Table 31. Segments of Access Control Permission ID

Segment	Description	Example
«app-id»	<p>Is a unique technical but human readable string of the application (or microservice). It shall not contain special characters and especially no dot or whitespace. We recommend to use lower-train-case-ascii-syntax. The identity and access management should be organized on enterprise level rather than application level. Therefore permissions of different apps might easily clash (e.g. two apps might both define a group ReadMasterData but some user shall get this group for only one of these two apps). Using the «app-id». prefix is a simple but powerful namespacing concept that allows you to scale and grow. You may also reserve specific «app-id»s for cross-cutting concerns that do not actually reflect a single app e.g to grant access to a geographic region.</p>	shop

Segment	Description	Example
«verb»	The action that is to be performed on «object». We use Find for searching and reading data. Save shall be used both for create and update. Only if you really have demands to separate these two you may use Create in addition to Save . Finally, Delete is used for deletions. For non CRUD actions you are free to use additional verbs such as Approve or Reject .	Find
«object»	The affected object or entity. Shall be named according to your data-model	Product

So as an example `shop.FindProduct` will reflect the permission to search and retrieve a **Product** in the `shop` application. The group `shop.ReadMasterData` may combine all permissions to read master-data from the `shop`. However, also a group `shop.Admin` may exist for the `Admin` role of the `shop` application. Here the «local-name» is `Admin` that does not follow the «verb»«object» schema.

devon4j-security

The module `devon4j-security` provides ready-to-use code based on `spring-security` that makes your life a lot easier.



Figure 6. `devon4j` Security Model

The diagram shows the model of `devon4j-security` that separates two different aspects:

- The *Identity- and Access-Management* is provided by according products and typically already available in the enterprise landscape (e.g. an active directory). It provides a hierarchy of *primary access control objects* (roles and groups) of a user. An administrator can grant and revoke permissions (indirectly) via this way.
- The application security defines a hierarchy of *secondary access control objects* (groups and permissions). This is done by configuration owned by the application (see following section). The "API" is defined by the IDs of the primary access control objects that will be referenced from the *Identity- and Access-Management*.

Access Control Config

In your application simply extend `AccessControlConfig` to configure your access control objects as code and reference it from your use-cases. An example config may look like this:

```

@Named
public class ApplicationAccessControlConfig extends AccessControlConfig {

    public static final String APP_ID = "MyApp";

    private static final String PREFIX = APP_ID + ".";

    public static final String PERMISSION_FIND_OFFER = PREFIX + "FindOffer";

    public static final String PERMISSION_SAVE_OFFER = PREFIX + "SaveOffer";

    public static final String PERMISSION_DELETE_OFFER = PREFIX + "DeleteOffer";

    public static final String PERMISSION_FIND_PRODUCT = PREFIX + "FindProduct";

    public static final String PERMISSION_SAVE_PRODUCT = PREFIX + "SaveProduct";

    public static final String PERMISSION_DELETE_PRODUCT = PREFIX + "DeleteProduct";

    public static final String GROUP_READ_MASTER_DATA = PREFIX + "ReadMasterData";

    public static final String GROUP_MANAGER = PREFIX + "Manager";

    public static final String GROUP_ADMIN = PREFIX + "Admin";

    public ApplicationAccessControlConfig() {

        super();
        AccessControlGroup readMasterData = group(GROUP_READ_MASTER_DATA,
PERMISSION_FIND_OFFER, PERMISSION_FIND_PRODUCT);
        AccessControlGroup manager = group(GROUP_MANAGER, readMasterData,
PERMISSION_SAVE_OFFER, PERMISSION_SAVE_PRODUCT);
        AccessControlGroup admin = group(GROUP_ADMIN, manager, PERMISSION_DELETE_OFFER,
PERMISSION_DELETE_PRODUCT);
    }
}

```

Configuration on Java Method level

In your use-case you can now reference a permission like this:

```
@Named  
public class UcSafeOfferImpl extends ApplicationUc implements UcSafeOffer {  
  
    @Override  
    @RolesAllowed(ApplicationAccessControlConfig.PERMISSION_SAVE_OFFER)  
    public OfferEto save(OfferEto offer) { ... }  
  
    ...  
}
```

Check Data-Permissions

See [data permissions](#)

Access Control Schema (deprecated)

The `access-control-schema.xml` approach is deprecated. The documentation can still be found in [access control schema](#).

12.11. Access Control Schema

With release [3.0.0](#) the `access-control-schema.xml` has been deprecated. You may still use it and find the documentation in this section. However, for new devonfw applications always start with the new approach described in [access control config](#).

12.11.1. Legacy Access Control Schema Documentation

The file `access-control-schema.xml` is used to define the mapping from groups to permissions (see [example from sample app](#)). The general terms discussed above can be mapped to the implementation as follows:

Table 32. General security terms related to devon4j access control schema

Term	devon4j-security implementation	Comment
Permission	<code>AccessControlPermission</code>	
Group	<code>AccessControlGroup</code>	When considering different levels of groups of different meanings, declare <code>type</code> attribute, e.g. as "group".
Role	<code>AccessControlGroup</code>	With <code>type="role"</code> .
Access Control	<code>AccessControl</code>	Super type that represents a tree of <code>AccessControlGroups</code> and <code>AccessControlPermissions</code> . If a principal "has" a <code>AccessControl</code> he also "has" all <code>AccessControls</code> with according permissions in the spanned sub-tree.

Listing 9. Example access-control-schema.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<access-control-schema>
    <group id="ReadMasterData" type="group">
        <permissions>
            <permission id="OfferManagement_GetOffer"/>
            <permission id="OfferManagement_GetProduct"/>
            <permission id="TableManagement_GetTable"/>
            <permission id="StaffManagement_GetStaffMember"/>
        </permissions>
    </group>

    <group id="Waiter" type="role">
        <inherits>
            <group-ref>Barkeeper</group-ref>
        </inherits>
        <permissions>
            <permission id="TableManagement_ChangeTable"/>
        </permissions>
    </group>
    ...
</access-control-schema>
```

This example `access-control-schema.xml` declares

- a group named `ReadMasterData`, which grants four different permissions, e.g., `OfferManagement_GetOffer`
- a group named `Waiter`, which
 - also grants all permissions from the group `Barkeeper`
 - in addition grants the permission `TableManagement_ChangeTable`
 - is marked to be a `role` for further application needs.

The devon4j-security module automatically validates the schema configuration and will throw an exception if invalid.

Unfortunately, Spring Security does not provide differentiated interfaces for authentication and authorization. Thus we have to provide an `AuthenticationProvider`, which is provided from Spring Security as an interface for authentication and authorization simultaneously. To integrate the devon4j-security provided access control schema, you can simply inherit your own implementation from the devon4j-security provided abstract class `AbstractAccessControlBasedAuthenticationProvider` and register your `ApplicationAuthenticationProvider` as an `AuthenticationManager`. Doing so, you also have to declare the two Beans `AccessControlProvider` and `AccessControlSchemaProvider`, which are precondition for the `AbstractAccessControlBasedAuthenticationProvider`.

As state of the art devon4j will focus on role-based authorization to cope with authorization for executing use case of an application. We will use the JSR250 annotations, mainly `@RolesAllowed`, for

authorizing method calls against the permissions defined in the annotation body. This has to be done for each use-case method in logic layer. Here is an example:

```
public class OrdermanagementImpl extends AbstractComponentFacade implements  
Ordermanagement {  
  
    @RolesAllowed(Roles.WAITER)  
    public PaginatedListTo<OrderCto> findOrdersByPost(OrderSearchCriteriaTo criteria) {  
  
        return findOrderCtos(criteria);  
    }  
}
```

Now this method can only be called if a user is logged-in that has the permission [FIND_TABLE](#).

12.11.2. Data-permissions

In some projects there are demands for permissions and authorization that is dependent on the processed data. E.g. a user may only be allowed to read or write data for a specific region. This is adding some additional complexity to your authorization. If you can avoid this it is always best to keep things simple. However, in various cases this is a requirement. Therefore the following sections give you guidance and patterns how to solve this properly.

Structuring your data

For all your business objects (entities) that have to be secured regarding to data permissions we recommend that you create a separate interface that provides access to the relevant data required to decide about the permission. Here is a simple example:

```
public interface SecurityDataPermissionCountry {  
  
    /**  
     * @return the 2-letter ISO code of the country this object is associated with.  
     * Users need  
     *      a data-permission for this country in order to read and write this  
     * object.  
     */  
    String getCountry();  
}
```

Now related business objects (entities) can implement this interface. Often such data-permissions have to be applied to an entire object-hierarchy. For security reasons we recommend that also all child-objects implement this interface. For performance reasons we recommend that the child-objects redundantly store the data-permission properties (such as `country` in the example above) and this gets simply propagated from the parent, when a child object is created.

Permissions for processing data

When saving or processing objects with a data-permission, we recommend to provide dedicated methods to verify the permission in an abstract base-class such as `AbstractUc` and simply call this explicitly from your business code. This makes it easy to understand and debug the code. Here is a simple example:

```
protected void verifyPermission(SecurityDataPermissionCountry entity) throws  
AccessDeniedException;
```

Beware of AOP

For simple but cross-cutting data-permissions you may also use [AOP](#). This leads to programming aspects that reflectively scan method arguments and magically decide what to do. Be aware that this quickly gets tricky:

- What if multiple of your method arguments have data-permissions (e.g. implement

SecurityDataPermission*)?

- What if the object to authorize is only provided as reference (e.g. `Long` or `IdRef`) and only loaded and processed inside the implementation where the AOP aspect does not apply?
- How to express advanced data-permissions in annotations?

What we have learned is that annotations like `@PreAuthorize` from `spring-security` easily lead to the "programming in string literals" anti-pattern. We strongly discourage to use this anti-pattern. In such case writing your own `verifyPermission` methods that you manually call in the right places of your business-logic is much better to understand, debug and maintain.

Permissions for reading data

When it comes to restrictions on the data to read it becomes even more tricky. In the context of a user only entities shall be loaded from the database he is permitted to read. This is simple for loading a single entity (e.g. by its ID) as you can load it and then if not permitted throw an exception to secure your code. But what if the user is performing a search query to find many entities? For performance reasons we should only find data the user is permitted to read and filter all the rest already via the database query. But what if this is not a requirement for a single query but needs to be applied cross-cutting to tons of queries? Therefore we have the following pattern that solves your problem:

For each data-permission attribute (or set of such) we create an abstract base entity:

```
@MappedSuperclass
@EntityListeners(PermissionCheckListener.class)
@FilterDef(name = "country", parameters = {@ParamDef(name = "countries", type =
"string")})
@Filter(name = "country", condition = "country in (:countries)")
public abstract class SecurityDataPermissionCountryEntity extends
ApplicationPersistenceEntity
    implements SecurityDataPermissionCountry {

    private String country;

    @Override
    public String getCountry() {
        return this.country;
    }

    public void setCountry(String country) {
        this.country = country;
    }
}
```

There are some special hibernate annotations `@EntityListeners`, `@FilterDef`, and `@Filter` used here allowing to apply a filter on the `country` for any (non-native) query performed by hibernate. The entity listener may look like this:

```

public class PermissionCheckListener {

    @PostLoad
    public void read(SecurityDataPermissionCountryEntity entity) {
        PermissionChecker.getInstance().requireReadPermission(entity);
    }

    @PrePersist
    @PreUpdate
    public void write(SecurityDataPermissionCountryEntity entity) {
        PermissionChecker.getInstance().requireWritePermission(entity);
    }
}

```

This will ensure that hibernate implicitly will call these checks for every such entity when it is read from or written to the database. Further to avoid reading entities from the database the user is not permitted to (and ending up with exceptions), we create an AOP aspect that automatically activates the above declared hibernate filter:

```

@Named
public class PermissionCheckerAdvice implements MethodBeforeAdvice {

    @Inject
    private PermissionChecker permissionChecker;

    @PersistenceContext
    private EntityManager entityManager;

    @Override
    public void before(Method method, Object[] args, Object target) {

        Collection<String> permittedCountries = this.permissionChecker
            .getPermittedCountriesForReading();
        if (permittedCountries != null) { // null is returned for admins that may access
            all countries
            if (permittedCountries.isEmpty()) {
                throw new AccessDeniedException("Not permitted for any country!");
            }
            Session session = this.entityManager.unwrap(Session.class);
            session.enableFilter("country").setParameterList("countries",
permittedCountries.toArray());
        }
    }
}

```

Finally to apply this aspect to all Repositories (can easily be changed to DAOs) implement the following advisor:

```
@Named
public class PermissionCheckerAdvisor implements PointcutAdvisor, Pointcut,
ClassFilter, MethodMatcher {

    @Inject
    private PermissionCheckerAdvice advice;

    @Override
    public Advice getAdvice() {
        return this.advice;
    }

    @Override
    public boolean isPerInstance() {
        return false;
    }

    @Override
    public Pointcut getPointcut() {
        return this;
    }

    @Override
    public ClassFilter getClassFilter() {
        return this;
    }

    @Override
    public MethodMatcher getMethodMatcher() {
        return this;
    }

    @Override
    public boolean matches(Method method, Class<?> targetClass) {
        return true; // apply to all methods
    }

    @Override
    public boolean isRuntime() {
        return false;
    }

    @Override
    public boolean matches(Method method, Class<?> targetClass, Object... args) {
        throw new IllegalStateException("isRuntime()==false");
    }

    @Override
    public boolean matches(Class<?> clazz) {
        // when using DAOs simply change to some class like ApplicationDao
        return DefaultRepository.class.isAssignableFrom(clazz);
    }
}
```

{
}

Managing and granting the data-permissions

Following our [authorization guide](#) we can simply create a permission for each country. We might simply reserve a prefix (as virtual `<>app-id>>`) for each data-permission to allow granting data-permissions to end-users across all applications of the IT landscape. In our example we could create access controls `country.DE`, `country.US`, `country.ES`, etc. and assign those to the users. The method `permissionChecker.getPermittedCountriesForReading()` would then scan for these access controls and only return the 2-letter country code from it.



Before you make your decisions how to design your access controls please clarify the following questions:

- Do you need to separate data-permissions independent of the functional permissions? E.g. may it be required to express that a user can read data from the countries `ES` and `PL` but is only permitted to modify data from `PL`? In such case a single assignment of "country-permissions" to users is insufficient.
- Do you want to grant data-permissions individually for each application (higher flexibility and complexity) or for the entire application landscape (simplicity, better maintenance for administrators)? In case of the first approach you would rather have access controls like `app1.country.GB` and `app2.country.GB`.
- Do your data-permissions depend on objects that can be created dynamically inside your application?
- If you want to grant data-permissions on other business objects (entities), how do you want to reference them (primary keys, business keys, etc.)? What reference is most stable? Which is most readable?

12.12. JSON Web Token(JWT)

JWT is an open standard ([RFC 7519](#)) for creating Json based access token that assert some number of claims.



For more information about JWT [click here](#)

12.12.1. Keystore

A KeyStore is a repository of certificates and keys (public key, private key, or secret key). They can be used for TSL transportation, for encryption and decryption as well as for signing. For demonstration you might create a keystore with openssl, with the following commands:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365
openssl pkcs12 -export -in cert.pem -inkey key.pem -out example.p12
```

For Java tooling you may also try the following instead:

```
keytool -genkeypair -alias devonfw -keypass "password" -storetype PKCS12 -keyalg RSA
-keysize 4096 -storepass "password" -keystore keystore.p12
```



Please use reasonable passwords instead of **password** what should be obvious. Also for the alias the value **devonfw** is just an example.

12.12.2. Dependency

To use JWT support from devon4j you have to add following required dependency.

```
<dependency>
    <groupId>com.devonfw.java.starters</groupId>
    <artifactId>devon4j-starter-security-jwt</artifactId>
</dependency>
```

12.12.3. Configuration

The following properties need to be configured in your `application.properties` file:

```
# location of the keystore file, can be any spring resource (such as file or classpath
# URIs)
security.keystore.location=classpath:config/keystore.pcks
# type of keystore e.g. "PKCS12" (recommended), "JKS", or "JCEKS"
security.keystore.type=PKCS12
# password the keystore is secured with. Consider using password encryption as
described in devon4j configuration guide
security.keystore.password=password
# the algorithm for encryption/decryption and signing - see
io.jsonwebtoken.SignatureAlgorithm
security.authentication.jwt.algorithm=RS256
# alias of public/private key in keystore (for validation only public key is used, for
creation private key is required)
security.authentication.jwt.alias=devonfw
# the following properties are used if you are validating JWTs (e.g. via
JwtAuthenticationFilter)
security.authentication.jwt.validation.expiration-required=false
security.authentication.jwt.validation.max-validity=42h
security.authentication.jwt.validation.not-before-required=false
# the following properties are only used if you are issuing JWTs (e.g. via
JwtLoginFilter)
security.authentication.jwt.creation.add-issued-at=true
security.authentication.jwt.creation.validity=4h
security.authentication.jwt.creation.not-before-delay=1m
```

12.12.4. Authentication with JWT via OAuth

The authentication with JWT via OAuth (HTTP header), will happen via `JwtAuthenticationFilter` that is automatically added by `devon4j-starter-security-jwt` via `JwtAutoConfiguration`. With the starter and auto-configuration we want to make it as easy as possible for you. In case you would like to build a server app that e.g. wants to issue JWTs but does not allow authentication via JWT itself, you can use `devon4j-security-jwt` as dependency instead of the starter and do the spring config yourself (pick and choose from `JwtAutoConfiguration`).

To do this, you need to add the following changes in your `BaseWebSecurityConfig`:

```

@Bean
public JwtAuthenticationFilter getJwtAuthenticationFilter() {
    return new JwtAuthenticationFilter();
}

@Override
public void configure(HttpSecurity http) throws Exception {
    // ...
    // add this line to the end of this existing method
    http.addFilterBefore(getJwtAuthenticationFilter(),
    UsernamePasswordAuthenticationFilter.class);
}

```

12.12.5. Login with Username and Password to get JWT

To allow a client to login with username and password to get a JWT for sub-sequent requests, you need to do the following changes in your `BaseWebSecurityConfig`:

```

@Bean
public JwtLoginFilter getJwtLoginFilter() throws Exception {

    JwtLoginFilter jwtLoginFilter = new JwtLoginFilter("/login");
    jwtLoginFilter.setAuthenticationManager(authenticationManager());
    jwtLoginFilter.setUserDetailsService(this.userDetailsService);
    return jwtLoginFilter;
}

@Override
public void configure(HttpSecurity http) throws Exception {
    // ...
    // add this line to the end of this existing method
    http.addFilterBefore(getJwtLoginFilter(), UsernamePasswordAuthenticationFilter
.class);
}

```

12.12.6. Authentication with Kafka

Authentication with JWT and Kafka is explained in the [Kafka guide](#).

12.13. Cross-site request forgery (CSRF)

CSRF is a type of malicious exploit of a web application that allows an attacker to induce users to perform actions that they do not intend to perform.



More details about csrf can be found at <https://owasp.org/www-community/attacks/csrf>.

12.13.1. Secure devon4j server against CSRF

In case your devon4j server application is not accessed by browsers or the web-client is using [JWT based authentication](#), you are already safe according to CSRF. However, if your application is accessed from a browser and you are using *form based authentication* (with session cookie) or *basic authentication*, you need to enable CSRF protection. This guide will tell you how to do this.

Dependency

To secure your devon4j application against CSRF attacks, you only need to add the following dependency:

```

<dependency>
    <groupId>com.devonfw.java.starters</groupId>
    <artifactId>devon4j-starter-security-csrf</artifactId>
</dependency>
  
```

Starting with devon4j version [2020.12.001](#) application template, this is all you need to do. However, if you have started from an older version or you want to understand more, please read on.

Pluggable web-security

To enable pluggable security via devon4j security starters you need to apply [WebSecurityConfigurer](#) to your [BaseWebSecurityConfig](#) (your class extending spring-boot's [WebSecurityConfigurerAdapter](#)) as following:

```

@Inject
private WebSecurityConfigurer webSecurityConfigurer;

public void configure(HttpSecurity http) throws Exception {
    // disable CSRF protection by default, use csrf starter to override.
    http = http.csrf().disable();
    // apply pluggable web-security from devon4j security starters
    http = this.webSecurityConfigurer.configure(http);
    ....
}

```

Custom CsrfRequestMatcher

If you want to customize which HTTP requests will require a CSRF token, you can implement your own `CsrfRequestMatcher` and provide it to the devon4j CSRF protection via qualified injection as following:

```

@Named("CsrfRequestMatcher")
public class CsrfRequestMatcher implements RequestMatcher {
    @Override
    public boolean matches(HttpServletRequest request) {
        ....
    }
}

```

Please note that the exact name (`@Named("CsrfRequestMatcher")`) is required here to ensure your custom implementation will be injected properly.

CsrfRestService

With the `devon4j-starter-security-csrf` the `CsrfRestService` gets integrated into your app. It provides an operation to get the CSRF token via an HTTP `GET` request. The URL path to retrieve this CSRF token is `services/rest/csrf/v1/token`. As a result you will get a JSON like the following:

```
{
    "token": "3a8a5f66-c9eb-4494-81e1-7cc58bc3a519",
    "parameterName": "_csrf",
    "headerName": "X-CSRF-TOKEN"
}
```

The `token` value is a strong random value that will differ for each user session. It has to be send with subsequent HTTP requests (when method is other than `GET`) in the specified header (`X-CSRF-TOKEN`).

How it works

Putting it all together, a browser client should call the `CsrfRestService` after successfull login to receive the current CSRF token. With every subsequent HTTP request (other than `GET`) the client has

to send this token in the according HTTP header. Otherwise the server will reject the request to prevent CSRF attacks. Therefore, an attacker might make your browser perform HTTP requests towards your devon4j application backend via <image> elements, <iframes>, etc. Your browser will then still include your session cookie if you are already logged in (e.g. from another tab). However, in case he wants to trigger **DELETE** or **POST** requests trying your browser to make changes in the application (delete or update data, etc.) this will fail without CSRF token. The attacker may make your browser retrieve the CSRF token but he will not be able to retrieve the result and put it into the header of other requests due to the same-origin-policy. This way your application will be secured against CSRF attacks.

12.13.2. Configure devon4ng client for CSRF

Devon4ng client configuration for CSRF is described [here](#)

12.14. Validation

Validation is about checking syntax and semantics of input data. Invalid data is rejected by the application. Therefore validation is required in multiple places of an application. E.g. the [GUI](#) will do validation for usability reasons to assist the user, early feedback and to prevent unnecessary server requests. On the server-side validation has to be done for consistency and [security](#).

In general we distinguish these forms of validation:

- *stateless validation* will produce the same result for given input at any time (for the same code/release).
- *stateful validation* is dependent on other states and can consider the same input data as valid in once case and as invalid in another.

12.14.1. Stateless Validation

For regular, stateless validation we use the JSR303 standard that is also called bean validation (BV). Details can be found in the [specification](#). As implementation we recommend [hibernate-validator](#).

Example

A description of how to enable BV can be found in the relevant [Spring documentation](#). For a quick summary follow these steps:

- Make sure that hibernate-validator is located in the classpath by adding a dependency to the pom.xml.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
</dependency>
```

- Add the `@Validated` annotation to the implementation (spring bean) to be validated. The standard use case is to annotate the logic layer implementation, i.e. the use case implementation or component facade in case of simple logic layer pattern. Thus, the validation will be executed for service requests as well as batch processing. For methods to validate go to their declaration and add constraint annotations to the method parameters.
 - `@Valid` annotation to the arguments to validate (if that class itself is annotated with constraints to check).
 - `@NotNull` for required arguments.
 - Other constraints (e.g. `@Size`) for generic arguments (e.g. of type `String` or `Integer`). However, consider to create [custom datatypes](#) and avoid adding too much validation logic (especially redundant in multiple places). [BookingmanagementRestServiceImpl.java](#)

```
@Validated
public class BookingmanagementRestServiceImpl implements BookingmanagementRestService {
    ...
    public BookingEto saveBooking(@Valid BookingCto booking) {
        ...
    }
}
```

- Finally add appropriate validation constraint annotations to the fields of the ETO class.
- .BookingCto.java**

```
@Valid
private BookingEto booking;
```

Listing 10. BookingEto.java

```
@NotNull
@Future
private Timestamp bookingDate;
```

A list with all bean validation constraint annotations available for hibernate-validator can be found [here](#). In addition it is possible to configure custom constraints. Therefore it is necessary to implement a annotation and a corresponding validator. A description can also be found in the [Spring documentation](#) or with more details in the [hibernate documentation](#).



Bean Validation in Wildfly >v8: Wildfly v8 is the first version of Wildfly implementing the JEE7 specification. It comes with bean validation based on [hibernate-validator out of the box](#). In case someone is running Spring in Wildfly for whatever reasons, the spring based annotation `@Validated` would duplicate bean validation at runtime and thus should be omitted.

GUI-Integration

TODO

Cross-Field Validation

BV has poor support for this. Best practice is to create and use beans for ranges, etc. that solve this. A bean for a range could look like so:

```

public class Range<V extends Comparable<V>> {

    private V min;
    private V max;

    public Range(V min, V max) {

        super();
        if ((min != null) && (max != null)) {
            int delta = min.compareTo(max);
            if (delta > 0) {
                throw new ValueOutOfRangeException(null, min, min, max);
            }
        }
        this.min = min;
        this.max = max;
    }

    public V getMin() ...
    public V getMax() ...
}

```

12.14.2. Stateful Validation

For complex and stateful business validations we do not use BV (possible with groups and context, etc.) but follow KISS and just implement this on the server in a straight forward manner. An example is the deletion of a table in the example application. Here the state of the table must be checked first: **BookingmanagementImpl.java**

```

private void sendConfirmationEmails(BookingEntity booking) {

    if (!booking.getInvitedGuests().isEmpty()) {
        for (InvitedGuestEntity guest : booking.getInvitedGuests()) {
            sendInviteEmailToGuest(guest, booking);
        }
    }

    sendConfirmationEmailToHost(booking);
}

```

Implementing this small check with BV would be a lot more effort.

12.15. Aspect Oriented Programming (AOP)

AOP is a powerful feature for cross-cutting concerns. However, if used extensive and for the wrong things an application can get unmaintainable. Therefore we give you the best practices where and how to use AOP properly.

12.15.1. AOP Key Principles

We follow these principles:

- We use [spring AOP](#) based on dynamic proxies (and fallback to cglib).
- We avoid AspectJ and other mighty and complex AOP frameworks whenever possible
- We only use AOP where we consider it as necessary (see below).

12.15.2. AOP Usage

We recommend to use AOP with care but we consider it established for the following cross cutting concerns:

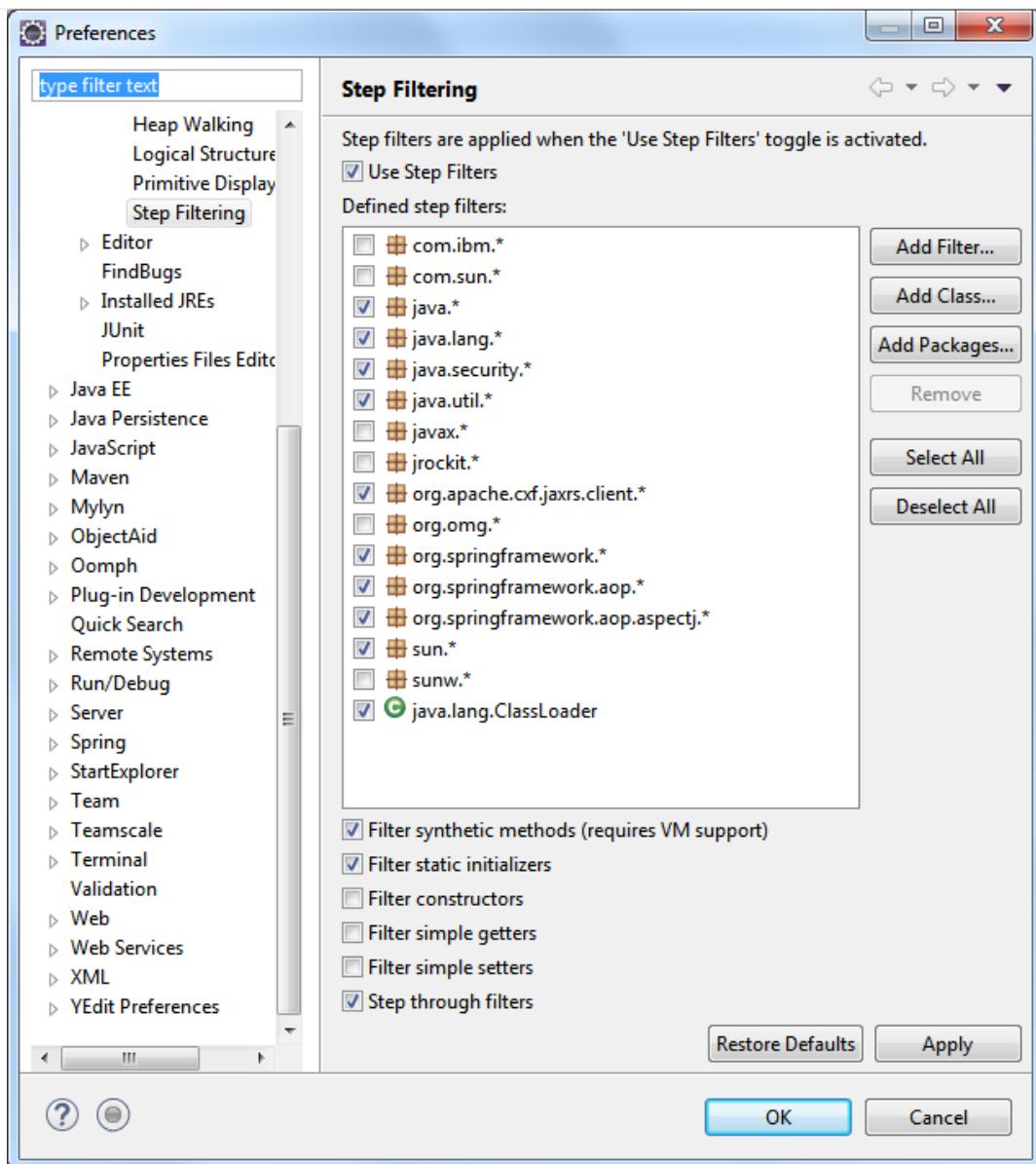
- [Transaction-Handling](#)
- [Authorization](#)
- [Validation](#)
- [Trace-Logging](#) (for testing and debugging)
- Exception facades for [services](#) but only if no other solution is possible (use alternatives such as [JAX-RS provider](#) instead).

12.15.3. AOP Debugging

When using AOP with dynamic proxies the debugging of your code can get nasty. As you can see by the red boxes in the call stack in the debugger there is a lot of magic happening while you often just want to step directly into the implementation skipping all the AOP clutter. When using Eclipse this can easily be archived by enabling *step filters*. Therefore you have to enable the feature in the Eclipse tool bar (highlighted in read).



In order to properly make this work you need to ensure that the step filters are properly configured:



Ensure you have at least the following step-filters configured and active:

```
ch.qos.logback.*  
com.devonfw.module.security.*  
java.lang.reflect.*  
java.security.*  
javax.persistence.*  
org.apache.commons.logging.*  
org.apache.cxf.jaxrs.client.*  
org.apache.tomcat.*  
org.h2.*  
org.springframework.*
```

12.16. Exception Handling

12.16.1. Exception Principles

For exceptions we follow these principles:

- We only use exceptions for *exceptional* situations and not for programming control flows, etc. Creating an exception in Java is expensive and hence you should not do it just for testing if something is present, valid or permitted. In the latter case design your API to return this as a regular result.
- We use unchecked exceptions (`RuntimeException`) [\[2\]](#)
- We distinguish *internal exceptions* and *user exceptions*:
 - Internal exceptions have technical reasons. For unexpected and exotic situations it is sufficient to throw existing exceptions such as `IllegalStateException`. For common scenarios a own exception class is reasonable.
 - User exceptions contain a message explaining the problem for end users. Therefore we always define our own exception classes with a clear, brief but detailed message.
- Our own exceptions derive from an exception base class supporting
 - [unique ID per instance](#)
 - [Error code per class](#)
 - [message templating](#) (see [I18N](#))
 - [distinguish between user exceptions and internal exceptions](#)

All this is offered by `mmm-util-core` that we propose as solution.

12.16.2. Exception Example

Here is an exception class from our sample application:

```
public class IllegalEntityStateException extends ApplicationBusinessException {  
  
    private static final long serialVersionUID = 1L;  
  
    public IllegalEntityStateException(Object entity, Object state) {  
  
        this((Throwable) null, entity, state);  
    }  
  
    public IllegalEntityStateException(Object entity, Object currentState, Object  
newState) {  
  
        this(null, entity, currentState, newState);  
    }  
  
    public IllegalEntityStateException(Throwable cause, Object entity, Object state) {  
  
        super(cause, createBundle(NlsBundleApplicationRoot.class).errorIllegalEntityState  
(entity, state));  
    }  
  
    public IllegalEntityStateException(Throwable cause, Object entity, Object  
currentState, Object newState) {  
  
        super(cause, createBundle(NlsBundleApplicationRoot.class)  
.errorIllegalEntityStateChange(entity, currentState,  
            newState));  
    }  
}
```

The message templates are defined in the interface NlsBundleRestaurantRoot as following:

```

public interface NlsBundleApplicationRoot extends NlsBundle {

    @NlsBundleMessage("The entity {entity} is in state {state}!")
    NlsMessage errorIllegalEntityState(@Named("entity") Object entity, @Named("state")
Object state);

    @NlsBundleMessage("The entity {entity} in state {currentState} can not be changed to
state {newState}!")
    NlsMessage errorIllegalEntityStateChange(@Named("entity") Object entity, @Named(
"currentState") Object currentState,
    @Named("newState") Object newState);

    @NlsBundleMessage("The property {property} of object {object} can not be changed!")
    NlsMessage errorIllegalPropertyChange(@Named("object") Object object, @Named(
"property") Object property);

    @NlsBundleMessage("There is currently no user logged in")
    NlsMessage errorNoActiveUser();
}

```

12.16.3. Handling Exceptions

For catching and handling exceptions we follow these rules:

- We do not catch exceptions just to wrap or to re-throw them.
- If we catch an exception and throw a new one, we always **have** to provide the original exception as `cause` to the constructor of the new exception.
- At the entry points of the application (e.g. a service operation) we have to catch and handle all `throwables`. This is done via the *exception-facade-pattern* via an explicit facade or aspect. The `devon4j-rest` module already provides ready-to-use implementations for this such as `RestServiceExceptionFacade`. The exception facade has to ...
 - log all errors (user errors on info and technical errors on error level)
 - ensure the entire exception is passed to the logger (not only the message) so that the logger can capture the entire stacktrace and the root cause is not lost.
 - convert the error to a result appropriate for the client and secure for `Sensitive Data Exposure`. Especially for security exceptions only a generic security error code or message may be revealed but the details shall only be logged but **not** be exposed to the client. All *internal exceptions* are converted to a generic error with a message like:

An unexpected technical error has occurred. We apologize any inconvenience. Please try again later.

12.16.4. Common Errors

The following errors may occur in any devon application:

Table 33. Common Exceptions

Code	Message	Link
TechnicalError	An unexpected error has occurred! We apologize any inconvenience. Please try again later.	TechnicalErrorUserException.java
ServiceInvoke	«original message of the cause»	ServiceInvocationFailedException.java

12.17. Internationalization

Internationalization (I18N) is about writing code independent from locale-specific information. For I18N of text messages we are suggesting [mmm native-language-support](#).

In devonfw we have developed a solution to manage text internationalization. devonfw solution comes into two aspects:

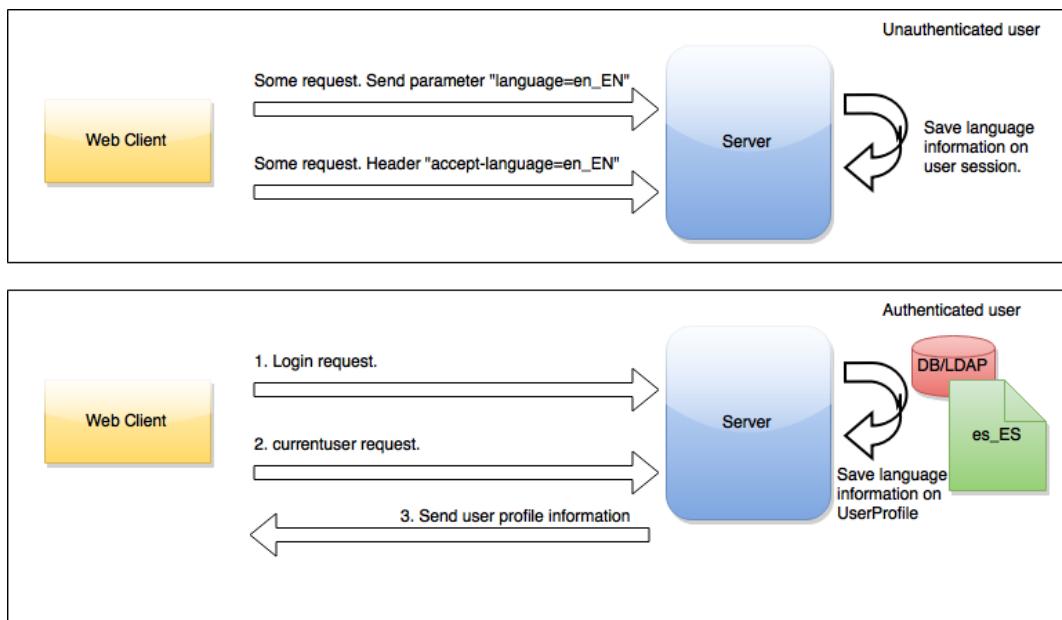
- Bind locale information to the user.
- Get the messages in the current user locale.

12.17.1. Binding locale information to the user

We have defined two different points to bind locale information to user, depending on user is authenticated or not.

- User not authenticated: devonfw intercepts unsecured request and extract locale from it. At first, we try to extract a `language` parameter from the request and if it is not possible, we extract locale from `Accept-language` header.`
- User authenticated. During login process, applications developers are responsible to fill `language` parameter in the `UserProfile` class. This `language` parameter could be obtain from DB, LDAP, request, etc. In devonfw sample we get the locale information from database.

This image shows the entire process:



12.17.2. Getting internationalized messages

devonfw has a bean that manage i18n message resolution, the `ApplicationLocaleResolver`. This bean is responsible to get the current user and extract locale information from it and read the correct properties file to get the message.

The i18n properties file must be called `ApplicationMessages_la_CO.properties` where la=language and CO=country. This is an example of a i18n properties file for English language to translate

devonfw sample user roles:

ApplicationMessages_en_US.properties

```
admin=Admin
```

You should define an ApplicationMessages_la_CO.properties file for every language that your application needs.

`ApplicationLocaleResolver` bean is injected in `AbstractComponentFacade` class so you have available this bean in logic layer so you only need to put this code to get an internationalized message:

```
String msg = getApplicationLocaleResolver().getMessage("mymessage");
```

12.18. XML

XML (Extensible Markup Language) is a W3C standard format for structured information. It has a large eco-system of additional standards and tools.

In Java there are many different APIs and frameworks for accessing, producing and processing XML. For the devonfw we recommend to use [JAXB](#) for mapping Java objects to XML and vice-versa. Further there is the popular [DOM API](#) for reading and writing smaller XML documents directly. When processing large XML documents [StAX](#) is the right choice.

12.18.1. JAXB

We use [JAXB](#) to serialize Java objects to XML or vice-versa.

JAXB and Inheritance

Use [@XmlSeeAlso](#) annotation to provide sub-classes. See section "Collective Polymorphism" described [here](#).

JAXB Custom Mapping

In order to map custom [datatypes](#) or other types that do not follow the Java bean conventions, you need to define a custom mapping. If you create dedicated objects for the XML mapping you can easily avoid such situations. When this is not suitable use [@XmlJavaTypeAdapter](#) and provide an [XmlAdapter](#) implementation that handles the mapping. For details see [here](#).

12.18.2. Security

To prevent XML External Entity attacks, follow [JAXP Security Guide](#) and enable [FSP](#).

12.19. JSON

JSON (JavaScript Object Notation) is a popular format to represent and exchange data especially for modern web-clients. For mapping Java objects to JSON and vice-versa there is no official standard API. We use the established and powerful open-source solution [Jackson](#). Due to problems with the wiki of fasterxml you should try this alternative link: [Jackson/AltLink](#).

12.19.1. Configure JSON Mapping

In order to avoid polluting business objects with proprietary Jackson annotations (e.g. `@JsonTypeInfo`, `@JsonSubTypes`, `@JsonProperty`) we propose to create a separate configuration class. Every devonfw application (sample or any app created from our [app-template](#)) therefore has a class called `ApplicationObjectMapperFactory` that extends `ObjectMapperFactory` from the `devon4j-rest` module. It looks like this:

```
@Named("ApplicationObjectMapperFactory")
public class ApplicationObjectMapperFactory extends ObjectMapperFactory {

    public RestaurantObjectMapperFactory() {
        super();
        // JSON configuration code goes here
    }
}
```

12.19.2. JSON and Inheritance

If you are using inheritance for your objects mapped to JSON then polymorphism can not be supported out-of-the box. So in general avoid polymorphic objects in JSON mapping. However, this is not always possible. Have a look at the following example from our sample application:



Figure 7. Transfer-Objects using Inheritance

Now assume you have a [REST service operation](#) as Java method that takes a `ProductEto` as argument. As this is an abstract class the server needs to know the actual sub-class to instantiate. We typically do not want to specify the classname in the JSON as this should be an implementation detail and not part of the public JSON format (e.g. in case of a service interface). Therefore we use a symbolic name for each polymorphic subtype that is provided as virtual attribute `@type` within the JSON data of the object:

```
{ "@type": "Drink", ... }
```

Therefore you add configuration code to the constructor of [ApplicationObjectMapperFactory](#). Here you can see an example from the sample application:

```
setBaseClasses(ProductEto.class);
addSubtypes(new NamedType(MealEto.class, "Meal"), new NamedType(DrinkEto.class, "Drink"),
new NamedType(SideDishEto.class, "SideDish"));
```

We use `setBaseClasses` to register all top-level classes of polymorphic objects. Further we declare all concrete polymorphic sub-classes together with their symbolic name for the JSON format via `addSubtypes`.

12.19.3. Custom Mapping

In order to map custom `datatypes` or other types that do not follow the Java bean conventions, you need to define a custom mapping. If you create objects dedicated for the JSON mapping you can easily avoid such situations. When this is not suitable follow these instructions to define the mapping:

- As an example, the use of JSR354 (`javax.money`) is appreciated in order to process monetary amounts properly. However, without custom mapping, the default mapping of Jackson will produce the following JSON for a `MonetaryAmount`:

```
"currency": {"defaultFractionDigits":2, "numericCode":978, "currencyCode":"EUR"},  
"monetaryContext": {...},  
"number":6.99,  
"factory": {...}
```

As clearly can be seen, the JSON contains too much information and reveals implementation secrets that do not belong here. Instead the JSON output expected and desired would be:

```
"currency":"EUR", "amount": "6.99"
```

Even worse, when we send the JSON data to the server, Jackson will see that `MonetaryAmount` is an interface and does not know how to instantiate it so the request will fail. Therefore we need a customized [Serializer](#).

- We implement `MonetaryAmountJsonSerializer` to define how a `MonetaryAmount` is serialized to JSON:

```

public final class MonetaryAmountJsonSerializer extends JsonSerializer<MonetaryAmount> {

    public static final String NUMBER = "amount";
    public static final String CURRENCY = "currency";

    public void serialize(MonetaryAmount value, JsonGenerator jgen,
    SerializerProvider provider) throws ... {
        if (value != null) {
            jgen.writeStartObject();
            jgen.writeFieldName(MonetaryAmountJsonSerializer.CURRENCY);
            jgen.writeString(value.getCurrency().getCurrencyCode());
            jgen.writeFieldName(MonetaryAmountJsonSerializer.NUMBER);
            jgen.writeString(value.getNumber().toString());
            jgen.writeEndObject();
        }
    }
}

```

For composite datatypes it is important to wrap the info as an object (`writeStartObject()` and `writeEndObject()`). `MonetaryAmount` provides the information we need by the `getCurrency()` and `getNumber()`. So that we can easily write them into the JSON data.

3. Next, we implement `MonetaryAmountJsonDeserializer` to define how a `MonetaryAmount` is deserialized back as Java object from JSON:

```

public final class MonetaryAmountJsonDeserializer extends AbstractJsonDeserializer<MonetaryAmount> {
    protected MonetaryAmount deserializeNode(JsonNode node) {
        BigDecimal number = getRequiredValue(node, MonetaryAmountJsonSerializer.NUMBER,
        BigDecimal.class);
        String currencyCode = getRequiredValue(node, MonetaryAmountJsonSerializer.CURRENCY,
        String.class);
        MonetaryAmount monetaryAmount =
            MonetaryAmounts.getAmountFactory().setNumber(number).setCurrency
        (currencyCode).create();
        return monetaryAmount;
    }
}

```

For composite datatypes we extend from `AbstractJsonDeserializer` as this makes our task easier. So we already get a `JsonNode` with the parsed payload of our datatype. Based on this API it is easy to retrieve individual fields from the payload without taking care of their order, etc. `AbstractJsonDeserializer` also provides methods such as `getRequiredValue` to read required fields and get them converted to the desired basis datatype. So we can easily read the amount and currency and construct an instance of `MonetaryAmount` via the official factory API.

4. Finally we need to register our custom (de)serializers with the following configuration code in the constructor of `ApplicationObjectMapperFactory`:

```
SimpleModule module = getExtensionModule();
module.addDeserializer(MonetaryAmount.class, new MonetaryAmountJsonDeserializer());
module.addSerializer(MonetaryAmount.class, new MonetaryAmountJsonSerializer());
```

Now we can read and write **MonetaryAmount** from and to JSON as expected.

12.20. REST

REST (REpresentational State Transfer) is an inter-operable protocol for [services](#) that is more lightweight than [SOAP](#). However, it is no real standard and can cause confusion. Therefore we define best practices here to guide you. **ATTENTION:** REST and RESTful often implies very strict and specific rules and conventions. However different people will often have different opinions of such rules. We learned that this leads to "religious discussions" (starting from [PUT](#) vs. [POST](#) and IDs in path vs. payload up to Hypermedia and [HATEOAS](#)). These "religious discussions" waste a lot of time and money without adding real value in case of common business applications (if you publish your API on the internet to billions of users this is a different story). Therefore we give best practices that lead to simple, easy and pragmatic "HTTP APIs" (to avoid the term "REST services" and end "religious discussions"). Please also note that we do not want to assault anybody nor force anyone to follow our guidelines. Please read the following best practices carefully and be aware that they might slightly differ from what your first hit on the web will say about REST (see e.g. [RESTful cookbook](#)).

12.20.1. URLs

URLs are not case sensitive. Hence, we follow the best practice to use only lower-case-letters-with-hyphen-to-separate-words. For operations in REST we distinguish the following types of URLs:

- A *collection URL* is build from the rest service URL by appending the name of a collection. This is typically the name of an entity. Such URI identifies the entire collection of all elements of this type. Example: <https://mydomain.com/myapp/services/rest/mycomponent/v1/myentity>
- An *element URL* is build from a collection URL by appending an element ID. It identifies a single element (entity) within the collection. Example: <https://mydomain.com/myapp/services/rest/mycomponent/v1/myentity/42>
- A *search URL* is build from a collection URL by appending the segment [search](#). The search criteria is send as [POST](#). Example: <https://mydomain.com/myapp/services/rest/mycomponent/v1/myentity/search>

This fits perfect for [CRUD](#) operations. For business operations (processing, calculation, etc.) we simply create a collection URL with the name of the business operation instead of the entity name (use a clear naming convention to avoid collisions). Then we can [POST](#) the input for the business operation and get the result back.

If you want to provide an entity with a different structure do not append further details to an element URL but create a separate collection URL as base. So use <https://mydomain.com/myapp/services/rest/mycomponent/v1/myentity-with-details/42> instead of <https://mydomain.com/myapp/services/rest/mycomponent/v1/myentity/42/with-details>. For offering a [CTO](#) simply append [-cto](#) to the collection URL (e.g. [.../myentity-cto](https://mydomain.com/myapp/services/rest/myentity-cto)).

12.20.2. HTTP Methods

While REST was designed as a pragmatical approach it sometimes leads to "religious discussions" e.g. about using [PUT](#) vs. [POST](#) (see ATTENTION notice above). As the devonfw has a strong focus on usual business applications it proposes a more "pragmatic" approach to REST services.

On the next table we compare the main differences between the "canonical" REST approach (or RESTful) and the devonfw proposal.

Table 34. Usage of HTTP methods

HTTP Method	RESTful Meaning	devonfw
GET	Read single element.	Read a single element.
	Search on an entity (with parametrized url)	
PUT	Replace entity data.	Not used
	Replace entire collection (typically not supported)	
POST	Create a new element in the collection	Create or update an element in the collection.
		Search on an entity (parametrized post body)
		Bulk deletion.
DELETE	Delete an entity.	Delete an entity.
	Delete an entire collection (typically not supported)	Delete an entire collection (typically not supported)

Please consider these guidelines and rationales:

- We use **POST** on the collection URL to save an entity (**create** if no ID provided in payload otherwise **update**). This avoids pointless discussions in distinctions between **PUT** and **POST** and what to do if a **create** contains an ID in the payload or if an **update** is missing the ID property or contains a different ID in payload than in URL.
- Hence, we do NOT use **PUT** but always use **POST** for write operations. As we always have a technical ID for each entity, we can simply distinguish create and update by the presence of the ID property.
- Please also note that for (large) bulk deletions you may be forced to used **POST** instead of **DELETE** as according to the HTTP standard **DELETE** must not have payload and URLs are limited in length.

12.20.3. HTTP Status Codes

Further we define how to use the HTTP status codes for REST services properly. In general the 4xx codes correspond to an error on the client side and the 5xx codes to an error on the server side.

Table 35. Usage of HTTP status codes

HTTP Code	Meaning	Response	Comment
200	OK	requested result	Result of successful GET
204	No Content	<i>none</i>	Result of successful POST, DELETE, or PUT (void return)
400	Bad Request	error details	The HTTP request is invalid (parse error, validation failed)
401	Unauthorized	<i>none</i> (security)	Authentication failed
403	Forbidden	<i>none</i> (security)	Authorization failed
404	Not found	<i>none</i>	Either the service URL is wrong or the requested resource does not exist
500	Server Error	error code, UUID	Internal server error occurred (used for all technical exceptions)

12.20.4. Metadata

devonfw has support for the following metadata in REST service invocations:

Name	Description	Further information
X-Correlation-Id	HTTP header for a <i>correlation ID</i> that is a unique identifier to associate different requests belonging to the same session / action	Logging guide
Validation errors	Standardized format for a service to communicate validation errors to the client	<p>Server-side validation is documented in the Validation guide.</p> <p>The protocol to communicate these validation errors is described in REST exception handling.</p>
Pagination	Standardized format for a service to offer paginated access to a list of entities	Server-side support for pagination is documented in the Repository Guide .

12.20.5. JAX-RS

For implementing REST services we use the [JAX-RS](#) standard. As an implementation we recommend [CXF](#). For [JSON](#) bindings we use [Jackson](#) while [XML](#) binding works out-of-the-box with [JAXB](#). To implement a service you write an interface with JAX-RS annotations for the API and a regular implementation class annotated with [@Named](#) to make it a spring-bean. Here is a simple example:

```

@Path("/imagemanagement/v1")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public interface ImagemanagementRestService {

    @GET
    @Path("/image/{id}/")
    public ImageEto getImage(@PathParam("id") long id);

}

@Named("ImagemanagementRestService")
public class ImagemanagementRestServiceImpl implements ImagemanagementRestService {

    @Inject
    private Imagemanagement imagemanagement;

    @Override
    public ImageEto getImage(long id) {

        return this.imagemanagement.findImage(id);
    }

}

```

Here we can see a REST service for the [business component](#) `imagemanagement`. The method `getImage` can be accessed via HTTP GET (see `@GET`) under the URL path `imagemanagement/image/{id}` (see `@Path` annotations) where `{id}` is the ID of the requested table and will be extracted from the URL and provided as parameter `id` to the method `getImage`. It will return its result (`ImageEto`) as [JSON](#) (see `@Produces` - should already be defined as defaults in `RestService` marker interface). As you can see it delegates to the [logic](#) component `imagemanagement` that contains the actual business logic while the service itself only exposes this logic via HTTP. The REST service implementation is a regular CDI bean that can use [dependency injection](#). The separation of the API as a Java interface allows to use it for [service client calls](#).



With JAX-RS it is important to make sure that each service method is annotated with the proper HTTP method (`@GET`,`@POST`,etc.) to avoid unnecessary debugging. So you should take care not to forget to specify one of these annotations.

JAX-RS Configuration

Starting from CXF 3.0.0 it is possible to enable the auto-discovery of JAX-RS roots.

When the jaxrs server is instantiated all the scanned root and provider beans (beans annotated with `javax.ws.rs.Path` and `javax.ws.rs.ext.Provider`) are configured.

12.20.6. REST Exception Handling

For exceptions a service needs to have an exception façade that catches all exceptions and handles them by writing proper log messages and mapping them to a HTTP response with an according [HTTP status code](#). Therefore the devonfw provides a generic solution via `RestServiceExceptionFacade`. You need to follow the [exception guide](#) so that it works out of the box because the façade needs to be able to distinguish between business and technical exceptions. Now your service may throw exceptions but the façade will automatically handle them for you.

12.20.7. Recommendations for REST requests and responses

The devonfw proposes, for simplicity, a deviation from the common REST pattern:

- Using `POST` for updates (instead of `PUT`)
- Using the payload for addressing resources on `POST` (instead of identifier on the `URL`)
- Using parametrized `POST` for searches

This use of REST will lead to simpler code both on client and on server. We discuss this use on the next points.

The following table specifies how to use the HTTP methods (verbs) for collection and element URIs properly (see [wikipedia](#)).

Unparameterized loading of a single resource

- **HTTP Method:** `GET`
- **URL example:** `/services/rest/productmanagement/v1/product/123`

For loading of a single resource, embed the `identifier` (e.g. `123`) of the resource in the URL.

The response contains the resource in JSON format, using a JSON object at the top-level, for example:

```
{  
  "id": 123,  
  "name": "Steak",  
  "color": "brown"  
}
```

Unparameterized loading of a collection of resources

- **HTTP Method:** GET
- **URL example:** /services/rest/productmanagement/v1/product

For loading of a collection of resources, make sure that the size of the collection can never exceed a reasonable maximum size. For parameterized loading (searching, pagination), see below.

The response contains the collection in JSON format, using a JSON object at the top-level, and the actual collection underneath a `result` key, for example:

```
{
  "result": [
    {
      "id": 123,
      "name": "Steak",
      "color": "brown"
    },
    {
      "id": 124,
      "name": "Broccoli",
      "color": "green"
    }
  ]
}
```

Saving a resource

- **HTTP Method:** POST
- **URL example:** /services/rest/productmanagement/v1/product

The resource will be passed via JSON in the request body. If updating an existing resource, include the resource's `identifier` in the JSON and not in the URL, in order to avoid ambiguity.

If saving was successful, the updated product (e.g. with assigned ID or updated modification counter) is returned.

If saving was unsuccessful, refer below for the format to return errors to the client.

Parameterized loading of a resource

- **HTTP Method:** POST
- **URL example:** /services/rest/productmanagement/v1/product/search

In order to differentiate from an unparameterized load, a special *subpath* (for example `search`) is introduced. The parameters are passed via JSON in the request body. An example of a simple, paginated search would be:

```
{
  "status": "OPEN",
  "pagination": {
    "page": 2,
    "size": 25
  }
}
```

The response contains the requested page of the collection in JSON format, using a JSON object at the top-level, the actual page underneath a `result` key, and additional pagination information underneath a `pagination` key, for example:

```
{
  "pagination": {
    "page": 2,
    "size": 25,
    "total": null
  },
  "result": [
    {
      "id": 123,
      "name": "Steak",
      "color": "brown"
    },
    {
      "id": 124,
      "name": "Broccoli",
      "color": "green"
    }
  ]
}
```

Compare the code needed on server side to accept this request:

```
@Path("/category/search")
@POST
public PaginatedListTo<CategoryEto> findCategoriesByPost(CategorySearchCriteriaTo
searchCriteriaTo) {
  return this.dishmanagement.findCategoryEtos(searchCriteriaTo);
}
```

With the equivalent code required if doing it the RESTful way by issuing a `GET` request:

```

@Path("/category/search")
@POST @Path("/order")
@GET
public PaginatedListTo<CategoryEto> findCategorysByPost( @Context UriInfo info) {

    RequestParameters parameters = RequestParameters.fromQuery(info);
    CategorySearchCriteriaTo criteria = new CategorySearchCriteriaTo();
    criteria.setName(parameters.get("name", Long.class, false));
    criteria.setDescription(parameters.get("description", OrderState.class, false));
    criteria.setShowOrder(parameters.get("showOrder", OrderState.class, false));
    return this.dishmanagement.findCategoryEtos(criteria);

}

```

Pagination details

The client can choose to request a count of the total size of the collection, for example to calculate the total number of available pages. It does so, by specifying the `pagination.total` property with a value of `true`.

The service is free to honour this request. If it chooses to do so, it returns the total count as the `pagination.total` property in the response.

Deletion of a resource

- **HTTP Method:** `DELETE`
- **URL example:** `/services/rest/productmanagement/v1/product/123`

For deletion of a single resource, embed the `identifier` of the resource in the URL.

Error results

The general format for returning an error to the client is as follows:

```
{
  "message": "A human-readable message describing the error",
  "code": "A code identifying the concrete error",
  "uuid": "An identifier (generally the correlation id) to help identify corresponding
requests in logs"
}
```

If the error is caused by a failed validation of the entity, the above format is extended to also include the list of individual validation errors:

```
{
  "message": "A human-readable message describing the error",
  "code": "A code identifying the concrete error",
  "uuid": "An identifier (generally the correlation id) to help identify corresponding requests in logs",
  "errors": {
    "property failing validation": [
      "First error message on this property",
      "Second error message on this property"
    ],
    // ....
  }
}
```

12.20.8. REST Media Types

The payload of a REST service can be in any format as REST by itself does not specify this. The most established ones that the devonfw recommends are [XML](#) and [JSON](#). Follow these links for further details and guidance how to use them properly. [JAX-RS](#) and [CXF](#) properly support these formats (`MediaType.APPLICATION_JSON` and `MediaType.APPLICATION_XML` can be specified for `@Produces` or `@Consumes`). Try to decide for a single format for all services if possible and NEVER mix different formats in a service.

12.20.9. REST Testing

For testing REST services in general consult the [testing guide](#).

For manual testing REST services there are browser plugins:

- Firefox: [rested](#)
- Chrome: [postman \(advanced-rest-client\)](#)

12.20.10. Security

Your services are the major entry point to your application. Hence security considerations are important here.

CSRF

A common security threat is [CSRF](#) for REST services. Therefore all REST operations that are performing modifications (PUT, POST, DELETE, etc. - all except GET) have to be secured against CSRF attacks. See [CSRF](#) how to do this.

JSON top-level arrays

OWASP suggests to prevent returning JSON arrays at the top-level, to prevent attacks (see https://www.owasp.org/index.php/OWASP_AJAX_Security_Guidelines). However, no rationale is given at OWASP. We digged deep and found [anatomy-of-a-subtle-json-vulnerability](#). To sum it up the

attack is many years old and does not work in any recent or relevant browser. Hence it is fine to use arrays as top-level result in a JSON REST service (means you can return `List<Foo>` in a Java JAX-RS service).

12.21. SOAP

SOAP is a common protocol for services that is rather complex and heavy. It allows to build interoperable and well specified services (see WSDL). SOAP is transport neutral what is not only an advantage. We strongly recommend to use HTTPS transport and ignore additional complex standards like WS-Security and use established HTTP-Standards such as RFC2617 (and RFC5280).

12.21.1. JAX-WS

For building web-services with Java we use the [JAX-WS](#) standard. There are two approaches:

- code first
- contract first

Here is an example in case you define a code-first service.

Web-Service Interface

We define a regular interface to define the API of the service and annotate it with JAX-WS annotations:

```
@WebService
public interface TablemanagementWebService {
    @WebMethod
    @WebResult(name = "message")
    TableEto getTable(@WebParam(name = "id") String id);
}
```

Web-Service Implementation

And here is a simple implementation of the service:

```
@Named
@WebService(endpointInterface =
"com.devonfw.application.mtsj.tablemanagement.service.api.ws.TablemanagementWebService")
public class TablemanagementWebServiceImpl implements TablemanagementWebService {

    private Tablemanagement tableManagement;

    @Override
    public TableEto getTable(String id) {
        return this.tableManagement.findTable(id);
    }
}
```

12.21.2. SOAP Custom Mapping

In order to map custom [datatypes](#) or other types that do not follow the Java bean conventions, you need to write adapters for JAXB (see [XML](#)).

12.21.3. SOAP Testing

For testing SOAP services in general consult the [testing guide](#).

For testing SOAP services manually we strongly recommend [SoapUI](#).

12.22. Service Client

This guide is about consuming (calling) services from other applications (micro-services). For providing services see the [Service-Layer Guide](#). Services can be consumed in the [client](#) or the server. As the client is typically not written in Java you should consult the according guide for your client technology. In case you want to call a service within your Java code this guide is the right place to get help.

12.22.1. Motivation

Various solutions already exist for calling services such as [RestTemplate](#) from spring or the JAX-RS client API. Further each and every service framework offers its own API as well. These solutions might be suitable for very small and simple projects (with one or two such invocations). However, with the trend of microservices the invocation of a service becomes a very common use-case that occurs all over the place. You typically need a solution that is very easy to use but supports flexible configuration, adding headers for authentication, mapping of errors from server, logging success/errors with duration for performance analysis, support for synchronous and asynchronous invocations, etc. This is exactly what this [devon4j](#) service-client solution brings for you.

12.22.2. Dependency

You need to add (at least one of) these dependencies to your application:

```
<!-- Starter for asynchronous consuming REST services via Java HTTP Client (Java11+)
-->
<dependency>
    <groupId>com.devonfw.java.starters</groupId>
    <artifactId>devon4j-starter-http-client-rest-async</artifactId>
</dependency>
<!-- Starter for synchronous consuming REST services via Apache CXF (Java8+) -->
<dependency>
    <groupId>com.devonfw.java.starters</groupId>
    <artifactId>devon4j-starter-cxf-client-rest</artifactId>
</dependency>
<!-- Starter for synchronous consuming SOAP services via Apache CXF (Java8+) -->
<dependency>
    <groupId>com.devonfw.java.starters</groupId>
    <artifactId>devon4j-starter-cxf-client-ws</artifactId>
</dependency>
```

12.22.3. Features

When invoking a service you need to consider many cross-cutting aspects. You might not think about them in the very first place and you do not want to implement them multiple times redundantly. Therefore you should consider using this approach. The following sub-sections list the covered features and aspects:

Simple usage

Assuming you already have a Java interface `MyService` of the service you want to invoke:

```
package com.company.department.foo.mycomponent.service.api.rest;
...
@Path("/myservice")
public interface MyService extends RestService {
    ...
    @POST
    @Path("/getresult")
    MyResult getResult(MyArgs myArgs);

    @DELETE
    @Path("/entity/{id}")
    void deleteEntity(@PathParam("id") long id);
}
```

Then all you need to do is this:

```
@Named
public class UcMyUseCaseImpl extends MyUseCaseBase implements UcMyUseCase {
    @Inject
    private ServiceClientFactory serviceClientFactory;

    ...
    private void callSynchronous(MyArgs myArgs) {
        MyService myService = this.serviceClientFactory.create(MyService.class);
        // call of service over the wire, synchronously blocking until result is received
        or error occurred
        MyResult myResult = myService.myMethod(myArgs);
        handleResult(myResult);
    }

    private void callAsynchronous(MyArgs myArgs) {
        AsyncServiceClient<MyService> client = this.serviceClientFactory.createAsync
        (MyService.class);
        // call of service over the wire, will return when request is send and invoke
        handleResult asynchronously
        client.call(client.get().myMethod(myArgs), this::handleResult);
    }

    private void handleResult(MyResult myResult) {
        ...
    }
    ...
}
```

As you can see both synchronous and asynchronous invocation of a service is very simple and type-safe. Still it is very flexible and powerful (see following features). The actual call of `myMethod` will technically call the remote service over the wire (e.g. via HTTP) including marshaling the arguments (e.g. converting `myArgs` to JSON) and unmarshalling the result (e.g. converting the received JSON to `myResult`).

Asynchronous Invocation of void Methods

If you want to call a service method with `void` as return type, the type-safe `call` method can not be used as `void` methods do not return a result. Therefore you can use the `callVoid` method as following:

```
private void callAsynchronousVoid(long id) {
    AsyncServiceClient<MyService> client = this.serviceClientFactory.createAsync
    (MyService.class);
    // call of service over the wire, will return when request is send and invoke
    resultHandler asynchronously
    Consumer<Void> resultHandler = r -> { System.out.println("Response received");};
    client.callVoid(() -> { client.get().deleteEntity(id);}, resultHandler);
}
```

You may also provide `null` as `resultHandler` for "fire and forget". However, this will lead to the result being ignored so even in case of an error you will not be notified.

Configuration

This solution allows a very flexible configuration on the following levels:

1. Global configuration (defaults)
2. Configuration per remote service application (microservice)
3. Configuration per invocation.

A configuration on a deeper level (e.g. 3) overrides the configuration from a higher level (e.g. 1).

The configuration on Level 1 and 2 are configured via `application.properties` (see [configuration guide](#)). For Level 1 the prefix `service.client.default.` is used for properties. Further, for level 2. the prefix `service.client.app.<<application>>.` is used where `<<application>>` is the technical name of the application providing the service. This name will automatically be derived from the java package of the service interface (e.g. `foo` in `MyService` interface before) following our [packaging conventions](#). In case these conventions are not met it will fallback to the fully qualified name of the service interface.

Configuration on Level 3 has to be provided as `Map` argument to the method `ServiceClientFactory.create(Class<S> serviceInterface, Map<String, String> config)`. The keys of this `Map` will not use prefixes (such as the ones above). For common configuration parameters a type-safe builder is offered to create such map via `ServiceClientConfigBuilder`. E.g. for testing you may want to do:

```
this.serviceClientFactory.create(MyService.class,
    new ServiceClientConfigBuilder().authBasic().userLogin(login).userPassword(password)
).buildMap());
```

Here is an example of a configuration block for your `application.properties`:

```
service.client.default.url=https://api.company.com/services/${type}
service.client.default.timeout.connection=120
service.client.default.timeout.response=3600

service.client.app.bar.url=https://bar.company.com:8080/services/rest
service.client.app.bar.auth=basic
service.client.app.bar.user.login=user4711
service.client.app.bar.user.password=ENC(jd5ZREpBqxuN9ok0IhnXabgw7V3EoG2p)

service.client.app.foo.url=https://foo.company.com:8443/services/rest
# authForward: simply forward Authorization header (e.g. with JWT) to remote service
service.client.app.bar.auth=authForward
```

Service Discovery

You do not want to hardwire service URLs in your code, right? Therefore different strategies might apply to *discover* the URL of the invoked service. This is done internally by an implementation of the interface `ServiceDiscoverer`. The default implementation simply reads the base URL from the configuration. So you can simply add this to your `application.properties` as in the above configuration example.

Assuming your service interface would have the fully qualified name `com.company.department.foo.mycomponent.service.api.rest.MyService` then the URL would be resolved to `https://foo.company.com:8443/services/rest` as the `<>application<>` is `foo`.

Additionally, the URL might use the following variables that will automatically be resolved:

- `${app}` to `<>application<>` (useful for default URL)
- `${type}` to the type of the service. E.g. `rest` in case of a `REST` service and `ws` for a `SOAP` service.
- `${local.server.port}` for the port of your current Java servlet container running the JVM. Should only used for testing with spring-boot random port mechanism (technically spring can not resolve this variable but we do it for you here).

Therefore, the default URL may also be configured as:

```
service.client.default.url=https://api.company.com/${app}/services/${type}
```

As you can use any implementation of `ServiceDiscoverer`, you can also easily use `eureka` (or anything else) instead to discover your services. However, we recommend to use `istio` instead as described below.

Headers

A very common demand is to tweak (HTTP) headers in the request to invoke the service. May it be for security (authentication data) or for other cross-cutting concerns (such as the [Correlation ID](#)). This is done internally by implementations of the interface [ServiceHeaderCustomizer](#). We already provide several implementations such as:

- [ServiceHeaderCustomizerBasicAuth](#) for basic authentication (`auth=basic`).
- [ServiceHeaderCustomizerOAuth](#) for OAuth: passes a security token from security context such as a [JWT](#) via OAuth (`auth=oauth`).
- [ServiceHeaderCustomizerAuthForward](#) forwards the [Authorization](#) HTTP header from the running request to the request to the remote service as is (`auth=authForward`). Be careful to avoid security pitfalls by misconfiguring this feature as it may also sensitive credentials (e.g. basic auth) to the remote service. Never use as default.
- [ServiceHeaderCustomizerCorrelationId](#) passed the [Correlation ID](#) to the service request.

Additionally, you can add further custom implementations of [ServiceHeaderCustomizer](#) for your individual requirements and additional headers.

Timeouts

You can configure timeouts in a very flexible way. First of all you can configure timeouts to establish the connection (`timeout.connection`) and to wait for the response (`timeout.response`) separately. These timeouts can be configured on all three levels as described in the configuration section above.

Error Handling

Whilst invoking a remote service an error may occur. This solution will automatically handle such errors and map them to a higher level [ServiceInvocationFailedException](#). In general we separate two different types of errors:

- **Network error**

In such case (host not found, connection refused, time out, etc.) there is not even a response from the server. However, in advance to a low-level exception you will get a wrapped [ServiceInvocationFailedException](#) (with code [ServiceInvoke](#)) with a readable message containing the service that could not be invoked.

- **Service error**

In case the service failed on the server-side the [error result](#) will be parsed and thrown as a [ServiceInvocationFailedException](#) with the received message and code.

This allows to catch and handle errors when a service-invocation failed. You can even distinguish business errors from the server-side from technical errors and implement retry strategies or the like. Further the created exception contains detailed contextual information about the service that failed (service interface class, method, URL) what makes it much easier to trace down errors. Here is an example from our tests:

```
While invoking the service
com.devonfw.test.app.myexample.service.api.rest.MyExampleRestService#businessError[htt
p://localhost:50178/app/services/rest/my-example/v1/business-error] the following
error occurred: Test of business error. Probably the service is temporary unavailable.
Please try again later. If the problem persists contact your system administrator.
2f43b03e-685b-45c0-9aae-23ff4b220c85:BusinessErrorCode
```

You may even provide your own implementation of `ServiceClientErrorHandler` instead to provide an own exception class for this purpose.

Handling Errors

In case of a synchronous service invocation an error will be immediately thrown so you can surround the call with a regular try-catch block:

```
private void callSynchronous(MyArgs myArgs) {
    MyService myService = this.serviceClientFactory.create(MyService.class);
    // call of service over the wire, synchronously blocking until result is received
    or error occurred
    try {
        MyResult myResult = myService.myMethod(myArgs);
        handleResult(myResult);
    } catch (ServiceInvocationFailedException e) {
        if (e.isTechnical()) {
            handleTechnicalError(e);
        } else {
            // error code you defined in the exception on the server side of the service
            String errorCode = e.getCode();
            handleBusinessError(e, errorCode);
        }
    } catch (Throwable e) { // you may not handle this explicitly here...
        handleTechnicalError(e);
    }
}
```

If you are using asynchronous service invocation an error can occur in a separate thread. Therefore you may and should define a custom error handler:

```

private void callAsynchronous(MyArgs myArgs) {
    AsyncServiceClient<MyService> client = this.serviceClientFactory.createAsync
    (MyService.class);
    Consumer<Throwalbe> errorHandler = this::handleError;
    client.setErrorHandler(errorHandler);
    // call of service over the wire, will return when request is send and invoke
    handleResult asynchronously
    client.call(client.get().myMethod(myArgs), this::handleResult);
}

private void handleError(Throwalbe error) {
    ...
}
}

```

The error handler consumes `Throwable` and not only `RuntimeException` so you can get notified even in case of an unexpected `OutOfMemoryError`, `NoClassDefFoundError`, or other technical problems. Please note that the error handler may also be called from the thread calling the service (e.g. if already creating the request fails). The default error handler used if no custom handler is set will only log the error and do nothing else.

Logging

By default this solution will log all invocations including the URL of the invoked service, success or error status flag and the duration in seconds (with decimal nano precision as available). Therefore you can easily monitor the status and performance of the service invocations. Here is an example from our tests:

```

Invoking service
com.devonfw.test.app.myexample.service.api.rest.MyExampleRestService#greet[http://loca
lhost:50178/app/services/rest/my-example/v1/greet/John%20Doe%20%26%20%3F%23] took
PT20.309756622S (20309756622ns) and succeeded with status 200.

```

Resilience

Resilience adds a lot of complexity and that typically means that addressing this here would most probably result in not being up-to-date and not meeting all requirements. Therefore we recommend something completely different: the *sidecar* approach (based on [sidecar pattern](#)). This means that you use a generic proxy app that runs as a separate process on the same host, VM, or container of your actual application. Then in your app you are calling the service via the sidecar proxy on `localhost` (service discovery URL is e.g. `http://localhost:8081/${app}/services/${type}`) that then acts as proxy to the actual remote service. Now aspects such as resilience with circuit breaking and the actual service discovery can be configured in the sidecar proxy app and independent of your actual application. Therefore, you can even share and reuse configuration and experience with such a sidecar proxy app even across different technologies (Java, .NET/C#, Node.JS, etc.). Further, you do not pollute the technology stack of your actual app with the infrastructure for resilience, throttling, etc. and can update the app and the side-card

independently when security-fixes are available.

Various implementations of such sidecar proxy apps are available as free open source software. Our recommendation in devonfw is to use [istio](#). This not only provides such a side-car but also an entire management solution for service-mesh making administration and maintenance much easier. Platforms like OpenShift support this out of the box.

However, if you are looking for details about side-car implementations for services you can have a look at the following links:

- Netflix Sidecar - see [Spring Cloud Netflix docs](#)
- [Envoy](#) - see [Microservices Patterns With Envoy Sidecar Proxy](#)
- [Prana](#) - see [Prana: A Sidecar for your Netflix PaaS based Applications and Services](#) ← **Not updated as it's not used internally by Netflix**
- Keycloak - see [Protecting Jaeger UI with a sidecar security proxy](#)

12.23. Testing

12.23.1. General best practices

For testing please follow our general best practices:

- Tests should have a clear goal that should also be documented.
- Tests have to be classified into different [integration levels](#).
- Tests should follow a clear naming convention.
- Automated tests need to properly assert the result of the tested operation(s) in a reliable way.
E.g. avoid stuff like `assertThat(service.getAllEntities()).hasSize(42)` or even worse tests that have no assertion at all.
- Tests need to be independent of each other. Never write test-cases or tests (in Java @Test methods) that depend on another test to be executed before.
- Use [AssertJ](#) to write good readable and maintainable tests that also provide valuable feedback in case a test fails. Do not use legacy JUnit methods like `assertEquals` anymore!
- For easy understanding divide your test in three commented sections:
 - `//given`
 - `//when`
 - `//then`
- Plan your tests and test data management properly before implementing.
- Instead of having a too strong focus on test coverage better ensure you have covered your critical core functionality properly and review the code including tests.
- Test code shall NOT be seen as second class code. You shall consider design, architecture and code-style also for your test code but do not over-engineer it.
- Test automation is good but should be considered in relation to cost per use. Creating full coverage via *automated system tests* can cause a massive amount of test-code that can turn out as a huge maintenance hell. Always consider all aspects including product life-cycle, criticality of use-cases to test, and variability of the aspect to test (e.g. UI, test-data).
- Use continuous integration and establish that the entire team wants to have clean builds and running tests.
- Prefer delegation over inheritance for cross-cutting testing functionality. Good places to put this kind of code can be realized and reused via the JUnit @Rule mechanism.

12.23.2. Test Automation Technology Stack

For test automation we use [JUnit](#). However, we are strictly doing all assertions with [AssertJ](#). For [mocking](#) we use [mockito](#). In order to mock remote connections we use [wiremock](#). For testing entire components or sub-systems we recommend to use [spring-boot-starter-test](#) as lightweight and fast testing infrastructure that is already shipped with [devon4j-test](#).

In case you have to use a full blown JEE application server, we recommend to use [arquillian](#). To get started with arquillian, look [here](#).

12.23.3. Test Doubles

We use [test doubles](#) as generic term for mocks, stubs, fakes, dummies, or spys to avoid confusion. Here is a short summary from [stubs VS mocks](#):

- **Dummy** objects specifying no logic at all. May declare data in a POJO style to be used as boiler plate code to parameter lists or even influence the control flow towards the test's needs.
- **Fake** objects actually have working implementations, but usually take some shortcut which makes them not suitable for production (an in memory database is a good example).
- **Stubs** provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test. Stubs may also record information about calls, such as an email gateway stub that remembers the messages it 'sent', or maybe only how many messages it 'sent'.
- **Mocks** are objects pre-programmed with expectations, which form a specification of the calls they are expected to receive.

We try to give some examples, which should make it somehow clearer:

Stubs

Best Practices for applications:

- A good way to replace small to medium large boundary systems, whose impact (e.g. latency) should be ignored during load and performance tests of the application under development.
- As stub implementation will rely on state-based verification, there is the threat, that test developers will partially reimplement the state transitions based on the replaced code. This will immediately lead to a black maintenance whole, so better use mocks to assure the certain behavior on interface level.
- Do NOT use stubs as basis of a large amount of test cases as due to state-based verification of stubs, test developers will enrich the stub implementation to become a large monster with its own hunger after maintenance efforts.

Mocks

Best Practices for applications:

- Replace not-needed dependencies of your system-under-test (SUT) to minimize the application context to start of your component framework.
- Replace dependencies of your SUT to impact the control flow under test without establishing all the context parameters needed to match the control flow.
- Remember: Not everything has to be mocked! Especially on lower levels of tests like isolated module tests you can be betrayed into a mocking delusion, where you end up in a hundred lines of code mocking the whole context and five lines executing the test and verifying the mocks behavior. Always keep in mind the benefit-cost ratio, when implementing tests using mocks.

Wiremock

If you need to mock remote connections such as HTTP-Servers, wiremock offers easy to use functionality. For a full description see the [homepage](#) or the [github repository](#). Wiremock can be used either as a JUnit Rule, in Java outside of JUnit or as a standalone process. The mocked server can be configured to respond to specific requests in a given way via a fluent Java API, JSON files and JSON over HTTP. An example as an integration to JUnit can look as follows.

```
import static
com.github.tomakehurst.wiremock.core.WireMockConfiguration.wireMockConfig;
import com.github.tomakehurst.wiremock.junit.WireMockRule;

public class WireMockOfferImport{

    @Rule
    public WireMockRule mockServer = new WireMockRule(wireMockConfig().dynamicPort());

    @Test
    public void requestDataTest() throws Exception {
        int port = this.mockServer.port();
        ...
    }
}
```

This creates a server on a randomly chosen free port on the running machine. You can also specify the port to be used if wanted. Other than that there are several options to further configure the server. This includes HTTPs, proxy settings, file locations, logging and extensions.

```
@Test
public void requestDataTest() throws Exception {
    this.mockServer.stubFor(get(urlEqualTo("/new/offers")).withHeader("Accept",
equalTo("application/json"))
    .withHeader("Authorization", containing("Basic")).willReturn(aResponse()
    .withStatus(200).withFixedDelay(1000)
    .withHeader("Content-Type", "application/json").withBodyFile(
"/wireMockTest/jsonBodyFile.json")));
}
```

This will stub the URL `localhost:port/new/offers` to respond with a status 200 message containing a header (`Content-Type: application/json`) and a body with content given in `jsonBodyFile.json` if the request matches several conditions. It has to be a GET request to `../new/offers` with the two given header properties.

Note that by default files are located in `src/test/resources/_files/`. When using only one WireMock server one can omit the `this.mockServer` in before the `stubFor` call (static method). You can also add a fixed delay to the response or processing delay with `WireMock.addRequestProcessingDelay(time)` in order to test for timeouts.

WireMock can also respond with different corrupted messages to simulate faulty behaviour.

```
@Test(expected = ResourceAccessException.class)
public void faultTest() {

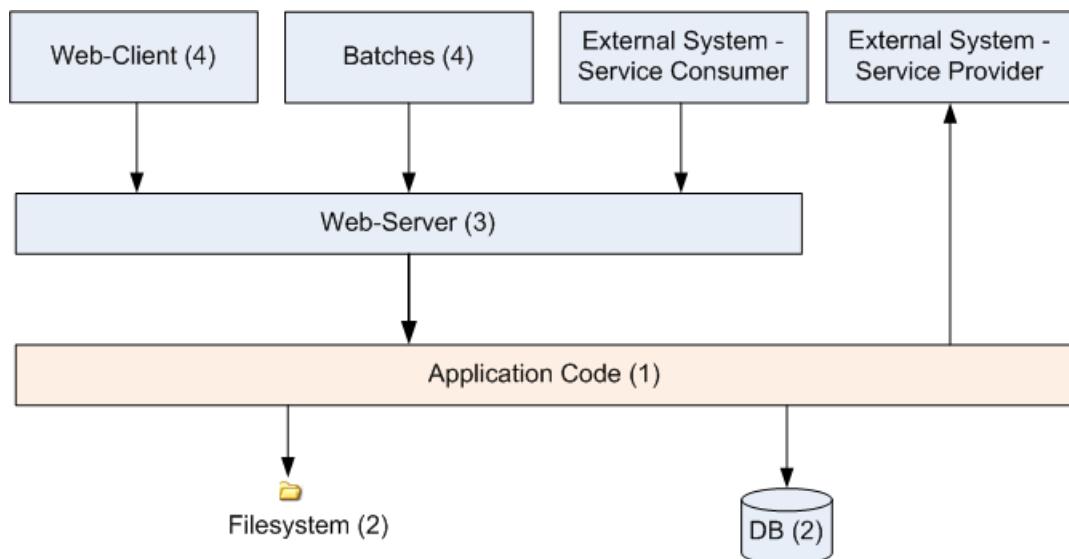
    this.mockServer.stubFor(get(urlEqualTo("/fault")).willReturn(aResponse()
        .withFault(Fault.MALFORMED_RESPONSE_CHUNK)));
...
}
```

A GET request to `.../fault` returns an OK status header, then garbage, and then closes the connection.

12.23.4. Integration Levels

There are many discussions about the right level of integration for test automation. Sometimes it is better to focus on small, isolated modules of the system - whatever a "module" may be. In other cases it makes more sense to test integrated groups of modules. Because there is no universal answer to this question, devonfw only defines a common terminology for what could be tested. Each project must make its own decision where to put the focus of test automation. There is no worldwide accepted terminology for the integration levels of testing. In general we consider [ISTQB](#). However, with a technical focus on test automation we want to get more precise.

The following picture shows a simplified view of an application based on the [devonfw reference architecture](#). We define four integration levels that are explained in detail below. The boxes in the picture contain parenthesized numbers. These numbers depict the lowest integration level, a box belongs to. Higher integration levels also contain all boxes of lower integration levels. When writing tests for a given integration level, related boxes with a lower integration level must be replaced by test [doubles](#) or drivers.



The main difference between the integration levels is the amount of infrastructure needed to test them. The more infrastructure you need, the more bugs you will find, but the more unstable and the slower your tests will be. So each project has to make a trade-off between pros and cons of including much infrastructure in tests and has to select the integration levels that fit best to the project.

Consider, that more infrastructure does not automatically lead to a better bug-detection. There may

be bugs in your software that are masked by bugs in the infrastructure. The best way to find those bugs is to test with very few infrastructure.

External systems do not belong to any of the integration levels defined here. devonfw does not recommend involving real external systems in test automation. This means, they have to be replaced by test [doubles](#) in automated tests. An exception may be external systems that are fully under control of the own development team.

The following chapters describe the four integration levels.

Level 1 Module Test

The goal of a *isolated module test* is to provide fast feedback to the developer. Consequently, isolated module tests must not have any interaction with the client, the database, the file system, the network, etc.

An isolated module test is testing a single classes or at least a small set of classes in isolation. If such classes depend on other components or external resources, etc. these shall be replaced with a [test double](#).

```
public class MyClassTest extends ModuleTest {

    @Test
    public void testMyClass() {

        // given
        MyClass myClass = new MyClass();
        // when
        String value = myClass.doSomething();
        // then
        assertThat(value).isEqualTo("expected value");
    }

}
```

For an advanced example see [here](#).

Level 2 Component Test

A *component test* aims to test components or component parts as a unit. These tests typically run with a (light-weight) infrastructure such as spring-boot-starter-test and can access resources such as a database (e.g. for DAO tests). Further, no remote communication is intended here. Access to external systems shall be replaced by a [test double](#).

With devon4j and spring you can write a component-test as easy as illustrated in the following example:

```

@SpringBootTest(classes = { MySpringBootApp.class }, webEnvironment = WebEnvironment
.NONE)
public class UcFindCountryTest extends ComponentTest {
    @Inject
    private UcFindCountry ucFindCountry;

    @Test
    public void testFindCountry() {

        // given
        String countryCode = "de";

        // when
        TestUtil.login("user", MyAccessControlConfig.FIND_COUNTRY);
        CountryEto country = this.ucFindCountry.findCountry(countryCode);

        // then
        assertThat(country).isNotNull();
        assertThat(country.getCountryCode()).isEqualTo(countryCode);
        assertThat(country.getName()).isEqualTo("Germany");
    }
}

```

This test will start the entire spring-context of your app ([MySpringBootApp](#)). Within the test spring will inject according spring-beans into all your fields annotated with `@Inject`. In the test methods you can use these spring-beans and perform your actual tests. This pattern can be used for testing DAOs/Repositories, Use-Cases, or any other spring-bean with its entire configuration including database and transactions.

When you are testing use-cases your `authorization` will also be in place. Therefore, you have to simulate a logon in advance what is done via the `login` method in the above example. The test-infrastructure will automatically do a `logout` for you after each test method in `doTearDown`.

Level 3 Subsystem Test

A *subsystem test* runs against the external interfaces (e.g. HTTP service) of the integrated subsystem. Subsystem tests of the client subsystem are described in the [devon4ng testing guide](#). In devon4j the server (JEE application) is the subsystem under test. The tests act as a client (e.g. service consumer) and the server has to be integrated and started in a container.

With devon4j and spring you can write a subsystem-test as easy as illustrated in the following example:

```

@SpringBootTest(classes = { MySpringBootApp.class }, webEnvironment = WebEnvironment
.RANDOM_PORT)
public class CountryRestServiceTest extends SubsystemTest {

    @Inject
    private ServiceClientFactory serviceClientFactory;

    @Test
    public void testFindCountry() {

        // given
        String countryCode = "de";

        // when
        CountryRestService service = this.serviceClientFactory.create(CountryRestService
.class);
        CountryEto country = service.findCountry(countryCode);

        // then
        assertThat(country).isNotNull();
        assertThat(country.getCountryCode()).isEqualTo(countryCode);
        assertThat(country.getName()).isEqualTo("Germany");
    }
}

```

Even though not obvious on the first look this test will start your entire application as a server on a free random port (so that it works in CI with parallel builds for different branches) and tests the invocation of a (REST) service including (un)marshalling of data (e.g. as JSON) and transport via HTTP (all in the invocation of the `findCountry` method).

Do not confuse a *subsystem test* with a [system integration test](#). A system integration test validates the interaction of several systems where we do not recommend test automation.

Level 4 System Test

A [system test](#) has the goal to test the system as a whole against its official interfaces such as its UI or batches. The system itself runs as a separate process in a way close to a regular deployment. Only external systems are simulated by [test doubles](#).

The devonfw only gives advice for automated system test (TODO see allure testing framework). In nearly every project there must be manual system tests, too. This manual system tests are out of scope here.

Classifying Integration-Levels

devon4j defines [Category-Interfaces](#) that shall be used as [JUnit Categories](#). Also devon4j provides [abstract base classes](#) that you may extend in your test-cases if you like.

devon4j further pre-configures the maven build to only run integration levels 1-2 by default (e.g. for

fast feedback in continuous integration). It offers the profiles subsystemtest (1-3) and systemtest (1-4). In your nightly build you can simply add -Psystemtest to run all tests.

12.23.5. Implementation

This section introduces how to implement tests on the different levels with the given devonfw infrastructure and the proposed frameworks.

Module Test

In devon4j you can extend the abstract class `ModuleTest` to basically get access to assertions. In order to test classes embedded in dependencies and external services one needs to provide mocks for that. As the [technology stack](#) recommends we use the Mockito framework to offer this functionality. The following example shows how to implement Mockito into a JUnit test.

```
import static org.mockito.Mockito.when;
import static org.mockito.Mockito.mock;
...

public class StaffmanagementImplTest extends ModuleTest {
    @Rule
    public MockitoRule rule = MockitoJUnit.rule();

    @Test
    public void testFindStaffMember() {
        ...
    }
}
```

Note that the test class does not use the `@SpringApplicationConfiguration` annotation. In a module test one does not use the whole application. The JUnit rule is the best solution to use in order to get all needed functionality of Mockito. Static imports are a convenient option to enhance readability within Mockito tests. You can define mocks with the `@Mock` annotation or the `mock(*.class)` call. To inject the mocked objects into your class under test you can use the `@InjectMocks` annotation. This automatically uses the setters of `StaffmanagementImpl` to inject the defined mocks into the *class under test (CUT)* when there is a setter available. In this case the `beanMapper` and the `staffMemberDao` are injected. Of course it is possible to do this manually if you need more control.

```
@Mock
private BeanMapper beanMapper;
@Mock
private StaffMemberEntity staffMemberEntity;
@Mock
private StaffMemberEto staffMemberEto;
@Mock
private StaffMemberDao staffMemberDao;
@InjectMocks
StaffmanagementImpl staffmanagementImpl = new StaffmanagementImpl();
```

The mocked objects do not provide any functionality at the time being. To define what happens on a method call on a mocked dependency in the CUT one can use `when(condition).thenReturn(result)`. In this case we want to test `findStaffMember(Long id)` in the `StaffmanagementImpl`.

```
public StaffMemberEto findStaffMember(Long id) {
    return getBeanMapper().map(getStaffMemberDao().find(id), StaffMemberEto.class);
}
```

In this simple example one has to stub two calls on the CUT as you can see below. For example the method call of the CUT `staffMemberDao.find(id)` is stubbed for returning a mock object `staffMemberEntity` that is also defined as mock.

Subsystem Test

devon4j provides a simple test infrastructure to aid with the implementation of subsystem tests. It becomes available by simply subclassing `AbstractRestServiceTest.java`.

```
//given
long id = 1L;
Class<StaffMemberEto> targetClass = StaffMemberEto.class;
when(this.staffMemberDao.find(id)).thenReturn(this.staffMemberEntity);
when(this.beanMapper.map(this.staffMemberEntity, targetClass)).thenReturn(this
    .staffMemberEto);

//when
StaffMemberEto resultEto = this.staffmanagementImpl.findStaffMember(id);

//then
assertThat(resultEto).isNotNull();
assertThat(resultEto).isEqualTo(this.staffMemberEto);
```

After the test method call one can verify the expected results. Mockito can check whether a mocked method call was indeed called. This can be done using Mockito `verify`. Note that it does not generate any value if you check for method calls that are needed to reach the asserted result anyway. Call verification can be useful e.g. when you want to assure that statistics are written out without actually testing them.

12.23.6. Regression testing

When it comes to complex output (even binary) that you want to regression test by comparing with an expected result, you could consider [Approval Tests](#) using `ApprovalTests.java`. If applied for the right problems, it can be very helpful.

12.23.7. Deployment Pipeline

A deployment pipeline is a semi-automated process that gets software-changes from version control into production. It contains several validation steps, e.g. automated tests of all integration levels.

Because devon4j should fit to different project types - from agile to waterfall - it does not define a standard deployment pipeline. But we recommend to define such a deployment pipeline explicitly for each project and to find the right place in it for each type of test.

For that purpose, it is advisable to have fast running test suite that gives as much confidence as possible without needing too much time and too much infrastructure. This test suite should run in an early stage of your deployment pipeline. Maybe the developer should run it even before he/she checked in the code. Usually lower integration levels are more suitable for this test suite than higher integration levels.

Note, that the deployment pipeline always should contain manual validation steps, at least manual acceptance testing. There also may be manual validation steps that have to be executed for special changes only, e.g. usability testing. Management and execution processes of those manual validation steps are currently not in the scope of devonfw.

12.23.8. Test Coverage

We are using tools (SonarQube/Jacoco) to measure the coverage of the tests. Please always keep in mind that the only reliable message of a code coverage of X% is that (100-X)% of the code is entirely untested. It does not say anything about the quality of the tests or the software though it often relates to it.

12.23.9. Test Configuration

This section covers test configuration in general without focusing on integration levels as in the first chapter.

Configure Test Specific Beans

Sometimes it can become handy to provide other or differently configured bean implementations via CDI than those available in production. For example, when creating beans using `@Bean`-annotated methods they are usually configured within those methods. [WebSecurityBeansConfig](#) shows an example of such methods.

```
@Configuration
public class WebSecurityBeansConfig {
    //...
    @Bean
    public AccessControlSchemaProvider accessControlSchemaProvider() {
        // actually no additional configuration is shown here
        return new AccessControlSchemaProviderImpl();
    }
    //...
}
```

`AccessControlSchemaProvider` allows to programmatically access data defined in some XML file, e.g. `access-control-schema.xml`. Now, one can imagine that it would be helpful if `AccessControlSchemaProvider` would point to some other file than the default within a test class. That

file could provide content that differs from the default. The question is: how can I change resource path of `AccessControlSchemaProviderImpl` within a test?

One very helpful solution is to use **static inner classes**. Static inner classes can contain `@Bean`-annotated methods, and by placing them in the `classes` parameter in `@SpringBootTest(classes = { /* place class here */ })` annotation the beans returned by these methods are placed in the application context during test execution. Combining this feature with inheritance allows to override methods defined in other configuration classes as shown in the following listing where `TempWebSecurityConfig` extends `WebSecurityBeansConfig`. This relationship allows to override `public AccessControlSchemaProvider accessControlSchemaProvider()`. Here we are able to configure the instance of type `AccessControlSchemaProviderImpl` before returning it (and, of course, we could also have used a completely different implementation of the `AccessControlSchemaProvider` interface). By overriding the method the implementation of the super class is ignored, hence, only the new implementation is called at runtime. Other methods defined in `WebSecurityBeansConfig` which are not overridden by the subclass are still dispatched to `WebSecurityBeansConfig`.

```
//... Other testing related annotations
@SpringBootTest(classes = { TempWebSecurityConfig.class })
public class SomeTestClass {

    public static class TempWebSecurityConfig extends WebSecurityBeansConfig {

        @Override
        @Bean
        public AccessControlSchemaProvider accessControlSchemaProvider() {

            ClassPathResource resource = new ClassPathResource(locationPrefix + "access-
control-schema3.xml");
            AccessControlSchemaProviderImpl accessControlSchemaProvider = new
            AccessControlSchemaProviderImpl();
            accessControlSchemaProvider.setAccessControlSchema(resource);
            return accessControlSchemaProvider;
        }
    }
}
```

The following [chapter of the Spring framework documentation](#) explains issue, but uses a slightly different way to obtain the configuration.

Test Data

It is possible to obtain test data in two different ways depending on your test's integration level.

12.23.10. Debugging Tests

The following two sections describe two debugging approaches for tests. Tests are either run from within the IDE or from the command line using Maven.

Debugging with the IDE

Debugging with the IDE is as easy as always. Even if you want to execute a **SubsystemTest** which needs a Spring context and a server infrastructure to run properly, you just set your breakpoints and click on Debug As → JUnit Test. The test infrastructure will take care of initializing the necessary infrastructure - if everything is configured properly.

Debugging with Maven

Please refer to the following two links to find a guide for debugging tests when running them from Maven.

- <http://maven.apache.org/surefire/maven-surefire-plugin/examples/debugging.html>
- <https://www.eclipse.org/jetty/documentation/9.3.x/debugging-with-eclipse.html>

In essence, you first have to start execute a test using the command line. Maven will halt just before the test execution and wait for your IDE to connect to the process. When receiving a connection the test will start and then pause at any breakpoint set in advance. The first link states that tests are started through the following command:

```
mvn -Dmaven.surefire.debug test
```

Although this is correct, it will run *every* test class in your project and - which is time consuming and mostly unnecessary - halt before each of these tests. To counter this problem you can simply execute a single test class through the following command (here we execute the **TablemanagementRestServiceTest** from the restaurant sample application):

```
mvn test -Dmaven.surefire.debug test -Dtest=TablemanagementRestServiceTest
```

It is important to notice that you first have to execute the Maven command in the according submodule, e.g. to execute the **TablemanagementRestServiceTest** you have first to navigate to the core module's directory.

12.24. Transfer-Objects

The technical data model is defined in form of [persistent entities](#). However, passing persistent entities via *call-by-reference* across the entire application will soon cause problems:

- Changes to a persistent entity are directly written back to the persistent store when the transaction is committed. When the entity is send across the application also changes tend to take place in multiple places endangering data sovereignty and leading to inconsistency.
- You want to send and receive data via services across the network and have to define what section of your data is actually transferred. If you have relations in your technical model you quickly end up loading and transferring way too much data.
- Modifications to your technical data model shall not automatically have impact on your external services causing incompatibilities.

To prevent such problems transfer-objects are used leading to a *call-by-value* model and decoupling changes to persistent entities.

In the following sections the different types of transfer-objects are explained. You will find all according naming-conventions in the [architecture-mapping](#)

12.24.1. ETO

For each [persistent entity](#) `<<BusinessObject>>Entity` we create or generate a corresponding *entity transfer object* (ETO) named `<<BusinessObject>>Eto`. It has the same properties except for relations.

12.24.2. BO

In order to centralize the properties (getters and setters with their javadoc) we create a common interface `<<BusinessObject>>` implemented both by the entity and its ETO. This also gives us compile-time safety that [bean-mapper](#) can properly map all properties between entity and ETO.

12.24.3. CTO

If we need to pass an entity with its relation(s) we create a corresponding *composite transfer object* (CTO) named `<<BusinessObject>><<Subset>>Cto` that only contains other transfer-objects or collections of them. Here `<<Subset>>` is empty for the canonical CTO that holds the ETO together with all its relations. This is what can be generated automatically with [CobiGen](#). However, be careful to generate CTOs without thinking and considering design. If there are no relations at all a CTO is pointless and shall be omitted. However, if there are multiple relations you typically need multiple CTOs for the same `<<BusinessObject>>` that define different subsets of the related data. These will typically be designed and implemented by hand. E.g. you may have `CustomerWithAddressCto` and `CustomerWithContractCto`. Most CTOs correspond to a specific `<<BusinessObject>>` and therefore contain a `<<BusinessObject>>Eto`. Such CTOs should inherit from `MasterCto`.

This pattern with entities, ETOs and CTOs is illustrated by the following UML diagram from our sample application.



Figure 8. ETOs and CTOs

12.24.4. TO

Finally, there are typically transfer-objects for data that is never persistent. For very generic cases these just carry the suffix **To**.

12.24.5. SearchCriteriaTo

For searching we create or generate a `<<BusinessObject>>SearchCriteriaTo` representing a query to find instances of `<<BusinessObject>>`.

12.24.6. STO

We can potentially create separate *service transfer objects* (STO) (if possible named `<<BusinessObject>>Sto`) to keep the `service` API stable and independent of the actual data-model. However, we usually do not need this and want to keep our architecture simple. Only create STOs if you need `service versioning` and support previous APIs or to provide legacy service technologies that require their own isolated data-model. In such case you also need `beanmapping` between STOs and ETOs what means extra effort and complexity that should be avoided.

12.25. Bean-Mapping

For decoupling you sometimes need to create separate objects (beans) for a different view. E.g. for an external service you will use a [transfer-object](#) instead of the [persistence entity](#) so internal changes to the entity do not implicitly change or break the service.

Therefore you have the need to map similar objects what creates a copy. This also has the benefit that modifications to the copy have no side-effect on the original source object. However, to implement such mapping code by hand is very tedious and error-prone (if new properties are added to beans but not to mapping code):

```
public UserEto mapUser(UserEntity source) {
    UserEto target = new UserEto();
    target.setUsername(source.getUsername());
    target.setEmail(source.getEmail());
    ...
    return target;
}
```

Therefore we are using a [BeanMapper](#) for this purpose that makes our lives a lot easier.

12.25.1. Bean-Mapper Dependency

To get access to the [BeanMapper](#) we have to use either of below dependency in our POM:

We started with [dozer.sourceforge.net/\[dozer\]](#) in devon4j and still support it. However, we now recommend [orika](#) (for new projects) as it is much faster (see [Performance of Java Mapping Frameworks](#)).

```
<dependency>
    <groupId>com.devonfw.java</groupId>
    <artifactId>devon4j-beanmapping-orika</artifactId>
</dependency>
```

or

```
<dependency>
    <groupId>com.devonfw.java</groupId>
    <artifactId>devon4j-beanmapping-dozer</artifactId>
</dependency>
```

12.25.2. Bean-Mapper Configuration using Dozer

The [BeanMapper](#) implementation is based on an existing open-source bean mapping framework. In case of Dozer the mapping is configured [src/main/resources/config/app/common/dozer-mapping.xml](#).

See the my-thai-star `dozer-mapping.xml` as an example. Important is that you configure all your custom datatypes as `<copy-by-reference>` tags and have the mapping from `PersistenceEntity` (`ApplicationPersistenceEntity`) to `AbstractEto` configured properly:

```
<mapping type="one-way">
    <class-a>com.devonfw.module.basic.common.api.entity.PersistenceEntity</class-a>
    <class-b>com.devonfw.module.basic.common.api.to.AbstractEto</class-b>
    <field custom-converter=
"com.devonfw.module.beanmapping.common.impl.dozer.IdentityConverter">
        <a>this</a>
        <b is-accessible="true">persistentEntity</b>
    </field>
</mapping>
```

12.25.3. Bean-Mapper Usage

Then we can get the `BeanMapper` via `dependency-injection` what we typically already provide by an abstract base class (e.g. `AbstractUc`). Now we can solve our problem very easy:

```
...
UserEntity resultEntity = ...;
...
return getBeanMapper().map(resultEntity, UserEto.class);
```

There is also additional support for mapping entire collections.

12.26. Datatypes

A datatype is an object representing a value of a specific type with the following aspects:

- It has a technical or business specific semantic.
- Its JavaDoc explains the meaning and semantic of the value.
- It is immutable and therefore stateless (its value assigned at construction time and can not be modified).
- It is serializable.
- It properly implements `#equals(Object)` and `#hashCode()` (two different instances with the same value are equal and have the same hash).
- It shall ensure syntactical validation so it is NOT possible to create an instance with an invalid value.
- It is responsible for formatting its value to a string representation suitable for sinks such as UI, loggers, etc. Also consider cases like a Datatype representing a password where `toString()` should return something like "****" instead of the actual password to prevent security accidents.
- It is responsible for parsing the value from other representations such as a string (as needed).
- It shall provide required logical operations on the value to prevent redundancies. Due to the immutable attribute all manipulative operations have to return a new Datatype instance (see e.g. `BigDecimal.add(java.math.BigDecimal)`).
- It should implement Comparable if a natural order is defined.

Based on the Datatype a presentation layer can decide how to view and how to edit the value. Therefore a structured data model should make use of custom datatypes in order to be expressive. Common generic datatypes are String, Boolean, Number and its subclasses, Currency, etc. Please note that both Date and Calendar are mutable and have very confusing APIs. Therefore, use JSR-310 or jodatime instead. Even if a datatype is technically nothing but a String or a Number but logically something special it is worth to define it as a dedicated datatype class already for the purpose of having a central javadoc to explain it. On the other side avoid to introduce technical datatypes like String32 for a String with a maximum length of 32 characters

as this is not adding value in the sense of a real Datatype. It is suitable and in most cases also recommended to use the class implementing the datatype as API omitting a dedicated interface.

— mmm project, datatype javadoc

See [mmm datatype javadoc](#).

12.26.1. Datatype Packaging

For the devonfw we use a common [packaging schema](#). The specifics for datatypes are as following:

Segment	Value	Explanation
<component>	*	Here we use the (business) component defining the datatype or general for generic datatypes.
<layer>	common	Datatypes are used across all layers and are not assigned to a dedicated layer.
<scope>	api	Datatypes are always used directly as API even though they may contain (simple) implementation logic. Most datatypes are simple wrappers for generic Java types (e.g. String) but make these explicit and might add some validation.

12.26.2. Technical Concerns

Many technologies like Dozer and QueryDSL's (alias API) are heavily based on reflection. For them to work properly with custom datatypes, the frameworks must be able to instantiate custom datatypes with no-argument constructors. It is therefore recommended to implement a no-argument constructor for each datatype of at least protected visibility.

12.26.3. Datatypes in Entities

The usage of custom datatypes in entities is explained in the [persistence layer guide](#).

12.26.4. Datatypes in Transfer-Objects

XML

For mapping datatypes with JAXB see [XML guide](#).

JSON

For mapping datatypes from and to JSON see [JSON custom mapping](#).

12.27. CORS support

When you are developing Javascript client and server application separately, you have to deal with cross domain issues. We have to request from a origin domain distinct to target domain and browser does not allow this.

So , we need to prepare server side to accept request from other domains. We need to cover the following points:

- Accept request from other domains.
- Accept devonfw used headers like `X-CSRF-TOKEN` or `correlationId`.
- Be prepared to receive secured request (cookies).

It is important to note that if you are using security in your request (sending cookies) you have to set `withCredentials` flag to `true` in your client side request and deal with special IE8 characteristics.

12.27.1. Configuring CORS support

Dependency

To enable the CORS support from the server side add the below dependency.

```
<dependency>
    <groupId>com.devonfw.java.starters</groupId>
    <artifactId>devon4j-starter-security-cors</artifactId>
</dependency>
```

Configuration

Add the below properties in your application.properties file.

```
#CORS support
security.cors.spring.allowCredentials=true
security.cors.spring.allowedOrigins=*
security.cors.spring.allowedHeaders=*
security.cors.spring.allowedMethods=OPTIONS,HEAD,GET,PUT,POST,DELETE,PATCH
security.cors.pathPattern=/**
```

Attribute	Description	HTTP Header
allowCredentials	Decides the browser should include any cookies associated with the request (<code>true</code> if cookies should be included).	Access-Control-Allow-Credentials
allowedOrigins	List of allowed origins (use <code>*</code> to allow all orgins).	Access-Control-Allow-Origin

Attribute	Description	HTTP Header
allowedMethods	List of allowed HTTP request methods (OPTIONS , HEAD , GET , PUT , POST , DELETE , PATCH , etc.).	-
allowedHeaders	List of allowed headers that can be used during the request (use * to allow all headers requested by the client)	Access-Control-Allow-Headers
pathPattern	Ant-style pattern for the URL paths where to apply CORS. Use "/**" to match all URL paths.	

More information about the CORS headers can be found [here](#)

12.28. BLOB support

BLOB stands for **B**inary **L**arge **O**bject. A BLOB may be an image, an office document, ZIP archive or any other multimedia object. Often these BLOBS are large. if this is the case you need to take care, that you do not copy all the blob data into you application heap, e.g. when providing them via a REST service. This could easily lead to performance problems or out of memory errors. As solution for that problem is "streaming" those BLOBS directly from the database to the client. To demonstrate how this can be accomplished, devonfw provides a [example](#).

12.28.1. Further Reading

- [BLOBs and the Data Access Layer](#)
- [Security Vulnerability Unrestricted File Upload](#)

12.29. Java Development Kit

The [Java Development Kit](#) is an implementation of the Java platform. It provides the [Java Virtual Machine](#) (JVM) and the Java Runtime Environment (JRE).

12.29.1. Editions

The JDK exists in different editions:

- [OpenJDK](#) is a free and open-source edition of the JDK.
- [OracleJDK](#) is a commercial edition of the JDK.
- [Various alternative JDK editions](#) either commercial (e.g. IBM's JVM) or open-source.

As Java is evolving and also complex maintaining a JVM requires a lot of energy. Therefore many alternative JDK editions are unable to cope with this and support latest Java versions and according compatibility. Unfortunately OpenJDK only maintains a specific version of Java for a relative short period of time before moving to the next major version. In the end, this technically means that OpenJDK is continuous beta and can not be used in production for reasonable software projects. As OracleJDK changed its licensing model and can not be used for commercial usage even during development, things can get tricky. You may want to use OpenJDK for development and OracleJDK only in production. However, e.g. OpenJDK 11 never released a version that is stable enough for reasonable development (e.g. javadoc tool is [broken](#) and fixes are not available of OpenJDK 11 - fixed in 11.0.3 what is only available as OracleJDK 11 or you need to go to OpenJDK 12+, what has other bugs) so in the end there is no working release of OpenJDK 11. This more or less forces you to use OracleJDK what requires you to buy a subscription so you can use it for commercial development. However, there is [AdoptOpenJDK](#) that provides forked releases of OpenJDK with bug-fixes what might be an option. Anyhow, as you want to have your development environment close to production, the productively used JDK (most likely OracleJDK) should be preferred also for development.

12.29.2. Upgrading

Until Java 8 compatibility was one of the key aspects for Java version updates (after the mess on the Swing updates with Java2 many years ago). However, Java 9 introduced a lot of breaking changes. This documentation wants to share the experience we collected in devonfw when upgrading from Java 8 to newer versions. First of all we separate runtime changes that you need if you want to build your software with JDK 8 but such that it can also run on newer versions (e.g. JRE 11) from changes required to also build your software with more recent JDKs (e.g. JDK 11 or 12).

Runtime Changes

This section describes required changes to your software in order to make it run also with versions newer than Java 8.

Classes removed from JDK

The first thing that most users hit when running their software with newer Java versions is a [ClassNotFoundException](#) like this:

Caused by: java.lang.ClassNotFoundException: javax.xml.bind.JAXBException

As Java 9 introduced a module system with [Jigsaw](#), the JDK that has been a monolithic mess is now a well-defined set of structured modules. Some of the classes that used to come with the JDK moved to modules that were not available by default in Java 9 and have even been removed entirely in later versions of Java. Therefore you should simply treat such code just like any other 3rd party component that you can add as a (maven) dependency. The following table gives you the required hints to make your software work even with such classes / modules removed from the JDK (please note that the specified version is just a suggestion that worked, feel free to pick a more recent or more appropriate version):

Table 36. Dependencies for classes removed from Java 8 since 9+

Class	GroupId	ArtifactId	Version
javax.xml.bind.*	javax.xml.bind	jaxb-api	2.3.1
com.sun.xml.bind.*	org.glassfish.jaxb	jaxb-runtime	2.3.1
java.activation.*	javax.activation	javax.activation-api	1.2.0
java.transaction.*	javax.transaction	javax.transaction-api	1.2
java.xml.ws.*	javax.xml.ws	jaxws-api	2.3.1
javax.jws.*	javax.jws	javax.jws-api	1.1
javax.annotation.*	javax.annotation	javax.annotation-api	1.3.2

3rd Party Updates

Further, internal and unofficial APIs (e.g. [sun.misc.Unsafe](#)) have been removed. These are typically not used by your software directly but by low-level 3rd party libraries like [asm](#) that need to be updated. Also simple things like the Java version have changed (from [1.8.x](#) to [9.x](#), [10.x](#), [11.x](#), [12.x](#), etc.). Some 3rd party libraries were parsing the Java version in a very naive way making them unable to be used with Java 9+:

Caused by: java.lang.NullPointerException
at
org.apache.maven.surefire.shade.org.apache.commons.lang3.SystemUtils.isJavaVersionAtLeast (SystemUtils.java:1626)

Therefore the following table gives an overview of common 3rd party libraries that have been affected by such breaking changes and need to be updated to at least the specified version:

Table 37. Minimum recommended versions of common 3rd party for Java 9+

GroupId	ArtifactId	Version	Issue
org.apache.commons	commons-lang3	3.7	LANG-1365
cglib	cglib	3.2.9	102 , 93 , 133
org.ow2.asm	asm	7.1	2941

GroupId	ArtifactId	Version	Issue
org.javassist	javassist	3.25.0-GA	194 , 228 , 246 , 171

ResourceBundles

For internationalization (i18n) and localization (l10n) [ResourceBundle](#) is used for language and country specific texts and configurations as properties (e.g. `MyResourceBundle_de.properties`). With Java modules there are changes and impacts you need to know to get things working. The most important change is documented in the [JavaDoc of ResourceBundle](#). However, instead of using [ResourceBundleProvider](#) and refactoring your entire code causing incompatibilities, you can simply put the resource bundles in a regular JAR on the classpath rather than a named module (or into the launching app). If you want to implement (new) Java modules with i18n support, you can have a look at [mmm-nls](#).

Buildtime Changes

If you also want to change your build to work with a recent JDK you also need to ensure that test frameworks and maven plugins properly support this.

Findbugs

Findbugs does not work with Java 9+ and is actually a dead project. The new findbugs is [SpotBugs](#). For maven the new solution is [spotbugs-maven-plugin](#):

```
<plugin>
  <groupId>com.github.spotbugs</groupId>
  <artifactId>spotbugs-maven-plugin</artifactId>
  <version>3.1.11</version>
</plugin>
```

Test Frameworks

Table 38. Minimum recommended versions of common 3rd party test frameworks for Java 9+

GroupId	ArtifactId	Version	Issue
org.mockito	mockito-core	2.23.4	1419 , 1696 , 1607 , 1594 , 1577 , 1482

Maven Plugins

Table 39. Minimum recommended versions of common maven plugins for Java 9+

GroupId	ArtifactId	(min.) Version	Issue
org.apache.maven.plugins	maven-compiler-plugin	3.8.1	x
org.apache.maven.plugins	maven-surefire-plugin	2.22.2	SUREFIRE-1439

GroupId	ArtifactId	(min.) Version	Issue
org.apache.maven.plugins	maven-surefire-report-plugin	2.22.2	SUREFIRE-1439
org.apache.maven.plugins	maven-archetype-plugin	3.1.0	x
org.apache.maven.plugins	maven-javadoc-plugin	3.1.0	x
org.jacoco	jacoco-maven-plugin	0.8.3	663

Maven Usage

With Java modules you can not run Javadoc standalone anymore or you will get this error when running `mvn javadoc:javadoc`:

```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-javadoc-plugin:3.1.1:javadoc (default-cli) on project mmm-base: An error has occurred in Javadoc report generation:
[ERROR] Exit code: 1 - error: module not found: io.github.mmm.base
[ERROR]
[ERROR] Command line was: /projects/mmm/software/java/bin/javadoc @options @packages @argfile
```

As a solution or workaround you need to include the `compile` goal into your build lifecycle so the module-path is properly configured:

```
mvn compile javadoc:javadoc
```

12.29.3. Sources and Links

We want to give credits and say thanks to the following articles that have been there before and helped us on our way:

- [Java 9 Migration Guide: The Seven Most Common Challenges](#)
- [It's time! Migrating to Java 11](#)
- [Migrate Maven Projects to Java 11](#)
- [JAXB on Java 9, 10, 11 and beyond](#)
- [JAXB Artifacts](#)

12.30. Microservices in devonfw

The Microservices architecture is an approach for application development based on a series of *small* services grouped under a business domain. Each individual service runs autonomously and communicating with each other through their APIs. That independence between the different services allows to manage (upgrade, fix, deploy, etc.) each one without affecting the rest of the system's services. In addition to that the microservices architecture allows to scale specific services when facing an increment of the requests, so the applications based on microservices are more flexible and stable, and can be adapted quickly to demand changes.

However, this new approach, developing apps based on microservices, presents some downsides.

Let's see the main challenges when working with microservices:

- Having the applications divided in different services we will need a component (router) to redirect each request to the related microservice. These redirection rules must implement filters to guarantee a proper functionality.
- In order to manage correctly the routing process, the application will also need a catalog with all the microservices and its details: IPs and ports of each of the deployed instances of each microservice, the state of each instance and some other related information. This catalog is called *Service Discovery*.
- With all the information of the *Service Discovery* the application will need to calculate and select between all the available instances of a microservice which is the suitable one. This will be figured out by the library *Client Side Load Balancer*.
- The different microservices will be likely interconnected with each other, that means that in case of failure of one of the microservices involved in a process, the application must implement a mechanism to avoid the error propagation through the rest of the services and provide an alternative as a process result. To solve this, the pattern *Circuit Breaker* can be implemented in the calls between microservices.
- As we have mentioned, the microservices will exchange calls and information with each other so our applications will need to provide a secured context to avoid not allowed operations or intrusions. In addition, since microservices must be able to operate in an isolated way, it is not recommended to maintain a session. To meet this need without using Spring sessions, a token-based authentication is used that exchanges information using the [json web token \(JWT\)](#) protocol.

In addition to all of this we will find other issues related to this particular architecture that we will address fitting the requirements of each project.

- Distributed data bases: each instance of a microservice should have only one data base.
- Centralized logs: each instance of a microservice creates a log and a trace that should be centralized to allow an easier way to read all that information.
- Centralized configuration: each microservice has its own configuration, so our applications should group all those configurations in only one place to ease the configuration management.
- Automatized deployments: as we are managing several components (microservices, catalogs, balancers, etc.) the deployment should be automatized to avoid errors and ease this process.

To address the above, devonfw microservices has an alternative approach [Microservices based on Netflix-Tools](#).

12.31. Application Performance Management

This guide gives hints how to manage, monitor and analyse performance of Java applications.

12.31.1. Temporary Analysis

If you are facing performance issues and want to do a punctual analysis we recommend you to use [glowroot](#). It is ideal in cases where monitoring in your local development environment is suitable. However, it is also possible to use it in your test environment. It is entirely free and open-source. Still it is very powerful and helps to trace down bottlenecks. To get a first impression of the tool take a look at the [demo](#).

JEE/WTP

In case you are forced to use an [JEE application server](#) and want to do a temporary analysis you can double click your server instance from the servers view in Eclipse and click on the link [Open launch configuration](#) in order to add the [-javaagent](#) JVM option.

12.31.2. Regular Analysis

In case you want to manage application performance regularly we recommend to use [JavaMelody](#) that can be integrated into your application. More information on javamelody is available on the [JavaMelody Wiki](#)

12.31.3. Alternatives

- [PinPoint](#)
- [OpenAPM](#)
- [AppDynamics](#)
- [Zabbix](#)

12.32. Caching

Caching is a technical approach to improve performance. While it may appear easy on the first sight it is an advanced topic. In general, try to use caching only when required for performance reasons. If you come to the point that you need caching first think about:

- What to cache?

Be sure about what you want to cache. Is it static data? How often will it change? What will happen if the data changes but due to caching you might receive "old" values? Can this be tolerated? For how long? This is not a technical question but a business requirement.

- Where to cache?

Will you cache data on client or server? Where exactly?

- How to cache?

Is a local cache sufficient or do you need a shared cache?

12.32.1. Local Cache

12.32.2. Shared Cache

Distributed Cache

12.32.3. Products

- <http://ehcache.org/>
- <http://hazelcast.org/>
- <http://terracotta.org/>
- <http://memcached.org/>

12.32.4. Caching of Web-Resources

- <http://www.mobify.com/blog/beginners-guide-to-http-cache-headers/>
- http://en.wikipedia.org/wiki/Web_cache#Cache_control
- http://en.wikipedia.org/wiki/List_of_HTTP_header_fields#Avoiding_caching

12.33. Feature-Toggles

The most software developing teams use Feature-Branching to be able to work in parallel and maintain a stable main branch in the VCS. However Feature-Branching might not be the ideal tool in every case because of big merges and isolation between development groups. In many cases, Feature-Toggles can avoid some of these problems, so these should definitely be considered to be used in the collaborative software development.

12.33.1. Implementation with the devonfw

To use Feature-Toggles with the devonfw, use the Framework [Togglz](#) because it has all the features generally needed and provides a great documentation.

For a pretty minimal working example, also see [this fork](#).

Preparation

The following example takes place in the [oasp-sample-core](#) project, so the necessary dependencies have to be added to the according [pom.xml](#) file. Required are the main Togglz project including Spring support, the Togglz console to graphically change the feature state and the Spring security package to handle authentication for the Togglz console.

```
<!-- Feature-Toggle-Framework togglz -->
<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-spring-boot-starter</artifactId>
    <version>2.3.0.RC2</version>
</dependency>

<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-console</artifactId>
    <version>2.3.0.RC2</version>
</dependency>

<dependency>
    <groupId>org.togglz</groupId>
    <artifactId>togglz-spring-security</artifactId>
    <version>2.3.0.RC2</version>
</dependency>
```

In addition to that, the following lines have to be included in the spring configuration file [application.properties](#)

```
# configuration for the togglz Feature-Toggle-Framework
togglz.enabled=true
togglz.console.secured=false
```

Small features

For small features, a simple query of the toggle state is often enough to achieve the desired functionality. To illustrate this, a simple example follows, which implements a toggle to limit the page size returned by the staffmanagement. See [here](#) for further details.

This is the current implementation to *toggle* the feature:

```
// Uncomment next line in order to limit the maximum page size for the staff member
// search
// criteria.limitMaximumPageSize(MAXIMUM_HIT_LIMIT);
```

To realise this more elegantly with Togglz, first an enum is required to configure the feature-toggle.

```
public enum StaffmanagementFeatures implements Feature {
    @Label("Limit the maximum page size for the staff members")
    LIMIT_STAFF_PAGE_SIZE;

    public boolean isActive() {
        return FeatureContext.getFeatureManager().isActive(this);
    }
}
```

To familiarize the Spring framework with the enum, add the following entry to the `application.properties` file.

```
togglz.feature-
enums=io.oasp.gastronomy.restaurant.staffmanagement.featuremanager.StaffmanagementFeat-
ures
```

After that, the toggle can be used easily by calling the `isActive()` method of the enum.

```
if (StaffmanagementFeatures.LIMIT_STAFF_PAGE_SIZE.isActive()) {
    criteria.limitMaximumPageSize(MAXIMUM_HIT_LIMIT);
}
```

This way, you can easily switch the feature on or off by using the administration console at <http://localhost:8081/devon4j-sample-server/togglz-console>. If you are getting redirected to the login page, just sign in with any valid user (eg. admin).

Extensive features

When implementing extensive features, you might want to consider using the strategy design pattern to maintain the overview of your software. The following example is an implementation of a feature which adds a 25% discount to all products managed by the offermanagement.

Therefore there are two strategies needed:

1. Return the offers with the normal price
2. Return the offers with a 25% discount

The implementation is pretty straight forward so use this as a reference. Compare [this](#) for further details.

```

@Override
@RolesAllowed(PermissionConstants.FIND_OFFER)
public PaginatedListTo<OfferEto> findOfferEtos(OfferSearchCriteriaTo criteria) {
    criteria.limitMaximumPageSize(MAXIMUM_HIT_LIMIT);
    PaginatedListTo<OfferEntity> offers = getOfferDao().findOffers(criteria);

    if (OffermanagementFeatures.DISCOUNT.isActive()) {
        return getOfferEtosDiscount(offers);
    } else {
        return getOfferEtosNormalPrice(offers);
    }
}

// Strategy 1: Return the OfferEtos with the normal price
private PaginatedListTo<OfferEto> getOfferEtosNormalPrice(PaginatedListTo<OfferEntity> offers) {
    return mapPaginatedEntityList(offers, OfferEto.class);
}

// Strategy 2: Return the OfferEtos with the new, discounted price
private PaginatedListTo<OfferEto> getOfferEtosDiscount(PaginatedListTo<OfferEntity> offers) {
    offers = addDiscountToOffers(offers);
    return mapPaginatedEntityList(offers, OfferEto.class);
}

private PaginatedListTo<OfferEntity> addDiscountToOffers(PaginatedListTo<OfferEntity> offers) {
    for (OfferEntity oe : offers.getResult()) {
        Double oldPrice = oe.getPrice().getValue().doubleValue();

        // calculate the new price and round it to two decimal places
        BigDecimal newPrice = new BigDecimal(oldPrice * 0.75);
        newPrice = newPrice.setScale(2, RoundingMode.HALF_UP);

        oe.setPrice(new Money(newPrice));
    }

    return offers;
}

```

12.33.2. Guidelines for a successful use of feature-toggles

The use of feature-toggles requires a specified set of guidelines to maintain the overview on the software. The following is a collection of considerations and examples for conventions that are reasonable to use.

Minimize the number of toggles

When using too many toggles at the same time, it is hard to maintain a good overview of the system and things like finding bugs are getting much harder. Additionally, the management of toggles in the configuration interface gets more difficult due to the amount of toggles.

To prevent toggles from piling up during development, a toggle and the associated obsolete source code should be removed after the completion of the corresponding feature. In addition to that, the existing toggles should be revisited periodically to verify that these are still needed and therefore remove legacy toggles.

Consistent naming scheme

A consistent naming scheme is the key to a structured and easily maintainable set of features. This should include the naming of toggles in the source code and the appropriate naming of commit messages in the VCS. The following section contains an example for a useful naming scheme including a small example.

Every Feature-Toggle in the system has to get its own unique name without repeating any names of features, which were removed from the system. The chosen names should be descriptive names to simplify the association between toggles and their purpose. If the feature should be split into multiple sub-features, you might want to name the feature like the parent feature with a describing addition. If for example you want to split the **DISCOUNT** feature into the logic and the UI part, you might want to name the sub-features **DISCOUNT_LOGIC** and **DISCOUNT_UI**.

The entry in the togglz configuration enum should be named identically to the aforementioned feature name. The explicitness of feature names prevents a confusion between toggles due to using multiple enums.

Commit messages are very important for the use of feature-toggles and also should follow a predefined naming scheme. You might want to state the feature name at the beginning of the message, followed by the actual message, describing what the commit changes to the feature. An example commit message could look like the following:

```
DISCOUNT: Add the feature-toggle to the offermanagement implementation.
```

Mentioning the feature name in the commit message has the advantage, that you can search your git log for the feature name and get every commit belonging to the feature. An example for this using the tool *grep* could look like this.

```
$ git log | grep -C 4 DISCOUNT
```

```
commit 034669a48208cb946cc6ba8a258bdab586929dd9
Author: Florian Luediger <florian.luediger@somemail.com>
Date:   Thu Jul 7 13:04:37 2016 +0100
```

```
DISCOUNT: Add the feature-toggle to the offermanagement implementation.
```

To keep track of all the features in your software system, a platform like GitHub offers issues. When creating an issue for every feature, you can retrace, who created the feature and who is assigned to completing its development. When referencing the issue from commits, you also have links to all the relevant commits from the issue view.

Placement of toggle points

To maintain a clean codebase, you definitely want to avoid using the same toggle in different places in the software. There should be one single query of the toggle which should be able to toggle the whole functionality of the feature. If one single toggle point is not enough to switch the whole feature on or off, you might want to think about splitting the feature into multiple ones.

Use of fine-grained features

Bigger features in general should be split into multiple sub-features to maintain the overview on the codebase. These sub-features get their own feature-toggle and get implemented independently.

12.34. Accessibility

TODO

<http://www.w3.org/TR/WCAG20/>

<http://www.w3.org/WAI/intro/aria>

<http://www.einfach-fuer-alle.de/artikel/bitv/>

<http://www.banu.bund.de>

<http://www.de.capgemini.com/public-sector/igov>

12.35. JEE

This section is about Java Enterprise Edition (JEE). Regarding to our [key principles](#) we focus on open standards. For Java this means that we consider official standards from Java Standard and Enterprise Edition as first choice for considerations. Therefore we also decided to recommend [JAX-RS](#) over [SpringMVC](#) as the latter is proprietary. Only if an existing Java standard is **not** suitable for current demands such as Java Server Faces (JSF), we do not officially recommend it (while you are still free to use it if you have good reasons to do so). In all other cases we officially suggest the according standard and use it in our guides, code-samples, sample application, modules, templates, etc. Examples for such standards are [JPA](#), [JAX-RS](#), [JAX-WS](#), [JSR330](#), [JSR250](#), [JAX-B](#), etc.

12.35.1. Application-Server

We designed everything based on standards to work with different technology stacks and servlet containers. However, we strongly encourage to use [spring](#) and [spring-boot](#). You are free to decide for something else but here is a list of good reasons for our decision:

- **Up-to-date**

With [spring](#) you easily keep up to date with evolving technologies (microservices, reactive, NoSQL, etc.). Most application servers put you in a jail with old legacy technology. In many cases you are even forced to use a totally outdated version of java (JVM/JDK). This may even cause severe IT-Security vulnerabilities but with expensive support you might get updates. Also with [spring](#) and open-source you need to be aware that for IT-security you need to update recently what can cost quite a lot of additional maintenance effort.

- **Development speed**

With [spring-boot](#) you can implement and especially test your individual logic very fast. Starting the app in your IDE is very easy, fast, and realistic (close to production). You can easily write JUnit tests that startup your server application to e.g. test calls to your remote services via HTTP fast and easy. For application servers you need to bundle and deploy your app what takes more time and limits you in various ways. We are aware that this has improved in the past but also [spring](#) continuously improves and is always way ahead in this area. Further, with [spring](#) you have your configurations bundled together with the code in version control (still with ability to handle different environments) while with application servers these are configured externally and can not be easily tested during development.

- **Documentation**

[Spring](#) has an extremely open and active community. There is documentation for everything available for free on the web. You will find solutions to almost any problem on platforms like stackoverflow. If you have a problem you are only a google search away from your solution. This is very much different for proprietary application server products.

- **Helpful Exception Messages**

[Spring](#) is really great for developers on exception messages. If you do something wrong you get detailed and helpful messages that guide you to the problem or even the solution. This is not as

great in application servers.

- **Future-proof**

Spring has evolved really awesome over time. Since its 1.0 release in 2004 spring has continuously been improved and always caught up with important trends and innovations. Even in critical situations, when the company behind it (interface21) was sold, spring went on perfectly. JEE went through a lot of trouble and crisis. Just look at the EJB pain stories. This happened often in the past and also recent. See [JEE 8 in crisis](#).

- **Free**

Spring and its ecosystem is free and open-source. It still perfectly integrates with commercial solutions for specific needs. Most application servers are commercial and cost a lot of money. As of today the ROI for this is of question.

- **Fun**

If you go to conferences or ask developers you will see that spring is popular and fun. If new developers are forced to use an old application server product they will be less motivated or even get frustrated. Especially in today's agile projects this is a very important aspect. In the end you will get into trouble with maintenance on the long run if you rely on a proprietary application server.

Of course the vendors of application servers will tell you a different story. This is simply because they still make a lot of money from their products. We do not get paid from application servers nor from spring. We are just developers who love to build great systems. A good reason for application servers is that they combine a set of solutions to particular aspects to one product that helps to standardize your IT. However, [devonfw](#) fills exactly this gap for the spring ecosystem in a very open and flexible way. However, there is one important aspect that you need to understand and be aware of:

Some big companies decided for a specific application server as their IT strategy. They may have hundreds of apps running with this application server. All their operators and developers have learned a lot of specific skills for this product and are familiar with it. If you are implementing yet another (small) app in this context it can make sense to stick with this application server. However, also they have to be aware that with every additional app they increase their technical debt.

12.36. Messaging Services

Messaging Services provide an asynchronous communication mechanism between applications. Technically this is implemented using [Apache Kafka](#).

Devonfw uses [Spring-Kafka](#) as kafka framework.

This guide explains how Spring kafka is used in devonfw applications. It focuses on aspects which are special to devonfw if you want to learn about spring-kafka you should adhere to springs references documentation.

There is an example of simple kafka implementation in the [devon4j-kafka-employeeapp](#).

The devon4j-kafka library consist of:

- Custom message processor with retry pattern
- Monitoring support
- Tracing support
- Logging support
- Configuration support for Kafka Producers, Consumers, brave tracer and message retry processing including defaults

12.36.1. How to use?

To use devon4j-kafka you have to add required starter dependencies which is "starter-kafka-sender" or "starter-kafka-receiver" from devon4j. These 2 starters are responsible for taking care of the required spring configuration. If you only want to produce messages "starter-kafka-sender" is enough. For consuming messages you need "starter-kafka-receiver" which also includes "starter-kafka-sender".

To use devon4j-kafka message sender add the below dependency:

```
<dependency>
  <groupId>com.devonfw.java.starters</groupId>
  <artifactId>devon4j-starter-kafka-sender</artifactId>
</dependency>
```

It includes the Tracer implementations from Spring cloud sleuth.

To use the devon4j-kafka message receiver configurations , loggers and message retry processor for processing message, add the below dependency:

```
<dependency>
  <groupId>com.devonfw.java.starters</groupId>
  <artifactId>devon4j-starter-kafka-receiver</artifactId>
</dependency>
```

12.36.2. Property Parameters

As written before kafka-producer and listener-specific configuration is done via properties classes. These classes provide useful defaults, at a minimum the following parameters have to be configured:

```
messaging.kafka.common.bootstrap-servers=kafka-broker:9092
messaging.kafka.consumer.group-id=<e.g. application name>
messaging.kafka.listener.container.concurrency=<Number of listener threads for each
listener container>
```

All the configuration beans for devon4j-kafka are annotated with `@ConfigurationProperties` and use common prefixes to read the property values from `application.properties` or `application.yml`.

Example:

```
@Bean
@ConfigurationProperties(prefix = "messaging.kafka.producer")
public KafkaProducerProperties messageKafkaProducerProperties() {
    return new KafkaProducerProperties();
}
```

For producer and consumer the prefixes are `messaging.kafka.producer...` and `'message.kafka.consumer...' and for retry the prefix is 'messaging.retry...`

See [devon4j-kafka-employeeapp's application.properties](#) file for an example.

We use the same properties defined by Apache Kafka or Spring Kafka. They are simply "mapped" to the above prefixes to allow easy access from your application properties. The java docs provided in each of the devon4j-kafka property classes which explains their use and what value has to be passed.

- The [Devon4j-Kafka-Producer properties](#) are defined from [kafka producer config](#).
- The [Devon4j-Kafka-Consumer properties](#) are defined from [kafka consumer config](#).
- The [Devon4j-Kafka-Listener container properties](#) are defined from [Spring kafka listener properties](#).

12.36.3. Naming convention for topics

For better managing of several Kafka topics in your application portfolio we devon strongly advice to introduce a naming scheme for your topics. The schema may depend on the actual usage pattern of kafka. For context where kafka is used in a 1-to-1-communication-scheme (not publish/subscribe) the following schema as been proven useful in practice:

```
<application name>-<service name>-<version>-<service-operation>
```

To keep things easy and prevent problems we suggest to use only small letters, hyphens but no other special characters.

12.36.4. Send Messages

As mentioned above the 'starter-kafka-sender' is required to be added as dependency to use MessageSender from kafka.

```
<dependency>
    <groupId>com.devonfw.java.starters</groupId>
    <artifactId>devon4j-starter-kafka-sender</artifactId>
</dependency>
```

The following example shows how to use MessageSender and its method to send message to kafka broker:

Example:

```
@Inject
private MessageSender messageSender;
private ProducerRecord<K,V> producerRecord;

public void sendMessageToKafka(){
producerRecord=new ProducerRecord<>("topic-name","message");
messageSender.sendMessage(this.producerRecord);
//Alternative
messageSender.sendMessageAndWait(this.producerRecord,10);
}
```

There are multiple methods available from MessageSender of devon4j-kafka. The ProducerListener will log the message sent to the kafka broker.

12.36.5. Receive Messages

To receive messages you have to define a listener. The listener is normally part of the service layer.



Figure 9. Architecture for Kafka services

Import the following `starter-kafka-receiver` dependency to use the listener configurations and loggers from devon4j-kafka.

```
<dependency>
  <groupId>com.devonfw.java.starters</groupId>
  <artifactId>devon4j-starter-kafka-receiver</artifactId>
</dependency>
```

The listener is defined by implementing and annotating a method like in the following example:

```

@KafkaListener(topics = "employeeapp-employee-v1-delete", groupId =
"${messaging.kafka.consumer.groupId}", containerFactory =
"kafkaListenerContainerFactory")
public void consumer(ConsumerRecord<Object, Object> consumerRecord, Acknowledgment
acknowledgment) {
    //user operation
    //To acknowledge listener after processing
    acknowledgement.acknowledge();
}

```

The group id can be mentioned in `application.properties` as listener properties.

```
messaging.kafka.consumer.groupId=default
```

if there are multiple topics and multiple listeners then we suggest to specify the topic names directly on each listeners instead reading from the property file. The container factory mentioned in the `@KafkaListener` is provided in the `KafkaListenerContainerProperties.java` to create default container factory with acknowledgement.

The default ack-mode is `manual_immediate`. It can be overridden by below example:

```
messaging.kafka.listener.container.ackMode=<ack-mode>
```

The other ack-mode values can be referred from [here](#).

12.36.6. Retry

The retry pattern in devon4j-kafka is invoked when a particular exception(described by user in `application.properties` file) is thrown while processing the consumed message and it is configured in `application.properties` file. Then general idea is to separate messages which could not be processed into dedicated retry-topics to allow fine control on how processing of the messages is retried and to not block newly arriving messages. Let us see more about handling retry in the below topics.



Handling retry in devon4j-kafka

The retry pattern is included in the starter dependency of "starter-kafka-receiver".

The `retryPattern` method is used by calling the method `processMessageWithRetry(ConsumerRecord<K, V> consumerRecord, MessageProcessor<K, V> processor)`. Please find the below Example:

```
@Inject
private MessageRetryOperations<K, V> messageRetryOperations;
@Inject
private DeleteEmployeeMessageProcessor<K, V> deleteEmployeeMessageProcessor;
@KafkaListener(topics = "employeeapp-employee-v1-delete", groupId =
"${messaging.kafka.consumer.groupId}", containerFactory =
"kafkaListenerContainerFactory")
public void consumer(ConsumerRecord<K, V> consumerRecord, Acknowledgment
acknowledgment) {
    this.messageRetryOperations.processMessageWithRetry(consumerRecord, this
        .deleteEmployeeMessageProcessor);
    // Acknowledge the listener.
    acknowledgment.acknowledge();
}
```

The implementation for `MessageProcessor` from `devon4j-kafka` is required to provide the implementation to process the `ConsumedRecord` from kafka broker. The implementation for `MessageProcessor` interface can look as below example:

```
import com.devonfw.module.kafka.common.messaging.retry.api.client.MessageProcessor;
@Named
public class DeleteEmployeeMessageProcessor<K, V> implements MessageProcessor<K, V> {
    @Override
    public void processMessage(ConsumerRecord<K, V> message) {
        //process message
    }
}
```

It works as follows:

- * The application gets a message from the topic.
- * During processing of the message an error occurs, the message will be written to the redelivery topic.
- * The message is acknowledged in the topic.
- * The message will be processed from the re-delivery topic after a delay.
- * Processing of the message fails again. It retries until the retry count gets over.
- * When the retry fails in all the retry then the message is logged and payload in the `ProducerRecord` is deleted for log compaction which is explained below.

Retry configuration and naming convention of redelivery topics.

The following properties should be added in the `application.properties` or `application.yml` file.

The retry pattern in `devon4j-kafka` will perform for specific topic of a message. So its mandatory to

specify the properties for each topic. Below properties are example,

```
# Back off policy properties for employeeapp-employee-v1-delete
messaging.retry.back-off-policy.retryReEnqueueDelay.employeeapp-employee-v1-
delete=1000
messaging.retry.back-off-policy.retryDelay.employeeapp-employee-v1-delete=600000
messaging.retry.back-off-policy.retryDelayMultiplier.employeeapp-employee-v1-
delete=1.0
messaging.retry.back-off-policy.retryMaxDelay.employeeapp-employee-v1-delete=600000
messaging.retry.back-off-policy.retryCount.employeeapp-employee-v1-delete=2

# Retry policy properties for employeeapp-employee-v1-delete
messaging.retry.retry-policy.retryPeriod.employeeapp-employee-v1-delete=1800
messaging.retry.retry-policy.retryableExceptions.employeeapp-employee-v1-delete=<Class
names of exceptions for which a retry should be performed>
messaging.retry.retry-policy.retryableExceptionsTraverseCauses.employeeapp-employee-
v1-delete=true

# Back off policy properties for employeeapp-employee-v1-add
messaging.retry.back-off-policy.retryReEnqueueDelay.employeeapp-employee-v1-add=1000
messaging.retry.back-off-policy.retryDelay.employeeapp-employee-v1-add=600000
messaging.retry.back-off-policy.retryDelayMultiplier.employeeapp-employee-v1-add=2.0
messaging.retry.back-off-policy.retryMaxDelay.employeeapp-employee-v1-add=600000
messaging.retry.back-off-policy.retryCount.employeeapp-employee-v1-add=4

# Retry policy properties for employeeapp-employee-v1-add
messaging.retry.retry-policy.retryPeriod.employeeapp-employee-v1-add=3000
messaging.retry.retry-policy.retryableExceptions.employeeapp-employee-v1-add=<Class
names of exceptions for which a retry should be performed>
messaging.retry.retry-policy.retryableExceptionsTraverseCauses.employeeapp-employee-
v1-add=true
```

If you notice the above properties, the **retry-policy** and **back-off policy** properties are repeated twice as i have 2 topics for the retry to be performed with different level of values. The topic name should be added at the last of attribute.

So, the retry will be performed for each topic according to their configuration values.

If you want to provide same/default values for all the topics, then its required to add **default** in the place of topic on the above properties example.

For example,

```

# Default back off policy properties
messaging.retry.back-off-policy.retryReEnqueueDelay.default=1000
messaging.retry.back-off-policy.retryDelay.default=600000
messaging.retry.back-off-policy.retryDelayMultiplier.default=1.0
messaging.retry.back-off-policy.retryMaxDelay.default=600000
messaging.retry.back-off-policy.retryCount.default=2

# Default retry policy properties
messaging.retry.retry-policy.retryPeriod.default=1800
messaging.retry.retry-policy.retryableExceptions.default=<Class names of exceptions
for which a retry should be performed>
messaging.retry.retry-policy.retryableExceptionsTraverseCauses.default=true

```

By giving properties like above , the same values will be passed for all the topics and the way of processing retry for all the topics are same.

All these above property values are mapped to the classes [DefaultBackOffPolicyProperties.java](#) and [DefaultRetryPolicyProperties.java](#) and configured by the class [MessageDefaultRetryConfig.java](#).

The MessageRetryContext in devon kafka is used to perform the retry pattern with the properties from DefaultBackOffPolicyProperties and DefaultRetryPolicyProperties.

The 2 main properties of MessageRetryContext is nextRetry and retryUntil which is a [Instant](#) date format and it is calculated internally using the properties given in DefaultBackOffPolicyProperties and DefaultRetryPolicyProperties.

You may change the behavior of this date calculation by providing your own implementation classes for [MessageBackOffPolicy.java](#) and [MessageRetryPolicy.java](#).

The naming convention for retry topic is the same topic name which you have given to publish the message and we add suffix [-retry](#) to it once it is consumed and given to process with retry.

If there is no topic found in the consumed record the default retry topic will be added which is [default-message-retry](#).

Retry topics

Devon4j-kafka uses a separate retry topic for each topic where retries occur. By default this topic is named [<topic name>-retry](#). You may change this behavior by providing your own implementation for [DefaultKafkaRecordSupport](#) which is an default implementation from devon4j-kafka for [KafkaRecordSupport](#).

Devon4-kafka enqueues a new message for each retry attempt. It is very important to configure your retry topics with [log compaction](#) enabled. More or less simplified, if log compaction is enabled Kafka keeps only one message per message key. Since each retry message has the same key, in fact only one message per retry attempt is stored. After the last retry attempt the message payload is removed from the message so, you do not keep unnecessary data in your topics.

Handling retry finally failed

Per default when the retry fails with final attempt we just log the message and delete the payload of ProducerRecord which comes to proceed the retry pattern.

You can change this behavior by providing the implementation class for the interface [MessageRetryHandler.java](#) which has two method `retryTimeout` and `retryFailedFinal`.

12.36.7. Tracer

We leverage [Spring Cloud Sleuth](#) for tracing in devon4j-kafka. This is used to trace the asynchronous process of kafka producing and consuming. In an asynchronous process it is important to maintain a id which will be same for all asynchronous process. However, devon uses its own correlation-id(UUID) to track the process. But devon4j-kafka uses additional tracing protocol which is [Brave Tracer](#).

This is a part of both starter dependencies [starter-kafka-receiver](#) and [starter-kafka-sender](#).

There are 2 important properties which will be automatically logged which is trace-id and span-id. The trace-id is same for all the asynchronous process and span-id is unique for each asynchronous process.

How devon4j-kafka handles tracer ?

We inject the trace-id and span-id in to the ProducerRecord headers which comes to publish into the kafka broker. Its injected in the headers with the key `traceId` for trace-id and `spanId` for span-id. Along with these, the correlation-id(UUID) is also injected in the headers of record with the key `correlationId`.

So, when you consume record from kafka broker, these values can be found in the consumed record's headers with these keys.

So, it is very helpful to track the asynchronous process of consuming the messages.

12.36.8. Logging

devon4j-kafka provides multiples support classes to log the published message and the consumed message. * The class [ProducerLoggingListener](#) which implements `ProducerListener<K,V>` from spring kafka uses to log the message as soon as it is published in the kafka broker.

- The aspect class [MessageListenerLoggingAspect](#) which is annotated with `@Aspect` and has a method `logMessageprocessing` which is annotated with `@Around("@annotation(org.springframework.kafka.annotation.KafkaListener)&&args(kafkaRecord, ..)")` used to listen to the classes which is annotated with `@KafkaListener` and logs the message as soon as it is consumed.
- The class [MessageLoggingSupport](#) has multiple methods to log different type of events like `MessageReceived`, `MessageSent`, `MessageProcessed`, `MessageNotProcessed`.
- The class [LoggingErrorHandler](#) which implements `ErrorHandler` from spring-kafka which logs the message when an error occurred while consuming message. You may change this behavior by

creating your own implementation class for the ErrorHandler.

12.36.9. Kafka Health check using Spring acutator

The spring config class MessageCommonConfig automatically provides a spring health indicator bean for kafka if the property endpoints. The health indicator will check for all topics listed in `messaging.kafka.health.topics-tocheck` if a leader is available. If this property is missing only the broker connection will be checked. The timeout for the check (default 60s) maybe changed via the property `messaging.kafka.health.timeout`. If an application uses multiple broker(-clusters) for each broker(-cluster) a dedicated health indicator bean has to be configured in the spring config.

The properties for the devon kafka health check should be given like below example:

```
management.endpoint.health.enabled=<true or false>
messaging.kafka.health.timeout=<the health check timeout seconds>
messaging.kafka.health.topicsToCheck=employeeapp-employee-v1-delete,employeeapp-
employee-v1-add
```

These properties are provided with default values except the topicsToCheck and health check will do happen only when the property is `management.endpoint.health.enabled=true`.

12.36.10. Authentication

JSON Web Token (JWT)

devon4j-kafka supports authentication via JSON Web Tokens (JWT) out-of-the-box. To use it add a dependency to the devon4j-starter-security-jwt:

```
<dependency>
  <groupId>com.devonfw.java.starters</groupId>
  <artifactId>devon4j-starter-security-jwt</artifactId>
</dependency>
```

The authentication via JWT needs some configuration, e.g. a keystore to verify the token signature. This is explained in the [JWT documentation](#).

To secure a message listener with jwt add the `@JwtAuthentication`:

```
@JwtAuthentication
@KafkaListener(topics = "employeeapp-employee-v1-delete", groupId =
"${messaging.kafka.consumer.groupId}")
public void consumer(ConsumerRecord<K, V> consumerRecord, Acknowledgment
acknowledgment) {
...
}
```

With this annotation in-place each message will be checked for a valid JWT in a message header with the name `Authorization`. If a valid annotation is found the spring security context will be initialized with the user roles and "normal" authorization e.g. with `@RolesAllowed` may be used. This is also demonstrated in the kafka sample application.

12.36.11. Using Kafka for internal parallel processing

Apart from the use of Kafka as "communication channel" it sometimes helpful to use Kafka internally to do parallel processing:

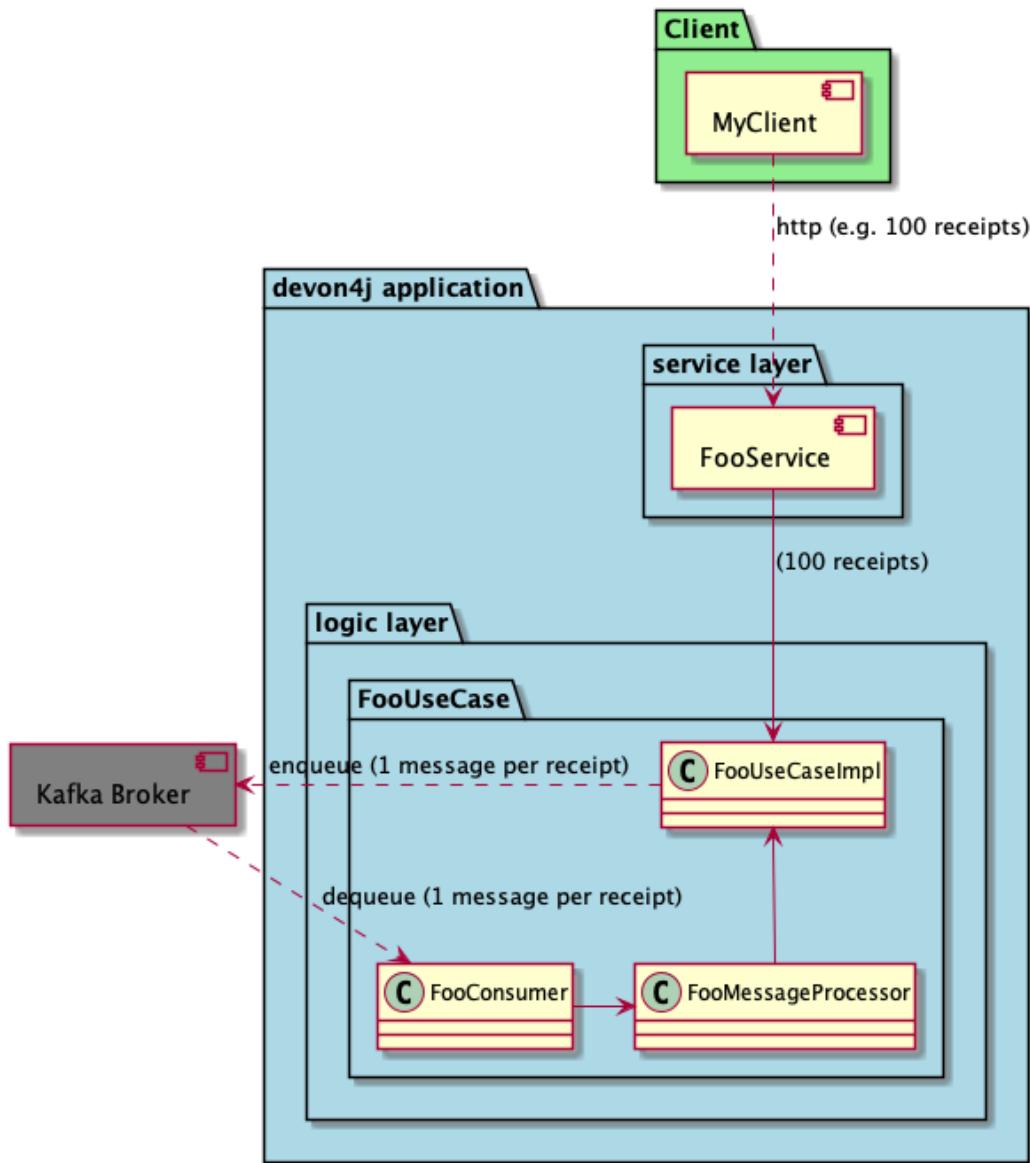


Figure 10. Architecture for internal parallel processing with Kafka

This examples shows a payment service which allows a to submit a list of receipt IDs for payment. We assume that the payment it self takes a long time and should be done asynchronously and in parallel. The general idea is to put a message for each receipt to pay into a topic. This is done in the use case implementation in a first step, if a rest call arrives. Also part of the use case is a listener which consumes the messages. For each message (e.g. payment to do) a processor is called, which actually does the payment via the use case. Since Kafka supports concurrency for the listeners easily the payment will also be done in parallel. All features of devon4j-kafka, like retry handling

could also be used.

12.37. Messaging

Messaging in Java is done using the [JMS](#) standard from JEE.

12.37.1. Products

For messaging you need to choose a JMS provider such as:

- [RabbitMQ](#)
- [ActiveMQ](#)
- [Oracle Advanced Queuing](#) (esp. if you already use [Oracle RDBMS](#))

12.37.2. Receiver

As a receiver of messages is receiving data from other systems it is located in the [service-layer](#).

JMS Listener

A [JmsListener](#) is a class listening and consuming JMS messages. It should carry the suffix `JmsListener` and implement the [MessageListener](#) interface or have its listener method annotated with `@JmsListener`. This is illustrated by the following example:

```
@Named
@Transactional
public class BookingJmsListener /* implements MessageListener */ {

    @Inject
    private Bookingmanagement bookingmanagement;

    @Inject
    private MessageConverter messageConverter;

    @JmsListener(destination = "BOOKING_QUEUE", containerFactory =
    "jmsListenerContainerFactory")
    public void onMessage(Message message) {
        try {
            BookingTo bookingTo = (BookingTo) this.messageConverter.fromMessage(message);
            this.bookingmanagement.importBooking(bookingTo);
        } catch (MessageConversionException | JMSException e) {
            throw new InvalidMessageException(message);
        }
    }
}
```

12.37.3. Sender

A sender of messages is writing to a data storage and hence is located in the [dataaccess-layer](#).

12.38. Monitoring

Monitoring is a very comprehensive topic. For [devon4j](#) we focus on selected core topics which are most important when developing production-ready applications. On a high level view we strongly suggest to separate the application to be monitored from the [monitoring system](#) itself. The monitoring system covers aspects like

- Collect monitoring information
- Aggregate, process and visualize data, e.g. in dashboards
- Provide alarms
- ...

In distributed systems it is crucial that a monitoring system provides a central overview over all applications in your application landscape. So it is not feasible to provide a monitoring system as part of your application. Your application is responsible for providing information to the monitoring system.

12.38.1. How to provide monitoring information

Java Management Extensions (JMX)

JMX is the official java monitoring solution. It is part of the JDK. Your application may provide monitoring information or receive monitoring related commands via [MBeans](#). There is a huge amount of information about JMX available. A good starting point might be [JMX on wikipedia](#). Traditionally JMX uses RMI for communication. In many environments HTTP(S) is preferred, so be careful on deciding if JMX is the right solution. Alternatively your application could provide a monitoring API via REST.

REST APIs

Since REST APIs are very popular it is quite natural to provide monitoring information via REST. If your application already uses REST and there is no JMX/RMI based monitoring solution in place, it is often a better approach to provide monitoring APIs via REST than introducing a new protocol with JMX.

Logging

Some aspects of monitoring are covered by "logging". A typical case might be the requirement to "monitor" the application REST APIs. For that it is perfectly fine to log the performance of all (or some) request and ship this information to a logging system like [Graylog](#). So please carefully read the [logging guide](#). To allow efficient processing of those logs you should use JSON based logs.

12.38.2. Which monitoring information to provide?

It is not possible to provide a final list of which monitoring information is exactly required for your application. But we give you a general advice what type of information your application should provide at a minimum.

General information

It is often useful if your application provides basic information about itself. This should cover

- The name of the application
- The version (or buildno., or commit-id, ...) of the application

This is especially useful in complex environments, e.g. to verify that deployments went correctly.

Metrics

Metric means providing key figures about your applications like

- performance key figures
 - request duration
 - queue lengths
 - duration of database queries
 - ..
- business related information
 - number of successful / unsuccessful requests
 - size of result sets
 - worth of shopping baskets
 - ..
- Technical key figures
 - JVM heap usage
 - cache sizes
 - (database) pool usage
 - ...
- ...

Remember that processing of this data should be done mainly in the monitoring system. You might have noticed that there are different types of metrics: those that represent current values (like JVM heap usage, queue length, ...), others base on (timed) events like (duration of requests). Handling of different types of metrics might be different. For handling events you may:

- Write log statements for each (or a sample of) event. These logs must then be shipped to your monitoring systems.
- Send data for the event via an API of your monitoring system
- Provide a REST API (or JMX MBeans) with pre-aggregated key figures, which is periodically polled by your monitoring system. This solution is a bit inferior since the aggregation is part of your application and might not fit to the desired visualization in your monitoring systems.

For actual values you may:

- Write them periodically to your log. These logs must then be shipped to your monitoring systems.
- Send them periodically via an API of your monitoring systems
- Provide a REST API (or JMX MBeans) which is periodically polled by your monitoring system.

Health (Watchdog)

For monitoring a complex application landscape it is crucial to have an exact overview if all applications are up and running. So your application should offer an API for the monitoring system which allows to easily check if the application is alive. Often this alive information is polled by the monitoring system with a kind of watchdog. The health check should include checks if the application is working "correctly". For that we suggest to check if all required neighbour systems and infrastructure components are usable:

- Check if your database can be queried (with a dummy query)
- Check if you can reach your messaging system
- Check if you can reach all your neighbour system, e.g. by querying their info-endpoint

You should be very careful to not cascade those requests, e.g. your system should only test their direct neighbours. This test should not lead to additional tests in these systems.

The healthcheck should return a simple OK/NOK result for use in dashboards, but additionally include detailed results for each check.

12.38.3. Implementation with Spring Boot Actuator

To implement a monitoring API for your systems we suggest to use [Spring Boot Actuator](#). Actuator offers APIs which provide monitoring information including metrics via HTTP and JMX. It also contains a framework to implement [health checks](#). Please consult the original documentation for information about how to use it. Basically to use it, add the following dependency to the [pom.xml](#) of your application core:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
</dependencies>
```

There will be several [endpoints](#) with monitoring information available out-of-the-box. We **strongly** advice to check carefully which information is required in your context, normally this is [info](#), [health](#) and [metrics](#). Be careful not to expose any security related information via this mechanism (e.g. by exposing those endpoints externally).

To make the info-endpoint useful you need to provide information to actuator. A good way to achieve this is by using the provided [maven module](#).

For first steps it might be useful to deactivate security for the actuator endpoints (this is **just for testing, never release it!**). This can be accomplished by implementing the following class:

```
@Configuration
@EnableWebSecurity
@Profile(SpringProfileConstants.NOT_JUNIT)
public class WebSecurityConfig extends BaseWebSecurityConfig {

    @Override
    public void configure(WebSecurity web) throws Exception {

        super.configure(web);
        web.ignoring().requestMatchers(EndpointRequest.toAnyEndpoint());
    }

}
```

Devonfw additions

Devonfw includes the following additions for Spring boot actuator:

- [Kafka Health Check](#) in devon4j-kafka (WIP)

12.38.4. Integration of monitoring information

Loadbalancers

To loadbalance HTTP requests the loadbalancers needs to know which instances of the desired application are available and functioning. Often loadbalancers support reacting on the HTTP status code of an HTTP request to the service. The loadbalancer will periodically poll the service to find out if it is available or not. To configure this you may use the healthcheck of the service to find out if the instance is functioning correctly or not.

Docker

Docker supports a [healthcheck](#). You may use a simple local `curl` to your application here to find out if the service is healthy or not. But be careful often unhealthy containers are automatically restarted. If you use the [health information](#) of your application this may lead to undesired effects. Since the health checks relies on querying all neighbour systems and infrastructure components, applications often become unhealthy because a 3rd system has problems. Restarting the application itself will not heal the problem and be inexpedient. So generally it is better you query the info endpoint of your application to just check if the service itself is up and running.

12.39. Full Text Search

If you want to all your users fast and simple searches with just a single search field (like in google), you need full text indexing and search support.

12.39.1. Solutions

- [Lucene](#)
- [NGram](#)
- [Solr](#)
- [elastic-search](#)

Maybe you also want to use native features of your database

- [SAP Hana Fuzzy Search](#)
- [Oracle Text](#)

12.39.2. Best Practices

TODO

[1] "Stammdaten" in German.

[2] Whether to use checked exceptions or not is a controversial topic. Arguments for both sides can be found under [The Trouble with Checked Exceptions](#), [Unchecked Exceptions — The Controversy](#), and [Checked Exceptions are Evil](#). The arguments in favor of unchecked exceptions tend to prevail for applications build with Devon4j. Therefore, unchecked exceptions should be used for a consistent style.

13. Tutorials

13.1. Creating a new application

13.1.1. Running the archetype

In order to create a new application you must use the archetype provided by devon4j which uses the maven archetype functionality.

To create a new application, you should have installed devonfw IDE. Follow the devon ide documentation to install the same. You can choose between 2 alternatives, create it from command line or, in more visual manner, within eclipse.

From command Line

To create a new devon4j application from command line, you can simply run the following command:

```
devon java create com.example.application.sampleapp
```

For low-level creation you can also manually call this command:

```
mvn -DarchetypeVersion=${devon4j.version}
-DarchetypeGroupId=com.devonfw.java.templates -DarchetypeArtifactId=devon4j-template
-server archetype:generate -DgroupId=com.example.application -DartifactId=sampleapp
-Dversion=1.0.0-SNAPSHOT -Dpackage=com.devonfw.application.sampleapp
```

Attention: The `archetypeVersion` (first argument) should be set to the latest version of `devon4j`. You can easily determine the version from this badge excluding the `v` prefix: [\[web artifact\]](#)

Further providing additional properties (using `-D` parameter) you can customize the generated app:

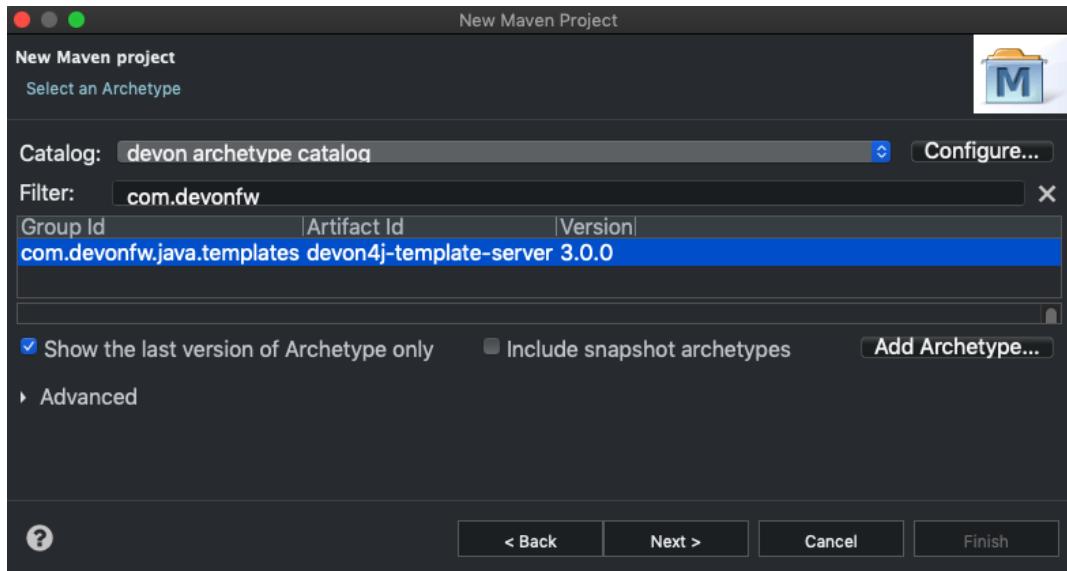
Table 40. Options for app template

property	comment	example
<code>dbType</code>	Choose the type of RDBMS to use (<code>hana</code> , <code>oracle</code> , <code>mssql</code> , <code>postgresql</code> , <code>mariadb</code> , <code>mysql</code> , etc.)	<code>-DdbType=postgresql</code>
<code>batch</code>	Option to add an <code>batch</code> module	<code>-Dbatch=batch</code>
<code>earProjectName</code>	Option to add an EAR module with the given name	<code>-DearProjectName=ear</code>

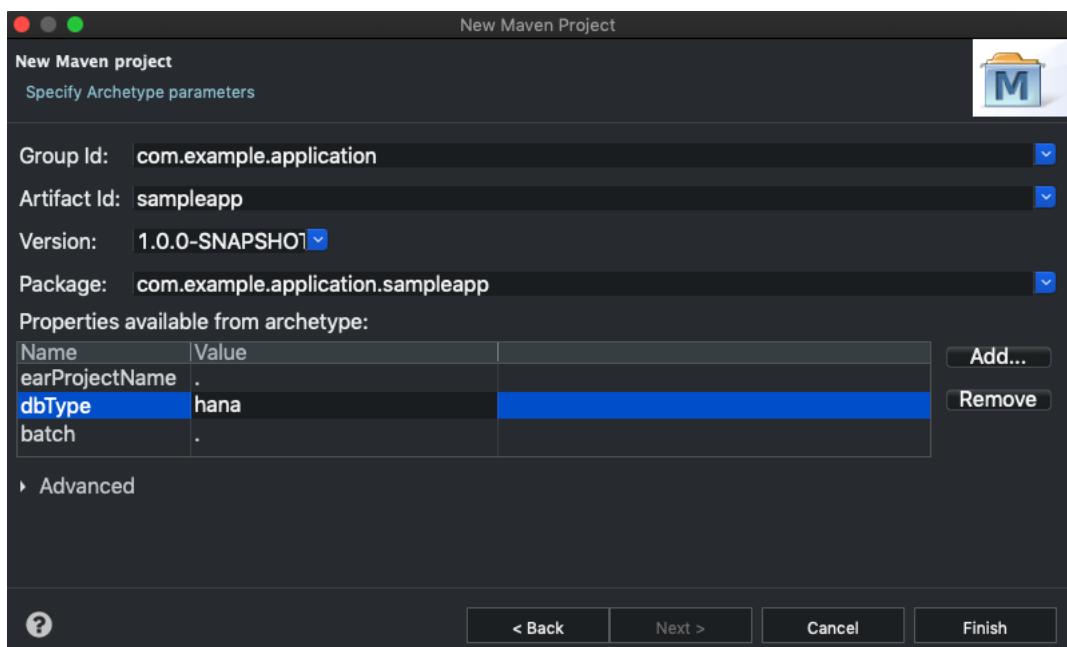
From Eclipse

After that, you should follow this Eclipse steps to create your application:

- Create a new Maven Project.
- Choose the devon4j-template-server archetype, just like the image.



- Fill the Group Id, Artifact Id, Version and Package for your project. If you want to add an EAR generation mechanism to your project, you should fill the property earProjectName with the value Artifact Id + "-ear". For example, "sampleapp-ear". If you only want WAR generation, you can remove the property earProjectName.



- Finish the Eclipse assistant and you are ready to start your project.

13.1.2. What is generated

The application template (archetype) generates a Maven multi-module project. It has the following modules:

- **api**: module with the API (REST service interfaces, transferobjects, datatypes, etc.) to be imported by other apps as a maven dependency in order to invoke and consume the offered (micro)services.
- **core**: maven module containing the core of the application.
- **batch**: optional module for **batch(es)**
- **server**: module that bundles the entire app (**core** with optional **batch**) as a WAR file.
- **ear**: optional maven module is responsible to packaging the application as a EAR file.

The toplevel **pom.xml** of the generated project has the following features:

- Properties definition: Spring-boot version, Java version, etc.
- Modules definition for the modules (described above)
- Dependency management: define versions for dependencies of the technology stack that are recommended and work together in a compatible way.
- Maven plugins with desired versions and configuration
- Profiles for **test stages**

13.1.3. How to run your app

Run app from IDE

To run your application from your favourite IDE, simply launch **SpringBootApp** as java application.

Run app as bootified jar or war

More details are available [here](#).

Part IV: devon4ng

14. Architecture

14.1. Architecture

The following principles and guidelines are based on Angular Styleguide - especially Angular modules (see [Angular Docs](#)). It extends those where additional guidance is needed to define an architecture which is:

- maintainable across applications and teams
- easy to understand, especially when coming from a classic Java/.Net perspective - so whenever possible the same principles apply both to the server and the client
- pattern based to solve common problems
- based on best of breed solutions coming from open source and Capgemini project experiences
- gives as much guidance as necessary and as little as possible

14.1.1. Overview

When using Angular the web client architecture is driven by the framework in a certain way Google and the Angular community think about web client architecture. Angular gives an opinion on how to look at architecture. It is component based like devon4j but uses different terms which are common language in web application development. The important term is *module* which is used instead of *component*. The primary reason is the naming collision with the *Web Components* standard (see [Web Components](#)).

To clarify this:

- A *component* describes an UI element containing HTML, CSS and JavaScript - structure, design and logic encapsulated inside a reusable container called component.
- A *module* describes an applications feature area. The application flight-app may have a module called booking.

An application developed using Angular consists of multiple modules. There are feature modules and special modules described by the Angular Styleguide - *core* and *shared*. Angular or Angular Styleguide give no guidance on how to structure a module internally. This is where this architecture comes in.

Layers

The architecture describes two layers. The terminology is based on common language in web development.

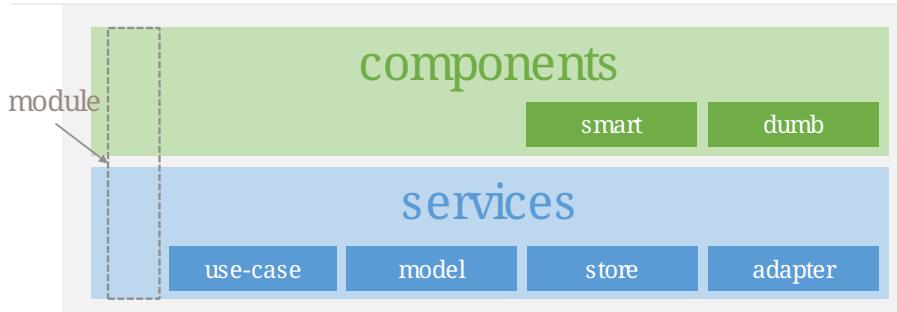


Figure 11. Layers

- **Components Layer** encapsulates components which present the current application state. Components are separated into *Smart* and *Dumb Components*. The only logic present is view logic inside *Smart Components*.
- **Services Layer** is more or less what we call 'business logic layer' on the server side. The layer defines the applications state, the transitions between state and classic business logic. Stores contain application state over time to which *Smart Components* subscribe to. Adapters are used to perform XHRs, WebSocket connections, etc. The business model is described inside the module. Use case services perform business logic needed for use cases. A use case services interacts with the store and adapters. Methods of use case services are the API for *Smart Components*. Those methods are *Actions* in reactive terminology.

Modules

Angular requires a module called *app* which is the main entrance to an application at runtime - this module gets bootstrapped. Angular Styleguide defines feature modules and two special modules - *core* and *shared*.

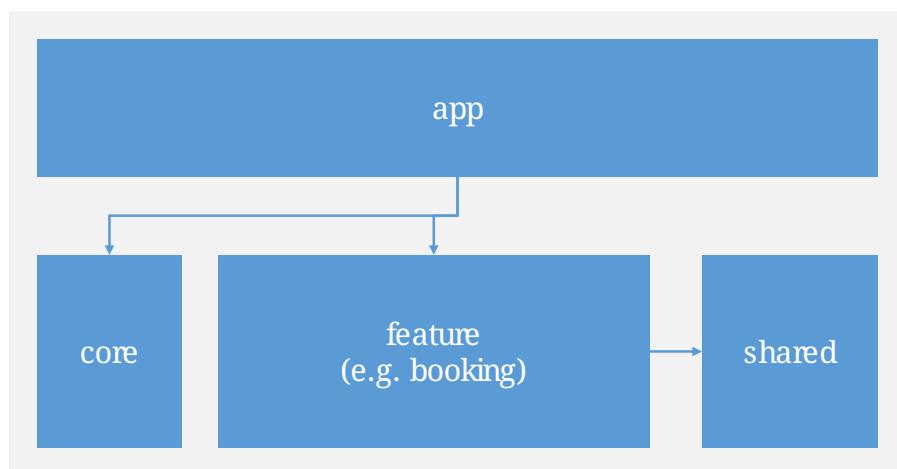


Figure 12. Modules

A feature module is basically a vertical cut through both layers. The *shared* module consists of components shared across feature modules. The *core* module holds services shared across modules. So *core* module is a module only having a services layer and *shared* module is a module only having a components layer.

14.2. Meta Architecture

14.2.1. Introduction

Purpose of this document

In our business applications, the client easily gets underestimated. Sometimes the client is more complex to develop and design than the server. While the server architecture is nowadays easily to agree as common sense, for clients this is not as obvious and stable especially as it typically depends on the client framework used. Finding a concrete architecture applicable for all clients may therefore be difficult to accomplish.

This document tries to define on a high abstract level, a reference architecture which is supposed to be a mental image and frame for orientation regarding the evaluation and appliance of different client frameworks. As such it defines terms and concepts required to be provided for in any framework and thus gives a common ground of understanding for those acquainted with the reference architecture. This allows better comparison between the various frameworks out there, each having their own terms for essentially the same concepts. It also means that for each framework we need to explicitly map how it implements the concepts defined in this document.

The architecture proposed herein is neither new nor was it developed from scratch. Instead it is the gathered and consolidated knowledge and best practices of various projects (s. References).

Goal of the Client Architecture

The goal of the client architecture is to support the non-functional requirements for the client, i.e. mostly maintainability, scalability, efficiency and portability. As such it provides a component-oriented architecture following the same principles listed already in the devonfw architecture overview. Furthermore it ensures a homogeneity regarding how different concrete UI technologies are being applied in the projects, solving the common requirements in the same way.

Architecture Views

As for the server we distinguish between the business and the technical architecture. Where the business architecture is different from project to project and relates to the concrete design of dialog components given concrete requirements, the technical architecture can be applied to multiple projects.

The focus of this document is to provide a technical reference architecture on the client on a very abstract level defining required layers and components. How the architecture is implemented has to be defined for each UI technology.

The technical infrastructure architecture is out of scope for this document and although it needs to be considered, the concepts of the reference architecture should work across multiple TI architecture, i.e. native or web clients.

14.2.2. devonfw Reference Client Architecture

The following gives a complete overview of the proposed reference architecture. It will be built up incrementally in the following sections.



Figure 1 Overview

Client Architecture

On the highest level of abstraction we see the need to differentiate between dialog components and their container they are managed in, as well as the access to the application server being the backend for the client (e.g. an devon4j instance). This section gives a summary of these components and how they relate to each other. Detailed architectures for each component will be supplied in subsequent sections



Figure 2 Overview of Client Architecture

Dialog Component

A dialog component is a logical, self-contained part of the user interface. It accepts user input and actions and controls communication with the user. Dialog components use the services provided by the dialog container in order to execute the business logic. They are self-contained, i.e. they possess their own user interface together with the associated logic, data and states.

- Dialog components can be composed of other dialog components forming a hierarchy
- Dialog components can interact with each other. This includes communication of a parent to its children, but also between components independent of each other regarding the hierarchy.

Dialog Container

Dialog components need to be managed in their lifecycle and how they can be coupled to each other. The dialog container is responsible for this along with the following:

- Bootstrapping the client application and environment
 - Configuration of the client
 - Initialization of the application server access component
- Dialog Component Management
 - Controlling the lifecycle
 - Controlling the dialog flow
 - Providing means of interaction between the dialogs
 - Providing application server access
 - Providing services to the dialog components
(e.g. printing, caching, data storage)
- Shutdown of the application

Application Server Access

Dialogs will require a backend application server in order to execute their business logic. Typically in an devonfw application the service layer will provide interfaces for the functionality exposed to the client. These business oriented interfaces should also be present on the client backed by a proxy handling the concrete call of the server over the network. This component provides the set of interfaces as well as the proxy.

Dialog Container Architecture

The dialog container can be further structured into the following components with their respective tasks described in own sections:



Figure 3 Dialog Container Architecture

Application

The application component represents the overall client in our architecture. It is responsible for bootstrapping all other components and connecting them with each other. As such it initializes the components below and provides an environment for them to work in.

Configuration Management

The configuration management manages the configuration of the client, so the client can be deployed in different environments. This includes configuration of the concrete application server to be called or any other environment-specific property.

Dialog Management

The Dialog Management component provides the means to define, create and destroy dialog components. It therefore offers basic lifecycle capabilities for a component. In addition it also allows composition of dialog components in a hierarchy. The lifecycle is then managed along the hierarchy, meaning when creating/destroying a parent dialog, this affects all child components, which are created/destroyed as well.

Service Registry

Apart from dialog components, a client application also consists of services offered to these. A service can thereby encompass among others:

- Access to the application server
- Access to the dialog container functions for managing dialogs or accessing the configuration
- Dialog independent client functionality such as Printing, Caching, Logging, Encapsulated business logic such as tax calculation
- Dialog component interaction

The service registry offers the possibility to define, register and lookup these services. Note that these services could be dependent on the dialog hierarchy, meaning different child instances could obtain different instances / implementations of a service via the service registry, depending on which service implementations are registered by the parents.

Services should be defined as interfaces allowing for different implementations and thus loose coupling.

Dialog Component Architecture

A dialog component has to support all or a subset of the following tasks:

- (T1) Displaying the user interface incl. internationalization
- (T2) Displaying business data incl. changes made to the data due to user interactions and localization of the data
- (T3) Accepting user input including possible conversion from e.g. entered Text to an Integer
- (T4) Displaying the dialog state
- (T5) Validation of user input
- (T6) Managing the business data incl. business logic altering it due to user interactions

(T7) Execution of user interactions

(T8) Managing the state of the dialog (e.g. Edit vs. View)

(T9) Calling the application server in the course of user interactions

Following the principle of separation of concerns, we further structure a dialog component in an own architecture allowing us the distribute responsibility for these tasks along the defined components:



Figure 4 Overview of dialog component architecture

Presentation Layer

The presentation layer generates and displays the user interface, accepts user input and user actions and binds these to the dialog core layer (T1-5). The tasks of the presentation layer fall into two categories:

- **Provision of the visual representation (View component)**

The presentation layer generates and displays the user interface and accepts user input and user actions. The logical processing of the data, actions and states is performed in the dialog core layer. The data and user interface are displayed in localized and internationalized form.

- **Binding of the visual representation to the dialog core layer**

The presentation layer itself does not contain any dialog logic. The data or actions entered by the user are then processed in the dialog core layer. There are three aspects to the binding to the dialog core layer. We refer to “data binding”, “state binding” and “action binding”. Syntactical and (to a certain extent) semantic validations are performed during data binding (e.g. cross-field plausibility checks). Furthermore, the formatted, localized data in the presentation layer is converted into the presentation-independent, neutral data in the dialog core layer (parsing) and vice versa (formatting).

Dialog Core Layer

The dialog core layer contains the business logic, the control logic, and the logical state of the dialog. It therefore covers tasks T5-9:

- **Maintenance of the logical dialog state and the logical data**

The dialog core layer maintains the logical dialog state and the logical data in a form which is independent of the presentation. The states of the presentation (e.g. individual widgets) must not be maintained in the dialog core layer, e.g. the view state could lead to multiple presentation states disabling all editable widgets on the view.

- **Implementation of the dialog and dialog control logic**

The component parts in the dialog core layer implement the client specific business logic and the dialog control logic. This includes, for example, the manipulation of dialog data and dialog states as well as the opening and closing of dialogs.

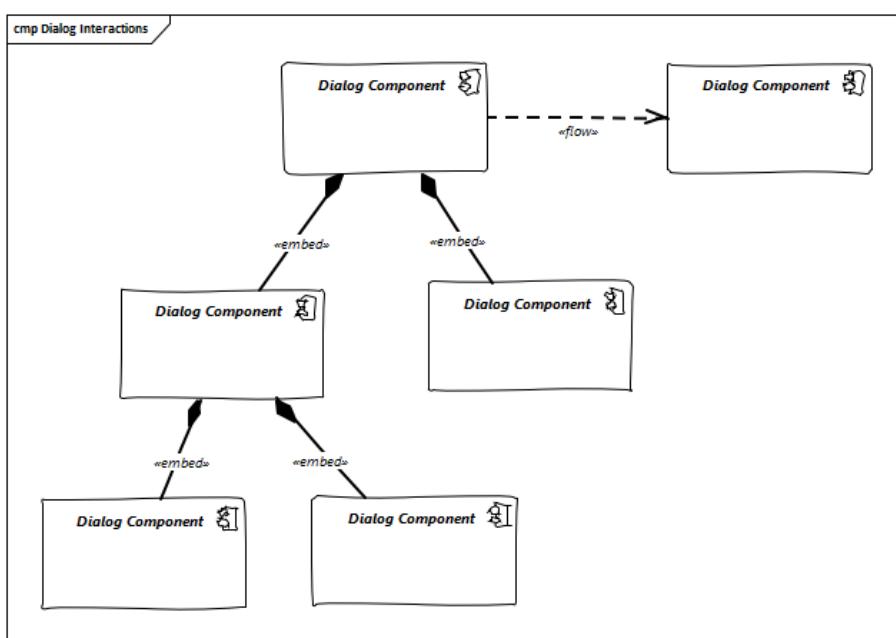
- **Communication with the application server**

The dialog core layer calls the interfaces of the application server via the application server access component services.

The dialog core layer should not depend on the presentation layer enforcing a strict layering and thus minimizing dependencies.

Interactions between dialog components

Dialog components can interact in the following ways:



- **Embedding of dialog components**

As already said dialog components can be hierarchically composed. This composition works by embedding one dialog component within the other. Apart from the lifecycle managed by the dialog container, the embedding needs to cope for the visual embedding of the presentation and core layer.

- **Embedding dialog presentation**

The parent dialog needs to either integrate the embedded dialog in its layout or open it in an own model window.

- **Embedding dialog core**

The parent dialog needs to be able to access the embedded instance of its children. This allows initializing and changing their data and states. On the other hand the children might

require context information offered by the parent dialog by registering services in the hierarchical service registry.

- **Dialog flow**

Apart from the embedding of dialog components representing a tight coupling, dialogs can interact with each other by passing the control of the UI, i.e. switching from one dialog to another.

When interacting, dialog components should interact only between the same or lower layers, i.e. the dialog core should not access the presentation layer of another dialog component.

14.2.3. Appendix

Notes about Quasar Client

The Quasar client architecture as the consolidated knowledge of our CSD projects is the major source for the above drafted architecture. However, the above is a much simplified and more agile version thereof:

- Quasar Client tried to abstract from the concrete UI library being used, so it could decouple the business from the technical logic of a dialog. The presentation layer should be the only one knowing the concrete UI framework used. This level of abstraction was dropped in this reference architecture, although it might of course still make sense in some projects. For fast-moving agile projects in the web however introducing such a level of abstraction takes effort with little gained benefits. With frameworks like Angular 2 we would even introduce one additional seemingly artificial and redundant layer, since it already separates the dialog core from its presentation.
- In the past and in the days of Struts, JSF, etc. the concept of session handling was important for the client since part of the client was sitting on a server with a session relating it to its remote counterpart on the users PC. Quasar Client catered for this need, by very prominently differentiating between session and application in the root of the dialog component hierarchy. However, in the current days of SPA applications and the lowered importance of servers-side web clients, this prominent differentiation was dropped. When still needed the referenced documents will provide in more detail how to tailor the respective architecture to this end.

14.2.4. References

- Architecture Guidelines for Application Design: https://troom.capgemini.com/sites/vcc/engineering/Cross%20Cutting/ArchitectureGuide/Architecture_Guidelines_for_Application_Design_v2.0.docx
- Quasar Client Architekturen: <https://troom.capgemini.com/sites/vcc/Shared%20Documents/CrossCuttingContent/TopicOrientedCCC/QuasarOverview/NCE%20Quasar%20Review%20Workshop%202009-11-17/Quasar%20Development/Quasar-Client-Architectures.doc>

15. Layers

15.1. Components Layer

The components layer encapsulates all components presenting the current application view state, which means data to be shown to the user. The term component refers to a component described by the standard [Web Components](#). So this layer has all Angular components, directives and pipes defined for an application. The main challenges are:

- how to structure the components layer (see [File Structure Guide](#))
- decompose components into maintainable chunks (see [Component Decomposition Guide](#))
- handle component interaction
- manage calls to the services layer
- apply a maintainable data and eventflow throughout the component tree

15.1.1. Smart and Dumb Components

The architecture applies the concept of *Smart* and *Dumb Components* (syn. *Containers* and *Presenters*). The concept means that components are devided into *Smart* and *Dumb Components*.

A *Smart Component* typically is a toplevel dialog inside the component tree.

- a component, that can be routed to
- a modal dialog
- a component, which is placed inside [AppComponent](#)

A *Dumb Component* can be used by one to many *Smart Components*. Inside the component tree a *Dumb Component* is a child of a *Smart Component*.

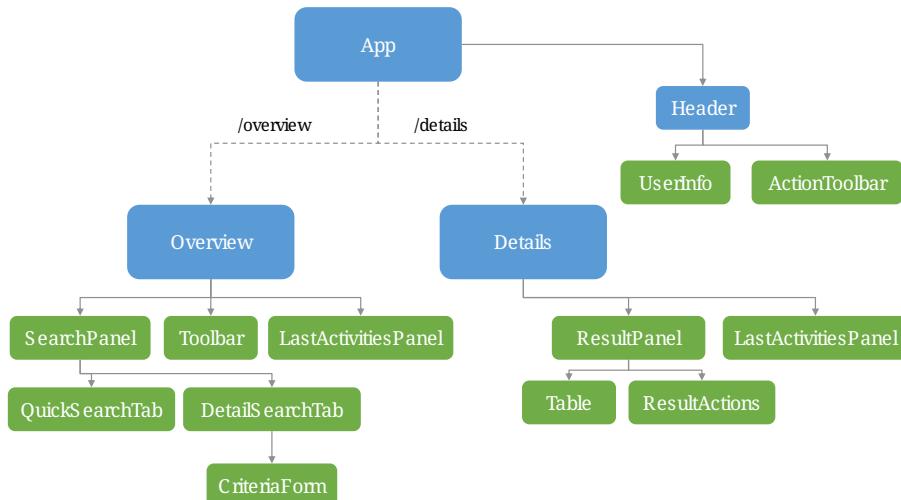


Figure 13. Component tree example

As shown the topmost component is always the [AppComponent](#) in Angular applications. The component tree describes the hierarchy of components starting from [AppComponent](#). The figure shows *Smart Components* in blue and *Dumb Components* in green. [AppComponent](#) is a *Smart*

Component by definition. Inside the template of `AppComponent` placed components are static components inside the component tree. So they are always displayed. In the example `OverviewComponent` and `DetailsComponent` are rendered by Angular compiler depending on current URL the application displays. So `OverviewComponents` subtree is displayed if the URL is `/overview` and `DetailsComponents` subtree is displayed if the URL is `/details`. To clarify this distinction further the following table shows the main differences.

Table 41. Smart vs Dumb Components

Smart Components	Dumb Components
contain the current view state	show data via binding (<code>@Input</code>) and contain no view state
handle events emitted by <i>Dumb Components</i>	pass events up the component tree to be handled by <i>Smart Components</i> (<code>@Output</code>)
call the services layer	never call the services layer
use services	do not use services
consists of n <i>Dumb Components</i>	is independent of <i>Smart Components</i>

15.1.2. Interaction of Smart and Dumb Components

With the usage of the *Smart* and *Dumb Components* pattern one of the most important part is component interaction. Angular comes with built in support for component interaction with `@Input()` and `@Output()` Decorators. The following figure illustrates an unidirectional data flow.

- Data always goes down the component tree - from a *Smart Component* down its children.
- Events bubble up, to be handled by a *Smart Component*.

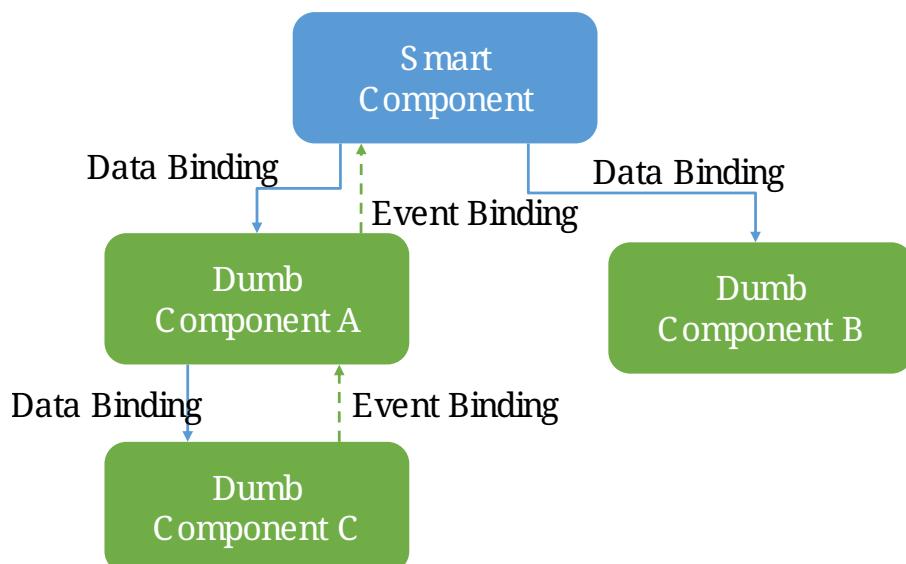


Figure 14. Smart and Dumb Component Interaction

As shown a *Dumb Components* role is to define a signature by declaring Input and Output Bindings.

- `@Input()` defines what data is necessary for that component to work
- `@Output()` defines which events can be listened on by the parent component

Listing 11. Dumb Components define a signature

```
export class ValuePickerComponent {

  @Input() columns: string[];
  @Input() items: {}[];
  @Input() selected: {};
  @Input() filter: string;
  @Input() isChunked = false;
  @Input() showInput = true;
  @Input() showDropdownHeader = true;

  @Output() elementSelected = new EventEmitter<{}>();
  @Output() filterChanged = new EventEmitter<string>();
  @Output() loadNextChunk = new EventEmitter();
  @Output() escapeKeyPressed = new EventEmitter();

}

}
```

The example shows the *Dumb Component* `ValuePickerComponent`. It describes seven input bindings with `isChunked`, `showHeader` and `showDropdownHeader` being non mandatory as they have a default value. Four output bindings are present. Typically, a *Dumb Component* has very little code to no code inside the TypeScript class.

Listing 12. Smart Components use the Dumb Components signature inside the template

```
<div>

  <value-input
    ...
  </value-input>

  <value-picker
    *ngIf="isValuePickerOpen"
    [columns]="columns"
    [items]="filteredItems"
    [isChunked]="isChunked"
    [filter]="filter"
    [selected]="selectedItem"
    [showDropdownHeader]="showDropdownHeader"
    (loadNextChunk)="onLoadNextChunk()"
    (elementSelected)="onElementSelected($event)"
    (filterChanged)="onFilterChanged($event)"
    (escapeKeyPressed)="onEscapePressedInsideChildTable()"
  </value-picker>

</div>
```

Inside the *Smart Components* template the events emitted by *Dumb Components* are handled. It is a good practice to name the handlers with the prefix `on*` (e.g. `onInputChanged()`).

15.2. Services Layer

The services layer is more or less what we call 'business logic layer' on the server side. It is the layer where the business logic is placed. The main challenges are:

- Define application state and an API for the components layer to use it
- Handle application state transitions
- Perform backend interaction (XHR, WebSocket, etc.)
- Handle business logic in a maintainable way
- Configuration management

All parts of the services layer are described in this chapter. An example which puts the concepts together can be found at the end [Interaction of Smart Components through the services layer](#).

15.2.1. Boundaries

There are two APIs for the components layer to interact with the services layer:

- A store can be subscribed to for receiving state updates over time
- A use case service can be called to trigger an action

To illustrate the fact the following figure shows an abstract overview.

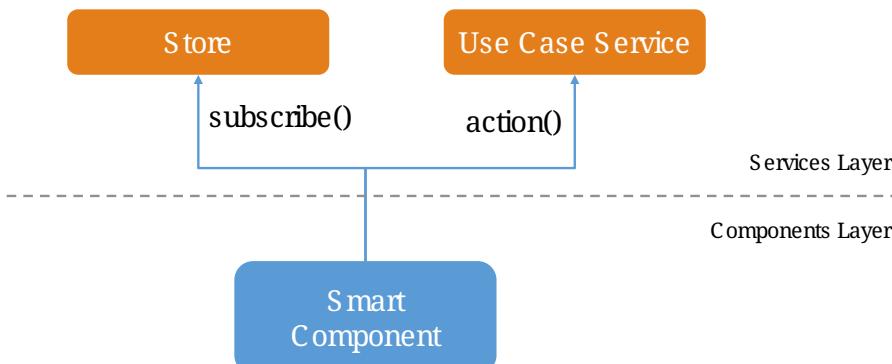


Figure 15. Boundaries to components layer

15.2.2. Store

A store is a class which defines and handles application state with its transitions over time. Interaction with a store is always synchronous. A basic implementation using `rxjs` can look like this.



A more profound implementation taken from a real-life project can be found here ([Abstract Class Store](#)).

Listing 13. Store defined using rxjs

```

@Injectable()
export class ProductSearchStore {

    private stateSource = new
BehaviorSubject<ProductSearchState>(defaultProductSearchState);
    state$ = this.stateSource.asObservable();

    setLoading(isLoading: boolean) {
        const currentState = this.stateSource.getValue();
        this.stateSource.next({
            isLoading: isLoading,
            products: currentState.products,
            searchCriteria: currentState.searchCriteria
        });
    }
}

```

In the example `ProductSearchStore` handles state of type `ProductSearchState`. The public API is the property `state$` which is an observable of type `ProductSearchState`. The state can be changed with method calls. So every desired change to the state needs to be modeled with a method. In reactive terminology this would be an *Action*. The store does not use any services. Subscribing to the `state$` observable leads to the subscribers receiving every new state.

This is basically the *Observer Pattern*:

The store consumer registeres itself to the observable via `state$.subscribe()` method call. The first parameter of `subscribe()` is a callback function to be called when the subject changes. This way the consumer - the observer - is registered. When `next()` is called with a new state inside the store, all callback functions are called with the new value. So every observer is notified of the state change. This equals the *Observer Pattern* push type.

A store is the API for *Smart Components* to receive state from the service layer. State transitions are handled automatically with *Smart Components* registering to the `state$` observable.

15.2.3. Use Case Service

A use case service is a service which has methods to perform asynchronous state transitions. In reactive terminology this would be an *Action of Actions* - a thunk (`redux`) or an effect (`@ngrx`).



Figure 16. Use case services are the main API to trigger state transitions

A use case services method - an action - interacts with adapters, business services and stores. So use case services orchestrate whole use cases. For an example see [use case service example](#).

15.2.4. Adapter

An adapter is used to communicate with the backend. This could be a simple XHR request, a WebSocket connection, etc. An adapter is simple in the way that it does not add anything other than the pure network call. So there is no caching or logging performed here. The following listing shows an example.

For further information on backend interaction see [Consuming REST Services](#)

Listing 14. Calling the backend via an adapter

```

@Injectable()
export class ProductsAdapter {

  private baseUrl = environment.baseUrl;

  constructor(private http: HttpClient) { }

  getAll(): Observable<Product[]> {
    return this.http.get<Product[]>(this.baseUrl + '/products');
  }

}
  
```

15.2.5. Interaction of Smart Components through the services layer

The interaction of smart components is a classic problem which has to be solved in every UI technology. It is basically how one dialog tells the other something has changed.

An example is *adding an item to the shopping basket*. With this action there need to be multiple state updates.

- The small logo showing how many items are currently inside the basket needs to be updated from 0 to 1
- The price needs to be recalculated

- Shipping costs need to be checked
- Discounts need to be updated
- Ads need to be updated with related products
- etc.

Pattern

To handle this interaction in a scalable way we apply the following pattern.

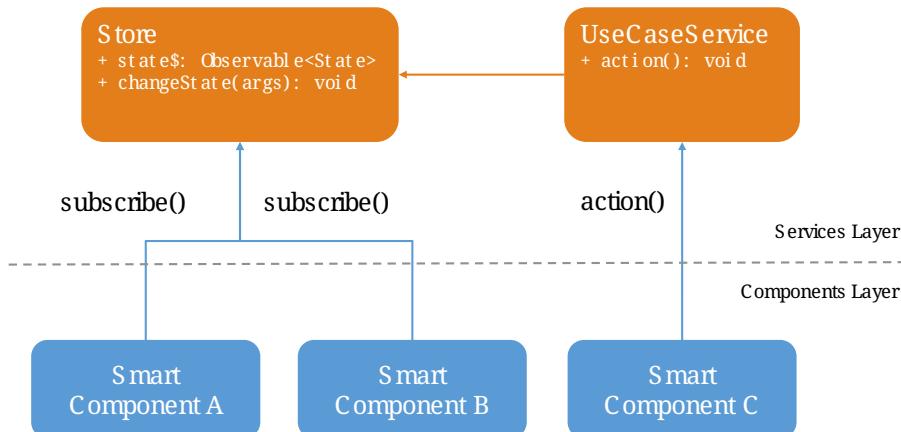


Figure 17. Smart Component interaction

The state of interest is encapsulated inside a store. All *Smart Components* interested in the state have to subscribe to the store's API served by the public observable. Thus, with every update to the store the subscribed components receive the new value. The components basically react to state changes. Altering a store can be done directly if the desired change is synchronous. Most actions are of asynchronous nature so the **UseCaseService** comes into play. Its actions are **void** methods, which implement a use case, i.e., adding a new item to the basket. It calls asynchronous actions and can perform multiple store updates over time.

To put this pattern into perspective the **UseCaseService** is a programmatic alternative to **redux-thunk** or **@ngrx/effects**. The main motivation here is to use the full power of TypeScript's **--strictNullChecks** and to let the learning curve not to become as steep as it would be when learning a new state management framework. This way actions are just **void** method calls.

Example

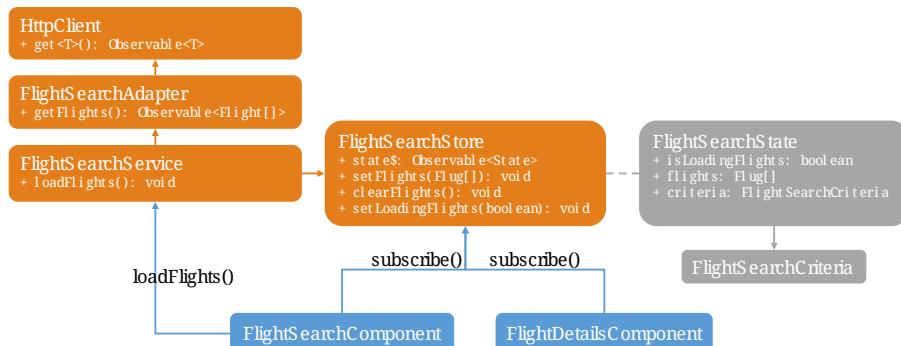


Figure 18. Smart Components interaction example

The example shows two *Smart Components* sharing the **FlightSearchState** by using the

[FlightSearchStore](#). The use case shown is started by an event in the *Smart Component* [FlightSearchComponent](#). The action `loadFlight()` is called. This could be submitting a search form. The UseCaseService is [FlightSearchService](#), which handles the use case *Load Flights*.

UseCaseService example

```
export class FlightSearchService {

    constructor(
        private flightSearchAdapter: FlightSearchAdapter,
        private store: FlightSearchStore
    ) { }

    loadFlights(criteria: FlightSearchCriteria): void {
        this.store.setLoadingFlights(true);
        this.store.clearFlights();

        this.flightSearchAdapter.getFlights(criteria.departureDate,
            {
                from: criteria.departureAirport,
                to: criteria.destinationAirport
            })
            .finally(() => this.store.setLoadingFlights(false))
            .subscribe((result: FlightTo[]) => this.store.setFlights(result, criteria));
    }

}
```

First the loading flag is set to `true` and the current flights are cleared. This leads the *Smart Component* showing a spinner indicating the loading action. Then the asynchronous XHR is triggered by calling the adapter. After completion the loading flag is set to `false` causing the loading indication no longer to be shown. If the XHR was successful, the data would be put into the store. If the XHR was not successful, this would be the place to handle a custom error. All general network issues should be handled in a dedicated class, i.e., an interceptor. So for example the basic handling of 404 errors is not done here.

16. Guides

16.1. Package Managers

There are two major package managers currently used for JavaScript / TypeScript projects which leverage node.js as a build platform.

1. [npm](#)
2. [yarn](#)

Our recommendation is to use yarn but both package managers are fine.



When using npm it is important to use a version greater 5.0 as npm 3 has major drawbacks compared to yarn. The following guide assumes that you are using npm ≥ 5 or yarn.

Before you start reading further, please take a look at the docs:

- [yarn getting started](#)
- [npm getting started](#)

The following guide will describe best practices for working with yarn / npm.

16.1.1. Semantic Versioning

When working with package managers it is very important to understand the concept of [semantic versioning](#).

Table 42. Version example 1.2.3

Version	1.	2.	3
Version name when incrementing	Major (2.0.0)	Minor (1.3.0)	Patch (1.2.4)
Has breaking changes	yes	no	no
Has features	yes	yes	no
Has bugfixes	yes	yes	yes

The table gives an overview of the most important parts of semantic versioning. In the header version 1.2.3 is displayed. The first row shows the name and the resulting version when incrementing a part of the version. The next rows show specifics of the resulting version - e.g. a major version can have breaking changes, features and bugfixes.

Packages from npm and yarn leverage semantic versioning and instead of selecting a fixed version one can specify a selector. The most common selectors are:

- **^1.2.3** At least 1.2.3 - 1.2.4 or 1.3.0 can be used, 2.0.0 can not be used

- **~1.2.3** At least 1.2.3 - 1.2.4 can be used, 2.0.0 and 1.3.0 can not be used
- **>=1.2.3** At least 1.2.3 - every version greater can also be used

This achieves a lower number of duplicates. To give an example:

If package A needs version 1.3.0 of package C and package B needs version 1.4.0 of package C one would end up with 4 packages.

If package A needs version ^1.3.0 of package C and package B needs version 1.4.0 of package C one would end up with 3 packages. A would use the same version of C as B - 1.4.0.

16.1.2. Do not modify package.json and lock files by hand

Dependencies are always added using a yarn or npm command. Altering the package.json, package.json.lock or yarn.lock file by hand is not recommended.

Always use a yarn or npm command to add a new dependency.

Adding the package **express** with yarn to dependencies.

```
yarn add express
```

Adding the package **express** with npm to dependencies.

```
npm install express
```

16.1.3. What does the lock file do

The purpose of files **yarn.lock** and **package-json.lock** is to freeze versions for a short time.

The following problem is solved:

- Developer A upgrades the dependency **express** to fixed version **4.16.3**.
- **express** has sub-dependency **accepts** with version selector **~1.3.5**
- His local **node_modules** folder receives **accepts** in version **1.3.5**
- On his machine everything is working fine
- Afterward version **1.3.6** of **accepts** is published - it contains a major bug
- Developer B now clones the repo and loads the dependencies.
- He receives version **1.3.6** of **accepts** and blames developer A for upgrading to a broken version.

Both **yarn.lock** and **package-json.lock** freeze all the dependencies. For example in **yarn.lock** you will find.

Listing 15. yarn.lock example (excerpt)

```

accepts@~1.3.5:
  version "1.3.5"
  resolved "[...URL to registry]"
  dependencies:
    mime-types "~2.1.18"
    negotiator "0.6.1"

mime-db@~1.33.0:
  version "1.33.0"
  resolved "[...URL to registry]"

mime-types@~2.1.18:
  version "2.1.18"
  resolved "[...URL to registry]"
  dependencies:
    mime-db "~1.33.0"

negotiator@0.6.1:
  version "0.6.1"
  resolved "[...URL to registry]"

```

The described problem is solved by the example yarn.lock file.

- `accepts` is frozen at version `~1.3.5`
- All of its sub-dependencies are also frozen. It needs `mime-types` at version `~2.1.18` which is frozen at `2.1.18`. `mime-types` needs `mime-db` at `~1.33.0` which is frozen at `1.33.0`

Every developer will receive the same versions of every dependency.



You have to make sure all your developers are using the same npm/yarn version - this includes the CI build.

16.2. Package Managers Workflow

16.2.1. Introduction

This document aims to provide you the necessary documentation and sources in order to help you understand the importance of dependencies between packages.

Projects in node.js make use of modules, chunks of reusable code made by other people or teams. These small chunks of reusable code are called packages ^[3]. Packages are used to solve specific problems or tasks. These relations between your project and the external packages are called dependencies.

For example, imagine we are doing a small program that takes your birthday as an input and tells you how many days are left until your birthday. We search in the repository if someone has

published a package to retrieve the actual date and manage date types, and maybe we could search for another package to show a calendar, because we want to optimize our time, and we wish the user to click a calendar button and choose the day in the calendar instead of typing it.

As you can see, packages are convenient. In some cases, they may be even needed, as they can manage aspects of your program you may not be proficient in, or provide an easier use of them.

For more comprehensive information visit [npm definition](#)

Package.json

Dependencies in your project are stored in a file called package.json. Every package.json must contain, at least, the name and version of your project.

Package.json is located in the root of your project.



If package.json is not on your root directory refer to [Problems you may encounter](#) section

If you wish to learn more information about package.json, click on the following links:

- [Yarn Package.json](#)
- [npm Package.json](#)

Content of package.json

As you noticed, package.json is a really important file in your project. It contains essential information about our project, therefore you need to understand what's inside.

The structure of package.json is divided in blocks, inside the first one you can find essential information of your project such as the name, version, license and optionally some [Scripts](#).

```
{
  "name": "exampleproject",
  "version": "0.0.0",
  "license": "MIT",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e"
  }
}
```

The next block is called *dependencies* and contains the packages that project needs in order to be developed, compiled and executed.

```
"private": true,
"dependencies": {
  "@angular/animations": "^4.2.4",
  "@angular/common": "^4.2.4",
  "@angular/forms": "^4.2.4",
  ...
  "zone.js": "^0.8.14"
}
```

After `dependencies` we find `devDependencies`, another kind of dependencies present in the development of the application but unnecessary for its execution. One example is typescript. Code is written in typescript, and then, *transpiled* to javascript. This means the application is not using typescript in execution and consequently not included in the deployment of our application.

```
"devDependencies": {
  "@angular/cli": "1.4.9",
  "@angular/compiler-cli": "^4.2.4",
  ...
  "@types/node": "~6.0.60",
  "typescript": "~2.3.3"
}
```

Having a peer dependency means that your package needs a dependency that is the same exact dependency as the person installing your package

```
"peerDependencies": {
  "package-123": "^2.7.18"
}
```

Optional dependencies are just that: optional. If they fail to install, Yarn will still say the install process was successful.

```
"optionalDependencies": {
  "package-321": "^2.7.18"
}
```

Finally you can have bundled dependencies which are packages bundled together when publishing your package in a repository.

```
{
  "bundledDependencies": [
    "package-4"
  ]
}
```

Here is the link to an in-depth explanation of [dependency types](#).

Scripts

Scripts are a great way of automating tasks related to your package, such as simple build processes or development tools.

For example:

```
{  
  "name": "exampleproject",  
  "version": "0.0.0",  
  "license": "MIT",  
  "scripts": {  
    "build-project": "node hello-world.js",  
  }  
}
```

You can run that script by running the command `yarn (run) script` or `npm run script`, check the example below:

```
$ yarn (run) build-project # run is optional  
$ npm run build-project
```

There are special reserved words for scripts, like `preinstall`, which will execute the script automatically before the package you install are installed.

Chech different uses for scripts in the following links:

- [Yarn scripts documentation](#)
- [npm scripts documentation](#)

Or you can go back to [Content of package.json](#).

Managing dependencies

In order to manage dependencies we recommend using package managers in your projects.

A big reason is their usability. Adding or removing a package is really easy, and by doing so, packet manager update the package.json and copies (or removes) the package in the needed location, with a single comand.

Another reason, closely related to the first one, is reducing human error by automating the package management process.

Two of the package managers you can use in node.js projects are "yarn" and "npm". While you can use both, we encourage you to use only one of them while working on projects. Using both may lead to different dependencies between members of the team.

npm

We'll start by installing npm following this small guide [here](#).

As stated on the web, npm comes inside of node.js, and must be updated after installing node.js, in the same guide you used earlier are written the instructions to update npm.

How npm works

In order to explain how npms works, let's take a command as an example:

```
$ npm install @angular/material @angular/cdk
```

This command tells npm to look for the packages @angular/material and @angular/cdk in the npm registry, download and decompress them in the folder node_modules along with their own dependencies. Additionally, npm will update package.json and create a new file called package-lock.json.

After initializing and installing the first package there will be a new folder called node_modules in your project. This folder is where your packages are unzipped and stored, following a tree scheme.

Take in consideration both npm and yarn need a package.json in the root of your project in order to work properly. If after creating your project don't have it, download again the package.json from the repository or you'll have to start again.

Brief overview of commands

If we need to create a package.json from scratch, we can use the comand **init**. This command asks the user for basic information about the project and creates a brand new package.json.

```
$ npm init
```

Install (or i) installs all modules listed as dependencies in package.json **locally**. You can also specify a package, and install that package. Install can also be used with the parameter **-g**, which tells npm to install the [Global package](#).

```
$ npm install
$ npm i
$ npm install Package
```



Earlier versions of npm did **not** add dependencies to package.json unless it was used with the flag **--save**, so npm install package would be npm install **--save** package, you have one example below.

```
$ npm install --save Package
```

Npm needs flags in order to know what kind of dependency you want in your project, in npm you need to put the flag **-D** or **--save-dev** to install devdependencies, for more information consult the links at the end of this section.

```
$ npm install -D package
$ npm install --save-dev package
```

The next command uninstalls the module you specified in the command.

```
$ npm uninstall Package
```

ls command shows us the dependencies like a nested tree, useful if you have few packages, not so useful when you need a lot of packages.

```
$ npm ls
```

```
npm@VERSION@ /path/to/npm
  └── init-package-json@0.0.4
    └── promzard@0.1.5
```

example tree

We recommend you to learn more about npm commands in the following [link](#), navigating to the section cli commands.

About Package-lock.json

Package-lock.json describes the dependency tree resulting of using package.json and npm. Whenever you update, add or remove a package, package-lock.json is deleted and redone with the new dependencies.

```
"@angular/animations": {
  "version": "4.4.6",
  "resolved": "https://registry.npmjs.org/@angular/animations/-/animations-4.4.6.tgz",
  "integrity": "sha1-+mYYmaik44y3xYPHpc185l1ZKjU=",
  "requires": {
    "tslib": "1.8.0"
  }
}
```

This lock file is checked everytime the command npm i (or npm install) is used without specifying a package, in the case it exists and it's valid, npm will install the exact tree that was generated, such that subsequent installs are able to generate identical dependency trees.



It is **not** recommended to modify this file yourself. It's better to leave its management to npm.

More information is provided by the npm team at [package-lock.json](#)

Yarn

Yarn is an alternative to npm, if you wish to install yarn follow the guide [getting started with yarn](#) and download the correct version for your operative system. Node.js is also needed you can find it [here](#).

Working with yarn

Yarn is used like npm, with small differences in syntax, for example *npm install module* is changed to *yarn add module*.

```
$ yarn add @covalent
```

This command is going to download the required packages, modify package.json, put the package in the folder node_modules and makes a new yarn.lock with the new dependency.

However, unlike npm, yarn maintains a cache with packages you download inside. You don't need to download every file every time you do a general installation. This means installations faster than npm.

Similarly to npm, yarn creates and maintains his own lock file, called yarn.lock. Yarn.lock gives enough information about the project for dependency tree to be reproduced.

yarn commands

Here we have a brief description of yarn's most used commands:

```
$ yarn add Package
$ yarn add --dev Package
```

Adds a package **locally** to use in your package. Adding the flags **--dev** or **-D** will add them to devDependencies instead of the default dependencies, if you need more information check the links at the end of the section.

```
$ yarn init
```

Initializes the development of a package.

```
$ yarn install
```

Installs all the dependencies defined in a package.json file, you can also write "yarn" to achieve the

same effect.

```
$ yarn remove Package
```

You use it when you wish to remove a package from your project.

```
$ yarn global add Package
```

Installs the [Global package](#).

Please, refer to the documentation to learn more about yarn commands and their attributes: [yarn commands](#)

yarn.lock

This file has the same purpose as Package-lock.json, to guide the packet manager, in this case yarn, to install the dependency tree specified in yarn.lock.

Yarn.lock and package.json are essential files when collaborating in a project more co-workers and may be a source of errors if programmers do not use the same manager.

Yarn.lock follows the same structure as package-lock.json, you can find an example of dependency below:

```
"@angular/animations@^4.2.4":  
  version "4.4.6"  
  resolved "https://registry.yarnpkg.com/@angular/animations/-/animations-  
4.4.6.tgz#fa661899a8a4e38cb7c583c7a5c97ce65d592a35"  
  dependencies:  
    tslib "^1.7.1"
```



As with package-lock.json, it's strongly **not** advised to modify this file. Leave its management to yarn

You can learn more about yarn.lock here: [yarn.lock](#)

Global package

Global packages are packages installed in your operative system instead of your local project, global packages useful for developer tooling that is not part of any individual project but instead is used for local commands.

A good example of global package is angular/cli, a command line interface for angular used in our projects. You can install a global package in npm with "npm install -g package" and "yarn global add package" with yarn, you have a npm example below:

Listing 16. npm global package

```
npm install -g @angular/cli
```

[Global npm](#)

[Global yarn](#)

Package version

Dependencies are critical to the success of a package. You must be extra careful about which version packages are using, one package in a different version may break your code.

Versioning in npm and yarn, follows a semantic called semver, following the logic MAJOR.MINOR.PATCH, like for example, @angular/animations: 4.4.6.

Different versions

Sometimes, packages are installed with a different version from the one initially installed. This happens because package.json also contains the range of versions we allow yarn or npm to install or update to, example:

```
"@angular/animations": "^4.2.4"
```

And here the installed one:

```
"@angular/animations": {
  "version": "4.4.6",
  "resolved": "https://registry.npmjs.org/@angular/animations/-/animations-4.4.6.tgz",
  "integrity": "sha1-+mYYmaik44y3xYPHpc185l1ZKjU=",
  "requires": {
    "tslib": "1.8.0"
  }
}
```

As you can see, the version we initially added is 4.2.4, and the version finally installed after a global installation of all packages, 4.4.6.

Installing packages without package-lock.json or yarn.lock using their respective packet managers, will always end with npm or yarn installing the latest version allowed by package.json.

"@angular/animations": "[^]4.2.4" contains not only the version we added, but also the range we allow npm and yarn to update. Here are some examples:

```
"@angular/animations": "<4.2.4"
```

The version installed must be lower than 4.2.4 .

```
"@angular/animations": ">=4.2.4"
```

The version installed must be greater than or equal to 4.2.4 .

```
"@angular/animations": "=4.2.4"
```

the version installed must be equal to 4.2.4 .

```
"@angular/animations": "^4.2.4"
```

The version installed cannot modify the first non zero digit, for example in this case it cannot surpass 5.0.0 or be lower than 4.2.4 .

You can learn more about this in [Versions](#)

Problems you may encounter

If you can't find package.json, you may have deleted the one you had previously, which means you have to download the package.json from the repository. In the case you are creating a new project you can create a new package.json. More information in the links below. Click on [Package.json](#) if you come from that section.

- [Creating new package.json in yarn](#)
- [Creating new package.json in npm](#)



Using npm install or yarn without package.json in your projects will result in compilation errors. As we mentioned earlier, Package.json contains essential information about your project.

If you have package.json, but you don't have package-lock.json or yarn.lock the use of command "npm install" or "yarn" may result in a different dependency tree.

If you are trying to import a module and visual code studio is not able to find it, is usually caused by error adding the package to the project, try to add the module again with yarn or npm, and restart Visual Studio Code.

Be careful with the semantic versioning inside your package.json of the packages, or you may find a new update on one of your dependencies breaking your code.



In the following [link](#) there is a solution to a problematic update to one package.

A list of common errors of npm can be found in: [npm errors](#)

Recomendations

Use yarn **or** npm in your project, reach an agreement with your team in order to choose one, this

will avoid undesired situations like forgetting to upload an updated yarn.lock or package-lock.json. Be sure to have the latest version of your project when possible.



Pull your project every time it's updated. Erase your node_modules folder and reinstall all dependencies. This assures you to be working with the same dependencies your team has.

AD Center recommends the use of yarn.

16.3. Yarn 2

Yarn v2 is a very different software from the v1. The following list contains the main new features:

- [Constraints](#)
- [Offline Cache](#)
- [Plug'n'Play](#)
- [Plugins](#)
- [Protocols](#)
- [Release Workflow](#)
- [Workspaces](#)
- [Zero-Installs](#)

Please, read them carefully to decide if your current project is suitable to use Yarn 2 as package manager.



Some features are still experimental, so please do not use them in production environments.

More info at <https://yarnpkg.com/>

16.3.1. Global Install

Installing Yarn 2.x globally is discouraged as Yarn team is moving to a per-project install strategy. We advise you to keep Yarn 1.x (Classic) as your global binary by installing it via the instructions you can find [here](#).

Once you've followed the instructions (running yarn --version from your home directory should yield something like 1.22.0), go to the next section to see how to enable Yarn 2 on your project.

16.3.2. Per-project install

Follow these instructions to update your current devon4ng project to Yarn 2:

1. Follow the global install instructions.
2. Move into your project folder:

```
cd ~/path/to/project
```

3. Run the following command:

```
yarn policies set-version berry # below v1.22
yarn set version berry          # on v1.22+
```

4. Since Angular CLI still is not fully supported with the new [Yarn architecture](#) as it is not compatible with PnP it is necessary to include the `node-modules` plugin adding the following line in the `.yarnrc.yml` file:

```
nodeLinker: node-modules
```

5. Commit the `.yarn` and `.yarnrc.yml` changes

6. Run again `yarn install`.



For more advanced migration topics please refer to <https://yarnpkg.com/advanced/migration>

16.3.3. Which files should be gitignored?

If you're using Zero-Installs:

```
.yarn/*
!.yarn/cache
!.yarn/releases
!.yarn/plugins
```

If you're not using Zero-Installs:

```
.yarn/*
!.yarn/releases
!.yarn/plugins
.pnp.*
```

More details at <https://yarnpkg.com/advanced/qa#details>

[3] A package is a file or directory that is described by a `package.json`.

17. Angular

17.1. Accessibility

Multiple studies suggest that around 15-20% of the population are living with a disability of some kind. In comparison, that number is higher than any single browser demographic currently, other than Chrome². Not considering those users when developing an application means excluding a large number of people from being able to use it comfortable or at all.

Some people are unable to use the mouse, view a screen, see low contrast text, Hear dialogue or music and some people having difficulty to understanding the complex language. This kind of people needed the support like Keyboard support, screen reader support, high contrast text, captions and transcripts and Plain language support. This disability may change the from permanent to the situation.

17.1.1. Key Concerns of Accessible Web Applications

- **Semantic Markup** - Allows the application to be understood on a more general level rather than just details of what's being rendered
- **Keyboard Accessibility** - Applications must still be usable when using only a keyboard
- **Visual Assistance** - color contrast, focus of elements and text representations of audio and events

Semantic Markup

If you're creating custom element directives, Web Components or HTML in general, use native elements wherever possible to utilize built-in events and properties. Alternatively, use ARIA to communicate semantic meaning.

HTML tags have attributes that provide extra context on what's being displayed on the browser. For example, the img tag's alt attribute lets the reader know what is being shown using a short description. However, native tags don't cover all cases. This is where ARIA fits in. ARIA attributes can provide context on what roles specific elements have in the application or on how elements within the document relate to each other.

A modal component can be given the role of dialog or alertdialog to let the browser know that that component is acting as a modal. The modal component template can use the ARIA attributes aria-labelledby and aria-describedby to describe to readers what the title and purpose of the modal is.

```

@Component({
  selector: 'ngc2-app',
  template: `
    <ngc2-notification-button
      message="Hello!"
      label="Greeting"
      role="button">
    </ngc2-notification-button>
    <ngc2-modal
      [title]="modal.title"
      [description]="modal.description"
      [visible]="modal.visible"
      (close)="modal.close()">
    </ngc2-modal>
  `
})
export class AppComponent {
  constructor(private modal: ModalService) { }
}

```

notification-button.component.ts

```

@Component({
  selector: 'ngc2-modal',
  template: `
    <div
      role="dialog"
      aria-labelledby="modal-title"
      aria-describedby="modal-description">
      <div id="modal-title">{{title}}</div>
      <p id="modal-description">{{description}}</p>
      <button (click)="close.emit()">OK</button>
    </div>
  `
})
export class ModalComponent {
  ...
}

```

Keyboard Accessibility

Keyboard accessibility is the ability of your application to be interacted with using just a keyboard. The more streamlined the site can be used this way, the more keyboard accessible it is. Keyboard accessibility is one of the largest aspects of web accessibility since it targets:

- those with motor disabilities who can't use a mouse
- users who rely on screen readers and other assistive technology, which require keyboard navigation

-
- those who prefer not to use a mouse

Focus

Keyboard interaction is driven by something called focus. In web applications, only one element on a document has focus at a time, and keypresses will activate whatever function is bound to that element. Focus element border can be styled with CSS using the outline property, but it should not be removed. Elements can also be styled using the :focus psuedo-selector.

Tabbing

The most common way of moving focus along the page is through the tab key. Elements will be traversed in the order they appear in the document outline - so that order must be carefully considered during development. There is way change the default behaviour or tab order. This can be done through the tabindex attribute. The tabindex can be given the values: * less than zero - to let readers know that an element should be focusable but not keyboard accessible * 0 - to let readers know that that element should be accessible by keyboard * greater than zero - to let readers know the order in which the focusable element should be reached using the keyboard. Order is calculated from lowest to highest.

Transitions

The majority of transitions that happen in an Angular application will not involve a page reload. This means that developers will need to carefully manage what happens to focus in these cases.

For example:

```

@Component({
  selector: 'ngc2-modal',
  template: `
    <div
      role="dialog"
      aria-labelledby="modal-title"
      aria-describedby="modal-description">
      <div id="modal-title">{{title}}</div>
      <p id="modal-description">{{description}}</p>
      <button (click)="close.emit()">OK</button>
    </div>
  `,
})
export class ModalComponent {
  constructor(private modal: ModalService, private element: ElementRef) { }

  ngOnInit() {
    this.modal.visible$.subscribe(visible => {
      if(visible) {
        setTimeout(() => {
          this.element.nativeElement.querySelector('button').focus();
        }, 0);
      }
    })
  }
}

```

17.1.2. Visual Assistance

One large category of disability is visual impairment. This includes not just the blind, but those who are color blind or partially sighted, and require some additional consideration.

Color Contrast

When choosing colors for text or elements on a website, the contrast between them needs to be considered. For WCAG 2.0 AA, this means that the contrast ratio for text or visual representations of text needs to be at least 4.5:1. There are tools online to measure the contrast ratio such as this color contrast checker from WebAIM or be checked with using automation tests.

Visual Information

Color can help a user's understanding of information, but it should never be the only way to convey information to a user. For example, a user with red/green color-blindness may have trouble discerning at a glance if an alert is informing them of success or failure.

Audiovisual Media

Audiovisual elements in the application such as video, sound effects or audio (ie. podcasts) need related textual representations such as transcripts, captions or descriptions. They also should never

auto-play and playback controls should be provided to the user.

17.1.3. Accessibility with Angular Material

The `a11y` package provides a number of tools to improve accessibility. Import

```
import { A11yModule } from '@angular/cdk/a11y';
```

ListKeyManager

`ListKeyManager` manages the active option in a list of items based on keyboard interaction. Intended to be used with components that correspond to a `role="menu"` or `role="listbox"` pattern . Any component that uses a `ListKeyManager` will generally do three things:

- Create a `@ViewChildren` query for the options being managed.
- Initialize the `ListKeyManager`, passing in the options.
- Forward keyboard events from the managed component to the `ListKeyManager`.

Each option should implement the `ListKeyManagerOption` interface:

```
interface ListKeyManagerOption {  
    disabled?: boolean;  
    getLabel?(): string;  
}
```

Types of ListKeyManager

There are two varieties of `ListKeyManager`, `FocusKeyManager` and `ActiveDescendantKeyManager`.

FocusKeyManager

Used when options will directly receive browser focus. Each item managed must implement the `FocusableOption` interface:

```
interface FocusableOption extends ListKeyManagerOption {  
    focus(): void;  
}
```

ActiveDescendantKeyManager

Used when options will be marked as active via `aria-activedescendant`. Each item managed must implement the `Highlightable` interface:

```
interface Highlightable extends ListKeyManagerOption {
  setActiveStyles(): void;
  setInactiveStyles(): void;
}
```

Each item must also have an ID bound to the listbox's or menu's aria-activedescendant.

FocusTrap

The `cdkTrapFocus` directive traps Tab key focus within an element. This is intended to be used to create accessible experience for components like modal dialogs, where focus must be constrained. This directive is declared in [A11yModule](#).

This directive will not prevent focus from moving out of the trapped region due to mouse interaction.

For example:

```
<div class="my-inner-dialog-content" cdkTrapFocus>
  <!-- Tab and Shift + Tab will not leave this element. -->
</div>
```

Regions

Regions can be declared explicitly with an initial focus element by using the `cdkFocusRegionStart`, `cdkFocusRegionEnd` and `cdkFocusInitial` DOM attributes. When using the tab key, focus will move through this region and wrap around on either end.

For example:

```
<a mat-list-item routerLink cdkFocusRegionStart>Focus region start</a>
<a mat-list-item routerLink>Link</a>
<a mat-list-item routerLink cdkFocusInitial>Initially focused</a>
<a mat-list-item routerLink cdkFocusRegionEnd>Focus region end</a>
```

InteractivityChecker

[InteractivityChecker](#) is used to check the interactivity of an element, capturing disabled, visible, tabbable, and focusable states for accessibility purposes.

LiveAnnouncer

[LiveAnnouncer](#) is used to announce messages for screen-reader users using an aria-live region.

For example:

```
@Component({...})  
export class MyComponent {  
  
  constructor(liveAnnouncer: LiveAnnouncer) {  
    liveAnnouncer.announce("Hey Google");  
  }  
}
```

API reference for Angular CDK a11y

[API reference for Angular CDK a11y](#)

17.2. Angular Elements

17.2.1. What are Angular Elements?

Angular elements are Angular components packaged as custom elements, a web standard for defining new HTML elements in a framework-agnostic way.

Custom elements are a Web Platform feature currently supported by Chrome, Firefox, Opera, and Safari, and available in other browsers through [Polyfills](#). A custom element extends HTML by allowing you to define a tag whose content is created and controlled by JavaScript code. The browser maintains a CustomElementRegistry of defined custom elements (also called Web Components), which maps an instantiable JavaScript class to an HTML tag.

17.2.2. Why use Angular Elements?

Angular Elements allows Angular to work with different frameworks by using input and output elements. This allows Angular to work with many different frameworks if needed. This is an ideal situation if a slow transformation of an application to [Angular](#) is needed or some Angular needs to be added in other web applications(For example. [ASP.net](#), [JSP](#) etc)

17.2.3. Negative points about Elements

Angular Elements is really powerful but since, the transition between views between views is going to be handled by another framework or html/javascript, using Angular [Router](#) is not possible. the view transitions have to be handled manually. This fact also eliminates the possibility of just porting an application completely.

17.2.4. How to use Angular Elements?

In a generalized way, a simple [Angular component](#) could be transformed to an [Angular Element](#) with this steps:

Installing Angular Elements

The first step is going to be install the library using our prefered packet manager:

NPM

```
npm install @angular/elements
```

YARN

```
yarn add @angular/elements
```

Preparing the components in the modules

Inside the `app.module.ts`, in addition to the normal declaration of the components inside `declarations`, the modules inside `imports` and the services inside `providers`, the components need to be added in `entryComponents`. If there are components that have their own module, the same logic is going to be applied for them, only adding in the `app.module.ts` the components that don't have their own module. Here is an example of this:

```
....  
@NgModule({  
    declarations: [  
        DishFormComponent,  
        DishViewComponent  
    ],  
    imports: [  
        CoreModule, // Module containing Angular Materials  
        FormsModule  
    ],  
    entryComponents: [  
        DishFormComponent,  
        DishViewComponent  
    ],  
    providers: [DishShareService]  
})  
....
```

After that is done, the constructor of the module is going to be modified to use injector and bootstrap the application defining the components. This is going to allow the `Angular Element` to get the injections and to define a component tag that will be used later:

```
....  
})  
export class AppModule {  
    constructor(private injector: Injector) {  
  
    }  
  
    ngDoBootstrap() {  
        const el = createCustomElement(DishFormComponent, {injector: this.injector});  
        customElements.define('dish-form', el);  
  
        const elView = createCustomElement(DishViewComponent, {injector: this.injector});  
        customElements.define('dish-view', elView);  
    }  
}  
....
```

A component example

In order to be able to use a component, `@Input()` and `@Output()` variables are used. These variables are going to be the ones that will allow the Angular Element to communicate with the framework/javascript:

Component html

```
<mat-card>
  <mat-grid-list cols="1" rowHeight="100px" rowWidth="50%">
    <mat-grid-tile colspan="1" rowspan="1">
      <span>{{ platename }}</span>
    </mat-grid-tile>
    <form (ngSubmit)="onSubmit(dishForm)" #dishForm="ngForm">
      <mat-grid-tile colspan="1" rowspan="1">
        <mat-form-field>
          <input matInput placeholder="Name" name="name" [(ngModel)]="dish.name">
        </mat-form-field>
      </mat-grid-tile>
      <mat-grid-tile colspan="1" rowspan="1">
        <mat-form-field>
          <textarea matInput placeholder="Description" name="description" [(ngModel)]="dish.description"></textarea>
        </mat-form-field>
      </mat-grid-tile>
      <mat-grid-tile colspan="1" rowspan="1">
        <button mat-raised-button color="primary" type="submit">
          Submit</button>
      </mat-grid-tile>
    </form>
  </mat-grid-list>
</mat-card>
```

Component ts

```

@Component({
  templateUrl: './dish-form.component.html',
  styleUrls: ['./dish-form.component.scss']
})
export class DishFormComponent implements OnInit {

  @Input() platename;

  @Input() platedescription;

  @Output()
  submitDishEvent = new EventEmitter();

  submitted = false;
  dish = {name: '', description: ''};

  constructor(public dishShareService: DishShareService) { }

  ngOnInit() {
    this.dish.name = this.platename;
    this.dish.description = this.platedescription;
  }

  onSubmit(dishForm: NgForm): void {
    this.dishShareService.createDish(dishForm.value.name, dishForm.value.description);
    this.submitDishEvent.emit('dishSubmited');
  }
}

```

In this file there are definitions of multiple variables that will be used as input and output. Since the input variables are going to be used directly by html, only lowercase and underscore strategies can be used for them. On the `onSubmit(dishForm: NgForm)` a service is used to pass this variables to another component. Finally, as a last thing, the selector inside `@Component` has been removed since a tag that will be used dynamically was already defined in the last step.

Solving the error

In order to be able to use this `Angular Element` a `Polyfills/Browser support` related error needs to solved. This error can be solved in two ways:

Changing the target

One solution is to change the target in `tsconfig.json` to `es2015`. This might not be doable for every application since maybe a specific target is required.

Installing Polyfaces

Another solution is to use AutoPollyfill. In order to do so, the library is going to be installed with a

packet manager:

Yarn

```
yarn add @webcomponents/webcomponentsjs
```

Npm

```
npm install @webcomponents/webcomponentsjs
```

After the packet manager has finished, inside the src folder a new file **polyfills.ts** is found. To solve the error, importing the corresponding adapter (**custom-elements-es5-adapter.js**) is necessary:

```
....  
*****  
*****  
* APPLICATION IMPORTS  
*/  
  
import '@webcomponents/webcomponentsjs/custom-elements-es5-adapter.js';  
....
```

If you want to learn more about polyfills in angular you can do it [here](#)

Building the Angular Element

First, before building the **Angular Element**, every element inside that app component except the module need to be removed. After that, a bash script is created in the root folder,. This script will allow to put every necessary file into a js.

```
ng build " projectName " --prod --output-hashing=none && cat  
dist/" projectName "/runtime.js dist/" projectName "/polyfills.js  
dist/" projectName "/scripts.js dist/" projectName "/main.js >  
. /dist/" projectName "/" "nameWantedAngularElement".js
```

After executing the bash script, it will generate inside the path **dist/" projectName "** a js file named **"nameWantedAngularElement".js** and a css file.

Building with **ngx-build-plus** (Recommended)

The library **ngx-build-plus** allows to add different options when building. In addition, it solves some errors that will occur when trying to use multiple angular elements in an application. In order to use it, yarn or npm can be used:

Yarn

```
yarn add ngx-build-plus
```

Npm

```
npm install ngx-build-plus
```

If you want to add it to a specific sub project in your projects folder, use the --project:

```
.... ngx-build-plus --project "project-name"
```

Using this library and the following command, an isolated [Angular Element](#) which won't have conflict with others can be generated. This [Angular Element](#) will not have a polyfill so, the project where we use them will need to include a [polyfill](#) with the [Angular Element](#) requirements.

```
ng build " projectName " --output-hashing none --single-bundle true --prod --bundle  
-styles false
```

This command will generate three things:

1. The main js bundle
2. The script js
3. The css

These files will be used later instead of the single js generated in the last step.

Extra parameters

Here are some extra useful parameters that [ngx-build-plus](#) provides:

- [--keep-polyfills](#): This parameter is going to allow us to keep the polyfills. This needs to be used with caution, avoiding using multiple different polyfills that could cause an error if necessary.
- [--extraWebpackConfig webpack.extra.js](#): This parameter allows us to create a javascript file inside our [Angular Elements](#) project with the name of different libraries. Using [webpack](#) these libraries will not be included in the [Angular Element](#). This is useful to lower the size of our [Angular Element](#) by removing libraries shared. Example:

```
const webpack = require('webpack');

module.exports = {
  "externals": {
    "rxjs": "rxjs",
    "@angular/core": "ng.core",
    "@angular/common": "ng.common",
    "@angular/common/http": "ng.common.http",
    "@angular/platform-browser": "ng.platformBrowser",
    "@angular/platform-browser-dynamic": "ng.platformBrowserDynamic",
    "@angular/compiler": "ng.compiler",
    "@angular/elements": "ng.elements",
    "@angular/router": "ng.router",
    "@angular/forms": "ng.forms"
  }
}
```



If some libraries are excluded from the 'Angular Element' you will need to add the bundled umd files of those libraries manually.

Using the Angular Element

The [Angular Element](#) that got generated in the last step can be used in almost every framework. In this case, the [Angular Element](#) is going to be used in html:

Listing 17. Sample index.html version without ngx-build-plus

```
<html>
  <head>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    <div id="container">

      </div>
      <!--Use of the element non dynamically-->
      <!--<plate-form platename="test" platedescription="test"></plate-form>-->
      <script src="./devon4ngAngularElements.js"> </script>
      <script>
        var elContainer = document.getElementById('container');
        var el= document.createElement('dish-form');
        el.setAttribute('platename','test');
        el.setAttribute('platedescription','test');
        el.addEventListener('submitDishEvent',(ev)=>{
          var elView= document.createElement('dish-view');
          elContainer.innerHTML = '';
          elContainer.appendChild(elView);
        });
        elContainer.appendChild(el);
      </script>
    </body>
</html>
```

Listing 18. Sample index.html version with ngx-build-plus

```

<html>
  <head>
    <link rel="stylesheet" href="styles.css">
  </head>
  <body>
    <div id="container">

      </div>
      <!--Use of the element non dynamically-->
      <!--<plate-form platename="test" platedescription="test"></plate-form>-->
      <script src="./polyfills.js"> </script> <!-- Created using --keep-polyfills
options -->
      <script src="./scripts.js"> </script>
      <script src="./main.js"> </script>
      <script>
        var elContainer = document.getElementById('container');
        var el= document.createElement('dish-form');
        el.setAttribute('platename','test');
        el.setAttribute('platedescription','test');
        el.addEventListener('submitDishEvent',(ev)=>{
          var elView= document.createElement('dish-view');
          elContainer.innerHTML = '';
          elContainer.appendChild(elView);
        });
        elContainer.appendChild(el);
      </script>
    </body>
</html>

```

In this html, the css generated in the last step is going to be imported inside the `<head>` and then, the javascript element is going to be imported at the end of the body. After that is done, There is two uses of **Angular Elements** in the html, one directly whith use of the `@input()` variables as parameters commented in the html:

```

.....
      <!--Use of the element non dynamically-->
      <!--<plate-form platename="test" platedescription="test"></plate-form>-->
.....

```

and one dynamically inside the script:

```
....  

<script>  

    var elContainer = document.getElementById('container');  

    var el= document.createElement('dish-form');  

    el.setAttribute('platename','test');  

    el.setAttribute('platedescription','test');  

    el.addEventListener('submitDishEvent',(ev)=>{  

        var elView= document.createElement('dish-view');  

        elContainer.innerHTML = '';  

        elContainer.appendChild(elView);  

    });  

    elContainer.appendChild(el);  

</script>  

....
```

This javascript is an example of how to create dynamically an **Angular Element** inserting attributed to fill our **@Input()** variables and listen to the **@Output()** that was defined earlier. This is done with:

```
el.addEventListener('submitDishEvent',(ev)=>{  

    var elView= document.createElement('dish-view');  

    elContainer.innerHTML = '';  

    elContainer.appendChild(elView);  

});
```

This allows javascript to hook with the **@Output()** event emitter that was defined. When this event gets called, another component that was defined gets inserted dynamically.

17.2.5. Angular Element within another Angular project

In order to use an **Angular Element** within another **Angular** project the following steps need to be followed:

Copy bundled script and css to resources

First copy the generated **.js** and **.css** inside assets in the corresponding folder.

Add bundled script to angular.json

Inside **angular.json** both of the files that were copied in the last step are going to be included. This will be done both, in **test** and in **build**. Including it on the test, will allow to perform unitary tests.

```
{
  ...
  "architect": {
    ...
    "build": {
      ...
      "styles": [
        ...
        "src/assets/css/devon4ngAngularElements.css"
        ...
      ]
      ...
      "scripts": [
        "src/assets/js/devon4ngAngularElements.js"
      ]
      ...
    }
    ...
    "test": {
      ...
      "styles": [
        ...
        "src/assets/css/devon4ngAngularElements.css"
        ...
      ]
      ...
      "scripts": [
        "src/assets/js/devon4ngAngularElements.js"
      ]
      ...
    }
  }
}
```

By declaring the files in the `angular.json` angular will take care of including them in a proper way.

Using Angular Element

There are two ways that `Angular Element` can be used:

Create component dynamically

In order to add the component in a dynamic way, first adding a container is necessary:

`app.component.html`

```
....  
<div id="container">  
</div>  
....
```

With this container created, inside the `app.component.ts` a method is going to be created. This method is going to find the container, create the dynamic element and append it into the container.

`app.component.ts`

```
export class AppComponent implements OnInit {  
    ....  
    ngOnInit(): void {  
        this.createComponent();  
    }  
    ....  
    createComponent(): void {  
        const container = document.getElementById('container');  
        const component = document.createElement('dish-form');  
        container.appendChild(component);  
    }  
    ....
```

Using it directly

In order to use it directly on the templates, in the `app.module.ts` the `CUSTOM_ELEMENTS_SCHEMA` needs to be added:

```
....  
import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core';  
....  
@NgModule({  
    ....  
    schemas: [ CUSTOM_ELEMENTS_SCHEMA ],
```

This is going to allow the use of the `Angular Element` in the templates directly:

`app.component.html`

```
....  
<div id="container">  
    <dish-form></dish-form>  
</div>
```

17.3. Angular Lazy loading

When the development of an application starts, it just contains a small set of features so the app usually loads fast. However, as new features are added, the overall application size grows up and its loading speed decreases, is in this context where Lazy loading finds its place. Lazy loading is a design pattern that defers initialization of objects until it is needed so, for example, Users that just access to a website's home page do not need to have other areas loaded. Angular handles lazy loading through the routing module which redirect to requested pages. Those pages can be loaded at start or on demand.

17.3.1. An example with Angular

To explain how lazy loading is implemented using angular, a basic sample app is going to be developed. This app will consist in a window named "level 1" that contains two buttons that redirect to other windows in a "second level". It is a simple example, but useful to understand the relation between angular modules and lazy loading.

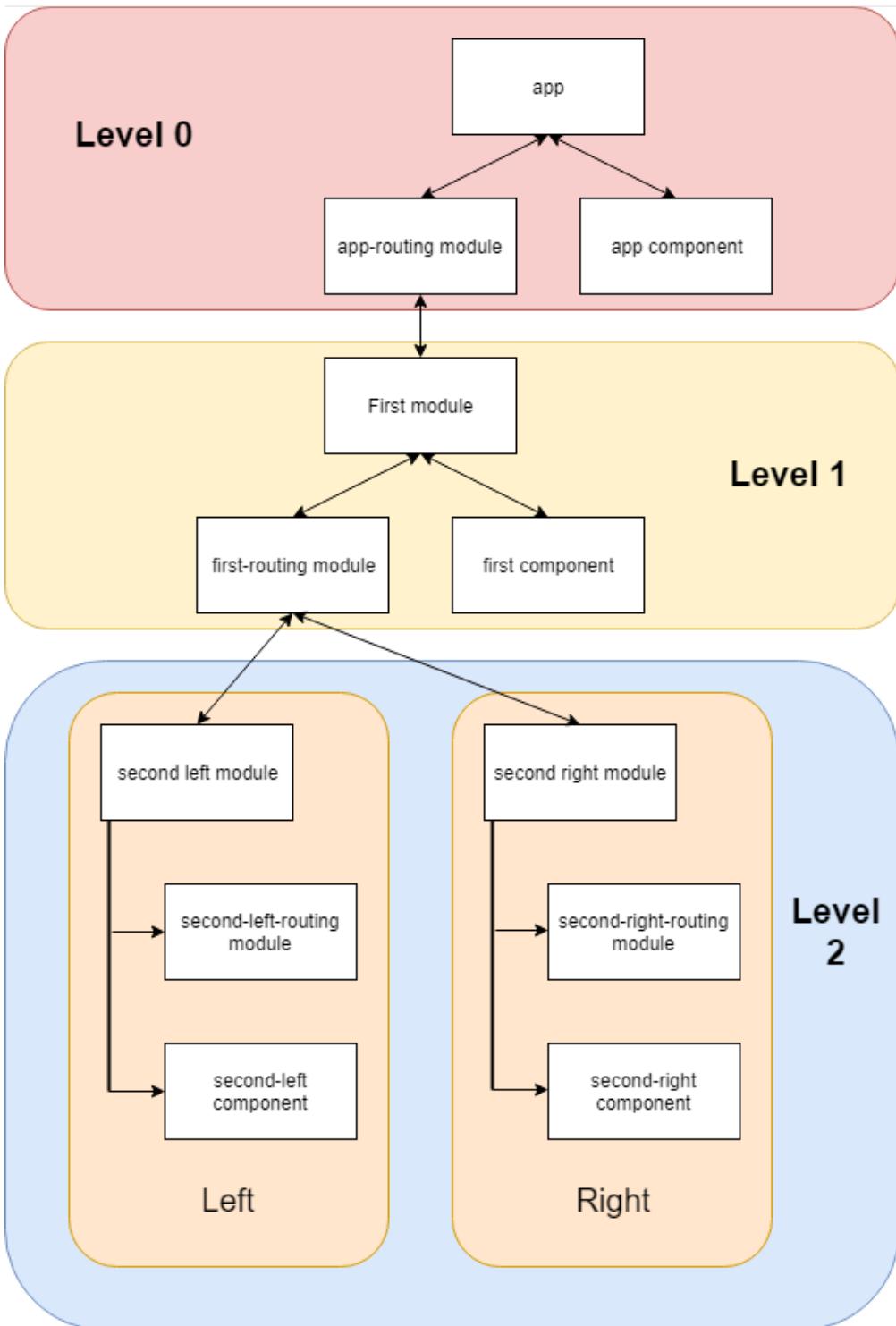


Figure 19. Levels app structure.

This graphic shows that modules acts as gates to access components "inside" them.

Because the objective of this guide is related mainly with logic, the html structure and scss styles are less relevant, but the complete code can be found as a sample [here](#).

Implementation

First write in a console `ng new level-app --routing`, to generate a new project called level-app including an `app-routing.module.ts` file (`--routing` flag).

In the file `app.component.html` delete all the content except the `router-outlet` tag.

Listing 19. File app.component.html

```
<router-outlet></router-outlet>
```

The next steps consists on creating features modules.

run `ng generate module first --routing` to generate a module named *first*.

- run `ng generate module first/second-left --routing` to generate a module named *second-left* under *first*.
- run `ng generate module first/second-right --routing` to generate a module *second-right* under *first*.
- run `ng generate component first/first` to generate a component named *first* inside the module *first*.
- run `ng generate component first/second-left/content` to generate a component *content* inside the module *second-left*.
- run `ng generate component first/second-right/content` to generate a component *content* inside the module *second-right*.

To move between components we have to configure the routes used:

In **app-routing.module.ts** add a path '**first**' to FirstComponent and a redirection from '' to '**first**'.

Listing 20. File app-routing.module.ts.

```
...
import { FirstComponent } from './first/first/first.component';

const routes: Routes = [
  {
    path: 'first',
    component: FirstComponent
  },
  {
    path: '',
    redirectTo: 'first',
    pathMatch: 'full'
  }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {}
```

In **app.module.ts** import the module which includes FirstComponent.

Listing 21. File app.module.ts

```
....  
import { FirstModule } from './first/first.module';  
  
@NgModule({  
  ...  
  imports: [  
    ....  
    FirstModule  
  ],  
  ...  
})  
export class AppModule { }
```

In **first-routing.module.ts** add routes that direct to the content of SecondRightModule and SecondLeftModule. The content of both modules have the same name so, in order to avoid conflicts the name of the components are going to be changed using **as** (original-name as new-name).

Listing 22. File first-routing.module.ts

```
....  
import { ContentComponent as ContentLeft} from './second-left/content/content.component';  
import { ContentComponent as ContentRight} from './second-right/content/content.component';  
import { FirstComponent } from './first/first.component';  
  
const routes: Routes = [  
  {  
    path: '',  
    component: FirstComponent  
  },  
  {  
    path: 'first/second-left',  
    component: ContentLeft  
  },  
  {  
    path: 'first/second-right',  
    component: ContentRight  
  }  
];  
  
@NgModule({  
  imports: [RouterModule.forChild(routes)],  
  exports: [RouterModule]  
})  
export class FirstRoutingModule { }
```

In first.module.ts import SecondLeftModule and SecondRightModule.

Listing 23. File first.module.ts

```

...
import { SecondLeftModule } from './second-left/second-left.module';
import { SecondRightModule } from './second-right/second-right.module';

@NgModule({
  ...
  imports: [
    ...
    SecondLeftModule,
    SecondRightModule,
  ]
})
export class FirstModule { }

```

Using the current configuration, we have a project that loads all the modules in a eager way. Run `ng serve` to see what happens.

First, during the compilation we can see that just a main file is built.

```

$ ng serve
** Angular Live Development Server is listing on localhost:4200, open your browser on http://localhost:4200/ **

Date: 2019-04-15T08:39:57.351Z
Hash: 7f5587d8d8b872e34b80
Time: 14121ms
chunk {es2015-polyfills} es2015-polyfills.js, es2015-polyfills.js.map (es2015-polyfills) 284 kB [initial] [rendered]
chunk {main} main.js, main.js.map (main) 33.4 kB [initial] [rendered] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 236 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.8 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.77 MB [initial] [rendered]
i [wdm]: Compiled successfully.

```

Figure 20. Compile eager.

If we go to `http://localhost:4200/first` and open developer options (F12 on Chrome), it is found that a document named "first" is loaded.



Figure 21. First level eager.

If we click on **[Go to right module]** a second level module opens, but there is no 'second-right' document.



Figure 22. Second level right eager.

But, typing the url directly will load 'second-right' but no 'first', even if we click on **[Go back]**



Figure 23. Second level right eager direct url.

Modifying an angular application to load its modules lazily is easy, you have to change the routing configuration of the desired module (for example FirstModule).

Listing 24. File app-routing.module.ts.

```
const routes: Routes = [
  {
    path: 'first',
    loadChildren: () => import('./first/first.module').then(m => m.FirstModule),
  },
  {
    path: '',
    redirectTo: 'first',
    pathMatch: 'full',
  },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
})
export class AppRoutingModule {}
```

Notice that instead of loading a component, you dynamically import it in a `loadChildren` attribute because modules acts as gates to access components "inside" them. Updating the app to load lazily has four consequences:

1. No component attribute.
2. No import of FirstComponent.
3. FirstModule import has to be removed from the imports array at app.module.ts.

4. Change of context.

If we check `first-routing.module.ts` again, we can see that the path for `ContentLeft` and `ContentRight` is set to '`first/second-left`' and '`first/second-right`' respectively, so writing '`http://localhost:4200/first/second-left`' will redirect us to `ContentLeft`. However, after loading a module with `loadChildren` setting the path to '`second-left`' and '`second-right`' is enough because it acquires the context set by `AppRoutingModule`.

Listing 25. File first-routing.module.ts

```
const routes: Routes = [
  {
    path: '',
    component: FirstComponent
  },
  {
    path: 'second-left',
    component: ContentLeft
  },
  {
    path: 'second-right',
    component: ContentRight
  }
];
```

If we go to 'first' then FirstModule is situated in '/first' but also its children ContentLeft and ContentRight, so it is not necessary to write in their path 'first/second-left' and 'first/second-right', because that will situate the components on 'first/first/second-left' and 'first/first/second-right'.



Figure 24. First level lazy wrong path.

When we compile an app with lazy loaded modules, files containing them will be generated

```
$ ng serve
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

Date: 2019-04-15T10:14:55.962Z
Hash: ef4fb28d33a264038a08
Time: 15058ms
chunk {es2015-polyfills} es2015-polyfills.js, es2015-polyfills.js.map (es2015-polyfills) 284 kB [initial] [rendered]
chunk {first-first-module} first-first-module.js, first-first-module.js.map (first-first-module) 22.4 kB [rendered]
chunk {main} main.js, main.js.map (main) 10.7 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 236 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 8.78 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.8 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.77 MB [initial] [rendered]
i [wdm]: Compiled successfully.
i [wdm]: Compiling...
```

Figure 25. First level lazy compilation.

And if we go to *developer tools* → *network*, we can find those modules loaded (if they are needed).

The screenshot shows a web browser window with the URL `localhost:4200/first`. The main content area displays a large red Angular logo with a white 'A' inside a pentagon, and two buttons at the bottom: 'Go to left module' and 'Go to right module'. The developer tools Network tab is active, showing a list of resources loaded from the server. The resources listed are: websocket, /sockjs-node/846/iehjyl5q; first (highlighted in blue); runtime.js; polyfills.js; styles.js; vendor.js; main.js; and first-first-module.js (highlighted in blue). The Network tab also includes a timeline at the top and detailed request and response headers on the right side.

Figure 26. First level lazy.

To load the component `ContentComponent` of `SecondLeftModule` lazily, we have to load `SecondLeftModule` as a children of `FirstModule`:

- Change **component** to **loadChildren** and reference `SecondLeftModule`.

Listing 26. File first-routing.module.ts.

```
const routes: Routes = [
  {
    path: '',
    component: FirstComponent
  },
  {
    path: 'second-left',
    loadChildren: () => import('./second-left/second-left.module').then(m =>
m.SecondLeftModule),
  },
  {
    path: 'second-right',
    component: ContentRight
  }
];
```

- Remove SecondLeftModule at first.component.ts
- Route the components inside SecondLeftModule. Without this step nothing would be displayed.

Listing 27. File second-left-routing.module.ts.

```
...
import { ContentComponent } from './content/content.component';

const routes: Routes = [
  {
    path: '',
    component: ContentComponent
  }
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})
export class SecondLeftRoutingModule { }
```

- run **ng serve** to generate files containing the lazy modules.

```
$ ng serve
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

Date: 2019-04-15T11:52:45.590Z
Hash: 64f55cca37225803551d
Time: 12632ms
chunk {es2015-polyfills} es2015-polyfills.js, es2015-polyfills.js.map (es2015-polyfills) 284 kB [initial] [rendered]
chunk {first-first-module} first-first-module.js, first-first-module.js.map (first-first-module) 14.8 kB [rendered]
chunk {main} main.js, main.js.map (main) 10.9 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 236 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 8.84 kB [entry] [rendered]
chunk {second-left-second-left-module} second-left-second-left-module.js, second-left-second-left-module.js.map (second-left-second-left-module) 7.14 kB [rendered]
chunk {styles} styles.js, styles.js.map (styles) 16.8 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.77 MB [initial] [rendered]
i [wdm]: Compiled successfully.
i [wdm]: Compiling...
```

Figure 27. Second level lazy loading compilation.

Clicking on **[Go to left module]** triggers the load of SecondLeftModule.



Figure 28. Second level lazy loading network.

17.3.2. Conclusion

Lazy loading is a pattern useful when new features are added, these features are usually identified as modules which can be loaded only if needed as shown in this document, reducing the time spent loading an application.

17.4. Angular Library

Angular CLI provides us with methods that allow the creation of a library. After that, using either packet manager (`npm` or `yarn`) the library can be build and packed which will allow later to install/publish it.

17.4.1. Whats a library?

From [wikipedia](#): a library is a collection of non-volatile resources used by computer programs, often for software development. These may include configuration data, documentation, help data, message templates, pre-written code and subroutines, classes, values or type specifications.

17.4.2. How to build a library

In this section, a library is going to be build step by step.

Creating an empty application

First, using Angular CLI we are going to generate a empty application which will be later filled with the generated library. In order to do so, Angular CLI allows us to add to `ng new "application-name"` an option `--create-application`. This option is going to tell Angular CLI not to create the initial app project. This is convenient since a library is going to be generated in later steps. Using this command `ng new "application-name" --create-application=false` an empty project with the name wanted is created.

```
ng new "application-name" --create-application=false
```

Generating a library

After generating an empty application, a library is going to be generated. Inside the folder of the project, the Angular CLI command `ng generate library "library-name"` is going to generate the library as a project (`projects/"library-name"`). As an addition, the option `--prefix="library-prefix-wanted"` allows us to switch the default prefix that Angular generated with (`lib`). Using the option to change the prefix the command will look like this `ng generate library "library-name" --prefix="library-prefix-wanted"`.

```
ng generate library "library-name" --prefix="library-prefix-wanted"
```

Generating/Modifying in our library

In the last step we generated a library. This generates automatically a `module`, `service` and `component` inside (`projects/"library-name"`) that we can modify adding new methods, components etc that we want to use in other projects. We can generate other elements, using the usual Angular CLI generate commands adding the option `--project="library-name"` is going to allow to generate elements within our project . An example of this is: `ng generate service "name" --project="library-name"`.

```
ng generate "element" "name" --project="library-name"
```

Exporting the generated things

Inside the library (`projects/"library-name"`) theres a `public_api.ts` which is the file that exports the elements inside the library. In case we generated other things, that file needs to be modified adding the extra exports with the generated elements. In addition, changing the library version is possible in the file `package.json`.

Building our library

Once we added the necessary exports, in order to use the library in other applications, we need to build the library. The command `ng build "library-name"` is going to build the library, generating in `"project-name"/dist/"library-name"` the necessary files.

```
ng build "library-name"
```

Packing the library

In this step we are going to pack the build library. In order to do so, we need to go inside `dist/"library-name"` and then run either `npm pack` or `yarn pack` to generate a `"library-name-version.tgz"` file.

Listing 28. Packing using npm

```
npm pack
```

Listing 29. Packing using yarn

```
yarn pack
```

Publishing to npm repository (optional)

- Add a `README.md` and `LICENSE` file. The text inside `README.md` will be used in you npm package web page as documentation.
- run `npm adduser` if you do not have a npm account to create it, otherwise run `npm login` and introduce your credentials.
- run `npm publish` inside `dist/"library-name"` folder.
- Check that the library is published: <https://npmjs.com/package/library-name>

Installing our library in other projects

In this step we are going to install/add the library on other projects.

npm

In order to add the library in other applications, there are two ways:

- **Option 1:** From inside the application where the library is going to get used, using the command `npm install "path-to-tgz"/"library-name-version.tgz"` allows us to install the `.tgz` generated in [Packing the library](#).
- **Option 2:** run `npm install "library-name"` to install it from npm repository.

yarn

To add the package using yarn:

- **Option 1:** From inside the application where the library is going to get used, using the command `yarn add "path-to-tgz"/"library-name-version.tgz"` allows us to install the `.tgz` generated in [Packing the library](#).
- **Option 2:** run `yarn add "library-name"` to install it from npm repository.

Using the library

Finally, once the library was installed with either packet manager, you can start using the elements from inside like they would be used in a normal element inside the application. Example `app.component.ts`:

```
import { Component, OnInit } from '@angular/core';
import { MyLibraryService } from 'my-library';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent implements OnInit {

  toUpper: string;

  constructor(private myLibraryService: MyLibraryService) {}
  title = 'devon4ng library test';
  ngOnInit(): void {
    this.toUpper = this.myLibraryService.firstLetterToUpper('test');
  }
}
```

Example `app.component.html`:

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  
</div>
<h2>Here is my library service being used: {{toUpperCase}}</h2>
<lib-my-library></lib-my-library>
```

Example `app.module.ts`:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

import { MyLibraryModule } from 'my-library';
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    MyLibraryModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

The result from using the library:

Welcome to devon4ng library test!



Here is my library service being used: Test

my-library works!

devon4ng libraries

In [devonfw/devon4ng-library](#) you can find some useful libraries:

- **Authorization module:** This devon4ng Angular module adds rights-based authorization to your Angular app.
- **Cache module:** Use this devon4ng Angular module when you want to cache requests to server. You may configure it to store in cache only the requests you need and to set the duration you want.

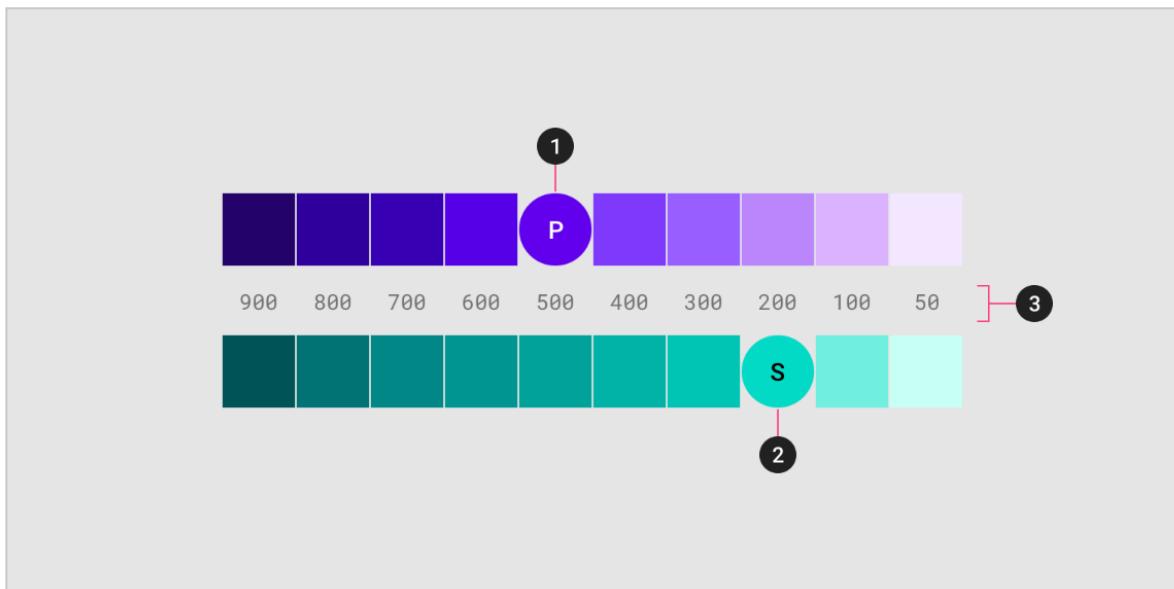
17.5. Angular Material Theming

Angular Material library offers UI components for developers, those components follows Google Material design baselines but characteristics like colors can be modified in order to adapt them to the needs of the client: corporative colors, corporate identity, dark themes, ...

17.5.1. Theming basics

In Angular Material, a theme is created mixing multiple colors. Colors and its light and dark variants conform a **palette**. In general, a theme consists of the following palettes:

- **primary:** Most used across screens and components.
- **accent:** Floating action button and interactive elements.
- **warn:** Error state.
- **foreground:** Text and icons.
- **background:** Element backgrounds.



A sample primary and secondary palette

1. Primary color indicator
2. Secondary color indicator
3. Light and dark variants

Figure 29. Palettes and variants.

In angular material, a palette is represented as a scss map.

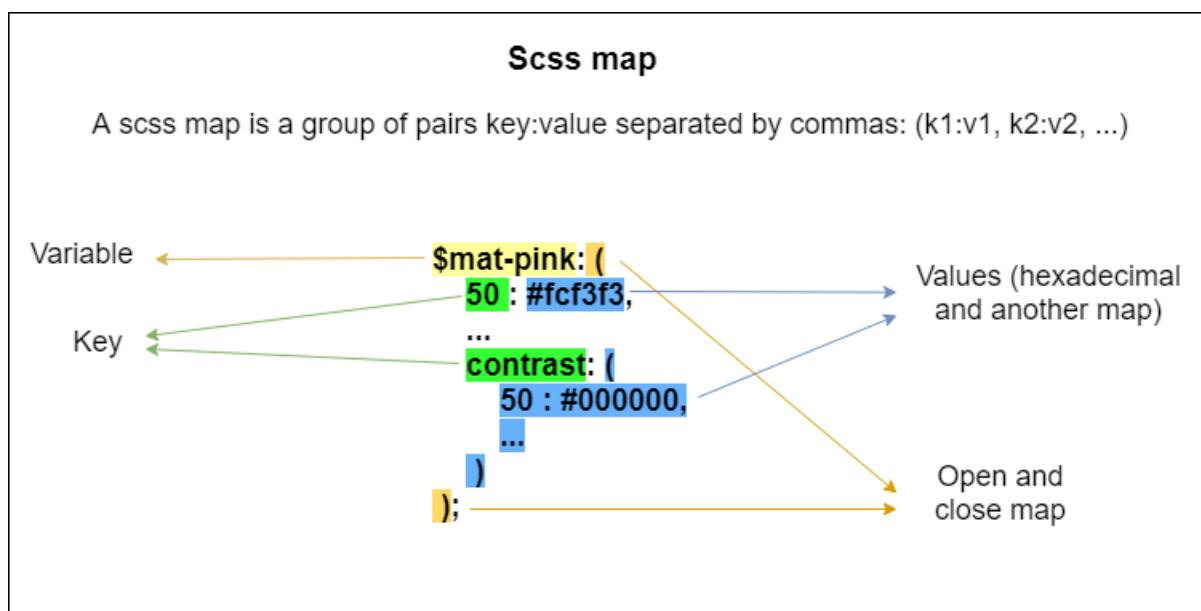


Figure 30. Scss map and palettes.



Some components can be forced to use primary, accent or warn palettes using the attribute **color**, for example: <mat-toolbar color="primary">.

17.5.2. Prebuilt themes

Available prebuilt themes:

- deeppurple-amber.css



Figure 31. *deeppurple-amber* theme.

- `indigo-pink.css`



Figure 32. *indigo-pink* theme.

- `pink-bluegrey.css`



Figure 33. *ink-bluegrey* theme.

- `purple-green.css`

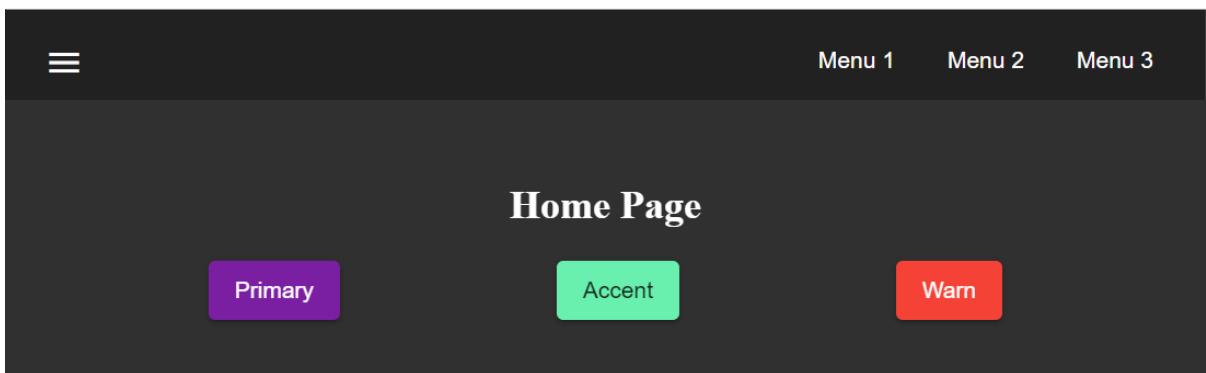


Figure 34. purple-green theme.

The prebuilt themes can be added using `@import`.

```
@import '@angular/material/prebuilt-themes/deeppurple-amber.css';
```

17.5.3. Custom themes

Sometimes prebuild themes do not meet the needs of a project, because color schemas are too specific or do not incorporate branding colors, in those situations custom themes can be built to offer a better solution to the client.

For this topic, we are going to use a basic layout project that can be found in [devon4ng repository](#).

Basics

Before starting writing custom themes, there are some necessary things that have to be mentioned:

- Add a default theme: The project mentioned before has just one global scss stylesheet `styles.scss` that includes `indigo-pink.scss` which will be the default theme.
- Add `@import '~@angular/material/theming'`; at the begining of the every stylesheet to be able to use angular material prebuilt color palettes and functions.
- Add `@include mat-core(); once` per project, so if you are writing multiple themes in multiple files you could import those files from a 'central' one (for example `styles.scss`). This includes all common styles that are used by multiple components.

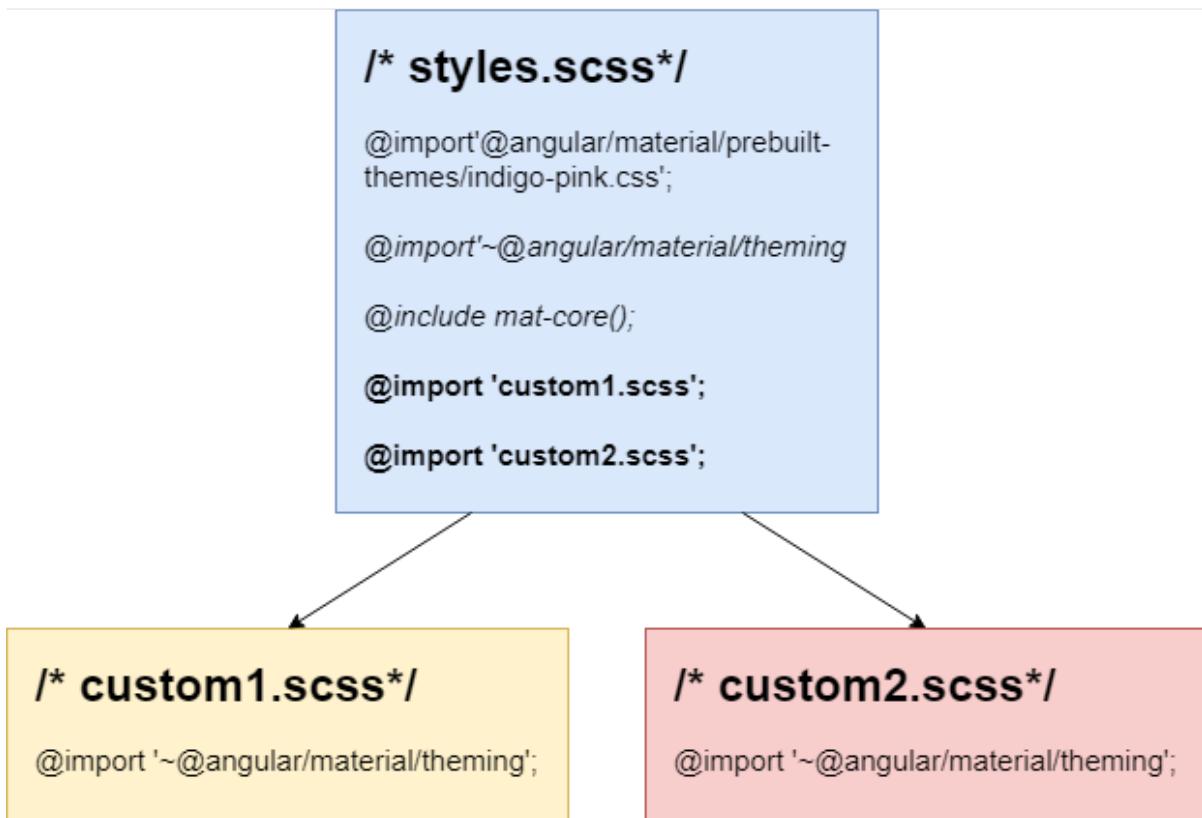


Figure 35. Theme files structure.

Basic custom theme

To create a new custom theme, the .scss file containing it has to have imported the angular_theming.scss file (angular/material/theming) file and mat-core included. _theming.scss includes multiple color palettes and some functions that we are going to see below. The file for this basic theme is going to be named **styles-custom-dark.scss**.

First, declare new variables for primary, accent and warn palettes. Those variables are going to store the result of the function **mat-palette**.

mat-palette accepts four arguments: base color palette, main, lighter and darker variants (See [\[id_palette_variants\]](#)) and returns a new palette including some additional map values: default, lighter and darker ([\[id_scss_map\]](#)). Only the first argument is mandatory.

Listing 30. File *styles-custom-dark.scss*.

```

$custom-dark-theme-primary: mat-palette($mat-pink);
$custom-dark-theme-accent: mat-palette($mat-blue);
$custom-dark-theme-warn: mat-palette($mat-red);
);

```

In this example we are using colors available in _theming.scss: mat-pink, mat-blue, mat-red. If you want to use a custom color you need to define a new map, for instance:

Listing 31. File styles-custom-dark.scss custom pink.

```
$my-pink: (
  50 : #fcf3f3,
  100 : #f9e0e0,
  200 : #f5cccc,
  300 : #f0b8b8,
  500 : #ea9999,
  900 : #db6b6b,
  A100 : #ffffff,
  A200 : #ffffff,
  A400 : #ffeaea,
  A700 : #ffd0d0,
  contrast: (
    50 : #000000,
    100 : #000000,
    200 : #000000,
    300 : #000000,
    900 : #000000,
    A100 : #000000,
    A200 : #000000,
    A400 : #000000,
    A700 : #000000,
  )
);
$custom-dark-theme-primary: mat-palette($my-pink);
...
```



Some pages allows to create these palettes easily, for instance:
<http://mcg.mbitson.com>

Until now, we just have defined primary, accent and warn palettes but what about foreground and background? Angular material has two functions to change both:

- **mat-light-theme:** Receives as arguments primary, accent and warn palettes and return a theme whose foreground is basically black (texts, icons, ...), the background is white and the other palettes are the received ones.



Figure 36. Custom light theme.

- **mat-dark-theme:** Similar to mat-light-theme but returns a theme whose foreground is basically white and background black.

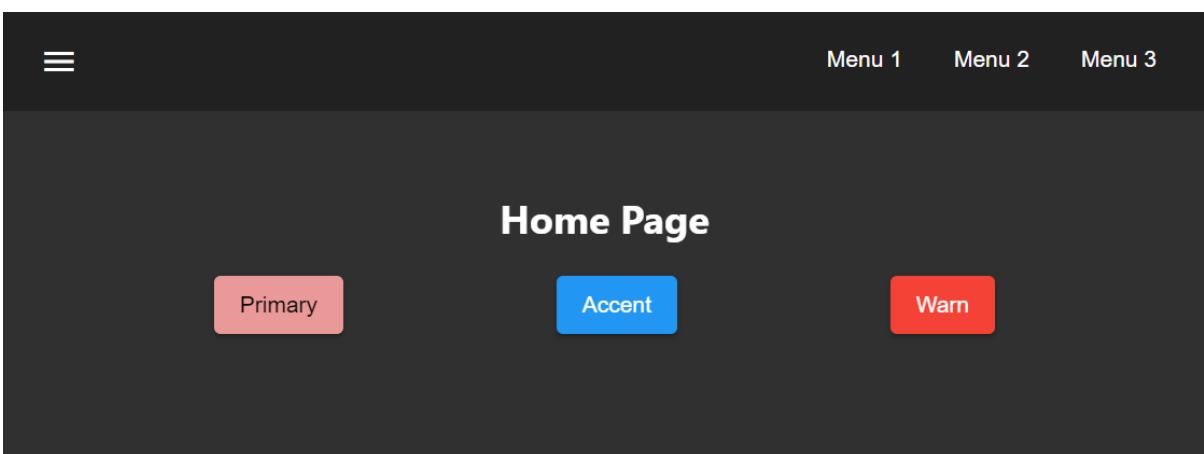


Figure 37. Custom dark theme.

For this example we are going to use mat-dark-theme and save its result in \$custom-dark-theme.

Listing 32. File styles-custom-dark.scss updated with mat-dark-theme.

```
...
$custom-dark-theme: mat-dark-theme(
  $custom-dark-theme-primary,
  $custom-dark-theme-accent,
  $custom-dark-theme-warn
);
```

To apply the saved theme, we have to go to **styles.scss** and import our **styles-custom-dark.scss** and include a function called **angular-material-theme** using the theme variable as argument.

Listing 33. File styles.scss.

```
...
@import 'styles-custom-dark.scss';
@include angular-material-theme($custom-dark-theme);
```

If we have multiple themes it is necessary to add the include statement inside a css class and use it in **src/index.html** → **app-root component**.

Listing 34. File styles.scss updated with custom-dark-theme class.

```
...
@import 'styles-custom-dark.scss';

.custom-dark-theme {
  @include angular-material-theme($custom-dark-theme);
}
```

Listing 35. File src/index.html.

```
...
<app-root class="custom-dark-theme"></app-root>
...
```

This will apply **\$custom-dark-theme** theme for the entire application.

Full custom theme

Sometimes it is needed to custom different elements from background and foreground, in those situations we have to create a new function similar to *mat-light-theme* and *mat-dark-theme*. Let's focus con *mat-light-theme*:

Listing 36. Source code of mat-light-theme

```
@function mat-light-theme($primary, $accent, $warn: mat-palette($mat-red)) {
  @return (
    primary: $primary,
    accent: $accent,
    warn: $warn,
    is-dark: false,
    foreground: $mat-light-theme-foreground,
    background: $mat-light-theme-background,
  );
}
```

As we can se, *mat-light-theme* takes three arguments and returs a map including them as primary, accent and warn color; but there are three more keys in that map: is-dark, foreground and background.

- **is-dark**: Boolean true if it is a dark theme, false otherwise.
- **background**: Map that stores the color for multiple background elements.
- **foreground**: Map that stores the color for multiple foreground elements.

To show which elements can be colored lets create a new theme in a file **styles-custom-cap.scss**:

Listing 37. File styles-custom-cap.scss: Background and foreground variables.

```
@import '~@angular/material/theming';

// custom background and foreground palettes
$my-cap-theme-background: (
  status-bar: #0070ad,
  app-bar: map_get($mat-blue, 900),
  background: #12abdb,
  hover: rgba(white, 0.04),
  card: map_get($mat-red, 800),
  dialog: map_get($mat-grey, 800),
  disabled-button: $white-12-opacity,
  raised-button: map-get($mat-grey, 800),
  focused-button: $white-6-opacity,
  selected-button: map_get($mat-grey, 900),
  selected-disabled-button: map_get($mat-grey, 800),
  disabled-button-toggle: black,
  unselected-chip: map_get($mat-grey, 700),
  disabled-list-option: black,
);

$my-cap-theme-foreground: (
  base: yellow,
  divider: $white-12-opacity,
  dividers: $white-12-opacity,
  disabled: rgba(white, 0.3),
  disabled-button: rgba(white, 0.3),
  disabled-text: rgba(white, 0.3),
  hint-text: rgba(white, 0.3),
  secondary-text: rgba(white, 0.7),
  icon: white,
  icons: white,
  text: white,
  slider-min: white,
  slider-off: rgba(white, 0.3),
  slider-off-active: rgba(white, 0.3),
);
```

Function which uses the variables defined before to create a new theme:

Listing 38. File styles-custom-cap.scss: Creating a new theme function.

```
// instead of creating a theme with mat-light-theme or mat-dark-theme,
// we will create our own theme-creating function that lets us apply our own
foreground and background palettes.
@function create-my-cap-theme($primary, $accent, $warn: mat-palette($mat-red)) {
  @return (
    primary: $primary,
    accent: $accent,
    warn: $warn,
    is-dark: false,
    foreground: $my-cap-theme-foreground,
    background: $my-cap-theme-background
  );
}
```

Calling the new function and storing its value in **\$custom-cap-theme**.

Listing 39. File styles-custom-cap.scss: Storing the new theme.

```
// We use create-my-cap-theme instead of mat-light-theme or mat-dark-theme
$custom-cap-theme-primary: mat-palette($mat-green);
$custom-cap-theme-accent: mat-palette($mat-blue);
$custom-cap-theme-warn: mat-palette($mat-red);

$custom-cap-theme: create-my-cap-theme(
  $custom-cap-theme-primary,
  $custom-cap-theme-accent,
  $custom-cap-theme-warn
);
```

After defining our new theme, we can import it from styles.scss.

Listing 40. File styles.scss updated with custom-cap-theme class.

```
...
@import 'styles-custom-cap.scss';
.custom-cap-theme {
  @include angular-material-theme($custom-cap-theme);
}
```

Multiple themes and overlay-based components

Certain components (e.g. menu, select, dialog, etc.) that are inside of a global overlay container, require an additional step to be affected by the theme's css class selector.

Listing 41. File app.module.ts

```
import {OverlayContainer} from '@angular/cdk/overlay';

@NgModule({
  // ...
})
export class AppModule {
  constructor(overlayContainer: OverlayContainer) {
    overlayContainer.getContainerElement().classList.add('custom-cap-theme');
  }
}
```

17.5.4. Useful resources

- [Angular Material's oficial theming guide](#)
- [Material Desing: Color theme creation](#)
- [Palette generator](#)
- [SCSS tutorial](#)

17.6. Angular Progressive Web App

Progresive web applications (PWAs) are web application that offer better user experience than the traditional ones. In general, they solve problems related with reliability and speed:

- *Reliability*: PWAs are stable. In this context stability means than even with slow connections or even with no network at all, the application still works. To achieve this, some basic resources like styles, fonts, requests, ... are stored; due to this caching, it is not possible to assure that the content is always up-to-date.
- *Speed*: When an users opens an application, he or she will expect it to load almost inmediately (almost 53% of users abandon sites that take longer than 3 seconds, source: <https://developers.google.com/web/progressive-web-apps/#fast>).

PWAs uses a script called [service worker](#), which runs in background and essentially act as proxy between web app and network, intercepting requests and acting depending on the network conditions.

17.6.1. Assumptions

This guide assumes that you already have installed:

- Node.js
- npm package manager
- Angular CLI

17.6.2. Sample Application



Figure 38. Basic angular PWA.

To explain how to build PWAs using angular, a basic application is going to be built. This app will be able to ask for resources and save in the cache in order to work even offline.

Step 1: Create a new project and install Angular PWA package

This step can be completed with one simple command: `ng new <name>`, where `<name>` is the name for the app. In this case, the app is going to be named **basic-ng-pwa**.

Step 2: Create a service

Web applications usually uses external resources, making necessary the addition of services which can get those resources. This application gets a dish from My Thai Star's back-end and shows it. To do so, a new service is going to be created.

- go to project folder: `cd basic-ng-pwa`
- run `ng generate service data`
- Modify `data.service.ts`, `environment.ts`, `environment.prod.ts`

To retrieve data with this service, you have to import the module `HttpClient` and add it to the service's constructor. Once added, use it to create a function `getDishes()` that sends http request to My Thai Start's back-end. The URL of the back-end can be stored as an environment variable `MY_THAI_STAR_DISH`.

`data.service.ts`

```
...
import { HttpClient } from '@angular/common/http';
import { MY_THAI_STAR_DISH } from '../environments/environment';
...

export class DataService {
  constructor(private http: HttpClient) {}

  /* Get data from Back-end */
  getDishes() {
    return this.http.get(MY_THAI_STAR_DISH);
  }
  ...
}
```

environments.ts

```
...
export const MY_THAI_STAR_DISH =
  'http://de-mucdevondepl01:8090/api/services/rest/dishmanagement/v1/dish/1';
...
```

environments.prod.ts

```
...
export const MY_THAI_STAR_DISH =
  'http://de-mucdevondepl01:8090/api/services/rest/dishmanagement/v1/dish/1';
...
```

Step 3: Use the service

The component AppComponent implements the interface OnInit and inside its method ngOnInit() the suscription to the services is done. When a dish arrives, it is saved and shown (app.component.html).

```

...
import { DataService } from './data.service';
export class AppComponent implements OnInit {
  dish: { name: string; description: string } = { name: '', description: ''};

...
ngOnInit() {
  this.data
    .getDishes()
    .subscribe(
      (dishToday: { dish: { name: string; description: string } }) => {
        this.dish = {
          name: dishToday.dish.name,
          description: dishToday.dish.description,
        };
      },
    );
}
}

```

Step 4: Structures, styles and updates

This step shows code interesting inside the sample app. The complete content can be found in [devon4ng samples](#).

index.html

To use the Montserrat font add the following link inside the tag header.

```
<link href="https://fonts.googleapis.com/css?family=Montserrat" rel="stylesheet">
```

styles.scss

```

body {
  ...
  font-family: 'Montserrat', sans-serif;
}

```

app.component.ts

This file is also used to reload the app if there are any changes.

- *SwUpdate*: This object comes inside the @angular/pwa package and it is used to detect changes and reload the page if needed.

```

...
import { SwUpdate } from '@angular/service-worker';

export class AppComponent implements OnInit {

...
constructor(updates: SwUpdate, private data: DataService) {
    updates.available.subscribe((event) => {
        updates.activateUpdate().then(() => document.location.reload());
    });
}
...
}

```

Step 5: Make it Progressive.

Install Angular PWA package with `ng add @angular/pwa --project <name>`. As before substitute `name` with **basic-ng-pwa**.

The above command completes the following actions:

1. Adds the `@angular/service-worker` package to your project.
2. Enables service worker build support in the CLI.
3. Imports and registers the service worker in the app module.
4. Updates the `index.html` file:
 - Includes a link to add the `manifest.json` file.
 - Adds meta tags for theme-color.
 - Installs icon files to support the installed Progressive Web App (PWA).
 - Creates the service worker configuration file called `ngsw-config.json`, which specifies the caching behaviors and other settings.

`manifest.json`

`manifest.json` is a file that allows to control how the app is displayed in places where native apps are displayed.

Fields

`name`: Name of the web application.

`short_name`: Short version of name.

`theme_color`: Default theme color for an application context.

`background_color`: Expected background color of the web application.

`display`: Preferred display mode.

scope: Navigation scope of this web application's application context.

start_url: URL loaded when the user launches the web application.

icons: Array of icons that serve as representations of the web app.

Additional information can be found [here](#).

ngsw-config.json

ngsw-config.json specifies which files and data URLs have to be cached and updated by the Angular service worker.

Fields

- *index*: File that serves as index page to satisfy navigation requests.
- *assetGroups*: Resources that are part of the app version that update along with the app.
 - *name*: Identifies the group.
 - *installMode*: How the resources are cached (prefetch or lazy).
 - *updateMode*: Caching behaviour when a new version of the app is found (prefetch or lazy).
 - *resources*: Resources to cache. There are three groups.
 - *files*: Lists patterns that match files in the distribution directory.
 - *urls*: URL patterns matched at runtime.
- *dataGroups*: UsefulIdentifies the group. for API requests.
 - *name*: Identifies the group.
 - *urls*: URL patterns matched at runtime.
 - *version*: Indicates that the resources being cached have been updated in a backwards-incompatible way.
 - *cacheConfig*: Policy by which matching requests will be cached
 - *maxSize*: The maximum number of entries, or responses, in the cache.
 - *maxAge*: How long responses are allowed to remain in the cache.
 - *d*: days. (5d = 5 days).
 - *h*: hours
 - *m*: minutes
 - *s*: seconds. (5m20s = 5 minutes and 20 seconds).
 - *u*: milliseconds
 - *timeout*: How long the Angular service worker will wait for the network to respond before using a cached response. Same dataformat as maxAge.
 - *strategy*: Caching strategies (performance or freshness).
- *navigationUrls*: List of URLs that will be redirected to the index file.

Additional information can be found [here](#).

Step 6: Configure the app

manifest.json

Default configuration.

ngsw-config.json

At *assetGroups* → *resources* → *urls*: In this field the google fonts api is added in order to use Montserrat font even without network.

```
"urls": [
    "https://fonts.googleapis.com/**"
]
```

At the root of the json: A data group to cache API calls.

```
{
  ...
  "dataGroups": [
    {
      "name": "mythaistar-dishes",
      "urls": [
        "http://de-mucdevondepl01:8090/api/services/rest/dishmanagement/v1/dish/1"
      ],
      "cacheConfig": {
        "maxSize": 100,
        "maxAge": "1h",
        "timeout": "10s",
        "strategy": "freshness"
      }
    }
  ]
}
```

Step 7: Check that your app is a PWA

To check if an app is a PWA lets compare its normal behaviour against itself but built for production. Run in the project's root folder the commands below:

`ng build --prod` to build the app using production settings.

`npm install http-server` to install an npm module that can serve your built application. Documentation [here](#).

Go to the `dist/basic-ng-pwa/` folder running `cd dist/basic-ng-pwa`.

`http-server -o` to serve your built app.

```
http-server stopped.

C:\Proyectos\devon4ng-projects\sample-ng\basic-ng-pwa\dist\basic-ng-pwa (master -> origin)
λ http-server -o
Starting up http-server, serving .
Available on:
  http://10.247.207.240:8081
  http://192.168.56.1:8081
  http://192.168.99.1:8081
  http://192.168.0.164:8081
  http://127.0.0.1:8081
Hit CTRL-C to stop the server
[Fri Apr 05 2019 12:54:29 GMT+0200 (GMT+02:00)] "GET /manifest.json" "Mozilla/5.0 (Windows NT 0.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36"
[Fri Apr 05 2019 12:54:31 GMT+0200 (GMT+02:00)] "GET /ngsw-worker.js" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36"
[Fri Apr 05 2019 12:54:35 GMT+0200 (GMT+02:00)] "GET /ngsw.json?ngsw-cache-bust=0.712252914348008" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36"
[Fri Apr 05 2019 12:54:35 GMT+0200 (GMT+02:00)] "GET /ngsw.json?ngsw-cache-bust=0.667599063365541" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36"
```

Figure 39. Http server running on localhost:8081.

In another console instance run `ng serve` to open the common app (not built).

```
C:\Proyectos\devon4ng-projects\sample-ng
λ cd basic-ng-pwa\

C:\Proyectos\devon4ng-projects\sample-ng\basic-ng-pwa (master -> origin)
λ ng serve
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
93% after chunk asset optimization SourceMapDevToolPlugin es2015-polyfills.js generate SourceMapDevToolPlugin styles.js map (styles) 17 kB [initial] [rendered]
Date: 2019-04-05T10:57:44.773Z
Hash: d173d05cc6a017858872
Time: 16642ms
chunk {es2015-polyfills} es2015-polyfills.js, es2015-polyfills.js.map (es2015-polyfills) 284 kB [initial] [rendered]
chunk {main} main.js, main.js.map (main) 16.9 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 236 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.08 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 17 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.86 MB [initial] [rendered]
i [wdm]: Compiled successfully.
```

Figure 40. Angular server running on localhost:4200.

The first difference can be found on *Developer tools* → *application*, here it is seen that the PWA application (left) has a service worker and the common (right) one does not.



Figure 41. Application service worker comparison.

If the "offline" box is checked, it will force a disconnection from network. In situations where users do not have connectivity or have a slow, one the PWA can still be accessed and used.



Figure 42. Offline application.

Finally, browser extensions like [Lighthouse](#) can be used to test whether an application is progressive or not.



Figure 43. Lighthouse report.

17.7. APP_INITIALIZER

17.7.1. What is the APP_INITIALIZER pattern

The APP_INITIALIZER pattern allows an application to choose which configuration is going to be used in the start of the application, this is useful because it allows to setup different configurations, for example, for docker or a remote configuration. This provides benefits since this is done on **runtime**, so there's no need to recompile the whole application to switch from configuration.

17.7.2. What is APP_INITIALIZER

APP_INITIALIZER allows to provide a service in the initialization of the application in a `@NgModule`. It also allows to use a factory, allowing to create a singleton in the same service. An example can be found in MyThaiStar `/core/config/config.module.ts`:



The provider expects the return of a `Promise`, if it is using Observables, a change with the method `toPromise()` will allow a switch from `Observable` to `Promise`

```
import { NgModule, APP_INITIALIZER } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

import { ConfigService } from './config.service';

@NgModule({
  imports: [HttpClientModule],
  providers: [
    ConfigService,
    {
      provide: APP_INITIALIZER,
      useFactory: ConfigService.factory,
      deps: [ConfigService],
      multi: true,
    },
  ],
})
export class ConfigModule {}
```

This is going to allow the creation of a [ConfigService](#) where, using a singleton, the service is going to load an external config depending on a route. This dependence with a route, allows to setup different configuration for docker etc. This is seen in the [ConfigService](#) of MyThaiStar:

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Config, config } from './config';

@Injectable()
export class ConfigService {
  constructor(private httpClient: HttpClient) {}

  static factory(appLoadService: ConfigService) {
    return () => appLoadService.loadExternalConfig();
  }

  // this method gets external configuration calling /config endpoint
  //and merges into config object
  loadExternalConfig(): Promise<any> {
    if (!environment.loadExternalConfig) {
      return Promise.resolve({});
    }

    const promise = this.httpClient
      .get('/config')
      .toPromise()
      .then((settings) => {
        Object.keys(settings || {}).forEach((k) => {
          config[k] = settings[k];
        });
        return settings;
      })
      .catch((error) => {
        return 'ok, no external configuration';
      });
  }

  return promise;
}

getValues(): Config {
  return config;
}
}

```

As it is mentioned earlier, you can see the use of a factory to create a singleton at the start. After that, `loadExternalConfig` is going to look for a boolean inside the corresponding environment file inside the path `src/environments/`, this boolean `loadExternalConfig` is going to easily allow to switch to a external config. If it is true, it generates a promise that overwrites the parameters of the local config, allowing to load the external config. Finally, the last method `getValues()` is going to allow to return the file config with the values (overwritten or not). The local `config` file from MyThaiStar can be seen here:

```
export enum BackendType {
```

```
IN_MEMORY,
REST,
GRAPHQL,
}

interface Role {
  name: string;
  permission: number;
}

interface Lang {
  label: string;
  value: string;
}

export interface Config {
  version: string;
  backendType: BackendType;
  restPathRoot: string;
  restServiceRoot: string;
  pageSizes: number[];
  pageSizesDialog: number[];
  roles: Role[];
  langs: Lang[];
}

export const config: Config = {
  version: 'dev',
  backendType: BackendType.REST,
  restPathRoot: 'http://localhost:8081/mythaistar/',
  restServiceRoot: 'http://localhost:8081/mythaistar/services/rest/',
  pageSizes: [8, 16, 24],
  pageSizesDialog: [4, 8, 12],
  roles: [
    { name: 'CUSTOMER', permission: 0 },
    { name: 'WAITER', permission: 1 },
  ],
  langs: [
    { label: 'English', value: 'en' },
    { label: 'Deutsch', value: 'de' },
    { label: 'Español', value: 'es' },
    { label: 'Català', value: 'ca' },
    { label: 'Français', value: 'fr' },
    { label: 'Nederlands', value: 'nl' },
    { label: 'Хълбоки', value: 'hi' },
    { label: 'Polski', value: 'pl' },
    { label: 'Русский', value: 'ru' },
    { label: 'български', value: 'bg' },
  ],
};
```

Finally, inside a environment file `src/environments/environment.ts` the use of the boolean `loadExternalConfig` is seen:

```
// The file contents for the current environment will overwrite these during build.
// The build system defaults to the dev environment which uses 'environment.ts', but
// if you do
// 'ng build --env=prod' then 'environment.prod.ts' will be used instead.
// The list of which env maps to which file can be found in '.angular-cli.json'.

export const environment: {
  production: boolean;
  loadExternalConfig: boolean;
} = { production: false, loadExternalConfig: false };
```

17.7.3. Creating a APP_INITIALIZER configuration

This section is going to be used to create a new `APP_INITIALIZER` basic example. For this, a basic app with angular is going to be generated using `ng new "appname"` substituting `appname` for the name of the app choosed.

17.7.4. Setting up the config files

Docker external configuration (Optional)

This section is only done if theres a docker configuration in the app you are setting up this type of configuration.

1.- Create in the root folder `/docker-external-config.json`. This external config is going to be used when the application is loaded with docker (if the boolean to load the external configuration is set to true). Here you need to add all the config parameter you want to load with docker:

```
{
  "version": "docker-version"
}
```

2.- In the root, in the file `/Dockerfile` angular is going to copy the `docker-external-config.json` that was created before into the nginx html route:

```
....  
COPY docker-external-config.json /usr/share/nginx/html/docker-external-config.json  
....
```

External json configuration

1.- Create a json file in the route `/src/external-config.json`. This external config is going to be used when the application is loaded with the start script (if the boolean to load the external

configuration is set to true). Here you need to add all the config parameter you want to load:

```
{
  "version": "external-config"
}
```

2.- The file named `/angular.json` located at the root is going to be modified to add the file `external-config.json` that was just created to both "assets" inside `Build` and `Test`:

```
.....
"build": {
  .....
  "assets": [
    "src/assets",
    "src/data",
    "src/favicon.ico",
    "src/manifest.json",
    "src/external-config.json"
  ]
  .....
  "test": {
    .....
    "assets": [
      "src/assets",
      "src/data",
      "src/favicon.ico",
      "src/manifest.json",
      "src/external-config.json"
    ]
  }
  .....
}
```

17.7.5. Setting up the proxies

This step is going to setup two proxies. This is going to allow to load the config desired by the context, in case that it is using docker to load the app or in case it loads the app with angular. Loading different files is made possible by the fact that the `ConfigService` method `loadExternalConfig()` looks for the path `/config`.

Docker (Optional)

1.- This step is going to be for docker. Add `docker-external-config.json` to nginx configuration (`/nginx.conf`) that is in the root of the application:

```
....  
location ~ ^/config {  
    alias /usr/share/nginx/html/docker-external-config.json;  
}  
....
```

External Configuration

1.- Now the file `/proxy.conf.json`, needs to be created/modified this file can be found in the root of the application. In this file you can add the route of the external configuration in `target` and the name of the file in `^/config`:

```
....  
"/config": {  
    "target": "http://localhost:4200",  
    "secure": false,  
    "pathRewrite": {  
        "^\u002fconfig": "/external-config.json"  
    }  
}  
....
```

2.- The file `package.json` found in the root of the application is gonna use the start script to load the proxy config that was just created:

```
"scripts": {  
....  
    "start": "ng serve --proxy-config proxy.conf.json -o",  
....
```

17.7.6. Adding the `loadExternalConfig` boolean to the environments

In order to load an external config we need to add the `loadExternalConfig` boolean to the environments. To do so, inside the folder `environments/` the files are going to get modified adding this boolean to each environment that is going to be used. In this case, only two environments are going to be modified (`environment.ts` and `environment.prod.ts`). Down below theres an example of the modification being done in the `environment.prod.ts`:

```
export const environment: {  
    production: boolean;  
    loadExternalConfig: boolean;  
} = { production: false, loadExternalConfig: false };
```

In the file in first instance theres the declaration of the types of the variables. After that, theres the definition of those variables. This variable `loadExternalConfig` is going to be used by the service,

allowing to setup a external config just by switching the `loadExternalConfig` to true.

17.7.7. Creating core configuration service

In order to create the whole configuration module three are going to be created:

- 1.- Create in the core `app/core/config/` a `config.ts`

```
export interface Config {  
    version: string;  
}  
  
export const config: Config = {  
    version: 'dev'  
};
```

Taking a look to this file, it creates a interface (`Config`) that is going to be used by the variable that exports (`export const config: Config`). This variable `config` is going to be used by the service that is going to be created.

- 2.- Create in the core `app/core/config/` a `config.service.ts`:

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Config, config } from './config';

@Injectable()
export class ConfigService {
  constructor(private httpClient: HttpClient) {}

  static factory(appLoadService: ConfigService) {
    return () => appLoadService.loadExternalConfig();
  }

  // this method gets external configuration calling /config endpoint
  // and merges into config object
  loadExternalConfig(): Promise<any> {
    if (!environment.loadExternalConfig) {
      return Promise.resolve({});
    }

    const promise = this.httpClient
      .get('/config')
      .toPromise()
      .then((settings) => {
        Object.keys(settings || {}).forEach((k) => {
          config[k] = settings[k];
        });
        return settings;
      })
      .catch((error) => {
        return 'ok, no external configuration';
      });
  }

  return promise;
}

getValues(): Config {
  return config;
}
}

```

As it was explained in previous steps, at first, there is a factory that uses the method `loadExternalConfig()`, this factory is going to be used in later steps in the module. After that, the `loadExternalConfig()` method checks if the boolean in the environment is false. If it is false it will return the promise resolved with the normal config. Else, it is going to load the external config in the path (`/config`), and overwrite the values from the external config to the config that's going to be used by the app, this is all returned in a promise.

3.- Create in the core a module for the config `app/core/config` a `config.module.ts`:

```

import { NgModule, APP_INITIALIZER } from '@angular/core';
import { HttpClientModule } from '@angular/common/http';

import { ConfigService } from './config.service';

@NgModule({
  imports: [HttpClientModule],
  providers: [
    ConfigService,
    {
      provide: APP_INITIALIZER,
      useFactory: ConfigService.factory,
      deps: [ConfigService],
      multi: true,
    },
  ],
})
export class ConfigModule {}

```

As seen earlier, the `ConfigService` is added to the module. In this addition, the app is initialized(`provide`) and it uses the factory that was created in the `ConfigService` loading the config with or without the external values depending on the boolean in the `config`.

Using the Config Service

As a first step, in the file `/app/app.module.ts` the `ConfigModule` created earlier in the other step is going to be imported:

```

imports: [
  ....
  ConfigModule,
  ....
]

```

After that, the `ConfigService` is going to be injected into the `app.component.ts`

```

.....
import { ConfigService } from './core/config/config.service';
.....
export class AppComponent {
  ....
  constructor(public configService: ConfigService) { }
  ....
}

```

Finally, for this demonstration app, the component `app/app.component.html` is going to show the version of the config it is using at that moment.

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
</div>
<h2>Here is the configuration version that is using angular right now:
{{configService.getValues().version}}</h2>
```

Final steps

The script `start` that was created earlier in the `package.json` (`npm start`) is going to be used to start the application. After that, modifying the boolean `loadExternalConfig` inside the corresponding environment file inside `/app/environments/` should show the different config versions.

Welcome to Devon4ngAppInitializer!

Here is the configuration version that is using angular right now: dev

```
export interface Config {
  version: string;
  loadExternalConfig: boolean;
}

export const config: Config = {
  version: 'dev',
  loadExternalConfig: false,
};
```

Welcome to Devon4ngAppInitializer!

Here is the configuration version that is using angular right now: external-config

```
export interface Config {
  version: string;
  loadExternalConfig: boolean;
}

export const config: Config = {
  version: 'dev',
  loadExternalConfig: true,
};
```

17.8. Component Decomposition

When implementing a new requirement there are a few design decisions, which need to be considered. A decomposition in *Smart* and *Dumb Components* should be done first. This includes the definition of state and responsibilities. Implementing a new dialog will most likely be done by defining a new *Smart Component* with multiple *Dumb Component* children.

In the component tree this would translate to the definition of a new subtree.

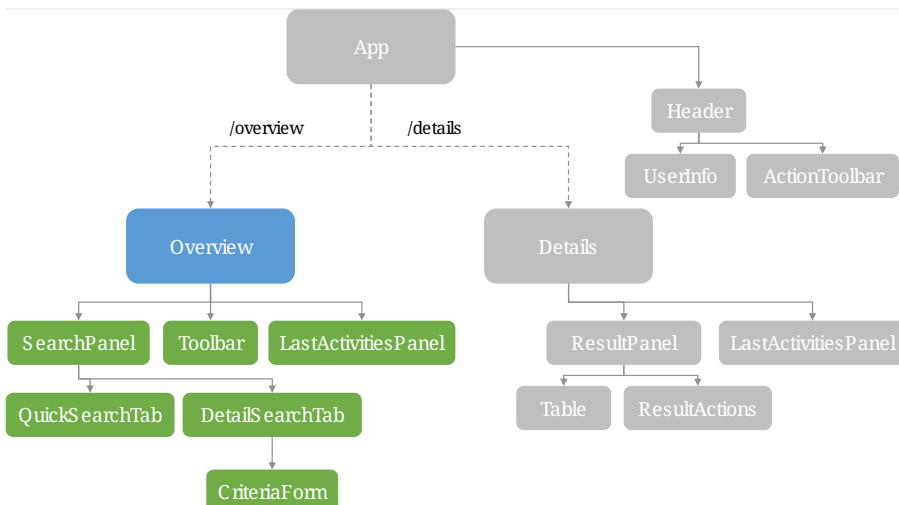


Figure 44. Component Tree with highlighted subtree

17.8.1. Defining Components

The following gives an example for component decomposition. Shown is a screenshot from a styleguide to be implemented. It is a widget called **Listpicker**.

The basic function is an **input** field accepting direct input. So typing **otto** puts **otto** inside the **FormControl**. With arrow down key or by clicking the icon displayed in the inputs right edge a dropdown is opened. Inside possible values can be selected and filtered beforehand. After pressing arrow down key the focus should move into the filter input field. Up and down arrow keys can be used to select an element from the list. Typing into the filter input field filters the list from which the elements can be selected. The current selected element is highlighted with green background color.



Figure 45. Component decomposition example before

What should be done, is to define small reusable *Dumb Components*. This way the complexity becomes manageable. In the example every colored box describes a component with the purple box being a *Smart Component*.



Figure 46. Component decomposition example after

This leads to the following component tree.



Figure 47. Component decomposition example component tree

Note the uppermost component is a *Dumb Component*. It is a wrapper for the label and the component to be displayed inside a form. The *Smart Component* is [Listpicker](#). This way the widget can be reused without a form needed.

A widgets is a typical *Smart Component* to be shared across feature modules. So the [SharedModule](#) is the place for it to be defined.

17.8.2. Defining state

Every UI has state. There are different kinds of state, for example

- View State: e.g. is a panel open, a css transition pending, etc.
- Application State: e.g. is a payment pending, current URL, user info, etc.
- Business Data: e.g. products loaded from backend

It is good practice to base the component decomposition on the state handled by a component and to define a simplified state model beforehand. Starting with the parent - the *Smart Component*:

- What overall state does the dialog have: e.g. loading, error, valid data loaded, valid input, invalid input, etc. Every defined value should correspond to an overall appearance of the whole dialog.
- What events can occur to the dialog: e.g. submitting a form, changing a filter, pressing buttons, pressing keys, etc.

For every *Dumb Component*:

- What data does a component display: e.g. a header text, user information to be displayed, a loading flag, etc.

This will be a slice of the overall state of the parent *Smart Component*. In general a *Dumb Component* presents a slice of its parent *Smart Components* state to the user.

- What events can occur: keyboard events, mouse events, etc.

These events are all handled by its parent *Smart Component* - every event is passed up the tree to be handled by a *Smart Component*.

These information should be reflected inside the modeled state. The implementation is a TypeScript type - an interface or a class describing the model.

So there should be a type describing all state relevant for a *Smart Component*. An instance of that type is send down the component tree at runtime. Not every *Dumb Component* will need the whole state. For instance a single *Dumb Component* could only need a single string.

The state model for the previous [Listpicker](#) example is shown in the following listing.

Listing 42. Listpicker state model

```
export class ListpickerState {

  items: {}[]|undefined;
  columns = ['key', 'value'];
  keyColumn = 'key';
  displayValueColumn = 'value';
  filteredItems: {}[]|undefined;
  filter = '';
  placeholder = '';
  caseSensitive = true;
  isDisabled = false;
  isDropdownOpen = false;
  selectedItem: {}|undefined;
  displayValue = '';

}
```

Listpicker holds an instance of **ListpickerState** which is passed down the component tree via **@Input()** bindings in the *Dumb Components*. Events emitted by children - *Dumb Components* - create a new instance of **ListpickerState** based on the current instance and the event and its data. So a state transition is just setting a new instance of **ListpickerState**. Angular Bindings propagate the value down the tree after exchanging the state.

Listing 43. Listpicker State transition

```
export class ListpickerComponent {

  // initial default values are set
  state = new ListpickerState();

  /** User changes filter */
  onFilterChange(filter: string): void {
    // apply filter ...
    const filteredList = this.filterService.filter(...);

    // important: A new instance is created, instead of altering the existing one.
    // This makes change detection easier and prevents hard to find bugs.
    this.state = Object.assign({}, this.state, {
      filteredItems: filteredList,
      filter: filter
    });
  }

}
```

Note:

It is not always necessary to define the model as independent type. So there would be no state

property and just properties for every state defined directly in the component class. When complexity grows and state becomes larger this is usually a good idea. If the state should be shared between *Smart Components* a store is to be used.

17.8.3. When are Dumb Components needed

Sometimes it is not necessary to perform a full decomposition. The architecture does not enforce it generally. What you should keep in mind is, that there is always a point when it becomes recommendable.

For example a template with 800 loc is:

- not understandable
- not maintainable
- not testable
- not reusable

So when implementing a template with more than 50 loc you should think about decomposition.

17.9. Consuming REST services

A good introduction to working with Angular HttpClient can be found in [Angular Docs](#)

This guide will cover, how to embed Angular HttpClient in the application architecture. For backend request a special service with the suffix **Adapter** needs to be defined.

17.9.1. Defining Adapters

It is a good practice to have a Angular service whose single responsibility is to call the backend and parse the received value to a transfer data model (e.g. Swagger generated TOs). Those services need to have the suffix **Adapter** to make them easy to recognize.

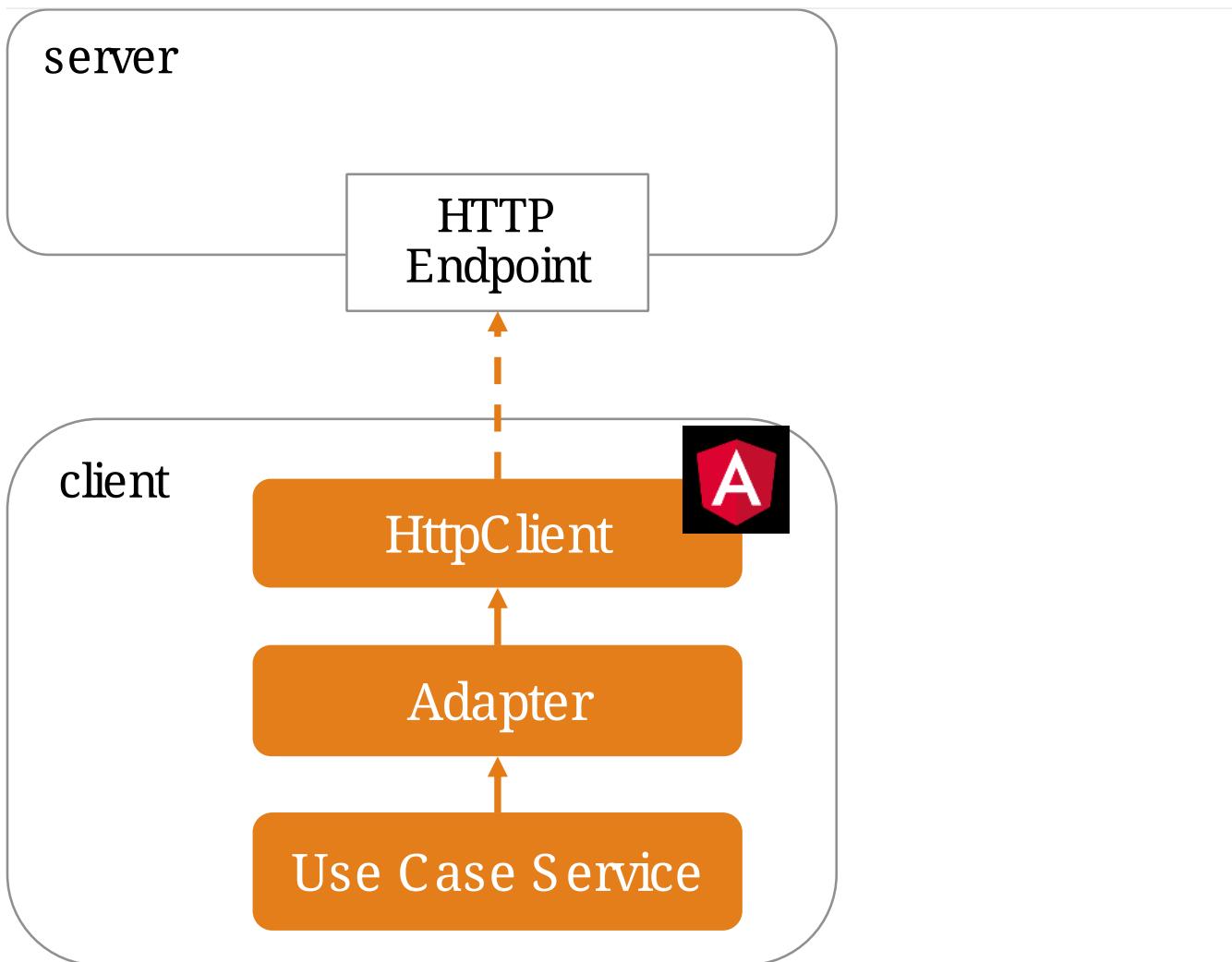


Figure 48. Adapters handle backend communication

As illustrated in the figure a Use Case service does not use Angular HttpClient directly but uses an adapter. A basic adapter could look like this:

Listing 44. Example adapter

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';

import { FlightTo } from './flight-to';

@Injectable({
  providedIn: 'root',
})
export class FlightsAdapter {

  constructor(
    private httpClient: HttpClient
  ) {}

  getFlights(): Observable<FlightTo> {
    return this.httpClient.get<FlightTo>('/relative/url/to/flights');
  }

}
```

The adapters should use a well-defined transfer data model. This could be generated from server endpoints with CobiGen, Swagger, typescript-maven-plugin, etc. If inside the application there is a business model defined, the adapter has to parse to the transfer model. This is illustrated in the following listing.

Listing 45. Example adapter mapping from business model to transfer model

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import { map } from 'rxjs/operators';

import { FlightTo } from './flight-to';
import { Flight } from '../../../../../model/flight';

@Injectable({
  providedIn: 'root',
})
export class FlightsAdapter {

  constructor(
    private httpClient: HttpClient
  ) {}

  updateFlight(flight: Flight): Observable<Flight> {
    const to = this.mapFlight(flight);

    return this.httpClient.post<FlightTo>('/relative/url/to/flights', to).pipe(
      map(to => this.mapFlightTo(to))
    );
  }

  private mapFlight(flight: Flight): FlightTo {
    // mapping logic
  }

  private mapFlightTo(flightTo: FlightTo): Flight {
    // mapping logic
  }
}

```

17.9.2. Token management

In most cases the access to backend API is secured using well known mechanisms as **CSRF**, **JWT** or both. In these cases the frontend application must manage the tokens that are generated when the user authenticates. More concretely it must store them to include them in every request automatically. Obviously, when user logs out these tokens must be removed from localStorage, memory, etc.

Store security token

In order to make this guide simple we are going to store the token in memory. Therefore, if we consider that we already have a login mechanism implemented we would like to store the token using a **auth.service.ts**:

```

import { Injectable } from '@angular/core';
import { Router } from '@angular/router';

@Injectable({
  providedIn: 'root',
})
export class AuthService {
  private loggedIn = false;
  private token: string;

  constructor(public router: Router) {}

  public isLoggedIn(): boolean {
    return this.loggedIn || false;
  }

  public setLoggedIn(login: boolean): void {
    this.loggedIn = login;
  }

  public getToken(): string {
    return this.token;
  }

  public setToken(token: string): void {
    this.token = token;
  }
}

```

Using the previous service we will be able to store the token obtained in the login request using the method `setToken(token)`. Please consider that, if you want a more sofisticated approach using localStorage API, you will need to modify this service accordingly.

Include token in every request

Now that the token is available in the application it is necessary to include it in every request to a protected API endpoint. Instead of modifying all the http requests in our application, Angular provides a class to intercept every request (and every response if we need to) called `HttpInterceptor`. Let's create a service called `http-interceptor.service.ts` to implement the `intercept` method of this class:

```

import {
  HttpEvent,
  HttpHandler,
  HttpInterceptor,
  HttpRequest,
} from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { environment } from '../../../../../environments/environment';
import { AuthService } from './auth.service';

@Injectable()
export class HttpRequestInterceptorService implements HttpInterceptor {

  constructor(private auth: AuthService) {}

  intercept(
    req: HttpRequest<any>,
    next: HttpHandler,
  ): Observable<HttpEvent<any>> {
    // Get the auth header from the service.
    const authHeader: string = this.auth.getToken();
    if (authHeader) {
      let authReq: HttpRequest<any>;

      // CSRF
      if (environment.security === 'csrf') {
        authReq = req.clone({
          withCredentials: true,
          setHeaders: { 'x-csrf-token': authHeader },
        });
      }

      // JWT
      if (environment.security === 'jwt') {
        authReq = req.clone({
          setHeaders: { Authorization: authHeader },
        });
      }

      return next.handle(authReq);
    } else {
      return next.handle(req);
    }
  }
}

```

As you may notice, this service is making use of an environment field `environment.security` to determine if we are using JWT or CSRF in order to inject the token accordingly. In your application

you can combine both if necessary.

Configure environment.ts file to use the CSRF/JWT.

```
security: 'csrf'
```

The `authHeader` used is obtained using the injected service `AuthService` already presented above.

In order to activate the interceptor we need to provide it in our `app.module.ts` or `core.module.ts` depending on the application structure. Let's assume that we are using the latter and the interceptor file is inside a `security` folder:

```
...
import { HttpRequestInterceptorService } from './security/http-request-
interceptor.service';
...

@NgModule({
  imports: [...],
  exports: [...],
  declarations: [],
  providers: [
    ...
    {
      provide: HTTP_INTERCEPTORS,
      useClass: HttpRequestInterceptorService,
      multi: true,
    },
  ],
})
export class CoreModule {}
```

Angular automatically will now modify every request and include in the header the token if it is convenient.

17.10. Error Handler in angular

Angular allows us to set up a custom error handler that can be used to control the different errors and them in a correct way. Using a global error handler will avoid mistakes and provide a user friendly interface allowing us to indicate the user what problem is happening.

17.10.1. What is ErrorHandler

`ErrorHandler` is the class that `Angular` uses by default to control the errors. This means that, even if the application doesn't have a `ErrorHandler` it is going to use the one setup by default in `Angular`. This can be tested by trying to find a page not existing in any app, instantly `Angular` will print the error in the console.

17.10.2. Creating your custom ErrorHandler step by step

In order to create a custom `ErrorHandler` three steps are going to be needed:

Creating the custom ErrorHandler class

In this first step the custom `ErrorHandler` class is going to be created inside the folder `/app/core/errors/errors-handler.ts`:

```
import { ErrorHandler, Injectable, Injector } from '@angular/core';
import { HttpErrorResponse } from '@angular/common/http';

@Injectable()
export class ErrorsHandler implements ErrorHandler {

    constructor(private injector: Injector) {}

    handleError(error: Error | HttpErrorResponse) {
        // To do: Use injector to get the necessary services to redirect or
        // show a message to the user
        const classname = error.constructor.name;
        switch (classname) {
            case 'HttpErrorResponse':
                console.error('HttpError:' + error.message);
                if (!navigator.onLine) {
                    console.error('Theres no internet connection');
                    // To do: control here in internet what you wanna do if user has no
internet
                } else {
                    console.error('Server Error:' + error.message);
                    // To do: control here if the server gave an error
                }
                break;
            default:
                console.error('Error:' + error.message);
                // To do: control here if the client/other things gave an error
        }
    }
}
```

This class can be used to control the different type of errors. If wanted, the `classname` variable could be used to add more switch cases. This would allow control of more specific situations.

Creating a ErrorInterceptor

Inside the same folder created in the last step we are going to create the `ErrorInterceptor(errors-handler-interceptor.ts)`. This `ErrorInterceptor` is going to retry any failed calls to the server to make sure it is not being found before showing the error:

```

import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent } from
'@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { retry } from 'rxjs/operators';

@Injectable()
export class ErrorsHandlerInterceptor implements HttpInterceptor {

  constructor() {}
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    return next.handle(req).pipe(
      retryWhen((errors: Observable<any>) => errors.pipe(
        delay(500),
        take(5),
        concatMap((error: any, retryIndex: number) => {
          if (++retryIndex === 5) {
            throw error;
          }
          return of(error);
        })
      ))
    );
  }
}

```

This custom made interceptor is implementing the `HttpInterceptor` and inside the method `intercept` using the method `pipe,retryWhen,delay,take` and `concatMap` from `RxJs` it is going to do the next things if there is errors:

1. With `delay(500)` do a delay to allow some time in between requests
2. With `take(5)` retry five times.
3. With `concatMap` if the index that `take()` gives is not 5 it returns the error, else, it throws the error.

Creating a Error Module

Finally, creating a module(`errors-handler.module.ts`) is necessary to include the `interceptor` and the custom error handler. In this case, the module is going to be created in the same folder as the last two:

```

import { NgModule, ErrorHandler } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ErrorsHandler } from './errors-handler';
import { HTTP_INTERCEPTORS } from '@angular/common/http';
import { ErrorsHandlerInterceptor } from './errors-handler-interceptor';

@NgModule({
  declarations: [], // Declare here component if you want to use routing to error
  component
  imports: [
    CommonModule
  ],
  providers: [
    {
      provide: ErrorHandler,
      useClass: ErrorsHandler,
    },
    {
      provide: HTTP_INTERCEPTORS,
      useClass: ErrorsHandlerInterceptor,
      multi: true,
    }
  ]
})
export class ErrorsHandlerModule { }

```

This module simply is providing the services that are implemented by our custom classes and then telling angular to use our custom made classes instead of the default ones. After doing this, the module has to be included in the app module `app.module.ts` in order to be used.

```

.....
imports: [
  ErrorsHandlerModule,
  .....

```

17.10.3. Handling Errors

As a final step, handling these errors is necessary. Theres different ways that can be used to control the errors, here are a few:

- Creating a custom page and using with `Router` to redirect to a page showing an error.
- Creating a service in the server side or `Backend` to create a log with the error and calling it with `HttpClient`.
- Showing a custom made `SnackBar` with the error message.

Using `SnackBarService` and `NgZone`

If the `SnackBar` is used directly, some errors can occur, this is due to `SnackBar` being out of the `Angular` zone. In order to use this service properly, `NgZone` is necessary. The method `run()` from `NgZone` will allow the service to be inside the `Angular Zone`. An example on how to use it:

```
import { ErrorHandler, Injectable, Injector, NgZone } from '@angular/core';
import { HttpErrorResponse } from '@angular/common/http';
import { MatSnackBar } from '@angular/material';

@Injectable()
export class ErrorsHandler implements ErrorHandler {

    constructor(private injector: Injector, private zone: NgZone) {}

    handleError(error: Error | HttpErrorResponse) {
        // Use injector to get the necessary services to redirect or
        const snackBar: MatSnackBar = this.injector.get(MatSnackBar);
        const classname = error.constructor.name;
        let message: string;
        switch (classname) {
            case 'HttpErrorResponse':
                message = !(navigator.onLine) ? 'There is no internet connection' :
error.message;
                break;
            default:
                message = error.message;
        }
        this.zone.run(
            () => snackBar.open(message, 'danger', { duration : 4000})
        );
    }
}
```

Using `Injector` the `MatSnackBar` is obtained, then the correct message is obtained inside the switch. Finally, using `NgZone` and `run()`, we open the `SnackBar` passing the message, and the parameters wanted.

17.11. File Structure

17.11.1. Toplevel

The toplevel file structure is defined by Angular CLI. You might put this "toplevel file structure" into a subdirectory to facilitate your build, but this is not relevant for this guide. So the applications file structure relevant to this guide is the folder `/src/app` inside the part managed by Angular CLI.

Listing 46. Toplevel file structure shows feature modules

```
/src
└── /app
    ├── /account-management
    ├── /billing
    ├── /booking
    ├── /core
    ├── /shared
    └── /status
    |
    ├── app.module.ts
    ├── app.component.spec.ts
    ├── app.component.ts
    └── app.routing-module.ts
```

Besides the definition of app module the `app` folder has feature modules on toplevel. The special modules `shared` and `core` are present as well.

17.11.2. Feature Modules

A feature module contains the modules definition and two folders representing both layers.

Listing 47. Feature module file structure has both layers

```
/src
└── /app
    └── /account-management
        ├── /components
        └── /services
        |
        ├── account-management.module.ts
        ├── account-management.component.spec.ts
        ├── account-management.component.ts
        └── account-management.routing-module.ts
```

Additionally an entry component is possible. This would be the case in lazy loading scenarios. So `account-management.component.ts` would be only present if `account-management` is lazy loaded. Otherwise, the module's routes would be defined *Component-less* (see [vsavkin blog post](#)).

17.11.3. Components Layer

The component layer reflects the distinction between *Smart Components* and *Dumb Components*.

Listing 48. Components layer file structure shows Smart Components on toplevel

```
/src
└── /app
    └── /account-management
        └── /components
            ├── /account-overview
            ├── /confirm-modal
            ├── /create-account
            ├── /forgot-password
            └── /shared
```

Every folder inside the `/components` folder represents a smart component. The only exception is `/shared`. `/shared` contains *Dumb Components* shared across *Smart Components* inside the components layer.

Listing 49. Smart components contain Dumb components

```
/src
└── /app
    └── /account-management
        └── /components
            └── /account-overview
                ├── /user-info-panel
                |   ├── /address-tab
                |   ├── /last-activities-tab
                |
                |   ├── user-info-panel.component.html
                |   ├── user-info-panel.component.scss
                |   ├── user-info-panel.component.spec.ts
                |   └── user-info-panel.component.ts
                |
                ├── /user-header
                └── /user-toolbar
                |
                ├── account-overview.component.html
                ├── account-overview.component.scss
                ├── account-overview.component.spec.ts
                └── account-overview.component.ts
```

Inside the folder of a *Smart Component* the component is defined. Besides that are folders containing the *Dumb Components* the *Smart Component* consists of. This can be recursive - a *Dumb Component* can consist of other *Dumb Components*. This is reflected by the file structure as well. This way the structure of a view becomes very readable. As mentioned before, if a *Dumb Component* is used by multiple *Smart Components* inside the components layer it is put inside the `/shared` folder inside the components layer.

With this way of thinking the *shared* module makes a lot of sense. If a *Dumb Component* is used by multiple *Smart Components* from different feature modules, the *Dumb Component* is placed into the

shared module.

Listing 50. The shared module contains Dumb Components shared across Smart Components from different feature modules

```
/src
  └── /app
    └── /shared
      └── /user-panel
        ├── user-panel.component.html
        ├── user-panel.component.scss
        ├── user-panel.component.spec.ts
        └── user-panel.component.ts
```

The layer folder `/components` is not necessary inside the *shared* module. The *shared* module only contains components!

17.12. Internationalization

Nowadays, a common scenario in front-end applications is to have the ability to translate labels and locate numbers, dates, currency and so on when the user clicks over a language selector or similar. devon4ng and specifically Angular has a default mechanism in order to fill the gap of such features, and besides there are some wide used libraries that make even easier to translate applications.

More info at [Angular i18n official documentation](#)

17.12.1. devon4ng i18n approach

The official approach could be a bit complicated, therefore the recommended one is to use the recommended library **Transloco** from <https://netbasal.gitbook.io/transloco/>.

Install and configure Transloco

In order to include this library in your devon4ng **Angular >= 7.2** project you will need to execute in a terminal:

```
$ ng add @ngneat/transloco
```

As part of the installation process you'll be presented with questions; Once you answer them, everything you need will automatically be created for you.

- First, Transloco creates boilerplate files for the requested translations.
- Next, it will create a new file, `transloco-root.module.ts` which exposes an Angular's `module` with a default configuration, and inject it into the `AppModule`.

```

import { HttpClient } from '@angular/common/http';
import {
  TRANSLOCO_LOADER,
  Translation,
  TranslocoLoader,
  TRANSLOCO_CONFIG,
  translocoConfig,
  TranslocoModule
} from '@ngneat/transloco';
import { Injectable, NgModule } from '@angular/core';
import { environment } from '../environments/environment';

@Injectable({ providedIn: 'root' })
export class TranslocoHttpLoader implements TranslocoLoader {
  constructor(private http: HttpClient) {}

  getTranslation(lang: string) {
    return this.http.get<Translation>(`/assets/i18n/${lang}.json`);
  }
}

@NgModule({
  exports: [ TranslocoModule ],
  providers: [
    {
      provide: TRANSLOCO_CONFIG,
      useValue: translocoConfig({
        availableLangs: ['en', 'es'],
        defaultLang: 'en',
        // Remove this option if your application doesn't support changing language in
        runtime.
        reRenderOnLangChange: true,
        prodMode: environment.production,
      })
    },
    { provide: TRANSLOCO_LOADER, useClass: TranslocoHttpLoader }
  ]
})
export class TranslocoRootModule {}

```



As you might have noticed it also set an **HttpLoader** into the module's providers. The **HttpLoader** is a class that implements the **TranslocoLoader** interface. It's responsible for instructing Transloco how to load the translation files. It uses Angular HTTP client to fetch the files, based on the given path.

Usage

In order to translate any label in any HTML template you will need to use the **transloco** pipe available:

```
{{ 'HELLO' | transloco }}
```

An **optional** parameter from the component TypeScript class could be included as follows:

```
{{ 'HELLO' | transloco: { value: dynamic } }}
```

It is possible to use with **inputs**:

```
<span [attr.alt]="'hello' | transloco">Attribute</span>
<span [title]="'hello' | transloco">Property</span>
```

In order to change the language used you will need to create a button or selector that calls the `this.translocoService.use(language: string)` method from `TranslocoService`. For example:

```
export class AppComponent {
  constructor(private translocoService: TranslocoService) {}

  changeLanguage(lang) {
    this.translocoService.setActiveLang(lang);
  }
}
```

The translations will be included in the `en.json`, `es.json`, `de.json`, etc. files inside the `/assets/i18n` folder. For example `en.json` would be (using the previous param):

```
{
  "HELLO": "hello"
}
```

Or with an **optional param**:

```
{
  "HELLO": "hello {{value}}"
}
```

Transloco understands nested JSON objects. This means that you can have a translation that looks like this:

```
{
  "HOME": {
    "HELLO": "hello {{value}}"
  }
}
```

In order to access access the value, use the dot notation, in this case `HOME.HELLO`.

Using the service, pipe or directive

Structural Directive

Using a structural directive is the **recommended** approach. It's DRY and efficient, as it creates **one** subscription per template:

```
<ng-container *transloco="let t">
  <p>{{ t('title') }}</p>

  <comp [title]="t('title')"></comp>
</ng-container>
```

Note that the `t` function is **memoized**. It means that given the same `key` it will return the result directly from the cache.

We can pass a `params` object as the second parameter:

```
<ng-container *transloco="let t">
  <p>{{ t('name', { name: 'Transloco' }) }}</p>
</ng-container>
```

We can instruct the directive to use a different language in our template:

```
<ng-container *transloco="let t; lang: 'es'">
  <p>{{ t('title') }}</p>
</ng-container>
```

Pipe

The use of pipes can be possible too:

template:

```
<div>{{ 'HELLO' | transloco:param }}</div>
```

component:

```
param = {value: 'world'};
```

Attribute Directive

The last option available with `transloco` is the attribute directive:

```
<div transloco="HELLO" [translocoParams]="{ value: 'world' }"></div>
```

Service

If you need to access translations in any component or service you can do it injecting the `TranslocoService` into them:

```
// Sync translation
translocoService.translate('HELLO', {value: 'world'});

// Async translation
translocoService.selectTranslate('HELLO', { value: 'world' }).subscribe(res => {
  console.log(res);
  //=> 'hello world'
});
```



You can find a complete example at <https://github.com/devonfw/devon4ng-application-template>.

Please, visit <https://netbasal.gitbook.io/transloco> for more info.

17.13. Routing

A basic introduction to the Angular Router can be found in [Angular Docs](#).

This guide will show common tasks and best practices.

17.13.1. Defining Routes

For each feature module and the app module all routes should be defined in a separate module with the suffix `RoutingModule`. This way the routing modules are the only place where routes are defined. This pattern achieves a clear separation of concerns. The following figure illustrates this.



Figure 49. Routing module declaration

It is important to define routes inside app routing module with `.forRoot()` and in feature routing modules with `.forChild()`.

Example 1 - No Lazy Loading

In this example two modules need to be configured with routes - `AppModule` and `FlightModule`.

The following routes will be configured

- `/` will redirect to `/search`
- `/search` displays `FlightSearchComponent` (`FlightModule`)
- `/search/print/:flightId/:date` displays `FlightPrintComponent` (`FlightModule`)
- `/search/details/:flightId/:date` displays `FlightDetailsComponent` (`FlightModule`)
- All other routes will display `ErrorPage404` (`AppModule`)

Listing 51. `app-routing.module.ts`

```

const routes: Routes = [
  { path: '', redirectTo: 'search', pathMatch: 'full' },
  { path: '**', component: ErrorPage404 }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
  
```

Listing 52. flight-search-routing.module.ts

```
const routes: Routes = [
  {
    path: 'search', children: [
      { path: '', component: FlightSearchComponent },
      { path: 'print/:flightId/:date', component: FlightPrintComponent },
      { path: 'details/:flightId/:date', component: FlightDetailsComponent }
    ]
  }
];
@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule],
})
export class FlightSearchRoutingModule { }
```



The import order inside AppModule is important. AppRoutingModule needs to be imported **after** FlightModule.

Example 2 - Lazy Loading

Lazy Loading is a good practice when the application has multiple feature areas and a user might not visit every dialog. Or at least he might not need every dialog up front.

The following example will configure the same routes as example 1 but will lazy load FlightModule.

Listing 53. app-routing.module.ts

```
const routes: Routes = [
  { path: '/search', loadChildren: 'app/flight-search/flight-
search.module#FlightSearchModule' },
  { path: '**', component: ErrorPage404 }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Listing 54. flight-search-routing.module.ts

```
const routes: Routes = [
  {
    path: '', children: [
      { path: '', component: FlightSearchComponent },
      { path: 'print/:flightId/:date', component: FlightPrintComponent },
      { path: 'details/:flightId/:date', component: FlightDetailsComponent }
    ]
  }
];
@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule],
})
export class FlightSearchRoutingModule { }
```

17.13.2. Triggering Route Changes

With Angular you have two ways of triggering route changes.

1. Declarative with bindings in component HTML templates
2. Programmatic with Angular **Router** service inside component classes

On the one hand, architecture-wise it is a much cleaner solution to trigger route changes in *Smart Components*. This way you have every UI event that should trigger a navigation handled in one place - in a *Smart Component*. It becomes very easy to look inside the code for every navigation, that can occur. Refactoring is also much easier, as there are no navigation events "hidden" in the HTML templates

On the other hand, in terms of accessibility and SEO it is a better solution to rely on bindings in the view - e.g. by using Angular's router-link directive. This way screen readers and the Google crawler can move through the page easily.



If you do not have to support accessibility (screen readers, etc.) and to care about SEO (Google rank, etc.), then you should aim for triggering navigations only in *Smart Components*.



Figure 50. Triggering navigation

17.13.3. Guards

Guards are Angular services implemented on routes which determines whether a user can navigate to/from the route. There are examples below which will explain things better. We have the following types of Guards:

- **CanActivate**: It is used to determine whether a user can visit a route. The most common scenario for this guard is to check if the user is authenticated. For example, if we want only logged in users to be able to go to a particular route, we will implement the **CanActivate** guard on this route.
- **CanActivateChild**: Same as above, only implemented on child routes.
- **CanDeactivate**: It is used to determine if a user can navigate away from a route. Most common example is when a user tries to go to a different page after filling up a form and does not save/submit the changes, we can use this guard to confirm whether the user really wants to leave the page without saving/submitting.
- **Resolve**: For resolving dynamic data.
- **CanLoad**: It is used to determine whether an *Angular module* can be loaded lazily. Example below will be helpful to understand it.

Let's have a look at some examples.

Example 1 - CanActivate and CanActivateChild guards

CanActivate guard

As mentioned earlier, a guard is an Angular service and services are simply TypeScript classes. So we begin by creating a class. This class has to implement the **CanActivate** interface (imported from `angular/router`), and therefore, must have a `canActivate` function. The logic of this function

determines whether the requested route can be navigated to or not. It returns either a `boolean` value or an `Observable` or a `Promise` which resolves to a `boolean` value. If it is true, the route is loaded, else not.

Listing 55. CanActivate example

```
...
import {CanActivate} from "@angular/router";

@Injectable()
class ExampleAuthGuard implements CanActivate {
  constructor(private authService: AuthService) {}

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    if (this.authService.isLoggedIn()) {
      return true;
    } else {
      window.alert('Please log in first');
      return false;
    }
  }
}
```

In the above example, let's assume we have a `AuthService` which has a `isLoggedIn()` method which returns a boolean value depending on whether the user is logged in. We use it to return `true` or `false` from the `canActivate` function. The `canActivate` function accepts two parameters (provided by Angular). The first parameter of type `ActivatedRouteSnapshot` is the snapshot of the route the user is trying to navigate to (where the guard is implemented); we can extract the route parameters from this instance. The second parameter of type `RouterStateSnapshot` is a snapshot of the router state the user is trying to navigate to; we can fetch the URL from its `url` property.



We can also redirect the user to another page (maybe a login page) if the `authService` returns false. To do that, inject `Router` and use its `navigate` function to redirect to the appropriate page.

Since it is a service, it needs to be provided in our module:

Listing 56. provide the guard in a module

```
@NgModule({
  ...
  providers: [
    ...
    ExampleAuthGuard
  ]
})
```

Now this guard is ready to use on our routes. We implement it where we define our array of routes in the application:

Listing 57. Implementing the guard

```
...
const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'page1', component: Page1Component, canActivate: [ExampleAuthGuard] }
];

```

As you can see, the `canActivate` property accepts an array of guards. So we can implement more than one guard on a route.

CanActivateChild guard

To use the guard on nested (children) routes, we add it to the `canActivateChild` property like so:

Listing 58. Implementing the guard on child routes

```
...
const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  { path: 'page1', component: Page1Component, canActivateChild: [ExampleAuthGuard],
    children: [
      {path: 'sub-page1', component: SubPageComponent},
      {path: 'sub-page2', component: SubPageComponent}
    ]
];

```

Example 2 - CanLoad guard

Similar to `CanActivate`, to use this guard we implement the `CanLoad` interface and overwrite its `canLoad` function. Again, this function returns either a boolean value or an `Observable` or a `Promise` which resolves to a boolean value. The fundamental difference between `CanActivate` and `CanLoad` is that `CanLoad` is used to determine whether an entire module can be lazily loaded or not. If the guard returns `false` for a module protected by `CanLoad`, the entire module is not loaded.

Listing 59. CanLoad example

```
...
import {CanLoad, Route} from "@angular/router";

@Injectable()
class ExampleCanLoadGuard implements CanLoad {
  constructor(private authService: AuthService) {}

  canLoad(route: Route) {
    if (this.authService.isLoggedIn()) {
      return true;
    } else {
      window.alert('Please log in first');
      return false;
    }
  }
}
```

Again, let's assume we have a `AuthService` which has a `isLoggedIn()` method which returns a boolean value depending on whether the user is logged in. The `canLoad` function accepts a parameter of type `Route` which we can use to fetch the path a user is trying to navigate to (using the `path` property of `Route`).

This guard needs to be provided in our module like any other service.

To implement the guard, we use the `canLoad` property:

Listing 60. Implementing the guard

```
...
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'admin', loadChildren: 'app/admin/admin.module#AdminModule', canLoad:
[ExampleCanLoadGuard] }
];
```

17.14. Testing

This guide will cover the basics of testing logic inside your code with UnitTests. The guide assumes that you are familiar with Angular CLI ([see the guide](#))

For testing your Angular application with UnitTests there are two main strategies:

1. Isolated UnitTests

Isolated unit tests examine an instance of a class all by itself without any dependence on Angular or any injected values. The amount of code and effort needed to create such tests is minimal.

2. Angular Testing Utilities

Let you test components including their interaction with Angular. The amount of code and effort needed to create such tests is a little higher.

17.14.1. Testing Concept

The following figure shows you an overview of the application architecture devided in testing areas.



Figure 51. Testing Areas

There are three areas, which need to be covered by different testing strategies.

1. Components:

Smart Components need to be tested because they contain view logic. Also the interaction with 3rd party components needs to be tested. When a 3rd party component changes with an upgrade a test will be failing and warn you, that there is something wrong with the new version. Most of the time Dumb Components do not need to be teste because they mainly display data and do not contain any logic. Smart Components are alway tested with **Angular Testing Utilities**. For example selectors, which select data from the store and transform it further, need to be tested.

2. Stores:

A store contains methods representing state transitions. If these methods contain logic, they need to be tested. Stores are always testet using **Isolated UnitTests**.

3. Services:

Services contain Business Logic, which needs to be tested. UseCase Services represent a whole business use case. For instance this could be initializing a store with all the data that is needed for a dialog - loading, transforming, storing. Often **Angular Testing Utilities** are the optimal solution for testing UseCase Services, because they allow for an easy stubbing of the backend. All other services should be tested with **Isolated UnitTests** as they are much easier to write and maintain.

17.14.2. Testing Smart Components

Testing Smart Components should assure the following.

1. Bindings are correct.
2. Selectors which load data from the store are correct.
3. Asynchronous behavior is correct (loading state, error state, "normal" state).
4. Oftentimes through testing one realizes, that important edge cases are forgotten.
5. Do these test become very complex, it is often an indicator for poor code quality in the component. Then the implementation is to be adjusted / refactored.
6. When testing values received from the native DOM, you will test also that 3rd party libraries did not change with a version upgrade. A failing test will show you what part of a 3rd party library has changed. This is much better than the users doing this for you. For example a binding might fail because the property name was changed with a newer version of a 3rd party library.

In the function `beforeEach()` the TestBed imported from **Angular Testing Utilities** needs to be initialized. The goal should be to define a minimal test-module with TestBed. The following code gives you an example.

Listing 61. Example test setup for Smart Components

```

describe('PrintFlightComponent', () => {

  let fixture: ComponentFixture<PrintCPrintFlightComponentcomponent>;
  let store: FlightStore;
  let printServiceSpy: jasmine.SpyObj<FlightPrintService>;

  beforeEach(() => {
    const urlParam = '1337';
    const activatedRouteStub = { params: of({ id: urlParam }) };
    printServiceSpy = jasmine.createSpyObj('FlightPrintService',
    ['initializePrintDialog']);
    TestBed.configureTestingModule({
      imports: [
        TranslateModule.forRoot(),
        RouterTestingModule
      ],
      declarations: [
        PrintFlightComponent,
        PrintContentComponent,
        GeneralInformationPrintPanelComponent,
        PassengersPrintPanelComponent
      ],
      providers: [
        FlightStore,
        {provide: FlightPrintService, useValue: printServiceSpy},
        {provide: ActivatedRoute, useValue: activatedRouteStub}
      ]
    });
    fixture = TestBed.createComponent(PrintFlightComponent);
    store = fixture.debugElement.injector.get(FlightStore);
    fixture.detectChanges();
  });

  // ... test cases
})

```

It is important:

- Use `RouterTestingModule` instead of RouterModule`
- Use `TranslateModule.forRoot()` without translations This way you can test language-neutral without translation marks.
- Do not add a whole module from your application - in declarations add the tested Smart Component with all its Dumb Components
- The store should never be stubbed. If you need a complex test setup, just use the regular methods defined on the store.
- Stub all services used by the Smart Component. These are mostly UseCase services. They should

not be tested by these tests. Only the correct call to their functions should be assured. The logic inside the UseCase services is tested with separate tests.

- `detectChanges()` performs an Angular Change Detection cycle (Angular refreshes all the bindings present in the view)
- `tick()` performs a virtual macro task, `tick(1000)` is equal to the virtual passing of 1s.

The following test cases show the testing strategy in action.

Listing 62. Example

```
it('calls initializePrintDialog for url parameter 1337', fakeAsync(() => {
  expect(printServiceSpy.initializePrintDialog).toHaveBeenCalledWith(1337);
});

it('creates correct loading subtitle', fakeAsync(() => {
  store.setPrintStateLoading(123);
  tick();
  fixture.detectChanges();

  const subtitle = fixture.debugElement.query(By.css('app-header-element .print-
header-container span:last-child'));
  expect(subtitle.nativeElement.textContent).toBe('PRINT_HEADER.FLIGHT
STATE.IS_LOADING');
});

it('creates correct subtitle for loaded flight', fakeAsync(() => {
  store.setPrintStateLoadedSuccess({
    id: 123,
    description: 'Description',
    iata: 'FRA',
    name: 'Frankfurt',
    // ...
  });
  tick();
  fixture.detectChanges();

  const subtitle = fixture.debugElement.query(By.css('app-header-element .print-
header-container span:last-child'));
  expect(subtitle.nativeElement.textContent).toBe('PRINT_HEADER.FLIGHT "FRA
(Frankfurt)" (ID: 123)');
});
```

The examples show the basic testing method

- Set the store to a well-defined state
- check if the component displays the correct values
- ... via checking values inside the native DOM.

17.14.3. Testing state transitions performed by stores

Stores are always tested with **Isolated UnitTests**.

Actions triggered by `dispatchAction()` calls are asynchronously performed to alter the state. A good solution to test such a state transition is to use the done callback from Jasmine.

Listing 63. Example for testing a store

```
let sut: FlightStore;

beforeEach(() => {
  sut = new FlightStore();
});

it('setPrintStateLoading sets print state to loading', (done: Function) => {
  sut.setPrintStateLoading(4711);

  sut.state$.pipe(first()).subscribe(result => {
    expect(result.print.isLoading).toBe(true);
    expect(result.print.loadingId).toBe(4711);
    done();
  });
});

it('toggleRowChecked adds flight with given id to selectedValues Property', (done: Function) => {
  const flight: FlightTO = {
    id: 12
    // dummy data
  };
  sut.setRegisterabgleichListe([flight]);
  sut.toggleRowChecked(12);

  sut.state$.pipe(first()).subscribe(result => {
    expect(result.selectedValues).toContain(flight);
    done();
  });
});
```

17.14.4. Testing services

When testing services both strategies - **Isolated UnitTests** and **Angular Testing Utilities** - are valid options.

The goal of such tests are

- assuring the behavior for valid data.
- assuring the behavior for invalid data.

- documenting functionality
- safely performing refactorings
- thinking about edge case behavior while testing

For simple services **Isolated UnitTests** can be written. Writing these tests takes lesser effort and they can be written very fast.

The following listing gives an example of such tests.

Listing 64. Testing a simple services with Isolated UnitTests

```
let sut: IsyDatePipe;

beforeEach(() => {
  sut = new IsyDatePipe();
});

it('transform should return empty string if input value is empty', () => {
  expect(sut.transform('')).toBe('');
});

it('transform should return empty string if input value is null', () => {
  expect(sut.transform(undefined)).toBe('');
});

// ...more tests
```

For testing Use Case services the Angular Testing Utilities should be used. The following listing gives an example.

Listing 65. Test setup for testing use case services with Angular Testing Utilities

```

let sut: FlightPrintService;
let store: FlightStore;
let httpController: HttpTestingController;
let flightCalculationServiceStub: jasmine.SpyObj<FlightCalculationService>;
const flight: FlightTo = {
  // ... valid dummy data
};

beforeEach(() => {
  flightCalculationServiceStub = jasmine.createSpyObj('FlightCalculationService',
  ['getFlightType']);
  flightCalculationServiceStub.getFlightType.and.callFake((catalog: string, type: string, key: string) => of(`${key}_long`));
  TestBed.configureTestingModule({
    imports: [
      HttpClientTestingModule,
      RouterTestingModule,
    ],
    providers: [
      FlightPrintService,
      FlightStore,
      FlightAdapter,
      {provide: FlightCalculationService, useValue: flightCalculationServiceStub}
    ]
  });
  sut = TestBed.get(FlightPrintService);
  store = TestBed.get(FlightStore);
  httpController = TestBed.get(HttpTestingController);
});

```

When using TestBed, it is important

- to import HttpClientTestingModule for stubbing the backend
- to import RouterTestingModule for stubbing the Angular router
- not to stub stores, adapters and business services
- to stub services from libraries like FlightCalculationService - the correct implementation of libraries should not be tested by these tests.

Testing backend communication looks like this:

Listing 66. Testing backend communication with Angular HttpTestingController

```
it('loads flight if not present in store', fakeAsync(() => {
  sut.initializePrintDialog(1337);
  const processRequest = httpController.expectOne('/path/to/flight');
  processRequest.flush(flight);

  httpController.verify();
}));

it('does not load flight if present in store', fakeAsync(() => {
  const flight = {...flight, id: 4711};
  store.setRegisterabgleich(flight);

  sut.initializePrintDialog(4711);
  httpController.expectNone('/path/to/flight');

  httpController.verify();
}));
```

The first test assures a correct XHR request is performed if `initializePrintDialog()` is called and no data is in the store. The second test assures no XHR request ist performed if the needed data is already in the store.

The next steps are checks for the correct implementation of logic.

Listing 67. Example testing a Use Case service

```
it('creates flight destination for valid key in svz', fakeAsync(() => {
  const flightTo: FlightTo = {
    ...flight,
    id: 4712,
    profile: '77'
  };
  store.setFlight(flightTo);
  let result: FlightPrintContent|undefined;

  sut.initializePrintDialog(4712);
  store.select(s => s.print.content).subscribe(content => result = content);
  tick();

  expect(result!.destination).toBe('77_long (ID: 77)');
}));
```

17.15. Update Angular CLI

17.15.1. Angular CLI common issues

There are constant updates for the official Angular framework dependencies. These dependencies

are directly related with the Angular CLI package. Since this package comes installed by default inside the devonfw distribution folder for Windows OS and the distribution is updated every few months it needs to be updated in order to avoid known issues.

17.15.2. Angular CLI update guide

For **Linux users** is as easy as updating the global package:

```
$ npm uninstall -g @angular/cli
$ npm install -g @angular/cli
```

For **Windows users** the process is only a bit harder. Open the **devonfw bundled console** and do as follows:

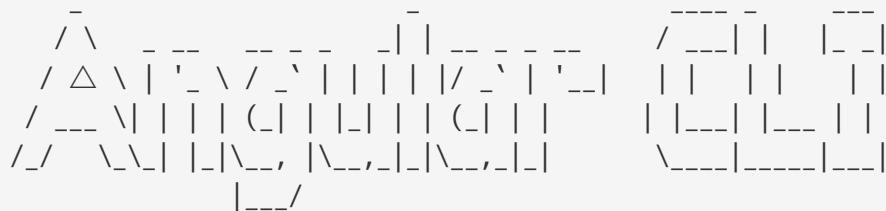
```
$ cd [devonfw_dist_folder]
$ cd software/nodejs
$ npm uninstall @angular/cli --no-save
$ npm install @angular/cli --no-save
```

After following these steps you should have the latest Angular CLI version installed in your system. In order to check it run in the distribution console:



At the time of this writing, the Angular CLI is at 1.7.4 version.

```
λ ng version
```



```
Angular CLI: 7.2.3
```

```
Node: 10.13.0
```

```
OS: win32 x64
```

```
Angular:
```

```
...
```

17.16. Upgrade devon4ng Angular and Ionic/Angular applications

Angular CLI provides a powerful tool to upgrade Angular based applications to the current stable release of the core framework.

This tool is `ng update`. It will not only upgrade dependencies and their related ones but also will perform some fixes in your code if available thanks to the provided *schematics*. It will check even if the update is not possible as there is another library or libraries that are not compatible with the versions of the upgraded dependencies. In this case it will keep your application untouched.



The repository must be in a clean state before executing a `ng update`. So, remember to commit your changes first.

17.16.1. Basic usage

In order to perform a basic upgrade we will execute:

```
$ ng update @angular/cli @angular/core
```

Upgrade to new Angular version

The process will be the same, but first we need to make sure that our devon4ng application is in the latest version of Angular 8, so the `ng update` command can perform the upgrade not only in the dependencies but also making code changes to reflect the new features and fixes.

- First, upgrade to latest Angular 9 version:

```
$ ng update @angular/cli@9 @angular/core@9
```

Optionally the flag `-C` can be added to previous command to make a commit automatically. This is also valid for the next steps.

- Then, upgrade Angular:

```
$ ng update @angular/cli @angular/core
```

- In case you use Angular Material:

```
$ ng update @angular/material
```

- If the application depends on third party libraries, the new tool `ngcc` can be run to make them compatible with the new **Ivy compiler**. In this case it is recommended to include a `postinstall` script in the `package.json`:

```
{
  "scripts": {
    "postinstall": "ngcc --properties es2015 browser module main --first-only --create-ivy-entry-points"
  }
}
```

More information at <https://angular.io/guide/updating-to-version-9> and <https://angular.io/guide/ivy>.

Important use cases:

- To update to the next beta or pre-release version, use the `--next=true` option.
- To update from one major version to another, use the format `ng update @angular/cli@^<major_version> @angular/core@^<major_version>`.
- In case your Angular application uses `@angular/material` include it in the first command:

```
$ ng update @angular/cli @angular/core @angular/material
```

Ionic/Angular applications

Just following the same procedure we can upgrade Angular applications, but we must take care of important specific Ionic dependencies:

```
$ ng update @angular/cli @angular/core @ionic/angular @ionic/angular-toolkit
[@ionic/...]
```

Other dependencies

Every application will make use of different dependencies. Angular CLI `ng upgrade` will also take care of these ones. For example, if you need to upgrade `@capacitor` you will perform:

```
$ ng update @capacitor/cli @capacitor/core [@capacitor/...]
```

Another example could be that you need to upgrade `@ngx-translate` packages. As always in this case you will execute:

```
$ ng update @ngx-translate/core @ngx-translate/http-loader
```

Angular Update Guide online tool

It is recommended to use the Angular Update Guide tool at <https://update.angular.io/> that will provide the necessary steps to upgrade any Angular application depending on multiple criteria.

17.17. Working with Angular CLI

Angular CLI provides a facade for building, testing, linting, debugging and generating code. Under the hood Angular CLI uses specific tools to achieve these tasks. The user does no need to maintain them and can rely on Angular to keep them up to date and maybe switch to other tools which come up in the future.

The Angular CLI provides a wiki with common tasks you encounter when working on applications with the Angular CLI. [The Angular CLI Wiki can be found here](#).

In this guide we will go through the most important tasks. To go into more details, please visit the Angular CLI wiki.

17.17.1. Installing Angular CLI

Angular CLI should be added as global and local dependency. The following commands add Angular CLI as global Dependency.

yarn command

```
yarn global add @angular/cli
```

npm command

```
npm install -g @angular/cli
```

You can check a successful installation with `ng --version`. This should print out the version installed.

```
C:\>ng --version
Angular CLI: 1.7.3
Node: 8.9.4
OS: win32 x64
```

Figure 52. Printing Angular CLI Version

17.17.2. Running a live development server

The Angular CLI can be used to start a live development server. First your application will be compiled and then the server will be started. If you change the code of a file, the server will reload the displayed page. Run your application with the following command:

```
ng serve -o
```

17.17.3. Running Unit Tests

All unit tests can be executed with the command:

```
ng test
```

To make a single run and create a code coverage file use the following command:

```
ng test -sr -cc
```



You can configure the output format for code coverage files to match your requirements in the file `karma.conf.js` which can be found on toplevel of your project folder. For instance, this can be useful for exporting the results to a SonarQube.

17.17.4. Linting the code quality

You can lint your files with the command

```
ng lint --type-check
```



You can adjust the linting rules in the file `tslint.json` which can be found on toplevel of your project folder.

17.17.5. Generating Code

Creating a new Angular CLI project

For creating a new Angular CLI project the command `ng new` is used.

The following command creates a new application named `my-app`.

```
ng create my-app
```

Creating a new feature module

A new feature module can be created via `ng generate module` command.

The following command generates a new feature module named `todo`.

```
ng generate module todo
```

```
C:\my-app>ng generate module todo
  create src/app/todo/todo.module.ts (188 bytes)
```

Figure 53. Generate a module with Angular CLI



The created feature module needs to be added to the AppModule by hand. Other option would be to define a lazy route in AppRoutingModule to make this a lazy loaded module.

Creating a new component

To create components the command `ng generate component` can be used.

The following command will generate the component todo-details inside the components layer of todo module. It will generate a class, a html file, a css file and a test file. Also, it will register this component as declaration inside the nearest module - this ist TodoModule.

```
ng generate component todo/components/todo-details
```

```
C:\my-app>ng generate component todo/components/todo-details
  create src/app/todo/components/todo-details/todo-details.component.html (31 bytes)
  create src/app/todo/components/todo-details/todo-details.component.spec.ts (664 bytes)
  create src/app/todo/components/todo-details/todo-details.component.ts (292 bytes)
  create src/app/todo/components/todo-details/todo-details.component.css (0 bytes)
  update src/app/todo/todo.module.ts (297 bytes)
```

Figure 54. Generate a component with Angular CLI



If you want to export the component, you have to add the component to exports array of the module. This would be the case if you generate a component inside shared module.

17.17.6. Configuring an Angular CLI project

Inside an Angular CLI project the file `.angular-cli.json` can be used to configure the Angular CLI.

The following options are very important to understand.

- The property `defaults`` can be used to change the default style extension. The following settings will make the Angular CLI generate `.less` files, when a new component is generated.

```
"defaults": {
  "styleExt": "less",
  "component": {}
}
```

- The property `apps` contains all applications maintained with Angular CLI. Most of the time you will have only one.
 - `assets` configures all the static files, that the application needs - this can be images, fonts, json files, etc. When you add them to assets the Angular CLI will put these files to the build

target and serve them while debugging. The following will put all files in `/i18n` to the output folder `/i18n`

```
"assets": [
  { "glob": "**/*.{json}", "input": "./i18n", "output": "./i18n" }
]
```

- `styles` property contains all style files that will be globally available. The Angular CLI will create a styles bundle that goes directly into index.html with it. The following will make all styles in `styles.less` globally available.

```
"styles": [
  "styles.less"
]
```

- `environmentSource` and `environments` are used to configure configuration with the Angular CLI. Inside the code always the file specified in `environmentSource` will be referenced. You can define different environments - eg. production, staging, etc. - which you list in `environments`. At compile time the Angular CLI will override all values in `environmentSource` with the values from the matching environment target. The following code will build the application for the environment staging.

```
ng build --environment=staging
```

18. Ionic

18.1. Ionic 5 Getting started

Ionic is a front-end focused framework which offers different tools for developing hybrid mobile applications. The web technologies used for this purpose are CSS, Sass, HTML5 and Typescript.

18.1.1. Why Ionic?

Ionic is used for developing hybrid applications, which means not having to rely on a specific IDE such as Android Studio or Xcode. Furthermore, development of native apps require learning different languages (Java/Kotlin for Android and Objective-C/Swift for Apple), with Ionic, a developer does not have to code the same functionality for multiple platforms, just use the adequate libraries and components.

18.1.2. Basic environment set up

Install Ionic CLI

Although the devonfw distribution comes with and already installed Ionic CLI, here are the steps to install it. The installation of Ionic is easy, just one command has to be written:

```
$ npm install -g @ionic/cli
```

Update Ionic CLI

If there was a previous installation of the Ionic CLI, it will need to be uninstalled due to a change in package name.

```
$ npm uninstall -g ionic  
$ npm install -g @ionic/cli
```

18.2. Basic project set up

The set up of an ionic application is pretty immediate and can be done in one line:

```
ionic start <name> <template> --type=angular
```

- ionic start: Command to create an app.
- <name>: Name of the application.
- <template>: Model of the application.
- --type=angular: With this flag, the app produced will be based on angular.

To create an empty project, the following command can be used:

```
ionic start MyApp blank --type=angular
```



The image above shows the directory structure generated.

There are more templates available that can be seen with the command `ionic start --list`

C:\Projects\ionic	ionic start template	updateIonic-cli.PNG	ionic blank project.PNG
name	project type	description	
blank	angular	A blank starter project	
sidemenu	angular	A starting project with a side menu with navigation in the content area	
tabs	angular	A starting project with a simple tabbed interface	
tabs	ionic-angular	A starting project with a simple tabbed interface	
blank	ionic-angular	A blank starter project	
sidemenu	ionic-angular	A starting project with a side menu with navigation in the content area	
super	ionic-angular	A starting project complete with pre-built pages, providers and best practices for Ionic development.	
tutorial	ionic-angular	A tutorial based project that goes along with the Ionic documentation	
aws	ionic-angular	AWS Mobile Hub Starter	
tabs	ionic1	A starting project for Ionic using a simple tabbed interface	
blank	ionic1	A blank starter project for Ionic	
sidemenu	ionic1	A starting project for Ionic using a side menu with navigation in the content area	
maps	ionic1	An Ionic starter project using Google Maps and a side menu	

The templates surrounded by red line are based on angular and comes with Ionic v5, while the others belong to earlier versions (before v4).



More info at <https://ionicframework.com/docs>. Remember to select **Angular documentation**, since Ionic supports React, Vue and Vanilla JS.

18.3. Ionic to android

This page is written to help developers to go from the source code of an ionic application to an android one, with this in mind, topics such as: environment, commands, modifications,... are covered.

18.3.1. Assumptions

This document assumes that the reader has already:

-
- Source code of an Ionic application and wants to build it on an android device,
 - A working installation of Node.js
 - An Ionic CLI installed and up-to-date.
 - Android Studio and Android SDK.

18.3.2. From Ionic to Android project

When a native application is being designed, sometimes, functionalities that uses camera, geolocation, push notification, ... are requested. To resolve these requests, Capacitor can be used.

In general terms, Capacitor wraps apps made with Ionic (HTML, SCSS, Typescript) into WebViews that can be displayed in native applications (Android, IOS) and allows the developer to access native functionalities like the ones said before.

Installing capacitor is as easy as installing any node module, just a few commands have to be run in a console:

- `cd name-of-ionic-4-app`
- `npm install --save @capacitor/core @capacitor/cli`

Then, it is necessary to initialize capacitor with some information: app id, name of the app and the directory where your app is stored. To fill this information, run:

- `npx cap init`

Modifications

Throughout the development process, usually back-end and front-end are on a local computer, so it's a common practice to have different configuration files for each environment (commonly production and development). Ionic uses an angular.json file to store those configurations and some rules to be applied.

If a back-end is hosted on <http://localhost:8081>, and that direction is used in every environment, the application built for android will not work because computer and device do not have the same localhost. Fortunately, different configurations can be defined.

Android Studio uses 10.0.0.2 as alias for 127.0.0.1 (computer's localhost) so adding `http://10.0.0.2:8081` in a new environment file and modifying angular.json accordingly, will make possible connect front-end and back-end.

```

environment.ts
1 // This file can be replaced during build by using the 'fileReplacements' in 'angular.json'
2 // The list of file replacements can be found in 'angular.json'
3
4 export const environment = {
5   production: false,
6 },
7
8 export const SERVER_URL = 'http://localhost:8081/';
9
10 /*
11 * For easier debugging in development mode, you can import the
12 * to ignore zone related error stack frames such as 'zone.run'
13 */
14
15 /* This import should be commented out in production mode because
16 * on performance if an error is thrown.
17 */
18 // Import 'zone.js/dist/zone-error'; // Included with Angular
19

environment.android.ts
1 Ambuludi Olmedo, a day ago | 1 author (Ambuludi Olmedo)
2
3 export const environment = {
4   production: false,
5 };
6
7 export const SERVER_URL = 'http://10.0.2.2:8081/';

angular.json
5 "newProjectRoot": "projects",
6 "projects": {
7   "app": {
8     "root": "",
9     "sourceRoot": "src",
10    "projectType": "application",
11    "prefix": "app",
12    "schematics": {},
13    "architect": {
14      "build": {
15        "builder": "@angular-devkit/build-angular:browser",
16        "options": {
17          "configuration": "production"
18        }
19      },
20      "serve": {
21        "builder": "@angular-devkit/build-angular:dev-server",
22        "options": {
23          "browserTarget": "app:build"
24        }
25      }
26    },
27    "configurations": {
28      "production": {
29        "fileReplacements": [
30          {
31            "replace": "src/environments/environment.ts",
32            "with": "src/environments/environment.android.ts"
33          }
34        ],
35        "i18n": {
36          "progress": false
37        }
38      }
39    }
40  }
41 }

"build": {
...
  "configurations": {
    ...
    "android": {
      "fileReplacements": [
        {
          "replace": "src/environments/environment.ts",
          "with": "src/environments/environment.android.ts"
        }
      ]
    },
  }
}

```

```

"build": {
...
  "configurations": {
    ...
    "android": {
      "fileReplacements": [
        {
          "replace": "src/environments/environment.ts",
          "with": "src/environments/environment.android.ts"
        }
      ]
    },
  }
}

```

Build

Once configured, it is necessary to build the Ionic app using this new configuration:

- `ionic build --configuration=android`

The next commands copy the build application on a folder named android and open android studio.

- `npx cap add android`
- `npx cap copy`
- `npx cap open android`

18.3.3. From Android project to emulated device

Once Android Studio is opened, follow these steps:

1. Click on "Build" → Make project.
2. Click on "Build" → Make Module 'app' (default name).



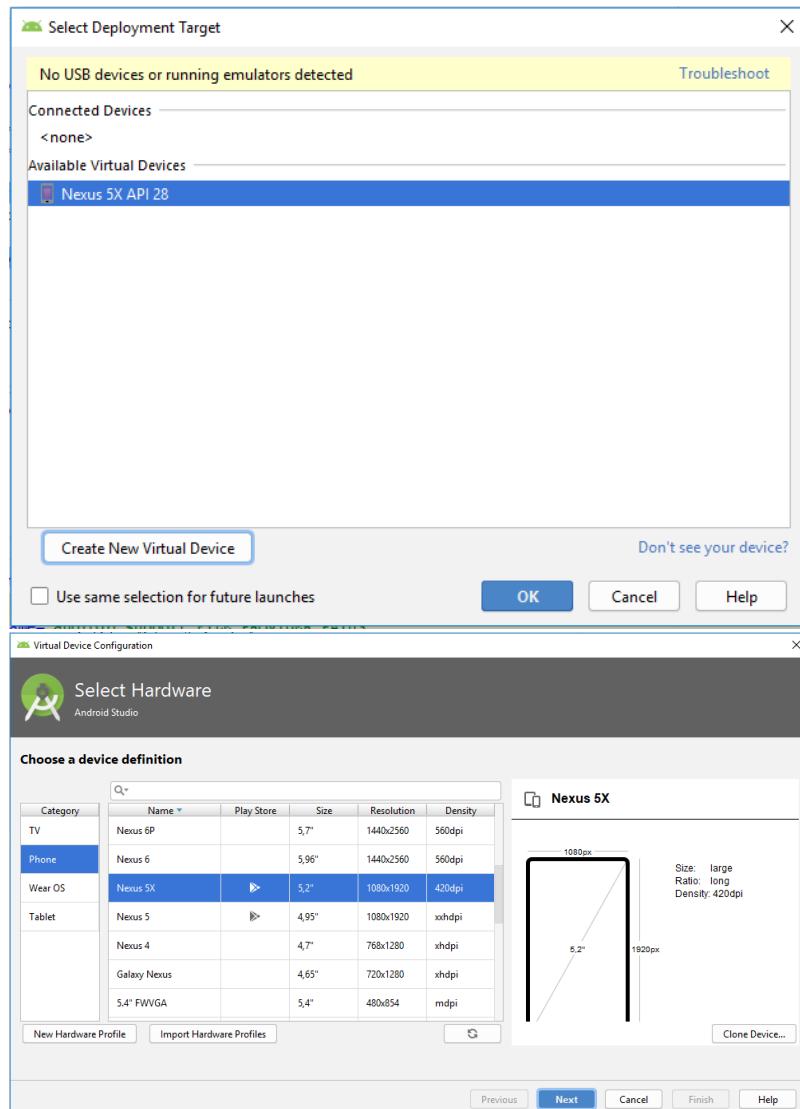
3. Click on "Build" → Build Bundle(s) / APK(s) → Build APK(s).

4. Click on run and choose a device.

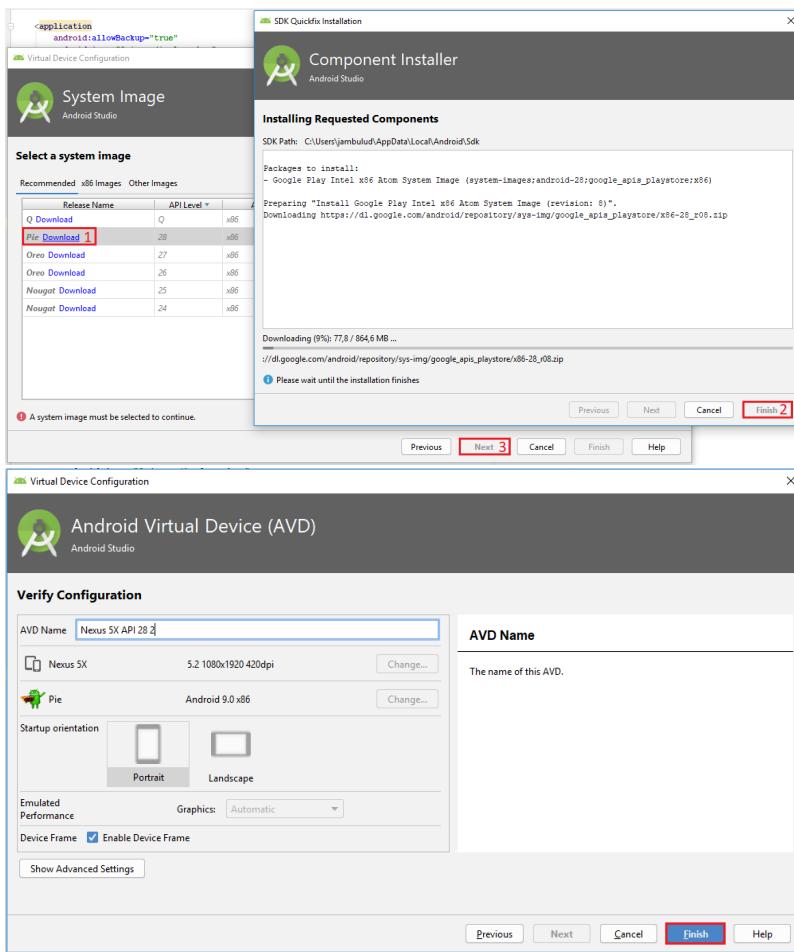


If there are no devices available, a new one can be created:

1. Click on "Create new device"
2. Select hardware and click "Next". For example: Phone → Nexus 5X.



3. Download a system image.
 1. Click on download.
 2. Wait until the installation finished and then click "Finish".
 3. Click "Next".
4. Verify configuration (default configuration should be enough) and click "Next".



5. Check that the new device is created correctly.



18.3.4. From Android project to real device

To test on a real android device, an easy approach to communicate a smartphone (front-end) and computer (back-end) is to configure a Wi-fi hotspot and connect the computer to it. A guide about this process can be found at <https://support.google.com/nexus/answer/9059108?hl=en>

Once connected, run `ipconfig` on a console if you are using windows or `ifconfig` on a linux machine

to get the IP address of your machine's Wireless LAN adapter Wi-fi.

```
Connection-specific DNS Suffix . . .
Wireless LAN adapter Wi-Fi:
  Connection-specific DNS Suffix . . .
  Link-Local IPv6 Address . . . . . : fe80::90a:3d66:7659:1963%17
  IPv4 Address. . . . . : 192.168.43.17
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . : 192.168.43.1

Ethernet adapter Bluetooth Network Connection:
  Media State . . . . . : Media disconnected
  Connection-specific DNS Suffix . . .

Tunnel adapter Teredo Tunneling Pseudo-Interface:
  Media State . . . . . : Media disconnected
  Connection-specific DNS Suffix . . .
```

This obtained IP must be used instead of "localhost" or "10.0.2.2" at environment.android.ts.



```
TS environment.android.ts x {} capacitor.config.json
1 export const environment = {
2   production: false,
3 };
4
5 export const SERVER_URL = 'http://192.168.43.17:8081/';
6 |
```

After this configuration, follow the build steps in "From Ionic to Android project" and the first three steps in "From Android project to emulated device".

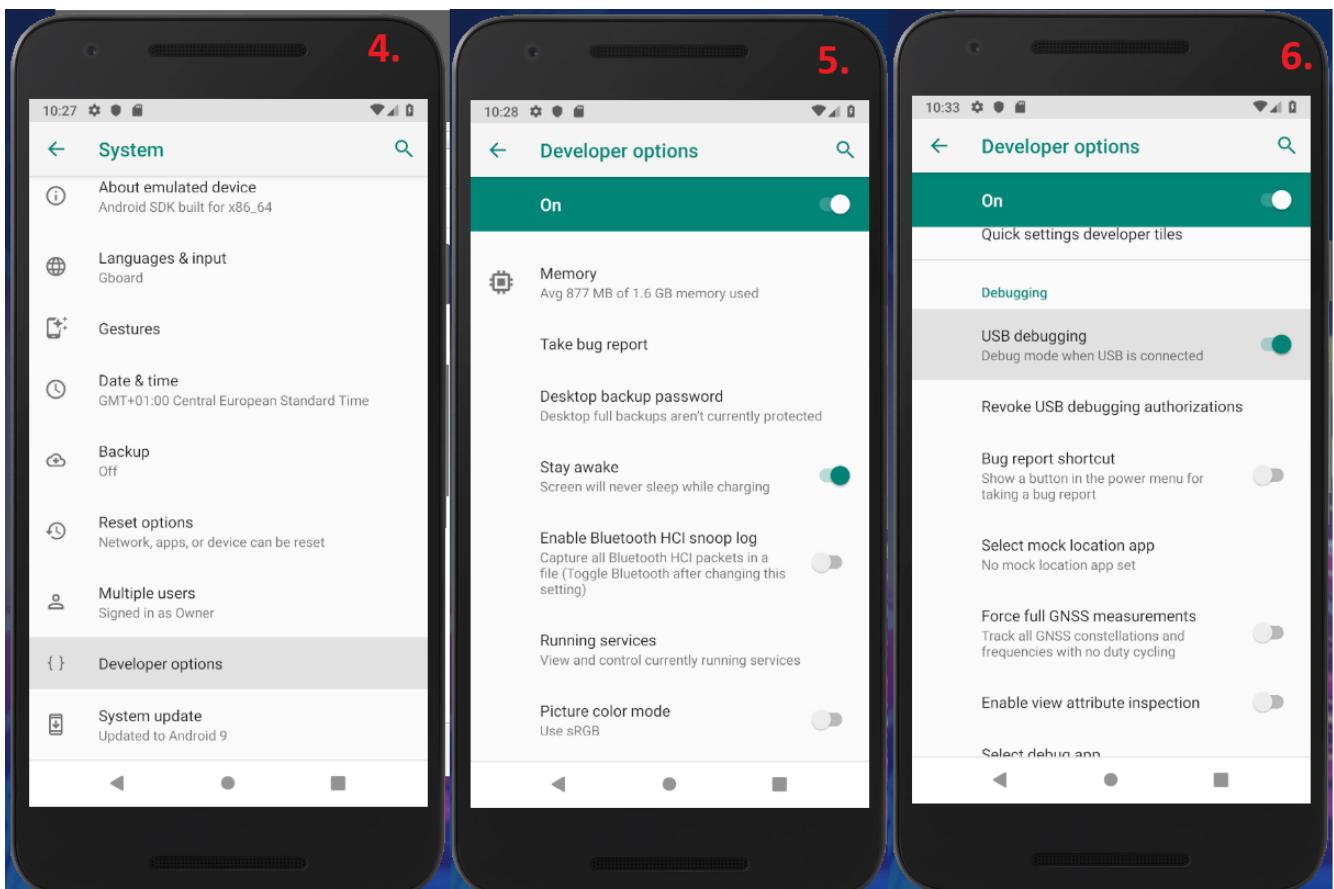
Send APK to Android through USB

To send the built application to a device, you can connect computer and mobile through USB, but first, it is necessary to unlock developer options.

1. Open "Settings" and go to "System".
2. Click on "About".
3. Click "Build number" seven times to unlock developer options.



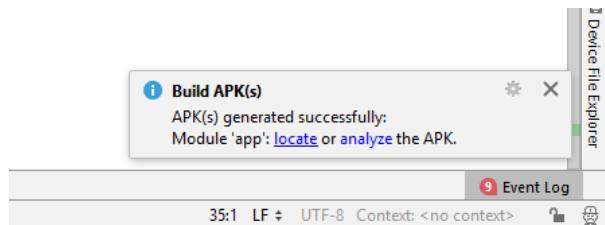
4. Go to "System" again and then to "Developer options"
5. Check that the options are "On".
6. Check that "USB debugging" is activated.



After this, do the step four in "From Android project to emulated device" and choose the connected smartphone.

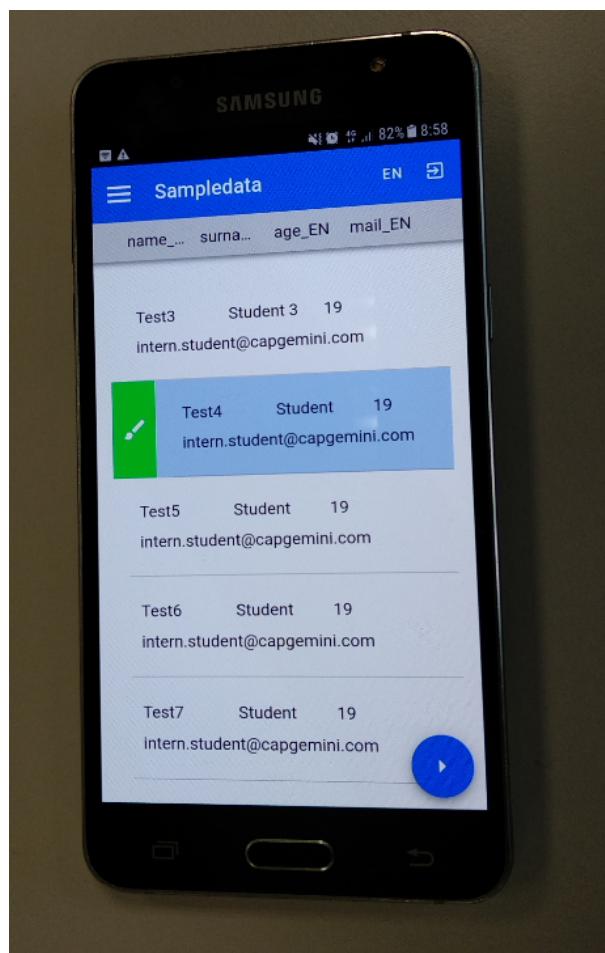
Send APK to Android through email

When you build an APK, a dialog gives two options: locate or analyze. If the first one is chosen, Windows file explorer will be opened showing an APK that can be sent using email. Download the APK on your phone and click it to install.



18.3.5. Result

If everything goes correctly, the Ionic application will be ready to be tested.



18.4. Ionic Progressive Web App

This guide is a continuation of the guide [Angular PWAs](#), therefore, valid concepts explained there are still valid in this page but focused on Ionic.

18.4.1. Assumptions

This guide assumes that you already have installed:

- Node.js
- npm package manager
- Angular CLI
- Ionic 5 CLI
- Capacitor

Also, it is a good idea to read the document about PWA using Angular.

18.4.2. Sample Application



Figure 55. Basic ionic PWA.

To explain how to build progressive web apps (PWA) using Ionic, a basic application is going to be built. This app will be able to take photos even without network using PWA elements.

Step 1: Create a new project

This step can be completed with one simple command: `ionic start <name> <template>`, where `<name>` is the name and `<template>` a model for the app. In this case, the app is going to be named **basic-ion-pwa**.

Step 2: Structures and styles

The styles (scss) and structures (html) do not have anything specially relevant, just colors and ionic web components. The code can be found in [devon4ng samples](#).

Step 3: Add functionality

After this step, the app will allow users take photos and display them in the main screen. First we have to import three important elements:

- DomSanitizer: Sanitizes values to be safe to use.
- SafeResourceUrl: Interface for values that are safe to use as URL.
- Plugins: Capacitor constant value used to access to the device's camera and toast dialogs.

```
import { DomSanitizer, SafeResourceUrl } from '@angular/platform-browser';
import { Plugins, CameraResultType } from '@capacitor/core';
const { Camera, Toast } = Plugins;
```

The process of taking a picture is enclosed in a **takePicture** method. takePicture calls the Camera's *getPhoto* function which returns an URL or an exception. If a photo is taken then the image displayed in the main page will be changed for the new picture, else, if the app is closed without changing it, a toast message will be displayed.

```
export class HomePage {
  image: SafeResourceUrl;
  ...

  async takePicture() {
    try {
      const image = await Camera.getPhoto({
        quality: 90,
        allowEditing: true,
        resultType: CameraResultType.Uri,
      });

      // Change last picture shown
      this.image = this.sanitizer.bypassSecurityTrustResourceUrl(image.webPath);
    } catch (e) {
      this.show('Closing camera');
    }
  }

  async show(message: string) {
    await Toast.show({
      text: message,
    });
  }
}
```

Step 4: PWA Elements

When Ionic apps are not running natively, some resources like Camera do not work by default but can be enabled using PWA Elements. To use Capacitor's PWA elements run `npm install @ionic/pwa-elements` and modify `src/main.ts` as shown below.

```
...
// Import for PWA elements
import { defineCustomElements } from '@ionic/pwa-elements/loader';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.log(err));

// Call the element loader after the platform has been bootstrapped
defineCustomElements(window);
```

Step 5: Make it Progressive.

Turining an Ionic 5 app into a PWA is pretty easy, the same module used to turn Angular apps into PWAs has to be added, to do so, run: `ng add @angular/pwa`. This command also creates an **icons** folder inside **src/assets** and contains angular icons for multiple resolutions. If you want use other images, be sure that they have the same resolution, the names can be different but the file **manifest.json** has to be changed accordingly.

Step 6: Configure the app

manifest.json

Default configuration.

ngsw-config.json

At `assetGroups → resources` add a `urls` field and a pattern to match PWA Elements scripts and other resources (images, styles, ...):

```
"urls": ["https://unpkg.com/@ionic/pwa-elements@1.0.2/dist/**"]
```

Step 7: Check that your app is a PWA

To check if an app is a PWA lets compare its normal behaviour against itself but built for production. Run in the project's root folder the commands below:

`ionic build --prod` to build the app using production settings.

`npm install http-server` to install an npm module that can serve your built application. Documentation [here](#). A good alternative is also `npm install serve`. It can be checked [here](#).

Go to the `www` folder running `cd www`.

`http-server -o` or `serve` to serve your built app.



In order not to install anything not necessary `npx` can be used directly to serve the app. i.e run `npx serve [folder]` will automatically download and run this HTTP server without installing it in the project dependencies.

```
C:\Proyectos\devon4ng-projects\sample-ion\basic-ion-pwa\www (master -> origin)
λ http-server -o
Starting up http-server, serving .
Available on:
  http://10.80.132.29:8081
  http://192.168.56.1:8081
  http://192.168.99.1:8081
  http://127.0.0.1:8081
Hit CTRL-C to stop the server
[Mon Apr 08 2019 08:35:37 GMT+0200 (GMT+02:00)] "GET /" "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36"
[Mon Apr 08 2019 08:35:42 GMT+0200 (GMT+02:00)] "GET /ngsw.json?ngsw-cache-bust=0366" "Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X) AppleWebKit/604.1 (KHTML, like Gecko) Version/11.0 Mobile/15A372 Safari/604.1"
```

Figure 56. Http server running on localhost:8081.

In another console instance run `ionic serve` to open the common app (not built).

```
$ ionic serve
> ng run app:serve --host=0.0.0.0 --port=8100
[ng] WARNING: This is a simple server for use in testing or debugging Angular applications
[ng] locally. It hasn't been reviewed for security issues.
[ng] Binding this server to an open connection can result in compromising your application or
[ng] computer. Using a different host than the one passed to the "--host" flag might result in
[ng] websocket connection issues. You might need to use "--disableHostCheck" if that's the
[ng] case.
[INFO] Waiting for connectivity with ng...
[INFO] Development server running!
Local: http://localhost:8100
External: http://10.80.132.29:8100, http://192.168.56.1:8100, http://192.168.99.1:8100
Use Ctrl+C to quit this process
[INFO] Browser window opened to http://localhost:8100!
```

Figure 57. Ionic server running on localhost:8100.

The first difference can be found on *Developer tools → application*, here it is seen that the PWA application (left) has a service worker and the common one does not.



Figure 58. Application service worker comparison.

If the "offline" box is checked, it will force a disconnection from network. In situations where users do not have connectivity or have a slow, one the PWA can still be accessed and used.



Figure 59. Offline application.

Finally, plugins like [Lighthouse](#) can be used to test whether an application is progressive or not.



Figure 60. Lighthouse report.

19. Layouts

19.1. Angular Material Layout

The purpose of this guide is to get a basic understanding of creating layouts using [Angular Material](#) in a devon4ng application. We will create an application with a header containing some menu links and a sidenav with some navigation links.



Figure 61. This is what the finished application will look like

19.1.1. Let's begin

We start with opening the console(running `console.bat` in the Devon distribution folder) and running the following command to start a project named `devon4ng-mat-layout`

- `ng new devon4ng-mat-layout`

Select `y` when it asks whether it would like to add Angular routing and select `SCSS` when it asks for the stylesheet format. You can also use the **devonfw-ide** CLI to create a new devon4ng application.

Once the creation process is complete, open your newly created application in Visual Studio Code. Try running the empty application by running the following command in the integrated terminal:

- `ng serve`

Angular will spin up a server and you can check your application by visiting <http://localhost:4200/> in your browser.

Welcome to devon4ng-mat-layout!



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

Figure 62. Blank application

19.1.2. Adding Angular Material library to the project

Next we will add Angular Material to our application. In the integrated terminal, press **Ctrl + C** to terminate the running application and run the following command:

- `npm install --save @angular/material @angular/cdk @angular/animations`

You can also use Yarn to install the dependencies if you prefer that:

- `yarn add @angular/material @angular/cdk @angular/animations`

Once the dependencies are installed, we need to import the `BrowserAnimationsModule` in our `AppModule` for animations support.

Listing 68. Importing `BrowserAnimationsModule` in `AppModule`

```
import {BrowserAnimationsModule} from '@angular/platform-browser/animations';

@NgModule({
  ...
  imports: [BrowserAnimationsModule],
  ...
})
export class AppModule { }
```

Angular Material provides a host of components for designing our application. All the components are well structured into NgModules. For each component from the Angular Material library that we want to use, we have to import the respective NgModule.

Listing 69. We will be using the following components in our application:

```
import { MatIconModule, MatButtonModule, MatMenuModule, MatListModule,
MatToolbarModule, MatSidenavModule } from '@angular/material';

@NgModule({
  ...
  imports: [
    ...
    MatIconModule,
    MatButtonModule,
    MatMenuModule,
    MatListModule,
    MatToolbarModule,
    MatSidenavModule,
    ...
  ],
  ...
})
export class AppModule { }
```

A better approach is to import and then export all the required components in a shared module. But for the sake of simplicity, we are importing all the required components in the AppModule itself.

Next, we include a theme in our application. Angular Material comes with four inbuilt themes: indigo-pink, deeppurple-amber, pink-bluegrey and purple-green. It is also possible to create our own custom theme, but that is beyond the scope of this guide. Including a theme is required to apply all of the core and theme styles to your application. We will include the indigo-pink theme in our application by importing the `indigo-pink.css` file in our `src/styles.scss`:

Listing 70. In `src/styles.scss`:

```
@import "~@angular/material/prebuilt-themes/indigo-pink.css";
```

Some Angular Material components depend on HammerJs for gestures. So it is a good idea to install HammerJs as a dependency in our application. To do so, run the following command in the terminal:

- `npm install --save hammerjs`

Then import it in the `src/main.ts` file

- `import 'hammerjs';`

To use [Material Design Icons](#) along with the `mat-icon` component, we will load the Material Icons library in our `src/index.html` file

Listing 71. In src/index.html:

```
<link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
```

19.1.3. Development

Now that we have all the Angular Material related dependencies set up in our project, we can start coding. Let's begin by adding a suitable `margin` and `font` to the `body` element of our single page application. We will add it in the `src/styles.scss` file to apply it globally:

Listing 72. In src/styles.scss:

```
body {
  margin: 0;
  font-family: "Segoe UI", Roboto, sans-serif;
}
```

At this point, if we run our application with `ng serve`, this is how it will look like:

Welcome to devon4ng-mat-layout!



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

Figure 63. Application with Angular Material set up

We will clear the `app.component.html` file and setup a header with a menu button and some navigational links. We will use `mat-toolbar`, `mat-button`, `mat-menu`, `mat-icon` and `mat-icon-button` for this:

Listing 73. app.component.html:

```
<mat-toolbar color="primary">
  <button mat-icon-button aria-label="menu">
    <mat-icon>menu</mat-icon>
  </button>
  <button mat-button [matMenuTriggerFor]="submenu">Menu 1</button>
  <button mat-button>Menu 2</button>
  <button mat-button>Menu 3</button>

  <mat-menu #submenu="matMenu">
    <button mat-menu-item>Sub-menu 1</button>
    <button mat-menu-item [matMenuTriggerFor]="submenu2">Sub-menu 2</button>
  </mat-menu>

  <mat-menu #submenu2="matMenu">
    <button mat-menu-item>Menu Item 1</button>
    <button mat-menu-item>Menu Item 2</button>
    <button mat-menu-item>Menu Item 3</button>
  </mat-menu>

</mat-toolbar>
```

The color attribute on the `mat-toolbar` element will give it the primary (indigo) color as defined by our theme. The color attribute works with most Angular Material components; the possible values are 'primary', 'accent' and 'warn'. The mat-toolbar is a suitable component to represent a header. It serves as a placeholder for elements we want in our header. Inside the mat-toolbar, we start with a button having `mat-icon-button` attribute, which itself contains a `mat-icon` element having the value `menu`. This will serve as a menu button which we can use to toggle the sidenav. We follow it with some sample buttons having the `mat-button` attribute. Notice the first button has a property `matMenuTriggerFor` binded to a local reference `submenu`. As the property name suggests, the click of this button will display the `mat-menu` element with the specified local reference as a drop-down menu. The rest of the code is self explanatory.



Figure 64. This is how our application looks with the first menu button (Menu 1) clicked.

We want to keep the sidenav toggling menu button on the left and move the rest to the right to make it look better. To do this we add a class to the menu icon button:

Listing 74. app.component.html:

```
...
<button mat-icon-button aria-label="menu" class="menu">
  <mat-icon>menu</mat-icon>
</button>
...
```

And in the `app.component.scss` file, we add the following style:

Listing 75. app.component.scss:

```
.menu {
  margin-right: auto;
}
```

The mat-toolbar element already has its display property set to `flex`. Setting the menu icon button's `margin-right` property to `auto` keeps itself on the left and pushes the other elements to the right.

Figure 65. Final look of the header.

Next, we will create a sidenav. But before that lets create a couple of components to navgate between, the links of which we will add to the sidenav. We will use the `ng generate component` (or `ng g c` command for short) to create `Home` and `Data` components. We nest them in the `pages` subdirectory since they represent our pages.

- `ng g c pages/home`
- `ng g c pages/data';`

Let us set up the routing such that when we visit `http://localhost:4200/` root url we see the `HomeComponent` and when we visit `http://localhost:4200/data` url we see the `DataComponent`. We had opted for routing while creating the application, so we have the routing module `app-routing.module.ts` setup for us. In this file, we have the empty `routes` array where we set up our routes.

Listing 76. `app-routing.module.ts`:

```
...
import { HomeComponent } from './pages/home/home.component';
import { DataComponent } from './pages/data/data.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'data', component: DataComponent }
];
...
```

We need to provide a hook where the components will be loaded when their respective URLs are loaded. We do that by using the `router-outlet` directive in the `app.component.html`.

Listing 77. app.component.html:

```
...
</mat-toolbar>
<router-outlet></router-outlet>
```

Now when we visit the defined URLs we see the appropriate components rendered on screen.

Lets change the contents of the components to have something better.

Listing 78. home.component.html:

```
<h2>Home Page</h2>
```

Listing 79. home.component.scss:

```
h2 {
  text-align: center;
  margin-top: 50px;
}
```

Listing 80. data.component.html:

```
<h2>Data Page</h2>
```

Listing 81. data.component.scss:

```
h2 {
  text-align: center;
  margin-top: 50px;
}
```

The pages look somewhat better now:



Home Page

Figure 66. Home page



Data Page

Figure 67. Data page

Let us finally create the sidenav. To implement the sidenav we need to use 3 Angular Material components: `mat-sidenav-container`, `mat-sidenav` and `mat-sidenav-content`. The `mat-sidenav-container`, as the name suggests, acts as a container for the sidenav and the associated content. So it is the parent element, and `mat-sidenav` and `mat-sidenav-content` are the children sibling elements. `mat-sidenav` represents the sidenav. We can put any content we want, though it is usually used to contain a list of navigational links. The `mat-sidenav-content` element is for containing our main page content. Since we need the sidenav application-wide, we will put it in the `app.component.html`.

Listing 82. app.component.html:

```
...
</mat-toolbar>

<mat-sidenav-container>
  <mat-sidenav mode="over" [disableClose]="false" #sidenav>
    Sidenav
  </mat-sidenav>
  <mat-sidenav-content>
    <router-outlet></router-outlet>
  </mat-sidenav-content>
</mat-sidenav-container>
```

The `mat-sidenav` has a `mode` property, which accepts one of the 3 values: `over`, `push` and `side`. It decides the behavior of the sidenav. `mat-sidenav` also has a `disableClose` property which accepts a boolean value. It toggles the behavior where we click on the backdrop or press the `Esc` key to close the sidenav. There are other properties which we can use to customize the appearance, behavior and position of the sidenav. You can find the properties documented online at <https://material.angular.io/components/sidenav/api> We moved the `router-outlet` directive inside the `mat-sidenav-content` where it will render the routed component. But if you check the running application in the browser, we don't see the sidenav yet. That is because it is closed. We want to have the sidenav opened/closed at the click of the menu icon button on the left side of the header we implemented earlier. Notice we have set a local reference `#sidenav` on the `mat-sidenav` element. We can access this element and call its `toggle()` function to toggle open or close the sidenav.

Listing 83. app.component.html:

```
...
<button mat-icon-button aria-label="menu" class="menu" (click)="sidenav.toggle()">
  <mat-icon>menu</mat-icon>
</button>
...
```

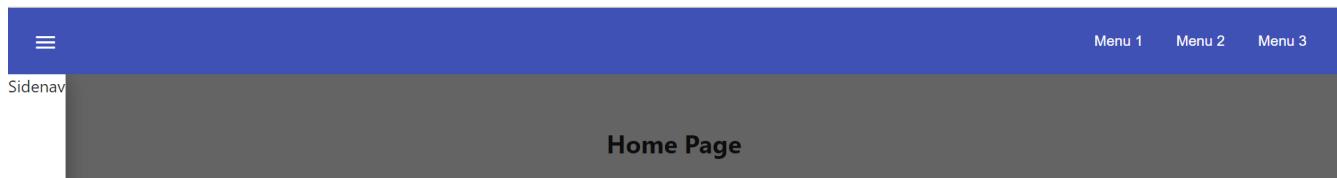


Figure 68. Sidenav is implemented

We can now open the sidenav by clicking the menu icon button. But it does not look right. The sidenav is only as wide as its content. Also the page does not stretch the entire viewport due to lack of content. Let's add the following styles to make the page fill the viewport:

Listing 84. `app.component.scss`:

```
...
mat-sidenav-container {
  position: absolute;
  top: 64px;
  left: 0;
  right: 0;
  bottom: 0;
}
```

The sidenav's width will be corrected when we add the navigational links to it. That is the only thing remaining to be done. Lets implement it now:

Listing 85. app.component.html:

```
...
<mat-sidenav [disableClose]="false" mode="over" #sidenav>
  <mat-nav-list>
    <a
      id="home"
      mat-list-item
      [routerLink]="/"
      (click)="sidenav.close()"
      routerLinkActive="active"
      [routerLinkActiveOptions]={{exact: true}}>
      >
        <mat-icon matListAvatar>home</mat-icon>
        <h3 matLine>Home</h3>
        <p matLine>sample home page</p>
    </a>
    <a
      id="sampleData"
      mat-list-item
      [routerLink]="/data"
      (click)="sidenav.close()"
      routerLinkActive="active">
      >
        <mat-icon matListAvatar>grid_on</mat-icon>
        <h3 matLine>Data</h3>
        <p matLine>sample data page</p>
    </a>
  </mat-nav-list>
</mat-sidenav>
...
```

We use the `mat-nav-list` element to set a list of navigational links. We use the `a` tags with `mat-list-item` directive. We implement a `click` listener on each link to close the sidenav when it is clicked. The `routerLink` directive is used to provide the URLs to navigate to. The `routerLinkActive` directive is used to provide the class name which will be added to the link when its URL is visited. Here we name the class `active`. To style it, let's modify the `app.component.scss` file:

Listing 86. app.component.scss:

```
...
mat-sidenav-container {
...
  a.active {
    background: #8e8d8d;
    color: #fff;

    p {
      color: #4a4a4a;
    }
  }
}
}
```

Now we have a working application with a basic layout: a header with some menu and a sidenav with some navigational links.

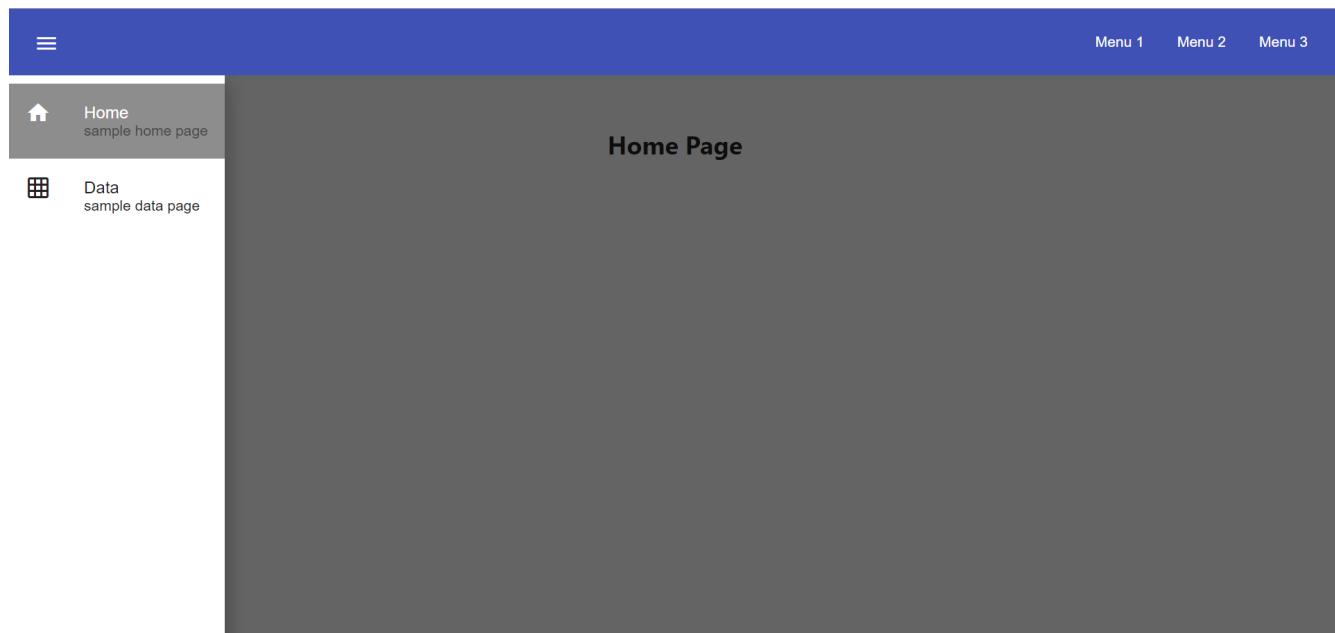


Figure 69. Finished application

19.1.4. Conclusion

The purpose of this guide was to provide a basic understanding of creating layouts with Angular Material. The Angular Material library has a huge collection of ready to use components which can be found at <https://material.angular.io/components/categories>. It has provided documentation and example usage for each of its components. Going through the documentation will give a better understanding of using Angular Material components in our devon4ng applications.

20. NgRx

20.1. Introduction to NgRx

NgRx is a state management framework for Angular based on the [Redux](#) pattern.

20.1.1. The need for client side state management

You may wonder why you should bother with state management. Usually data resides in a backend storage system, e.g. a database, and is retrieved by the client on a per-need basis. To add, update, or delete entities from this store, clients have to invoke API endpoints at the backend. Mimicking database-like transactions on the client side may seem redundant. However, there are many use cases for which a global client-side state is appropriate:

- the client has some kind of global state which should survive the destruction of a component, but does not warrant server side persistence, for example: volume level of media, expansion status of menus
- sever side data should not be retrieved every time it is needed, either because multiple components consume it, or because it should be cached, e.g. the personal watchlist in an online streaming app
- the app provides a rich experience with offline functionality, e.g. a native app built with [Ionic](#)

Saving global states inside the services they originates from results in a data flow that is hard to follow and state becoming inconsistent due to unordered state mutations. Following the *single source of truth* principle, there should be a central location holding all your application's state, just like a server side database does. State management libraries for Angular provide tools for storing, retrieving, and updating client-side state.

20.1.2. Why NgRx?

As stated in the [introduction](#), devon4ng does not stipulate a particular state library, or require using one at all. However, NgRx has proven to be a robust, mature solution for this task, with good tooling and 3rd-party library support. Albeit introducing a level of indirection that requires additional effort even for simple features, the redux concept enforces a clear separation of concerns leading to a cleaner architecture.

Nonetheless, you should always compare different approaches to state management and pick the best one suiting your use case. Here's a (non-exhaustive) list of competing state management libraries:

- Plain Rxjs using the simple store described in [Abstract Class Store](#)
- [NgXS](#) reduces some boilerplate of NgRx by leveraging the power of decorators and moving side effects to the store
- [MobX](#) follows a more imperative approach in contrast to the functional Redux pattern
- [Akita](#) also uses an imperative approach with direct setters in the store, but keeps the concept of immutable state transitions

20.1.3. Setup

To get a quick start, use the provided [template for devon4ng + NgRx](#).

To manually install the core store package together with a set of useful extensions:

NPM:

```
npm install @ngrx/store @ngrx/effects @ngrx/entity @ngrx/store-devtools --save
```

Yarn:

```
yarn add @ngrx/store @ngrx/effects @ngrx/entity @ngrx/store-devtools
```

We recommend to add the NgRx schematics to your project so you can create code artifacts from the command line:

NPM:

```
npm install @ngrx/schematics --save-dev
```

Yarn:

```
yarn add @ngrx/schematics --dev
```

Afterwards, make NgRx your default schematics provider, so you don't have to type the qualified package name every time:

```
ng config cli.defaultCollection @ngrx/schematics
```

If you have custom settings for Angular schematics, you have to [configure them as described here](#).

20.1.4. Concept

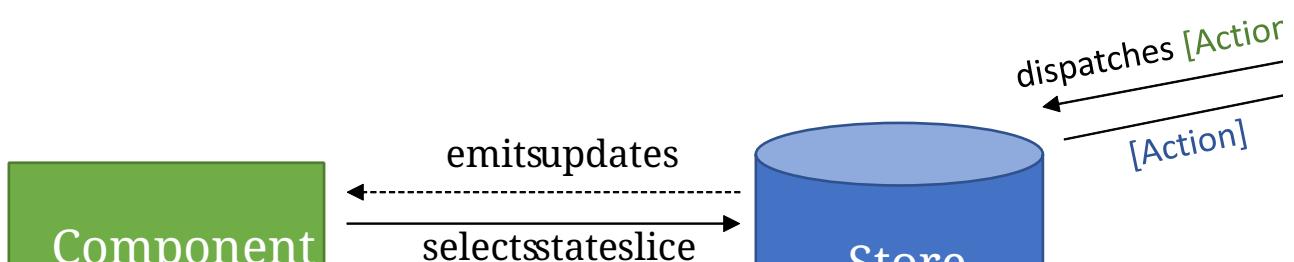


Figure 70. NgRx architecture overview

Figure 1 gives an overview of the NgRx data flow. The single source of truth is managed as an immutable state object by the store. Components dispatch actions to trigger state changes. Actions are handed over to reducers, which take the current state and action data to compute the next state. Actions are also consumed by effects, which perform side-effects such as retrieving data from the backend, and may dispatch new actions as a result. Components subscribe to state changes using selectors.

Continue with [Creating a Simple Store](#).

20.2. State, Selection and Reducers

20.2.1. Creating a Simple Store

In the following pages we use the example of an online streaming service. We will model a particular feature, a watchlist that can be populated by the user with movies she or he wants to see in the future.

Initializing NgRx

If you're starting fresh, you first have to initialize NgRx and create a root state. The fastest way to do this is using the schematic:

```
ng generate @ngrx/schematics:store State --root --module app.module.ts
```

This will automatically generate a root store and register it in the app module. Next we generate a feature module for the watchlist:

```
ng generate module watchlist
```

and create a corresponding feature store:

```
ng generate store watchlist/Watchlist -m watchlist.module.ts
```

This generates a file [watchlist/reducers/index.ts](#) with the reducer function, and registers the store in the watchlist module declaration.



If you're getting an error *Schematic "store" not found in collection "@schematics/angular"*, this means you forgot to register the NgRx schematics as default.

Next, add the WatchlistModule to the AppModule imports so the feature store is registered when the application starts. We also added the **store devtools** which we will use later, resulting in the following file:

app.module.ts

```

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { EffectsModule } from '@ngrx/effects';
import { AppEffects } from './app.effects';
import { StoreModule } from '@ngrx/store';
import { reducers, metaReducers } from './reducers';
import { StoreDevtoolsModule } from '@ngrx/store-devtools';
import { environment } from '../environments/environment';
import { WatchlistModule } from './watchlist/watchlist.module';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    WatchlistModule,
    StoreModule.forRoot(reducers, { metaReducers }),
    // Instrumentation must be imported after importing StoreModule (config is
    optional)
    StoreDevtoolsModule.instrument({
      maxAge: 25, // Retains last 25 states
      logOnly: environment.production, // Restrict extension to log-only mode
    }),
    !environment.production ? StoreDevtoolsModule.instrument() : []
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Create an entity model and initial state

We need a simple model for our list of movies. Create a file `watchlist/models/movies.ts` and insert the following code:

```

export interface Movie {
  id: number;
  title: string;
  releaseYear: number;
  runtimeMinutes: number;
  genre: Genre;
}

export type Genre = 'action' | 'fantasy' | 'sci-fi' | 'romantic' | 'comedy' |
'mystery';

export interface WatchlistItem {
  id: number;
  movie: Movie;
  added: Date;
  playbackMinutes: number;
}

```



We discourage putting several types into the same file and do this only for the sake of keeping this tutorial brief.

Later we will learn how to retrieve data from the backend using effects. For now we will create an initial state for the user with a default movie.

State is defined and transforms by a reducer function. Let's create a watchlist reducer:

```

cd watchlist/reducers
ng g reducer WatchlistData --reducers index.ts

```

Open the generated file `watchlist-data.reducer.ts`. You see three exports: The **State** interface defines the shape of the state. There is only one instance of a feature state in the store at all times. The **initialState** constant is the state at application creation time. The **reducer** function will later be called by the store to produce the next state instance based on the current state and an action object.

Let's put a movie into the user's watchlist:

watchlist-data.reducer.ts

```

export interface State {
  items: WatchlistItem[];
}

export const initialState: State = {
  items: [
    {
      id: 42,
      movie: {
        id: 1,
        title: 'Die Hard',
        genre: 'action',
        releaseYear: 1988,
        runtimeMinutes: 132
      },
      playbackMinutes: 0,
      added: new Date(),
    }
  ]
};

```

Select the current watchlist

State slices can be retrieved from the store using selectors.

Create a watchlist component:

```
ng g c watchlist/Watchlist
```

and add it to the exports of WatchlistModule. Also, replace `app.component.html` with

```
<app-watchlist></app-watchlist>
```

State observables are obtained using selectors. They are memoized by default, meaning that you don't have to worry about performance if you use complicated calculations when deriving state—these are only performed once per state emission.

Add a selector to `watchlist-data.reducer.ts`:

```
export const getAllItems = (state: State) => state.items;
```

Next, we have to re-export the selector for this substate in the feature reducer. Modify the `watchlist/reducers/index.ts` like this:

watchlist/reducers/index.ts

```

import {
  ActionReducer,
  ActionReducerMap,
  createFeatureSelector,
  createSelector,
  MetaReducer
} from '@ngrx/store';
import { environment } from 'src/environments/environment';
import * as fromWatchlistData from './watchlist-data.reducer';
import * as fromRoot from 'src/app/reducers/index';

export interface WatchlistState { ①
  watchlistData: fromWatchlistData.State;
}

export interface State extends fromRoot.State { ②
  watchlist: WatchlistState;
}

export const reducers: ActionReducerMap<WatchlistState> = { ③
  watchlistData: fromWatchlistData.reducer,
};

export const metaReducers: MetaReducer<WatchlistState>[] = !environment.production ?
[] : [];

export const getFeature = createFeatureSelector<State, WatchlistState>('watchlist'); ④

export const getWatchlistData = createSelector( ⑤
  getFeature,
  state => state.watchlistData
);

export const getAllItems = createSelector( ⑥
  getWatchlistData,
  fromWatchlistData.getAllItems
);

```

① The feature state, each member is managed by a different reducer

② Feature states are registered by the `forFeature` method. This interface provides a typesafe path from root to feature state.

③ Tie substates of a feature state to the corresponding reducers

④ Create a selector to access the 'watchlist' feature state

⑤ select the watchlistData sub state

⑥ re-export the selector

Note how `createSelector` allows to chain selectors. This is a powerful tool that also allows for

selecting from multiple states.

You can use selectors as pipeable operators:

watchlist.component.ts

```
export class WatchlistComponent {
  watchlistItems$: Observable<WatchlistItem[]>

  constructor(
    private store: Store<fromWatchlist.State>
  ) {
    this.watchlistItems$ = this.store.pipe(select(fromWatchlist.getAllItems));
  }
}
```

watchlist.component.html

```
<h1>Watchlist</h1>
<ul>
  <li *ngFor="let item of watchlistItems$ | async">{{item.movie.title}}
  ({{item.movie.releaseYear}}): {{item.playbackMinutes}}/{{item.movie.runtimeMinutes}}
  min watched</li>
</ul>
```

Dispatching an action to update watched minutes

We track the user's current progress at watching a movie as the `playbackMinutes` property. After closing a video, the watched minutes have to be updated. In NgRx, state is being updated by dispatching actions. An action is an option with a (globally unique) type discriminator and an optional payload.

Creating the action

Create a file `playback/actions/index.ts`. In this example, we do not further separate the actions per sub state. Actions can be defined by using action creators:

playback/actions/index.ts

```

import { createAction, props, union } from '@ngrx/store';

export const playbackFinished = createAction('[Playback] Playback finished', props<{
  movieId: number, stoppedAtMinute: number }>());

const actions = union({
  playbackFinished
});

export type ActionsUnion = typeof actions;

```

First we specify the type, followed by a call to the payload definition function. Next, we create a union of all possible actions for this file using `union`, which allows us to access action payloads in the reducer in a typesafe way.



Action types should follow the naming convention [\[Source\] Event](#), e.g. [\[Recommended List\] Hide Recommendation](#) or [\[Auth API\] Login Success](#). Think of actions rather as events than commands. You should never use the same action at two different places (you can still handle multiple actions the same way). This facilitates tracing the source of an action. For details see [Good Action Hygiene with NgRx](#) by Mike Ryan (video).

Dispatch

We skip the implementation of an actual video playback page and simulate watching a movie in 10 minute segments by adding a link in the template:

watchlist-component.html

```

<li *ngFor="let item of watchlistItems$ | async">... <button
(click)="stoppedPlayback(item.movie.id, item.playbackMinutes + 10)">Add 10
Minutes</button></li>

```

watchlist-component.ts

```

import * as playbackActions from 'src/app/playback/actions';
...
  stoppedPlayback(movieId: number, stoppedAtMinute: number) {
    this.store.dispatch(playbackActions.playbackFinished({ movieId, stoppedAtMinute
}));}
}

```

State reduction

Next, we handle the action inside the `watchlistData` reducer. Note that actions can be handled by multiple reducers and effects at the same time to update different states, for example if we'd like to show a rating modal after playback has finished.

watchlist-data.reducer.ts

```

export function reducer(state = initialState, action: playbackActions.ActionsUnion): State {
  switch (action.type) {
    case playbackActions.playbackFinished.type:
      return {
        ...state,
        items: state.items.map(updatePlaybackMinutesMapper(action.movieId, action.stoppedAtMinute))
      };

    default:
      return state;
  }
}

export function updatePlaybackMinutesMapper(movieId: number, stoppedAtMinute: number) {
  return (item: WatchlistItem) => {
    if (item.movie.id === movieId) {
      return {
        ...item,
        playbackMinutes: stoppedAtMinute
      };
    } else {
      return item;
    }
  };
}

```

Note how we changed the reducer's function signature to reference the actions union. The switch-case handles all incoming actions to produce the next state. The default case handles all actions a reducer is not interested in by returning the state unchanged. Then we find the watchlist item corresponding to the movie with the given id and update the playback minutes. Since state is immutable, we have to clone all objects down to the one we would like to change using the object spread operator (...).



Selectors rely on object identity to decide whether the value has to be recalculated. Do not clone objects that are not on the path to the change you want to make. This is why `updatePlaybackMinutesMapper` returns the same item if the movie id does not match.

Alternative state mapping with immer

It can be hard to think in immutable changes, especially if your team has a strong background in imperative programming. In this case, you may find the `immer` library convenient, which allows to produce immutable objects by manipulating a proxied draft. The same reducer can then be written as:

watchlist-data.reducer.ts with immer

```
import { produce } from 'immer';
...
case playbackActions.playbackFinished.type:
    return produce(state, draft => {
        const itemToUpdate = draft.items.find(item => item.movie.id ===
action.movieId);
        if (itemToUpdate) {
            itemToUpdate.playbackMinutes = action.stoppedAtMinute;
        }
    });
});
```

Immer works out of the box with plain objects and arrays.

Redux devtools

If the [StoreDevToolsModule](#) is instrumented as described above, you can use the browser extension [Redux devtools](#) to see all dispatched actions and the resulting state diff, as well as the current state, and even travel back in time by undoing actions.

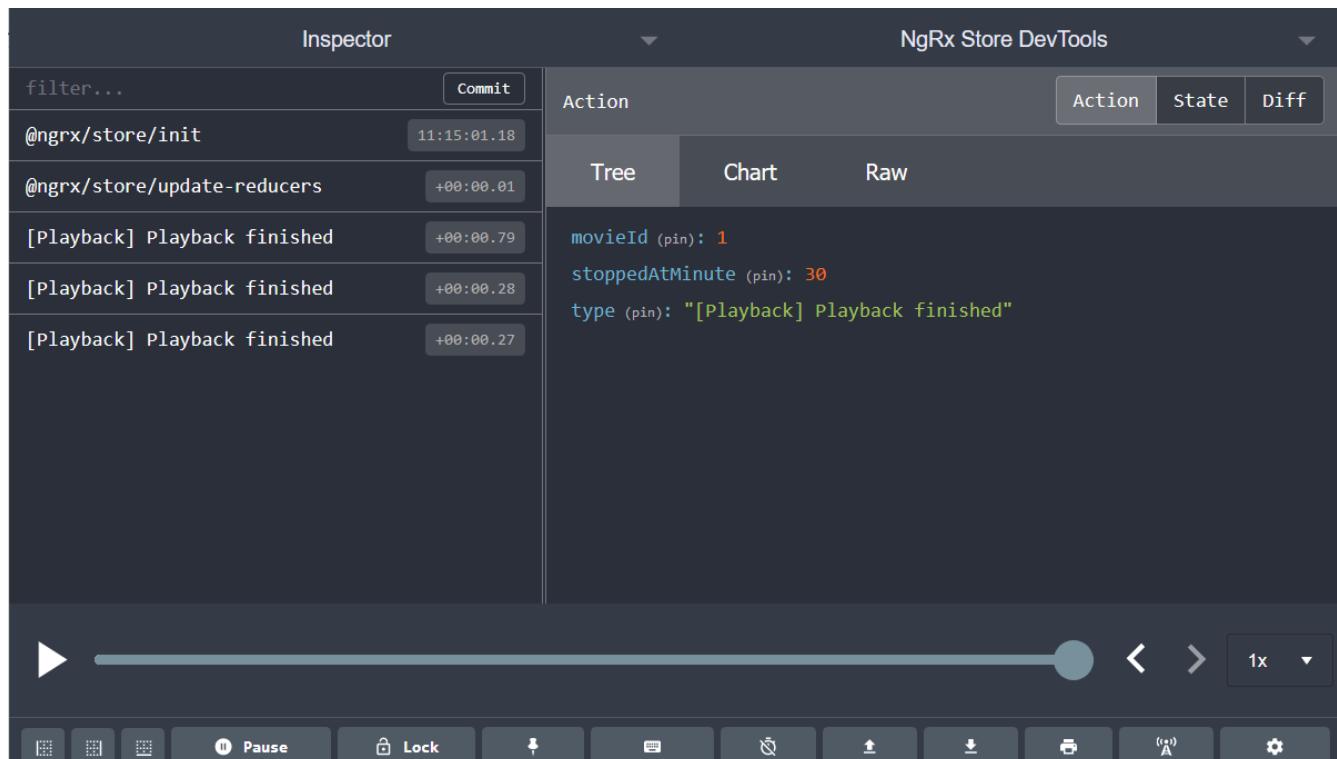


Figure 71. Redux devtools

Continue with [learning about effects](#)

20.3. Side effects with NgRx/Effects

Reducers are pure functions, meaning they are side-effect free and deterministic. Many actions however have side effects like sending messages or displaying a toast notification. NgRx encapsulates these actions in effects.

Let's build a recommended movies list so the user can add movies to their watchlist.

20.3.1. Obtaining the recommendation list from the server

Create a module for recommendations and add stores and states as in the previous chapter. Add `EffectsModule.forRoot([])` to the imports in `AppModule` below `StoreModule.forRoot()`. Add effects to the feature module:

```
ng generate effect recommendation/Recommendation -m
recommendation/recommendation.module.ts
```

We need actions for loading the movie list, success and failure cases:

`recommendation/actions/index.ts`

```
import { createAction, props, union } from '@ngrx/store';
import { Movie } from 'src/app/watchlist/models/movies';

export const loadRecommendedMovies = createAction('[Recommendation List] Load
movies');
export const loadRecommendedMoviesSuccess = createAction('[Recommendation API] Load
movies success', props<{movies: Movie[]}>());
export const loadRecommendedMoviesFailure = createAction('[Recommendation API] Load
movies failure', props<{error: any}>());

const actions = union({
  loadRecommendedMovies,
  loadRecommendedMoviesSuccess,
  loadRecommendedMoviesFailure
});

export type ActionsUnion = typeof actions;
```

In the reducer, we use a loading flag so the UI can show a loading spinner. The store is updated with arriving data.

`recommendation/actions/index.ts`

```

export interface State {
  items: Movie[];
  loading: boolean;
}

export const initialState: State = {
  items: [],
  loading: false
};

export function reducer(state = initialState, action:
recommendationActions.ActionsUnion): State {
  switch (action.type) {
    case '[Recommendation List] Load movies':
      return {
        ...state,
        items: [],
        loading: true
      };

    case '[Recommendation API] Load movies failure':
      return {
        ...state,
        loading: false
      };

    case '[Recommendation API] Load movies success':
      return {
        ...state,
        items: action.movies,
        loading: false
      };

    default:
      return state;
  }
}

export const getAll = (state: State) => state.items;
export const isLoading = (state: State) => state.loading;

```

We need an API service to talk to the server. For demonstration purposes, we simulate an answer delayed by one second:

recommendation/services/recommendation-api.service.ts

```
@Injectable({
  providedIn: 'root'
})
export class Recommendation ApiService {

  private readonly recommendedMovies: Movie[] = [
    {
      id: 2,
      title: 'The Hunger Games',
      genre: 'sci-fi',
      releaseYear: 2012,
      runtimeMinutes: 144
    },
    {
      id: 4,
      title: 'Avengers: Endgame',
      genre: 'fantasy',
      releaseYear: 2019,
      runtimeMinutes: 181
    }
  ];

  loadRecommendedMovies(): Observable<Movie[]> {
    return of(this.recommendedMovies).pipe(delay(1000));
  }
}
```

Here are the effects:

recommendation/services/recommendation-api.service.ts

```

@Injectable()
export class RecommendationEffects {

  constructor(
    private actions$: Actions,
    private recommendationApi: Recommendation ApiService,
  ) { }

  @Effect()
  loadBooks$ = this.actions$.pipe(
    ofType(recommendationActions.loadRecommendedMovies.type),
    switchMap(() => this.recommendationApi.loadRecommendedMovies().pipe(
      map(movies => recommendationActions.loadRecommendedMoviesSuccess({ movies })),
      catchError(error => of(recommendationActions.loadRecommendedMoviesFailure({
        error })))
    ))
  );
}

```

Effects are always observables and return actions. In this example, we consume the actions observable provided by NgRx and listen only for the `loadRecommendedMovies` actions by using the `ofType` operator. Using `switchMap`, we map to a new observable, one that loads movies and maps the successful result to a new `loadRecommendedMoviesSuccess` action or a failure to `loadRecommendedMoviesFailure`. In a real application we would show a notification in the error case.



If an effect should not dispatch another action, return an empty observable.

[Continue reading how to simplify CRUD \(Create Read Update Delete\) operations using @ngrx/entity.](#)

20.4. Simplifying CRUD with NgRx/Entity

Most of the time when manipulating entries in the store, we like to create, add, update, or delete entries (CRUD). NgRx/Entity provides convenience functions if each item of a collection has an id property. Luckily all our entities already have this property.

Let's add functionality to add a movie to the watchlist. First, create the required action:

recommendation/actions/index.ts

```

export const addToWatchlist = createAction('[Recommendation List] Add to watchlist',
  props<{ watchlistItemId: number, movie: Movie, addedAt: Date }>());

```



You may wonder why the `Date` object is not created inside the reducer instead, since it should always be the current time. However, remember that reducers should be deterministic state machines—State A + Action B should always result in the same State C. This makes reducers easily testable.

Then, rewrite the watchlistData reducer to make use of NgRx/Entity:

recommendation/actions/index.ts

```
export interface State extends EntityState<WatchlistItem> { ①
}

export const entityAdapter = createEntityAdapter<WatchlistItem>(); ②

export const initialState: State = entityAdapter.getInitialState(); ③

const entitySelectors = entityAdapter.getSelectors();

export function reducer(state = initialState, action: playbackActions.ActionsUnion | recommendationActions.ActionsUnion): State {
  switch (action.type) {
    case playbackActions.playbackFinished.type:
      const itemToUpdate = entitySelectors
        .selectAll(state) ④
        .find(item => item.movie.id === action.movieId);
      if (itemToUpdate) {
        return entityAdapter.updateOne({ ⑤
          id: itemToUpdate.id,
          changes: { playbackMinutes: action.stoppedAtMinute } ⑥
        }, state);
      } else {
        return state;
      }

    case recommendationActions.addToWatchlist.type:
      return entityAdapter.addOne({id: action.watchlistItemId, movie: action.movie, added: action.addedAt, playbackMinutes: 0}, state);

    default:
      return state;
  }
}

export const getAllItems = entitySelectors.selectAll;
```

- ① NgRx/Entity requires state to extend EntityState. It provides a list of ids and a dictionary of id ⇒ entity entries
- ② The entity adapter provides data manipulation operations and selectors
- ③ The state can be initialized with `getInitialState()`, which accepts an optional object to define any additional state beyond EntityState
- ④ `selectAll` returns an array of all entities
- ⑤ All adapter operations consume the state object as the last argument and produce a new state

-
- ⑥ Update methods accept a partial change definition; you don't have to clone the object

This concludes the tutorial on NgRx. If you want to learn about advanced topics such as selectors with arguments, testing, or router state, head over to the [official NgRx documentation](#).

21. Cookbook

21.1. Abstract Class Store

The following solution presents a base class for implementing stores which handle state and its transitions. Working with the base class achieves:

- common API across all stores
- logging (when activated in the constructor)
- state transitions are asynchronous by design - sequential order problems are avoided

Listing 87. Usage Example

```
@Injectable()
export class ModalStore extends Store<ModalState> {

    constructor() {
        super({ isOpen: false }, !environment.production);
    }

    closeDialog() {
        this.dispatchAction('Close Dialog', (currentState) => ({...currentState, isOpen: false}));
    }

    openDialog() {
        this.dispatchAction('Open Dialog', (currentState) => ({...currentState, isOpen: true}));
    }
}
```

Listing 88. Abstract Base Class Store

```
import { OnDestroy } from '@angular/core';
import { BehaviorSubject } from 'rxjs/BehaviorSubject';
import { Observable } from 'rxjs/Observable';
import { intersection, difference } from 'lodash';
import { map, distinctUntilChanged, observeOn } from 'rxjs/operators';
import { Subject } from 'rxjs/Subject';
import { queue } from 'rxjs/scheduler/queue';
import { Subscription } from 'rxjs/Subscription';

interface Action<T> {
    name: string;
    actionFn: (state: T) => T;
}
```

```

/** Base class for implementing stores. */
export abstract class Store<T> implements OnDestroy {

    private actionSubscription: Subscription;
    private actionSource: Subject<Action<T>>;
    private stateSource: BehaviorSubject<T>;
    state$: Observable<T>;

    /**
     * Initializes a store with initial state and logging.
     * @param initialState Initial state
     * @param logChanges When true state transitions are logged to the console.
     */
    constructor(initialState: T, public logChanges = false) {
        this.stateSource = new BehaviorSubject<T>(initialState);
        this.state$ = this.stateSource.asObservable();
        this.actionSource = new Subject<Action<T>>();

        this.actionSubscription =
            this.actionSource.pipe(observeOn(queue)).subscribe(action => {
                const currentState = this.stateSource.getValue();
                const nextState = action.actionFn(currentState);

                if (this.logChanges) {
                    this.log(action.name, currentState, nextState);
                }

                this.stateSource.next(nextState);
            });
    }

    /**
     * Selects a property from the stores state.
     * Will do distinctUntilChanged() and map() with the given selector.
     * @param selector Selector function which selects the needed property from the state.
     * @returns Observable of return type from selector function.
     */
    select<TX>(selector: (state: T) => TX): Observable<TX> {
        return this.state$.pipe(
            map(selector),
            distinctUntilChanged()
        );
    }

    protected dispatchAction(name: string, action: (state: T) => T) {
        this.actionSource.next({ name, actionFn: action });
    }

    private log(actionName: string, before: T, after: T) {
        const result: { [key: string]: { from: any, to: any } } = {};

```

```

const sameProps = intersection(Object.keys(after), Object.keys(before));
const newProps = difference(Object.keys(after), Object.keys(before));
for (const prop of newProps) {
  result[prop] = { from: undefined, to: (<any>after)[prop] };
}

for (const prop of sameProps) {
  if ((<any>before)[prop] !== (<any>after)[prop]) {
    result[prop] = { from: (<any>before)[prop], to: (<any>after)[prop] };
  }
}

console.log(this.constructor.name, actionPerformed, result);
}

ngOnDestroy() {
  this.actionSubscription.unsubscribe();
}

}

```

21.1.1. Add Electron to an Angular application

This cookbook recipe explains how to integrate Electron in an Angular 10+ application. [Electron](#) is a framework for creating native applications with web technologies like JavaScript, HTML, and CSS. As an example, very well known applications as Visual Studio Code, Atom, Slack or Skype (and many more) are using Electron too.



At the moment of this writing Angular 11.2.0, Electron 11.2.3 and Electron-builder 22.9.1 were the versions available.

Here are the steps to achieve this goal. Follow them in order.

Add Electron and other relevant dependencies

There are two different approaches to add the dependencies in the `package.json` file:

- Writing the dependencies directly in that file.
- Installing using `npm install` or `yarn add`.



Please remember if the project has a `package-lock.json` or `yarn.lock` file use `npm` or `yarn` respectively.

In order to add the dependencies directly in the `package.json` file, include the following lines in the `devDependencies` section:

```
"devDependencies": {
  ...
  "electron": "^11.2.3",
  "electron-builder": "^22.9.1",
  ...
},
```

As indicated above, instead of this `npm install` can be used:

```
$ npm install -D electron electron-builder
```

Or with `yarn`:

```
$ yarn add -D electron electron-builder
```

Create the necessary typescript configurations

In order to initiate electron in an angular app we need to modify the `tsconfig.json` file and create a `tsconfig.serve.json` and a `tsconfig.base.json` in the root folder.

`tsconfig.json`

This file needs to be modified to create references to `./src/tsconfig.app.json` and `./src/tsconfig.spec.json` to support different configurations.

```
{
  "files": [],
  "references": [
    {
      "path": "./src/tsconfig.app.json"
    },
    {
      "path": "./src/tsconfig.spec.json"
    }
  ]
}
```

`tsconfig.app.json`

```
{
  "extends": "../tsconfig.base.json",
  "compilerOptions": {
    "outDir": "../app",
    "module": "es2015",
    "baseUrl": "",
    "types": []
  },
  "include": [
    "**/*.ts",
  ],
  "exclude": [
    "**/*.spec.ts"
  ],
  "angularCompilerOptions": {
    "fullTemplateTypeCheck": true,
    "strictInjectionParameters": true,
    "preserveWhitespaces": true
  }
}
```

tsconfig.spec.json

```
{
  "extends": "../tsconfig.base.json",
  "compilerOptions": {
    "outDir": "../spec",
    "module": "commonjs",
    "types": [
      "jasmine",
      "node"
    ]
  },
  "files": [
    "test.ts",
  ],
  "include": [
    "**/*.spec.ts",
    "**/*.d.ts"
  ],
  "exclude": [
    "dist",
    "release",
    "node_modules"
  ]
}
```

tsconfig.base.json

This is shared between tsconfig.app.json and tsconfig.spec.json and it will be extended on each config file.

```
{  
  "compileOnSave": false,  
  "compilerOptions": {  
    "outDir": "./dist",  
    "sourceMap": true,  
    "declaration": false,  
    "moduleResolution": "node",  
    "emitDecoratorMetadata": true,  
    "experimentalDecorators": true,  
    "target": "es5",  
    "typeRoots": [  
      "node_modules/@types"  
    ],  
    "lib": [  
      "es2017",  
      "es2016",  
      "es2015",  
      "dom"  
    ]  
  },  
  "files": [  
    "electron-main.ts"  
    "src/polyfills.ts"  
  ],  
  "include": [  
    "src/**/*.d.ts"  
  ],  
  "exclude": [  
    "node_modules"  
  ]  
}
```

tsconfig.serve.json

In the root, `tsconfig.serve.json` needs to be created. This typescript config file is going to be used when we serve electron:

```
{  
  "compilerOptions": {  
    "outDir": ".",  
    "sourceMap": true,  
    "declaration": false,  
    "moduleResolution": "node",  
    "emitDecoratorMetadata": true,  
    "experimentalDecorators": true,  
    "target": "es5",  
    "typeRoots": [  
      "node_modules/@types"  
    ],  
    "lib": [  
      "es2017",  
      "dom"  
    ]  
  },  
  "include": [  
    "electron-main.ts"  
  ],  
  "exclude": [  
    "node_modules",  
    "**/*spec.ts"  
  ]  
}
```

Add Electron build configuration

In order to configure electron builds properly we need to create a new json on our application, let's call it `electron-builder.json`. For more information and fine tuning please refer to the [Electron Builder official documentation](#).

The contents of the file will be something similar to the following:

```
{
  "productName": "devon4ngElectron",
  "directories": {
    "output": "./builder-release"
  },
  "win": {
    "icon": "dist/assets/icons",
    "target": [
      "portable"
    ]
  },
  "mac": {
    "icon": "dist/assets/icons",
    "target": [
      "dmg"
    ]
  },
  "linux": {
    "icon": "dist/assets/icons",
    "target": [
      "AppImage"
    ]
  }
}
```

Theres two important things in this files:

1. "output": this is where electron builder is going to build our application
2. "icon": in every OS possible theres an icon parameter, the route to the icon folder that will be created after building with angular needs to be used here. This will make it so the electron builder can find the icons and build.

Modify angular.json

`angular.json` has to be modified so the project is build inside `/dist` without an intermediate folder.

```
{
  "architect": {
    "build": {
      "outputPath": "dist"
    }
  }
}
```

Create the electron window in `electron-main.ts`

In order to use electron, a file needs to be created at the root of the application (`main.ts`). This file will create a window with different settings checking if we are using `--serve` as an argument:

```
import { app, BrowserWindow } from 'electron';
import * as path from 'path';
import * as url from 'url';

let win: any;
const args: any = process.argv.slice(1);
const serve: any = args.some((val) => val === '--serve');

const createWindow: any = ()=>{
    // Create the browser window.
    win = new BrowserWindow({
        fullscreen: true,
        webPreferences: {
            nodeIntegration: true,
        }
    });

    if (serve) {
        require('electron-reload')(__dirname, {
            electron: require(`.${__dirname}/node_modules/electron`)
        });
        win.loadURL('http://localhost:4200');
    } else {
        win.loadURL(
            url.format({
                pathname: path.join(__dirname, 'dist/index.html'),
                protocol: 'file:',
                slashes: true
            })
        );
    }
}

if (serve) {
    win.webContents.openDevTools();
}

// Emitted when the window is closed.
win.on('closed', () => {
    // Dereference the window object, usually you would store window
    // in an array if your app supports multi windows, this is the time
    // when you should delete the corresponding element.
    // tslint:disable-next-line:no-null-keyword
    win = null;
});

try {
    // This method will be called when Electron has finished
    // initialization and is ready to create browser windows.
    // Some APIs can only be used after this event occurs.
```

```

app.on('ready', createWindow);

// Quit when all windows are closed.
app.on('window-all-closed', () => {
  // On OS X it is common for applications and their menu bar
  // to stay active until the user quits explicitly with Cmd + Q
  if (process.platform !== 'darwin') {
    app.quit();
  }
});

app.on('activate', () => {
  // On OS X it's common to re-create a window in the app when the
  // dock icon is clicked and there are no other windows open.
  if (win === null) {
    createWindow();
  }
});
} catch (e) {
  // Catch Error
  // throw e;
}

```

Add the electron window and improve the `package.json` scripts

Inside `package.json` the electron window that will be transformed to `electron-main.js` when building needs to be added.

```
{
  ...
  "main": "electron-main.js",
  "scripts": {...}
  ...
}
```

The `scripts` section in the `package.json` can be improved to avoid running too verbose commands. As a very complete example we can take a look to the My Thai Star's `scripts` section and copy the lines useful in your project. In any case, at least we recommend to add the following lines:

```

"scripts": {
  "ng": "ng",
  "start": "ng serve",
  "build": "ng build",
  "test": "ng test",
  "lint": "ng lint",
  "e2e": "ng e2e",
  "electron:tsc": "tsc -p tsconfig.serve.json",
  "electron:run": "npm run electron:tsc && ng build --base-href ./ && npx electron
  .",
  "electron:serve": "npm run electron:tsc && npx electron . --serve",
  "electron:pack": "npm run electron:tsc && electron-builder --dir --config
electron-builder.json",
  "electron:build": "npm run electron:tsc && electron-builder --config electron-
builder.json build"
},

```

The `electron:` scripts do the following:

- `electron:tsc`: Compiles electron TS files.
- `electron:run`: Serves Angular app and runs electron.
- `electron:serve`: Serves electron with an already running angular app (i.e. a `ng serve` command running on another terminal).
- `electron:pack`: Packs electron app.
- `electron:build`: Builds electron app.

21.2. Angular Mock Service

We've all been there: A new idea comes, let's quickly prototype it. But wait, there's no backend. What can we do?

Below you will find a solution that will get you started quick and easy. The idea is to write a simple mock service that helps us by feeding data into our components.

21.2.1. The app we start with

Let's say you have a simple boilerplate code, with your favorite styling library hooked up and you're ready to go. The [Angular Material](#) sample is a good starting place.

21.2.2. The Components

Components - are the building blocks of our application. Their main role is to enable fragments of user interfaces. They will either display data (a list, a table, a chart, etc.), or 'collect' user interaction (e.g: a form, a menu, etc.)

Components stay at the forefront of the application. They should also be reusable (as much as possible). Reusability is key for what we are trying to achieve - a stable, maintainable frontend

where multiple people can contribute and collaborate.

In our project, we are at the beginning. That means we may have more ideas than plans. We are exploring possibilites. In order to code eficiently:

- 1) We will not store mock data in the components.
- 2) We will not fetch or save data directly in the components.

[Learn more about Angular Components](#)

21.2.3. The Service

So, how do we get data in our app? How do we propagate the data to the components and how can we send user interaction from the components to the our data "manager" logic.

The answer to all these questions is an Angular Service (that we will just call a service from now on).

A service is an injectable logic that can be consumed by all the components that need it. It can carry manipulation functions and ,in our case, fetch data from a provider.

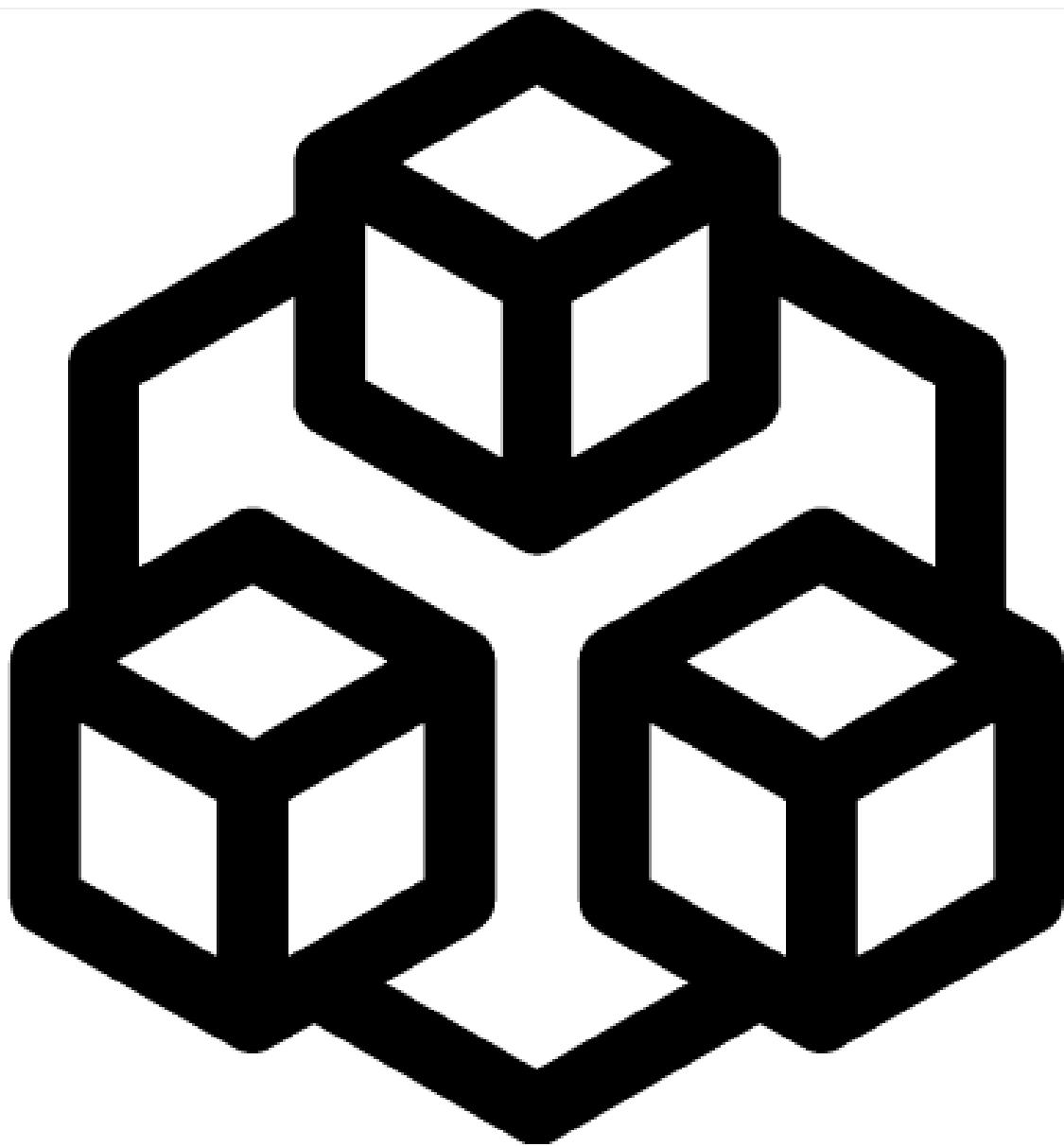


Figure 72. Angular Components & Services architecture.

Inside the Angular App, an Injector gives access to each component to their required services. It's good coding practice to use a distinct service to each data type you want to manipulate. The type is described in a interface.

Still, our ideas drive in different ways, so we have to stay flexible. We cannot use a database at the moment, but we want a way to represent data on screen, which can grow organically.

[Learn more about Angular Services](#)

21.2.4. The Model

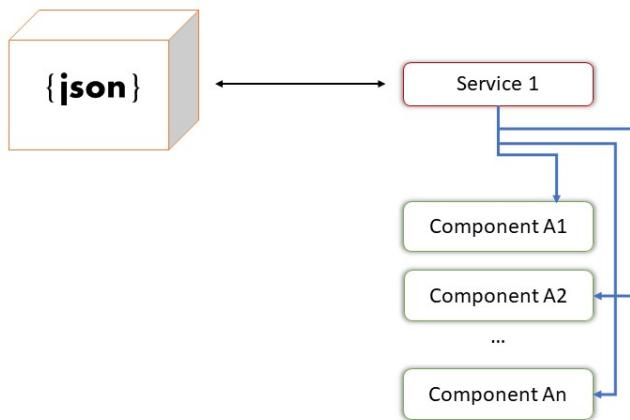


Figure 73. Data box in relation to services and components.

Let's consider a 'box of data' represented in JSON. Physically this means a folder with some JSON/TS files in it. They are located in the **app/mock** folder. The example uses only one mock data file. The file is typed according to our data model.

Pro tip: separate your files based on purpose. In your source code, put the **mock files** in the **mock folder**, **components** in the **components folder**, **services** in the **services folder** and **data models** in the **models folder**.

Figure 74. Project structure.

Aligned with the Angular way of development, we are implementing a model-view-controller pattern.

The **model** is represented by the interfaces we make. These interfaces describe the data structures we will use in our application. In this example, there is one data model, corresponding with the 'type' of data that was mocked. In the models folder you will find the .ts script file that describes chemical elements. The corresponding mock file defines a set of chemical elements objects, in accordance to our interface definition.

21.2.5. Use case

Enough with the theory, let's see what we have here. The app presents 3 pages as follows:

- A leader board with the top 3 elements
- A data table with all the elements
- A details page that reads a route parameter and displays the details of the element.

There are a lot of business cases which have these requirements:

- A leader board can be understood as "the most popular items in a set", "the latest updated

items", "you favorite items" etc.

- A data table with CRUD operations is very useful (in our case we only view details or delete an item, but they illustrate two important things: the details view shows how to navigate and consume a parametric route, the delete action shows how to invoke service operations over the loaded data - this means that the component is reusable and when the data comes with an API, only the service will need its implementation changed)

Check out the [Angular Mock Service](#) sample from the samples folder and easily get started with fast data roundtrips between your mock data and your components.

21.3. Testing e2e with Cypress

This guide will cover the basics of e2e testing using Cypress.

Cypress is a framework “all in one” that provides the necessary libraries to write specific e2e tests, without the need of Selenium.

Why Cypress?

- Uses javascript
- It works directly with the browser so the compatibility with the frontend framework the project uses (in this case Angular) is not a problem.
- Easy cross browser testing

21.3.1. Setup

Install First of all we need to install it, we can use `npm install`:

```
$ npm install -D cypress
```

Or we can install it with `yarn`:

```
$ yarn add -D cypress
```

We need to run Cypress in order to get the folder tree downloaded, then create a `tsconfig.json` file inside `cypress folder` to add the typescript configuration.

```
$ . /node_modules/.bin/cypress open
```

Listing 89. tsconfig.json

```
{
  "compilerOptions": {
    "strict": true,
    "baseUrl": "../node_modules",
    "target": "es5",
    "lib": ["es5", "dom"],
    "types": ["cypress"]
  },
  "include": [
    "**/*.ts"
  ]
}
```

BaseUrl

Let's setup the base url so when we run the tests cypress will "navegate" to the right place, go to **cypress.json** on the root of the project.

Listing 90. cypress.json

```
{
  "baseUrl": "http://localhost:4200"
}
```

21.3.2. Files / Structure

```
/cypress
  tsconfig.json
  /fixtures
    - example.json
  /integration
    - button.spec.ts
    - test.spec.ts
    /examples
  /plugins
    - index.js
  /support
    - commands.js
    - index.js
```

tsconfig.json for typescript configuration.

fixtures to store our mock data or files (img, mp3...) to use on our tests.

integration is where our tests go, by default it comes with an examples folder with tests samples.

plugins is where the configuration files of the plugins go.

support to add custom commands.

21.3.3. Tests

The structure is the same than Mocha.

First, we create a file, for example `form.spec.ts`, inside we define a context to group all our tests referred to the same subject.

Listing 91. form.spec.ts

```
context('Button page', () => {
  beforeEach(() => {
    cy.visit('/');
  });
  it('should have button', ()=>{
    cy.get('button').should('exist');
  });
  it('should contain PRESS',()=>{
    cy.contains('button', 'PRESS');
  });
});
```

beforeEach

Visit '/' before every test.

it

Inside we write the test.

The result:

```
cypress\integration\button.spec.ts

- Button page
  ✓ should have button
    - BEFORE EACH
      1 visit /
    - TEST
      1 get button
      2 - assert expected <button.mat-focus-indicator.mat-raised-button.mat-button-base.mat-primary> to exist in the DOM
  ✓ should contain Submit
```

For more info check [Cypress documentation](#)

On [kitchensink](#) you can find an official cypress demo with all the commands being used.

21.3.4. Fixtures

We use fixtures to mock data, it can be a json, an img, video...

```
{
  "name": "Dummy name",
  "phone": 999 99 99 99,
  "body": "Mock data"
}
```

You can store multiple mocks on the same fixture file.

```
{
  "create": {"name": "e2etestBox"},
  "boxFruit": {
    "uuid": "3376339576e33dfb9145362426a33333",
    "name": "e2etestBox",
    "visibility": true,
    "items": [
      {"name": "apple", "units": 3},
      {"name": "kiwi", "units": 2},
    ]
  },
}
```

To access data we don't need to import any file, we just call `cy.fixture(filename)` inside the `**.spec.ts`. We can name it as we want.

```
cy.fixture('box.json').as('fruitBox')
```

`cy.fixture('box.json')` we get access to `box.json .as(fruitBox)` is used to create an alias (fruitBox) to the fixture.

For more info check [Fixtures documentation](#)

21.3.5. Request / Route

With cypress you can test your application with real data or with mocks.

Not using mocks guarantees that your tests are real e2e test but makes them vulnerable to external issues. When you mock data you don't know exactly if the data and the structure received from the backend is correct because you are forcing a mock on the response, but you can avoid external issues, run test faster and have better control on the structure and status.

To get more information go to [Testing Strategies](#)

Route

Cypress can intercept a XHR request and interact with it.

```
cy.server();
cy.route(
  'GET',
  '/apiUrl/list',
  [{"name": "apple", "units": 3}, {"name": "kiwi", "units": 2}]
)
```

`cy.server(options)` start a server to interact with the responses.

`cy.route(options)` intercepts a XMLHttpRequests

- method `GET`
- url `/apiUrl/list'`
- response `[{"name": "apple", "units": 3}, {"name": "kiwi", "units": 2}]`

Waits

Every cypress action has a default await time to avoid asynchronous issues, but this time can be short for some particular actions like api calls, for those cases we can use `cy.wait()`.

```
cy.server();
cy.route('/apiUrl/list').as('list');
cy.visit('/boxList');
cy.wait('@list');
```

You can find more information about `cy.wait()` [here](#)

To mock data with fixtures:

```
cy.fixture('box')
  .then(({boxFruit}) => {
    cy.route(
      'GET',
      '/apiUrl/list',
      boxFruit
    ).as('boxFruit');
    cy.get('#button').click();
    cy.wait('@journalsList');
    cy.get('#list').contains('apple');
  })
```

We get `boxFruit` data from the `box` fixture and then we mock the api call with it so now the response of the call is `boxFruit` object. When the button is clicked, it waits to receive the response of the call and then checks if the list contains one of the elements of the `fruitBox`.

Request

Make a HTTP request.

```
cy.server();
cy.request('http://localhost:4200/')
  .its('body')
  .should('include', '<h1>Welcome to Devon4ngAngularElementsTest!</h1>');
```

If we have '<http://localhost:4200>' as baseUrl on `cypress.json`

```
cy.server();
cy.request('/')
  .its('body')
  .should('include', '<h1>Welcome to Devon4ngAngularElementsTest!</h1>');
// Goes to http://localhost:4200/
```

We can add other options, like we can send the body of a form.

```
cy.server();
cy.request({
  method: 'POST',
  url: '/send',
  form: true,
  body: {
    name: 'name task',
    description: 'description of the task'
  }
});
```

21.3.6. Custom commands

If you see yourself writing the same test more than once (login is a common one), you can create a custom command to make things faster.

`Cypress.Commands.add('name', ()=>{})` to create the test.

Listing 92. commands.ts

```
Cypress.Commands.add('checkPlaceholder', (name) => {
  cy.get(`[name='${name}']`).click();
  cy.get('mat-form-field.mat-focused').should('exist');
});
```

index.ts

To use the commands we need to import the files on support/index.ts

Listing 93. index.ts

```
import './commands'
import './file1'
import './folder/file2'
```

index.ts is where all our custom commands files unite so Cypress knows where to find them.

And as we are using typescript we need to define a **namespace**, **interface** and define our function.

- index.d.ts

```
declare namespace Cypress {
  interface Chainable<Subject> {
    checkPlaceholder(name:string):Chainable<void>
  }
}
```

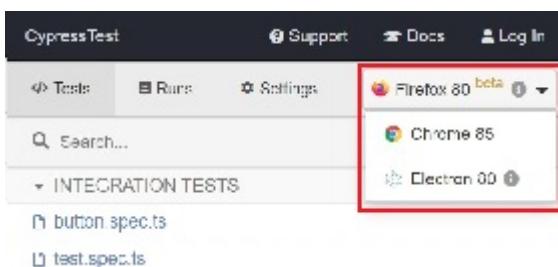
Check [typescript custom commands](#)

21.3.7. Cross browser testing

By default the browser used by Cypress is Chrome, it has compatibility with its family browsers (including Microsoft Edge) and has beta support for Mozilla Firefox.

To change the browser on the panel we can do it by selecting the desired one on the browsers tab before running the spec file.

Cypress will detect and display, except electron, only the browsers that you have already installed on your machine.



Once the browser is selected, you can run your tests.

To change the browser on the automatic test run, you can add a flag on the node command

```
cypress run --browser edge
```

Only if we use the **cypress run** command.

Or we can change the script file.

- cypress/script.js

```
const runTests= async ()=>{
  ...
  const {totalFailed} = await cypress.run({browser:'edge'});
  ...
};
```

[Cypress documentation](#)

21.3.8. Viewport

Cypress allow us to create tests depending on the viewport, so we can test responsiveness.

There are different ways to use it:

Inside a test case

```
it('should change title when viewport is less than 320px', ()=>{
  cy.get('.title-l').should('be.visible');
  cy.get('.title-s').should('not.be.visible');
  cy.viewport(320, 480);
  cy.get('.title-l').should('not.be.visible');
  cy.get('.title-s').should('be.visible');
})
```

Passing the configuration as an option

```
describe('page display on medium size screen', {
  viewportHeight: 1000,
  viewportWidth: 400
}, () => {
  ...
})
```

Or we can set a default

- cypress.json

```
...
{
  "viewportHeight": 1000
  "viewportWidth": 400,
}
...
```

21.3.9. Test retries

We can get false negatives intermittently due external issues that can affect our tests, because of that we can add, in the configuration, a retries entry so cypress can run again a certain failed test the selected number of times to verify that the error is real.

We can set retries for run or open mode.

- cypress.json

```
...
  "retries": {
    "runMode": 3,
    "openMode": 3
  }
...
...
```

The retries can be configured on the [cypress.json](#) or directly on a specific test.

```
it('should get button', {
  retries: {
    runMode: 2,
    openMode: 2
  },
  () => {
    ...
  }
})
```

This retries thoes not show on the test log.

Check more on [retries documentation](#)

21.3.10. Reporter

The tests results appear on the terminal, but to have a more friendly view we can add a reporter.

CypressTest

4.217s 2 3 3 0

Test Suite	Test Status	Time
Title	Pending 2 Passed 2	2.745s
Button	Pending 1 Passed 1	1.472s

Mochawesome

In this case we are going to use Mochawesome, initially its a Mocha reporter but as Cypress uses Mocha it works the same.

Install

npm

```
npm install --save-dev mochawesome
```

yarn

```
yarn add -D mochawesome
```

To run the reporter:

```
cypress run --reporter mochawesome
```

Mochawesome saves by default the generated files on `./mochawesome-report/` but we can add options to change this behaviour.

Options can be passed to the reporter in two ways

Using a flag

```
cypress run --reporter mochawesome --reporter-options reportDir=report
```

Or on `cypress.json`

```
{
  "baseUrl": "http://localhost:4200",
  "reporter": "mochawesome",
  "reporterOptions": {
    "overwrite": false,
    "html": false,
    "json": true,
    "reportDir": "cypress/report"
  }
}
```

Overwrite:false to not overwrite every **:spec.ts test report, we want them to create a merged version later.

reportDir to set a custom directory.

html:false because we don't need it.

json:true to save them on json.

Mochawesome only creates the html file of the last .spec.ts file that the tests run, that's why we don't generate html reports directly, in order to stack them all on the same final html we need to merge the reports.

Check the [mochawesome documentation](#)

mochawesome-merge

Mochawesome-merge is a library that helps us to merge the different json.

npm

```
npm install --save-dev mochawesome-merge
npm install --save-dev mochawesome-report-generator
```

yarn

```
yarn add -D mochawesome-merge
yarn add -D mochawesome-report-generator
```

To merge the files we execute this command:

```
mochawesome-merge cypress/report/*.json > cypress/reportFinal.json
```

reportFinal.json is the result of this merge, with that we have the data of all the spec files in one json.

We can also automate the test, merge and conversion to html using a script.

```
const cypress = require('cypress');
const fse = require('fs-extra');
const { merge } = require('mochawesome-merge');
const generator = require('mochawesome-report-generator');
const runTests= async ()=>{
    await fse.remove('mochawesome-report');
    await fse.remove('cypress/report');
    const {totalFailed} = await cypress.run();
    const reporterOptions = {
        files: ["cypress/report/*.json"]
    };
    await generateReport(reporterOptions);
    if(totalFailed !== 0){
        process.exit(2);
    };
};
const generateReport = (options)=> {
    return merge(options).then((jsonReport)=>{
        generator.create(jsonReport).then(()=>{
            process.exit();
        });
    });
};
runTests();
```

`fse.remove()` to remove older reports data.

`cypress.run()` to run the tests.

`merge(options)` we merge the jsons output from running the tests.

`generator.create(jsonReport)` then we generate the html view of the report.

Check the [mochawesome-merge documentation](#)

On [kitchensink](#) you can find an official cypress demo with all the commands being used.

21.4. Angular ESLint support



ESLint is supported in Angular 10.1.0 onwards.

21.4.1. What about TSLint?

TSLint is a fantastic tool. It is a linter that was written specifically to work based on the TypeScript AST format. This has advantages and disadvantages, as with most decisions we are faced with in software engineering!

One advantage is there is no tooling required to reconcile differences between ESLint and TypeScript AST formats, but the major disadvantage is that the tool is therefore unable to reuse any of the previous work which has been done in the JavaScript ecosystem around linting, and it has to reimplement everything from scratch. Everything from rules to auto-fixing capabilities and more.

However, the backers behind TSLint announced in 2019 that **they would be deprecating TSLint in favor of supporting `typescript-eslint`** in order to benefit the community. You can read more about that here: <https://medium.com/palantir/tslint-in-2019-1a144c2317a9>

The TypeScript Team themselves also announced their plans to move the TypeScript codebase from TSLint to `typescript-eslint`, and they have been big supporters of this project. More details at <https://github.com/microsoft/TypeScript/issues/30553>

Angular ESLint support comes from the `angular-eslint` tooling package. Angular documentation also links to this repository as you can check in the `ng lint` section of the Angular CLI documentation.

21.4.2. Quick start with Angular and ESLint

In order to create a brand new Angular CLI workspace which uses ESLint instead of TSLint and Codelyzer, simply run the following commands:

```
# Install the Angular CLI and @angular-eslint/schematics globally however you want
(e.g. npm, yarn, volta etc)

$ npm i -g @angular/cli @angular-devkit/core @angular-devkit/schematics @angular-
eslint/schematics

# Create a new Angular CLI workspace using the @angular-eslint/schematics collection
(instead of the default)

$ ng new --collection=@angular-eslint/schematics
```

21.4.3. Migrating an Angular CLI project from Codelyzer and TSLint

1 - Add relevant dependencies

The first step is to run the schematic to add `@angular-eslint` to your project:

```
$ ng add @angular-eslint/schematics
```

This will handle installing the latest version of all the relevant packages for you and adding them to the `devDependencies` of your `package.json`.

2 - Run the `convert-tslint-to-eslint` schematic on a project

The next thing to do is consider which "project" you want to migrate to use ESLint. If you have a single application in your workspace you will likely have just a single entry in the projects

configuration object within your `angular.json` file. If you have a `projects/`` directory in your workspace, you will have multiple entries in your `projects` configuration and you will need to chose which one you want to migrate using the `convert-tslint-to-eslint` schematic.

You can run it like so:

```
$ ng g @angular-eslint/schematics:convert-tslint-to-eslint  
{{YOUR_PROJECT_NAME_Goes_Here}}
```

From now on, `ng lint` will use ESLint!

3 - Remove root TSLint configuration and use only ESLint

Once you are happy with your ESLint setup, you simply need to remove the root-level `tslint.json` and potentially uninstall TSLint and any TSLint-related plugins/dependencies if your Angular CLI workspace is now no longer using TSLint at all.

More info at <https://github.com/angular-eslint/angular-eslint>

Part V: devon4net

22. Architecture basics

22.1. Introduction

The *devonfw platform* provides a solution to building applications which combine best-in-class frameworks and libraries as well as industry proven practices and code conventions. It massively speeds up development, reduces risks and helps you to deliver better results.

22.1.1. Overview Onion Design

This guide shows the overall proposed architecture in terms of separated layers making use the Onion architecture pattern. Each layers represents a logical group of components and functionality. In this guide you will learn the basics of the proposed architecture based in layers in order to develop software making use of the best practices.

22.1.2. Layer specification

It is important to understand the distinction between layers and tiers. *Layers* describe the logical groupings of the functionality and components in an application; whereas *tiers* describe the physical distribution of the functionality and components on separate servers, computers, networks, or remote locations. Although both layers and tiers use the same set of names (presentation, business, services, and data), remember that only tiers imply a physical separation. It is quite common to locate more than one layer on the same physical machine (the same tier). You can think of the term tier as referring to physical distribution patterns such as two-tier, three-tier, and n -tier.

— Layered Application Guidelines, MSDN Microsoft

The proposed architecture makes use of cooperating components called layers. Each layer contains a set of components capable to develop a specific functionality.

The next figure represents the different layers:

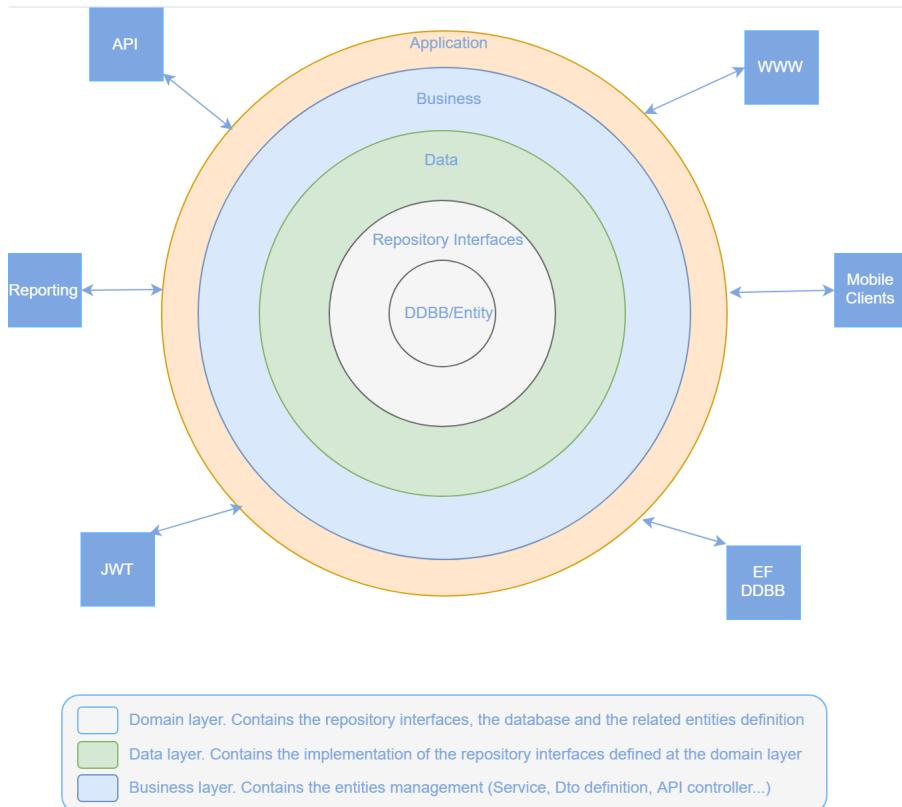


Figure 75. High level architecture representation

The layers are separated in physical tiers making use of interfaces. This pattern makes possible to be flexible in different kind of projects maximizing performance and deployment strategies (synchronous/asynchronous access, security, component deployment in different environments, microservices...). Another important point is to provide automated unit testing or test-driven development (TDD) facilities.

Application layer

The *Application Layer* encapsulates the different .Net projects and its resource dependencies and manages the user interaction depending on the project's nature.

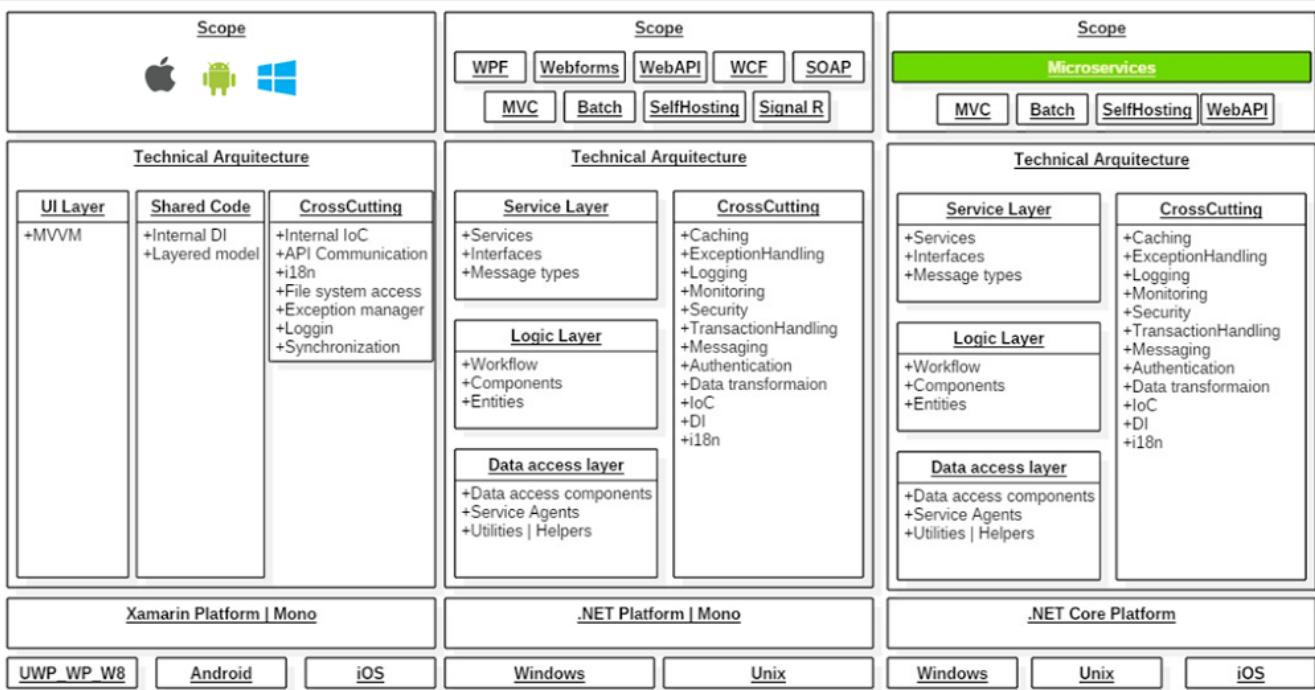


Figure 76. Net application stack

The provided application template implements an dotnet API application. Also integrates by default the Swagger client. This provides the possibility to share the contract with external applications (angular, mobile apps, external services...).

Business layer

The business layer implements the core functionality of the application and encapsulates the component's logic. This layer provides the interface between the data transformation and the application exposition. This allow the data to be optimized and ready for different data consumers.

This layer may implement for each main entity the API controller, the entity related service and other classes to support the application logic.

In order to implement the service logic, the services class must follow the next specification:

```
public class Service<TContext> : IService where TContext: DbContext
```

PE: devon4Net API template shows how to implement the TODOs service as follows:

```
public class TodoService: Service<TodoContext>, ITodoService
```

Where *Service* is the base service class to be inherited and have full access for the *Unit of work*, *TodoContext* is the TODOs database context and *ITodoService* is the interface of the service, which exposes the public extended methods to be implemented.

Data layer

The data layer orchestrates the data obtained between the *Domain Layer* and the *Business Layer*.

Also transforms the data to be used more efficiently between layers.

So, if a service needs the help of another service or repository, the implemented Dependency Injection is the solution to accomplish the task.

The main aim of this layer is to implement the repository for each entity. The repository's interface is defined in the Domain layer.

In order to implement the repository logic, the repository class must follow the next specification:

```
Repository<T> : IRepository<T> where T : class
```

PE: devon4Net API template shows how to implement the TODOs repository as follows:

```
public class TodoRepository : Repository<Todos>, ITodoRepository
```

Where *Repository* is the the base repository class to be inherited and have full access for the basic CRUD operations, *Todos* is the entity defined in the database context. *ITodoRepository* is the interface of the repository, which exposes the public extended methods to be implemented.



Please remember that <T> is the mapped class which reference the entity from the database context. This abstraction allows to write services implementation with different database contexts

Domain layer

The domain layer provides access to data directly exposed from other systems. The main source is used to be a data base system. The provided template makes use of *Entity Framework* solution from Microsoft in order to achieve this functionality.

To make a good use of this technology, *Repository Pattern* has been implemented with the help of *Unit Of Work* pattern. Also, the use of generic types are makes this solution to be the most flexible.

Regarding to data base source, each entity is mapped as a class. Repository pattern allows to use this mapped classes to access the data base via Entity framework:

```
public class UnitOfWork<TContext> : IUnitOfWork<TContext> where TContext : DbContext
```



Where <T> is the mapped class which reference the entity from the database.

The repository and unit of work patterns are create an abstraction layer between the data access layer and the business logic layer of an application.



Domain Layer has no dependencies with other layers. It contains the Entities, datasources and the Repository Interfaces.

devon4Net architecture layer implementation

The next picture shows how the devon4Net API template implements the architecture described in previous points:

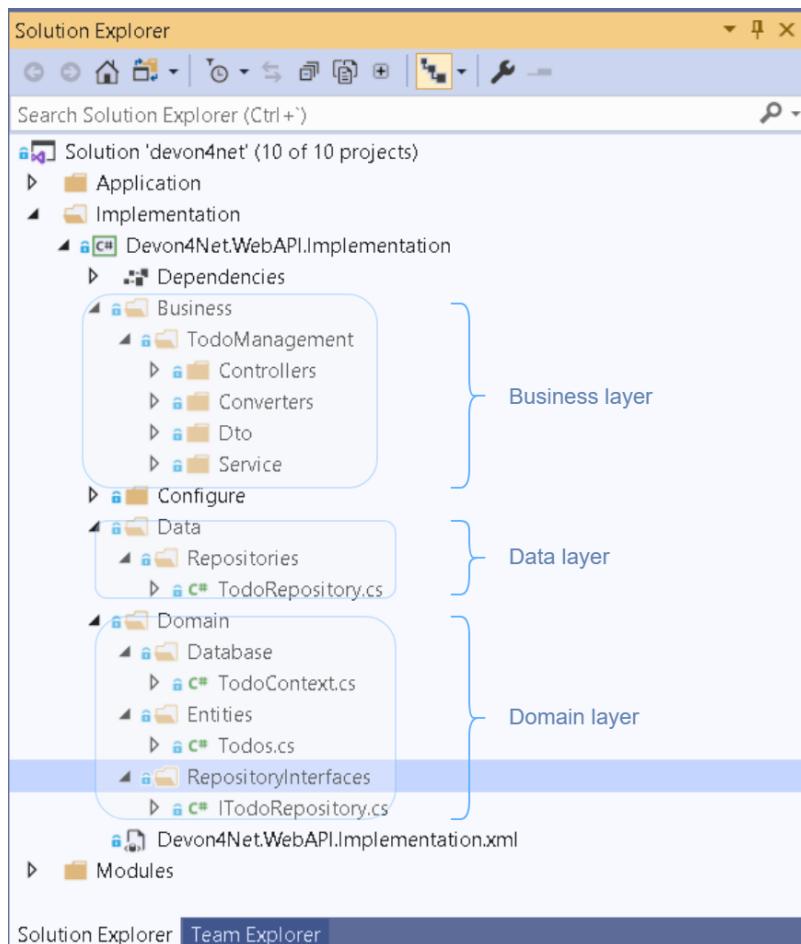


Figure 77. devon4Net architecture implementations

Cross-Cutting concerns

Cross-cutting provides the implementation functionality that spans layers. Each functionality is implemented through components able to work stand alone. This approach provides better reusability and maintainability.

A common component set of cross cutting components include different types of functionality regarding to authentication, authorization, security, caching, configuration, logging, and communication.

22.1.3. Communication between Layers: Interfaces

The main target of the use of interfaces is to loose coupling between layers and minimize dependencies.

Public interfaces allow to hide implementation details of the components within the layers making use of dependency inversion.

In order to make this possible, we make use of *Dependency Injection Pattern* (implementation of dependency inversion) given by default in .Net Core.

The provided *Data Layer* contains the abstract classes to inherit from. All new repository and service classes must inherit from them, also they must implement their own interfaces.

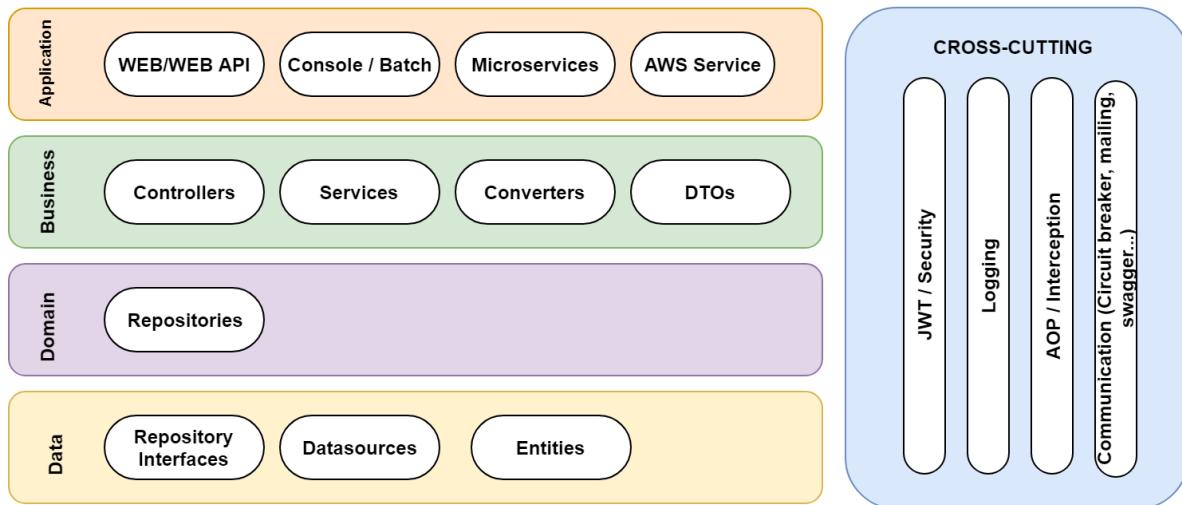


Figure 78. Architecture representation in deep

22.1.4. Templates

State of the art

The provided bundle contains the devon4Net API template based on .net core. The template allows to create a microservice solution with minimal configuration.

Also, the devon4Net framework can be added to third party templates such as the Amazon API template to use lambdas in serverless environments.

Included features:

- Logging:
- Text File
- Sqlite database support
- Serilog Seq Server support
- Graylog integration ready through TCP/UDP/HTTP protocols
- API Call params interception (simple and compose objects)
- API error exception management
- Swagger:
 - Swagger autogenerating client from comments and annotations on controller classes
 - Full swagger client customization (Version, Title, Description, Terms, License, Json end point definition)
 - Easy configuration with just one configuration node in your settings file
- JWT:
 - Issuer, audience, token expiration customization by external file configuration

- Token generation via certificate
- MVC inherited classes to access JWT user properties
- API method security access based on JWT Claims
- CORS:
 - Simple CORS definition ready
 - Multiple CORS domain origin definition with specific headers and verbs
- Headers:
 - Automatic header injection with middleware.
 - Supported header definitions: AccessControlExposeHeader, StrictTransportSecurityHeader, XFrameOptionsHeader, XSSProtectionHeader, XContentTypeOptionsHeader, ContentSecurityPolicyHeader, PermittedCrossDomainPoliciesHeader, ReferrerPolicyHeader
- Reporting server:
 - Partial implementation of reporting server based on My-FyiReporting (now runs on linux container)
- Testing:
 - Integration test template with sqlite support
 - Unit test template
 - Moq, xunit frameworks integrated
- Circuit breaker:
 - Integrated with HttpClient factory
 - Client Certificate customization
 - Number of retries customizables
- LiteDb:
 - Support for LiteDB
 - Provided basic repository for CRUD operations
- RabbitMq:
 - Use of EasyQNet library to perform CQRS main functions between different microservices
 - Send commands / Subscribe queues with one C# sentence
 - Events management: Handled received commands to subscribed messages
 - Automatic messaging backup when sent and handled (Internal database via LiteDB and database backup via Entity Framework)
- MediatR:
 - Use of MediatR library to perform CQRS main functions in memory
 - Send commands / Subscribe queues with one C# sentence
 - Events management: Handled received commands to subscribed messages
 - Automatic messaging backup when sent and handled (Internal database via LiteDB and

- database backup via Entity Framework)
- SmaxHcm:
 - Component to manage Microfocus SMAX for cloud infrastructure services management
 - CyberArk:
 - Manage safe credentials with CyberArk
 - AnsibleTower:
 - Ansible automates the cloud infrastructure. devon4net integrates with Ansible Tower via API consumption endpoints
 - gRPC+Protobuf:
 - Added Client + Server basic templates sample gRPC with Google's Protobuf protocol using devon4net
 - Kafka:
 - Added Apache Kafka support for deliver/consume messages and create/delete topics as well

Software stack

Table 43. Technology Stack of devon4Net

Topic	Detail	Implementation
runtime	language & VM	.Net Core Version 3.0
persistence	OR-mapper	Entity Framework Core
service	REST services	Web API
service - integration to external systems - optional	SOAP services	WCF
logging	framework	Serilog
validation	framework	NewtonSoft Json, DataAnnotations
component management	dependency injection	Unity
security	Authentication & Authorization	JWT .Net Security - Token based, local Authentication Provider
unit tests	framework	xUnit
Circuit breaker	framework, allows retry pattern on http calls	Polly
CQRS	Memory events and queue events	MediatR - EasyNetQ - Kafka
Kafka	Kafka support for enterprise applications	Confluent.Kafka
Fluent Validation	Fluent validation for class instances	Fluent validation

Target platforms

Thanks to the new .Net Core platform from Microsoft, the developed software can be published on Windows, Linux, OS X and Android platforms.

22.1.5. External links

[.Net Frameworks](#)

[Entity Framework documentation from Microsoft](#)

[Swagger API tooling](#)

[Dependency Injection in .NET Core](#)

[Json Web Token](#)

[Unit Testing \(xUnit\)](#)

[Runtime IDentifier for publishing](#)

23. User guide



23.1. devon4net Guide

23.1.1. Introduction

Welcome to devon4net framework user guide. In this document you will find the information regarding how to start and deploy your project using the guidelines proposed in our solution.

All the guidelines shown and used in this document are a set of rules and conventions proposed and supported by Microsoft and the industry.

23.1.2. The package

Devon4Net package solution contains:

File / Folder	Content
Documentation	User documentation in HTML format
Modules	Contains the source code of the different devon4net modules
Samples	Different samples implemented in .NET and .NET Core. Also includes My Thai Star Devon flagship restaurant application
Templates	Main .net Core template to start developing from scratch
License	License agreement
README.md	Github main page
TERMS_OF_USE.asciidoc	The devon4net terms of use
LICENSE	The devon license
Other files	Such the code of conduct and contributing guide

Application templates

The application templates given in the bundle are ready to use.

At the moment .net Core template is supported. The template is ready to be used as a simple console

Kestrel application or being deployed in a web server like IIS.

Samples

My Thai Star

You can find My Thai Star .NET port application at [Github](#).



As devon4net has been migrated to the latest version of .net core, the template is not finished yet.

23.1.3. Cookbook

Data management

To use EF Core, install the package for the database provider(s) you want to target. This walk-through uses SQL Server.

For a list of available providers see [Database Providers](#)

- Go to **Tools > NuGet Package Manager > Package Manager Console**
- Run **Install-Package Microsoft.EntityFrameworkCore.SqlServer**

We will be using some Entity Framework Tools to create a model from the database. So we will install the tools package as well:

- Run **Install-Package Microsoft.EntityFrameworkCore.Tools**

We will be using some ASP.NET Core Scaffolding tools to create controllers and views later on. So we will install this design package as well:

- Run **Install-Package Microsoft.VisualStudio.Web.CodeGeneration.Design**

EF Code first

In order to design your database model from scratch, we encourage to follow the Microsoft guidelines described [here](#).

EF Database first

- Go to **Tools > NuGet Package Manager > Package Manager Console**
- Run the following command to create a model from the existing database:

```
Scaffold-DbContext "Your connection string to existing database"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

The command will create the database context and the mapped entities as well inside of Models folder.

Register your context with dependency injection

Services are registered with dependency injection during application startup.

In order to register your database context (or multiple database context as well) you can add the following line at ConfigureDbService method at startup.cs:

```
private void SetupDatabase(IServiceCollection services)
{
    services.SetupDatabase<TodoContext>(Configuration, "Default",
WebAPI.Configuration.Enums.DatabaseType.InMemory);
}
```

Where:

Param	Description
TodoContext	Is the database context definition
Default	Is the connection string defined at <i>ConnectionString</i> node at the appsettings configuration file
WebAPI.Configuration.Enums.DatabaseType.InMemory	Is the database driver selection. In this case InMemory data base is chosen

The supported databases are:

- SqlServer
- Sqlite
- InMemory
- Cosmos
- PostgreSQL
- MySql
- MariaDb
- FireBird
- Oracle
- MSAccess

Repositories and Services

Services and *Repositories* are an important part of devon4net proposal. To make them work properly, first of all must be declared and injected at Startup.cs at *DI Region*.

Services are declared in devon4net.Business.Common and injected in Controller classes when needed. Use services to build your application logic.

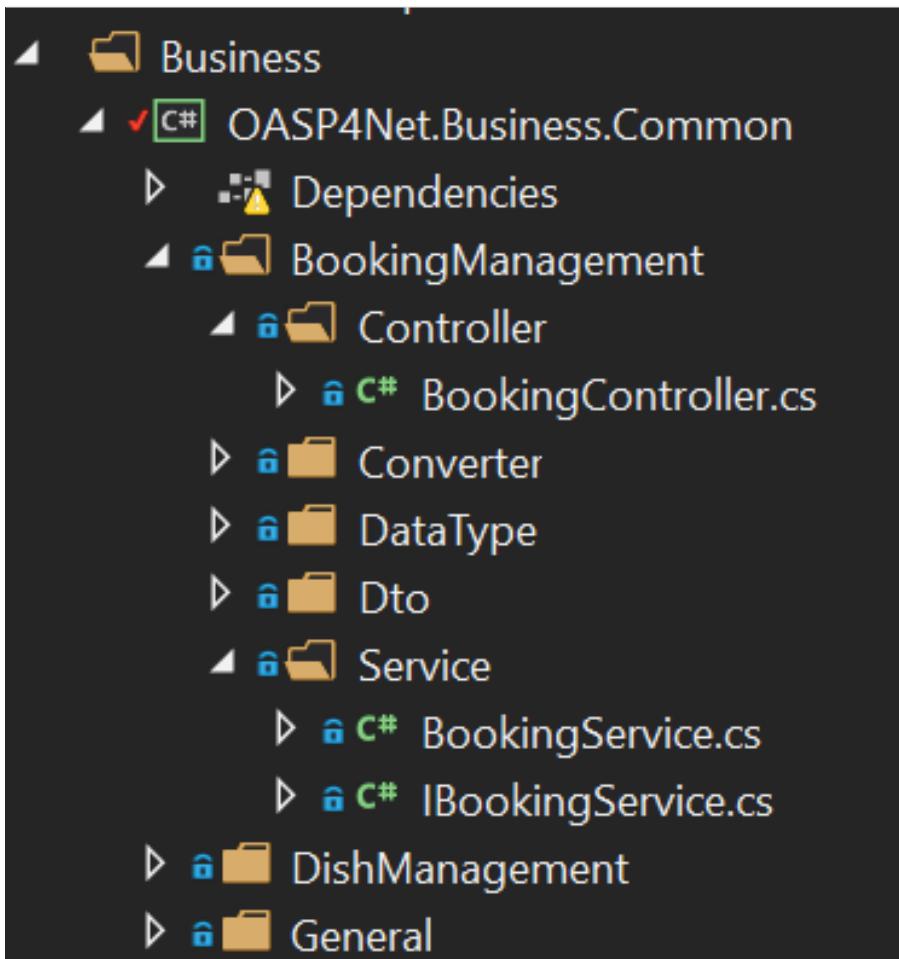


Figure 79. Screenshot of devon4net.Business.Common project in depth

For example, My Thai Star Booking controller constructor looks like this:

```
public BookingController(IBookingService bookingService, IMapper mapper)
{
    BookingService = bookingService;
    Mapper = mapper;

}
```

Currently devon4net has a *Unit of Work* class in order to perform CRUD operations to database making use of your designed model context.

Repositories are declared at *devon4net.Domain.UnitOfWork* project and make use of *Unit of Work* class.

The common methods to perform CRUD operations (where <T> is an entity from your model) are:

- Sync methods:

```
IList<T> GetAll(Expression<Func<T, bool>> predicate = null);
T Get(Expression<Func<T, bool>> predicate = null);
IList<T> GetAllInclude(IList<string> include, Expression<Func<T, bool>> predicate =
null);
T Create(T entity);
void Delete(T entity);
void DeleteById(object id);
void Delete(Expression<Func<T, bool>> where);
void Edit(T entity);
```

- Async methods:

```
Task<IList<T>> GetAllAsync(Expression<Func<T, bool>> predicate = null);
Task<T> GetAsync(Expression<Func<T, bool>> predicate = null);
Task<IList<T>> GetAllIncludeAsync(IList<string> include, Expression<Func<T, bool>>
predicate = null);
```

If you perform a Commit operation and an error happens, changes will be rolled back.

Swagger integration

The given templates allow you to specify the API contract through Swagger integration and the controller classes are the responsible of exposing methods making use of comments in the source code.

The next example shows how to comment the method with summaries in order to define the contract. Add (Triple Slash) XML Documentation To Swagger:

```
/// <summary>
/// Method to get reservations
/// </summary>
/// <response code="201">Ok.</response>
/// <response code="400">Bad request. Parser data error.</response>
/// <response code="401">Unauthorized. Authentication fail.</response>
/// <response code="403">Forbidden. Authorization error.</response>
/// <response code="500">Internal Server Error. The search process ended with
error.</response>
[HttpPost]
[Route("/mythaistar/services/rest/bookingmanagement/v1/booking/search")]
//[Authorize(Policy = "MTSWaiterPolicy")]
[AllowAnonymous]
[EnableCors("CorsPolicy")]
public async Task<IActionResult> BookingSearch([FromBody]BookingSearchDto
bookingSearchDto)
{
```

In order to be effective and make use of the comments to build the API contract, the project which

contains the controller classes must generate the XML document file. To achieve this, the XML documentation file must be checked in project settings tab:

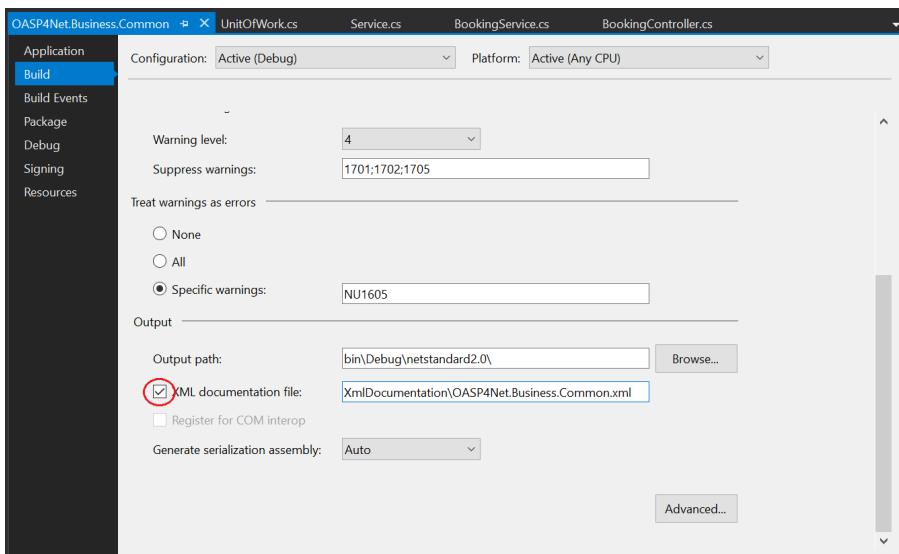


Figure 80. Project settings tab

We propose to generate the file under the XmlDocument folder. For example in devon4net.Domain.Entities project in My Thai Star .NET implementation the output folder is:

XmlDocumentation\devon4net.Business.Common.xml

The file *devon4net.Business.Common.xml* won't appear until you build the project. Once the file is generated, please modify its properties as a resource and set it to be *Copy always*.

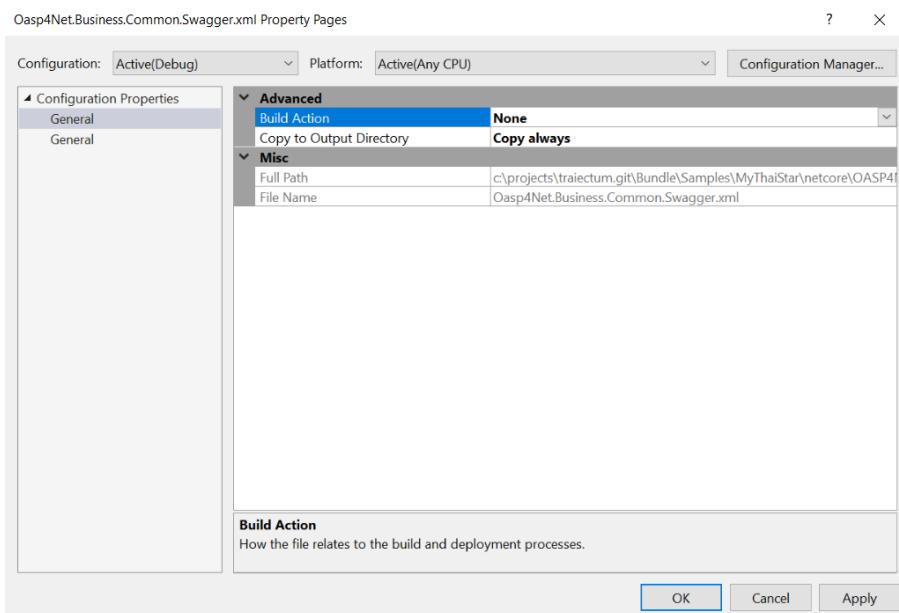


Figure 81. Swagger XML document file properties

Once you have this, the swagger user interface will show the method properties defined in your controller comments.

Making use of this technique controller are not encapsulated to the application project. Also, you can develop your controller classes in different projects obtain code reusability.

Swagger comment:

Comment	Functionality
<summary>	Will map to the operation's summary
<remarks>	Will map to the operation's description (shown as "Implementation Notes" in the UI)
<response code="####">	Specifies the different response of the target method
<param>	Will define the parameter(s) of the target method

Please check [Microsoft's site](#) regarding to summary notations.

Logging module

An important part of life software is the need of using log and traces. devon4net has a log module pre-configured to achieve this important point.

By default Microsoft provides a logging module on .NET Core applications. This module is open and can it can be extended. devon4net uses the [serilog](#) implementation. This implementation provides a huge quantity information about events and traces.

Log file

devon4net can write the log information to a simple text file. You can configure the file name and folder at appsettings.json file (LogFile attribute) at devon4net.Application.WebApi project.

Database log

devon4net can write the log information to a SQLite database. You can configure the file name and folder at appsettings.json file (LogDatabase attribute) at devon4net.Application.WebApi project.

With this method you can launch queries in order to search the information you are looking for.

Seq log

devon4net can write the log information to a serilog server. You can configure the serilog URL at appsettings.json file (SeqLogServerUrl attribute) at devon4net.Application.WebApi project.

With this method you can make queries via HTTP.

The screenshot shows the Seq log viewer interface. The left pane displays a list of log entries from February 5, 2018, at 15:41:42.199 to 15:40:36.320. The right pane contains navigation and configuration controls.

SIGNALS

- (*) None
- (*) @Level
 - Errors
 - Warnings
 - Exceptions

QUERIES

- Available Properties
- Count by Hour
- Latest as Table
- Space by Event Type

By default you can find the log information at *Logs* folder.

JWT module

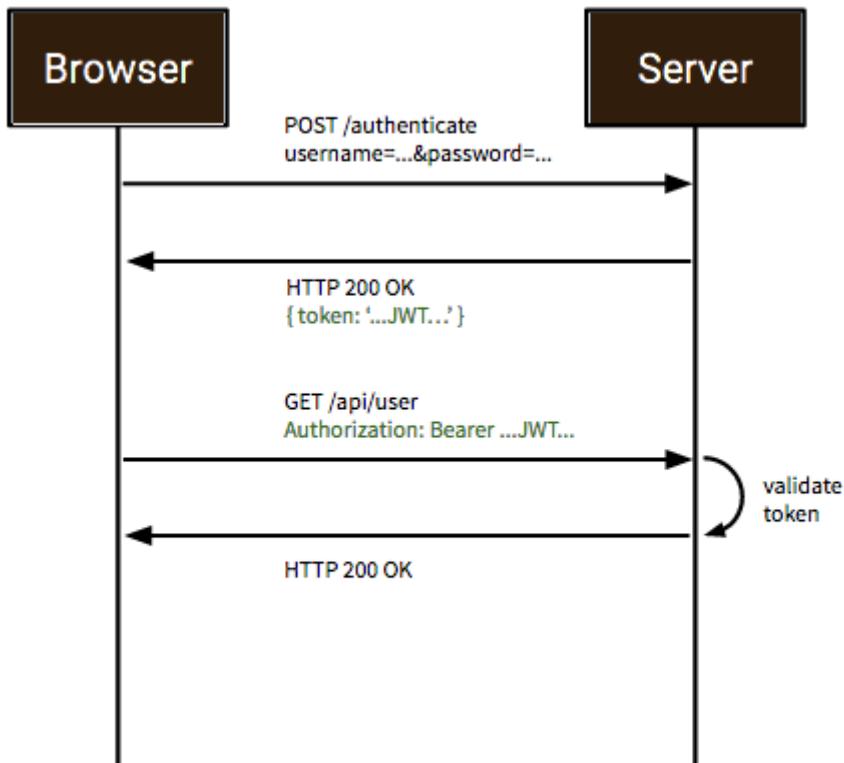
JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties allowing you to decode, verify and generate JWT.

You should use JWT for:

- Authentication : allowing the user to access routes, services, and resources that are permitted with that token.
- Information Exchange: JSON Web Tokens are a good way of securely transmitting information between parties. Additionally, as the signature is calculated using the header and the payload, you can also verify that the content.

The JWT module is configured at `Startup.cs` inside `devon4net.Application.WebApi` project from .NET Core template. In this class you can configure the different authentication policy and JWT properties.

Once the user has been authenticated, the client perform the call to the backend with the attribute `Bearer` plus the token generated at server side.



On My Thai Star sample there are two predefined users: user0 and Waiter. Once they log in the application, the client (Angular/Xamarin) will manage the server call with the json web token. With this method we can manage the server authentication and authorization.

You can find more information about JWT at jwt.io

AOP module

AOP (Aspect Oriented Programming) tracks all information when a method is called. AOP also tracks the input and output data when a method is called.

By default devon4net has AOP module pre-configured and activated for controllers at Startup.cs file at devon4net.Application.WebApi:

```

options.Filters.Add(new Infrastructure.AOP.AopControllerAttribute(Log.Logger));

options.Filters.Add(new Infrastructure.AOP.AopExceptionFilter(Log.Logger));
  
```

This configuration allows all Controller classes to be tracked. If you don't need to track the info comment the lines written before.

Docker support

devon4net Core projects are ready to be integrated with docker.

[My Thai Star application](#) sample is ready to be used with linux docker containers. The Readme file explains how to launch and setup the sample application.

- **angular** : Angular client to support backend. Just binaries.
- **database** : Database scripts and .bak file
- **mailservice**: Microservice implementation to send notifications.
- **netcore**: Server side using .net core 2.0.x.
- **xamarin**: Xamarin client based on Excalibur framework from The Netherlands using XForms.

Docker configuration and docker-compose files are provided.

23.1.4. Testing with XUnit

xUnit.net is a free, open source, community-focused unit testing tool for the .NET Framework. Written by the original inventor of NUnit v2, xUnit.net is the latest technology for unit testing C#, F#, VB.NET and other .NET languages. xUnit.net works with ReSharper, CodeRush, TestDriven.NET and Xamarin. It is part of the .NET Foundation, and operates under their code of conduct. It is licensed under Apache 2 (an OSI approved license).

— About xUnit.net, <https://xunit.github.io/#documentation>

Facts are tests which are always true. They test invariant conditions.

Theories are tests which are only true for a particular set of data.

The first test

```

using Xunit;

namespace MyFirstUnitTests
{
    public class Class1
    {
        [Fact]
        public void PassingTest()
        {
            Assert.Equal(4, Add(2, 2));
        }

        [Fact]
        public void FailingTest()
        {
            Assert.Equal(5, Add(2, 2));
        }

        int Add(int x, int y)
        {
            return x + y;
        }
    }
}

```

The first test with theory

Theory attribute is used to create tests with input params:

```

[Theory]
[InlineData(3)]
[InlineData(5)]
[InlineData(6)]
public void MyFirstTheory(int value)
{
    Assert.True(IsOdd(value));
}

bool IsOdd(int value)
{
    return value % 2 == 1;
}

```

Cheat Sheet

Operation	Example
Test	[Fact] public void Test() { }
Setup	public class TestFixture { public TestFixture() { ... } }
Teardown	public class TestFixture : IDisposable { public void Dispose() { ... } }

Console runner return codes

Code	Meaning
0	The tests ran successfully.
1	One or more of the tests failed.
2	The help page was shown, either because it was requested, or because the user did not provide any command line arguments.
3	There was a problem with one of the command line options passed to the runner.
4	There was a problem loading one or more of the test assemblies (for example, if a 64-bit only assembly is run with the 32-bit test runner).

23.1.5. Publishing

Nginx

In order to deploy your application to a Nginx server on Linux platform you can follow the instructions from [Microsoft here](#).

IIS

In this point is shown the configuration options that must implement the .Net Core application.

Supported operating systems:

- Windows 7 and newer
- Windows Server 2008 R2 and newer*

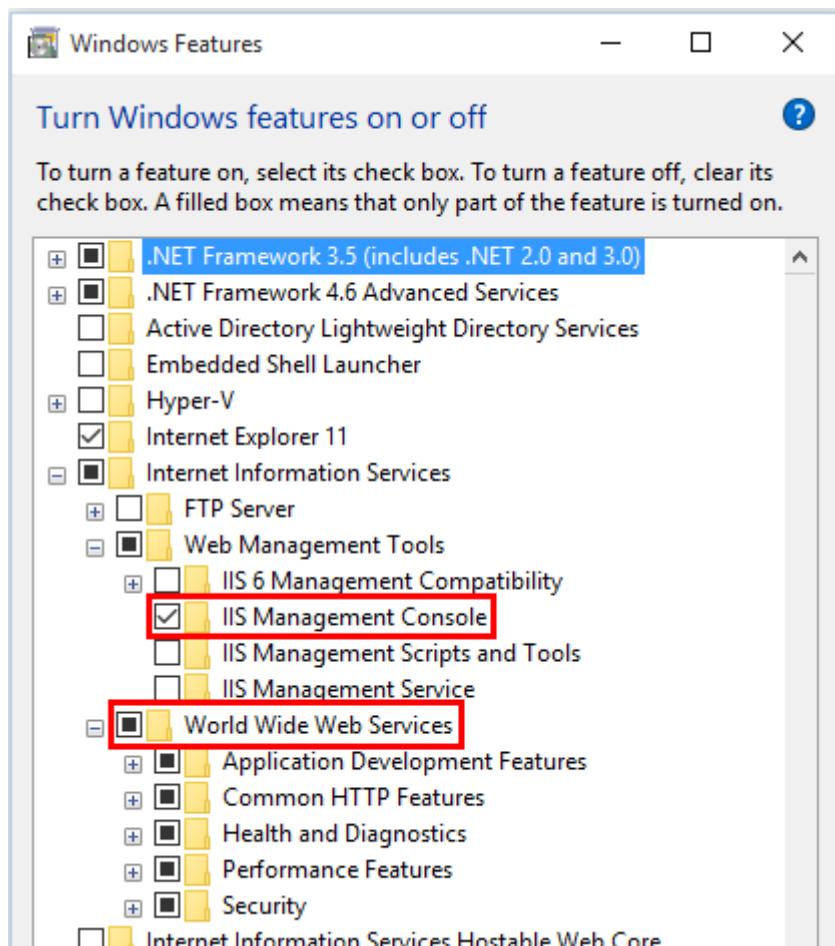
WebListener server will not work in a reverse-proxy configuration with IIS. You must use the [Kestrel server](#).

IIS configuration

Enable the Web Server (IIS) role and establish role services.

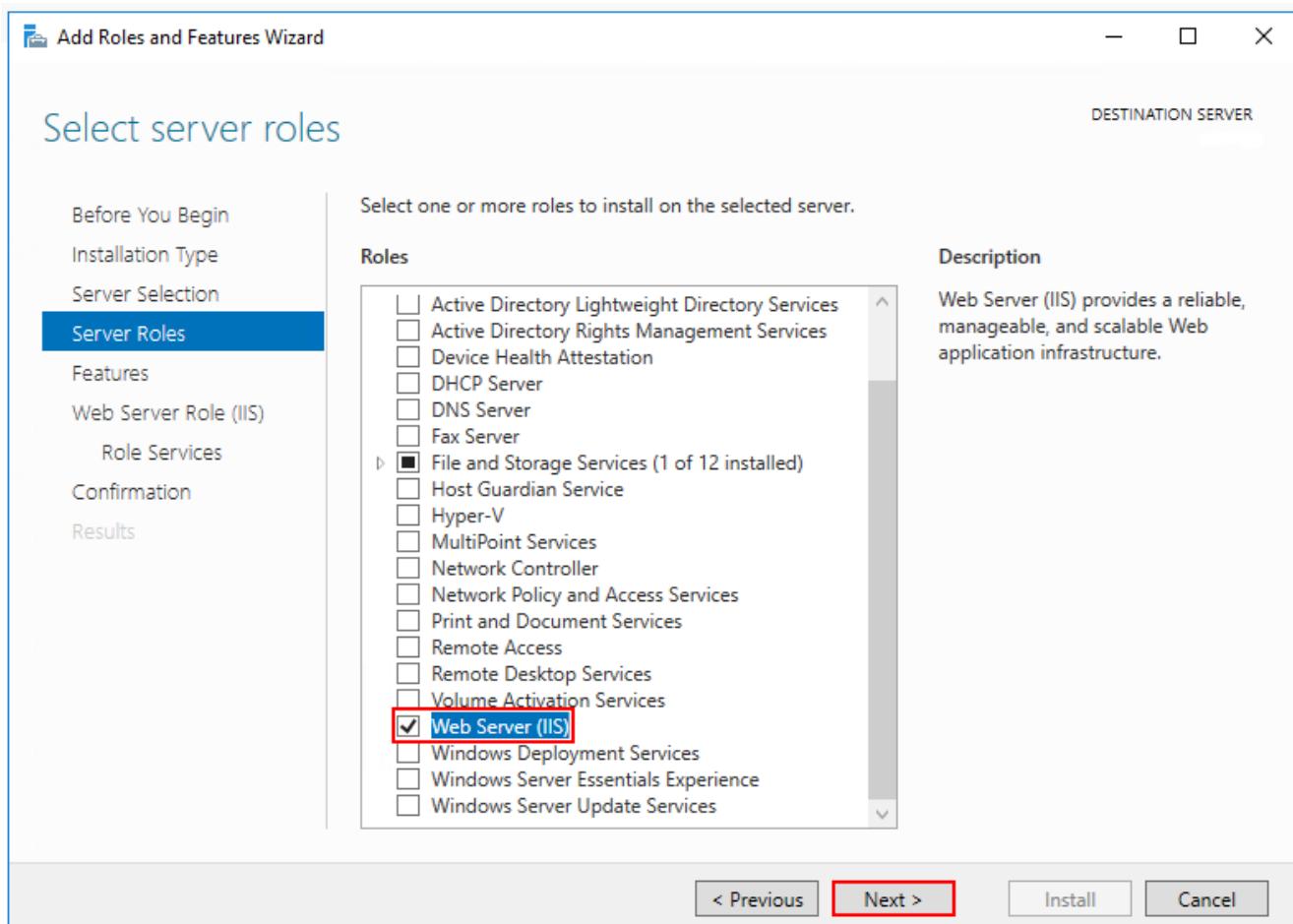
Windows desktop operating systems

Navigate to Control Panel > Programs > Programs and Features > Turn Windows features on or off (left side of the screen). Open the group for Internet Information Services and Web Management Tools. Check the box for IIS Management Console. Check the box for World Wide Web Services. Accept the default features for World Wide Web Services or customize the IIS features to suit your needs.

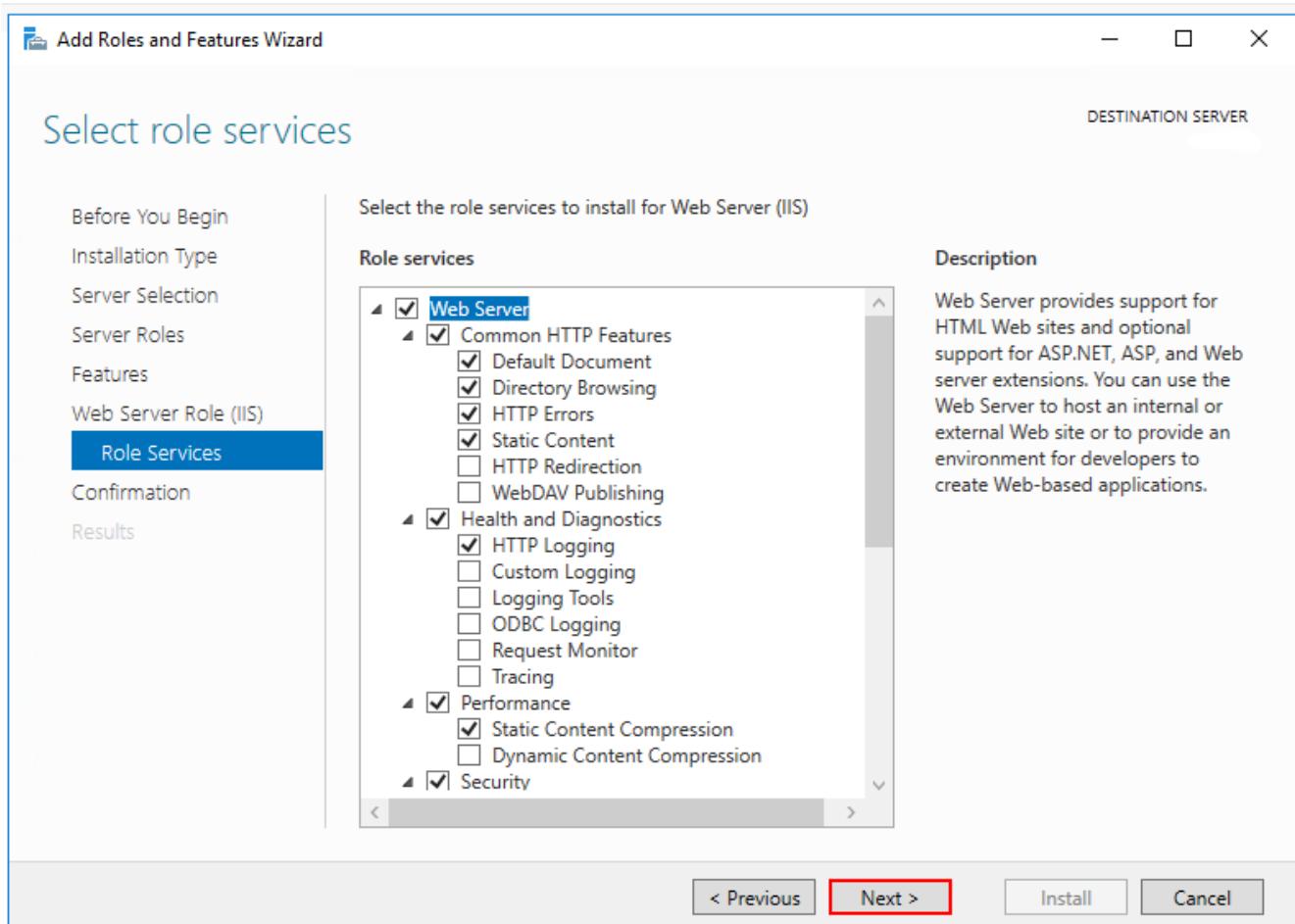


*Conceptually, the IIS configuration described in this document also applies to hosting ASP.NET Core applications on Nano Server IIS, but refer to [ASP.NET Core with IIS on Nano Server](#) for specific instructions.

Windows Server operating systems For server operating systems, use the Add Roles and Features wizard via the Manage menu or the link in Server Manager. On the Server Roles step, check the box for Web Server (IIS).



On the Role services step, select the IIS role services you desire or accept the default role services provided.



Proceed through the Confirmation step to install the web server role and services. A server/IIS restart is not required after installing the Web Server (IIS) role.

Install the .NET Core Windows Server Hosting bundle

1. Install the .NET Core Windows Server Hosting bundle on the hosting system. The bundle will install the .NET Core Runtime, .NET Core Library, and the ASP.NET Core Module. The module creates the reverse-proxy between IIS and the Kestrel server. Note: If the system doesn't have an Internet connection, obtain and install the Microsoft Visual C++ 2015 Redistributable before installing the .NET Core Windows Server Hosting bundle.
2. Restart the system or execute net stop was /y followed by net start w3svc from a command prompt to pick up a change to the system PATH.



If you use an IIS Shared Configuration, see ASP.NET Core Module with IIS Shared Configuration.

To configure IISIntegration service options, include a service configuration for IISOptions in ConfigureServices:

```
services.Configure<IISOptions>(options =>
{
    ...
});
```

Option	Default	Setting
AutomaticAuthentication	true	If true, the authentication middleware sets the <code>HttpContext.User</code> and responds to generic challenges. If false, the authentication middleware only provides an identity (<code>HttpContext.User</code>) and responds to challenges when explicitly requested by the <code>AuthenticationScheme</code> . Windows Authentication must be enabled in IIS for <code>AutomaticAuthentication</code> to function.
AuthenticationDisplayName	null	Sets the display name shown to users on login pages.
ForwardClientCertificate	true	If true and the <code>MS-ASPNETCORE-CLIENTCERT</code> request header is present, the <code>HttpContext.Connection.ClientCertificate</code> is populated.

web.config

The `web.config` file configures the ASP.NET Core Module and provides other IIS configuration. Creating, transforming, and publishing `web.config` is handled by `Microsoft.NET.Sdk.Web`, which is included when you set your project's SDK at the top of your `.csproj` file, `<Project Sdk="Microsoft.NET.Sdk.Web">`. To prevent the MSBuild target from transforming your `web.config` file, add the `<IsTransformWebConfigDisabled>` property to your project file with a setting of true:

```
<PropertyGroup>
  <IsTransformWebConfigDisabled>true</IsTransformWebConfigDisabled>
</PropertyGroup>
```

Azure

In order to deploy your application to Azure platform you can follow the instructions from [Microsoft](#):

Set up the development environment

- Install the latest [Azure SDK for Visual Studio](#). The SDK installs Visual Studio if you don't already have it.
- Verify your [Azure account](#). You can [open a free Azure account](#) or [Activate Visual Studio subscriber benefits](#).

Create a web app

In the Visual Studio Start Page, select **File > New > Project...**

[File menu] | ./offline/azure_files/file_new_project.png

Complete the **New Project** dialog:

- In the left pane, select **.NET Core**.
- In the center pane, select **ASP.NET Core Web Application**.
- Select **OK**.

[New Project dialog] | ./offline/azure_files/new_prj.png

In the **New ASP.NET Core Web Application** dialog:

- Select **Web Application**.
- Select **Change Authentication**.

[New Project dialog] | ./offline/azure_files/new_prj_2.png

The **Change Authentication** dialog appears.

- Select **Individual User Accounts**.
- Select **OK** to return to the **New ASP.NET Core Web Application**, then select **OK** again.

[New ASP.NET Core Web authentication dialog] | ./offline/azure_files/new_prj_auth.png

Visual Studio creates the solution.

Run the app locally

- Choose **Debug** then **Start Without Debugging** to run the app locally.
- Click the **About** and **Contact** links to verify the web application works.

[Web application open in Microsoft Edge on localhost] | ./offline/azure_files/show.png

- Select **Register** and register a new user. You can use a fictitious email address. When you submit, the page displays the following error:

"Internal Server Error: A database operation failed while processing the request. SQL exception: Cannot open the database. Applying existing migrations for Application DB context may resolve this issue."

- Select **Apply Migrations** and, once the page updates, refresh the page.

[Internal Server Error: A database operation failed while processing the request. SQL exception:

Cannot open the database. Applying existing migrations for Application DB context may resolve this

issue.] |/offline/azure_files/mig.png

The app displays the email used to register the new user and a **Log out** link.

[Web application open in Microsoft Edge. The Register link is replaced by the text Hello

email@domain.com!] | ./offline/azure_files/hello.png

Deploy the app to Azure

Close the web page, return to Visual Studio, and select **Stop Debugging** from the **Debug** menu.

Right-click on the project in Solution Explorer and select **Publish....**

[Contextual menu open with Publish link highlighted] | ./offline/azure_files/pub.png

In the **Publish** dialog, select **Microsoft Azure App Service** and click **Publish**.

[Publish dialog] | ./offline/azure_files/maas1.png

- Name the app a unique name.
- Select a subscription.
- Select **New...** for the resource group and enter a name for the new resource group.
- Select **New...** for the app service plan and select a location near you. You can keep the name that is generated by default.

[App Service dialog] | ./offline/azure_files/newrg1.png

- Select the **Services** tab to create a new database.
- Select the green + icon to create a new SQL Database

[New SQL Database] | ./offline/azure_files/sql.png

- Select **New...** on the **Configure SQL Database** dialog to create a new database.

[New SQL Database and server] | ./offline/azure_files/conf.png

The **Configure SQL Server** dialog appears.

- Enter an administrator user name and password, and then select **OK**. Don't forget the user name and password you create in this step. You can keep the default **Server Name**.
- Enter names for the database and connection string.

Note

"admin" is not allowed as the administrator user name.

[Configure SQL Server dialog] | ./offline/azure_files/conf_servername.png

- Select **OK**.

Visual Studio returns to the **Create App Service** dialog.

- Select **Create** on the **Create App Service** dialog.

[Configure SQL Database dialog] | ./azure_files/conf_final.png

- Click the **Settings** link in the **Publish** dialog.

[Publish dialog: Connection panel] | ./offline/azure_files/pubc.png

On the **Settings** page of the **Publish** dialog:

- Expand **Databases** and check **Use this connection string at runtime**.
- Expand **Entity Framework Migrations** and check **Apply this migration on publish**.
- Select **Save**. Visual Studio returns to the **Publish** dialog.

[Publish dialog: Settings panel] | ./offline/azure_files/pubs.png

Click **Publish**. Visual Studio will publish your app to Azure and launch the cloud app in your browser.

Test your app in Azure

- Test the **About** and **Contact** links
- Register a new user

[Web application opened in Microsoft Edge on Azure App Service] | ./offline/azure_files/register.png

Update the app

- Edit the *Pages/About.cshtml* Razor page and change its contents. For example, you can modify the paragraph to say "Hello ASP.NET Core!":

```
html<button class="action copy" data-bi-name="copy">Copy</button>
```

```
@page
@model AboutModel
 @{
    ViewData["Title"] = "About";
}
<h2>@ViewData["Title"]</h2>
<h3>@Model.Message</h3>

<p>Hello ASP.NET Core!</p>
```

- Right-click on the project and select **Publish...** again.

[Contextual menu open with Publish link highlighted] | ./offline/azure_files/pub.png

- After the app is published, verify the changes you made are available on Azure.

[Verify task is complete] | ./offline/azure_files/final.png

Clean up

When you have finished testing the app, go to the [Azure portal](#) and delete the app.

- Select **Resource groups**, then select the resource group you created.

[Azure Portal: Resource Groups in sidebar menu] | ./offline/azure_files/portalrg.png

- In the **Resource groups** page, select **Delete**.

[Azure Portal: Resource Groups page] | ./offline/azure_files/rgd.png

- Enter the name of the resource group and select **Delete**. Your app and all other resources created in this tutorial are now deleted from Azure.

23.1.6. External links

[Publishing .Net Core on IIS](#)

[IIS Shared configuration](#)

[Publishing to Nginx](#)

[Publishing to Docker](#)

[Connection strings](#)

[EF basics](#)

[Entity framework advanced design](#)

[Swagger annotations](#)

[Summary notation](#)

[JWT Official Site](#)

[Serilog](#)

24. How To section

24.1. Introduction

The aim of this document is to show how to get devon4net things done in a easy way.

24.2. How to

24.2.1. Start a new devonfw project

The *.Net Core 3.1* template allows you to start developing an n-layer server application to provide the latest features. The template can be used in Visual Studio Code and Visual Studio 2019.

The application result can be deployed as a console application, microservice or web page.

To start developing with devon4Net template, please follow this instructions:

Using devon4Net template

Option 1

```
Open your favourite terminal (Win/Linux/iOS)
Go to future project's path
Type dotnet new --install Devon4Net.WebAPI.Template
Type dotnet new Devon4NetAPI
Go to project's path
You are ready to start developing with devon4Net
```

Option 2

```
Create a new dotnet API project from scratch
Add the nuget package reference to your project:
Install-Package Devon4Net.Application.WebAPI.Configuration
```

Set up your project as follows in program.cs file:

```

public static void Main(string[] args)
{
    // Please use
    // Devonfw.Configure<Startup>(args);
    // Or :

    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .InitializeDevonFw()
        .Build()
        .Run();
}

```

Set up your project as follows in startup.cs file:

```

private IConfiguration Configuration { get; }

public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}

public void ConfigureServices(IServiceCollection services)
{

    services.ConfigureDevonFw(Configuration);
    SetupDatabase(services);

    ...
}

private void SetupDatabase(IServiceCollection services)
{
    // Default is the database connection name in appsettings.json file
    services.SetupDatabase<TodoContext>(Configuration, "Default",
    DatabaseType.InMemory);
}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.ConfigureDevonFw();
    ...
}

```

Add the devonfw configuration options in your appsettings.json file

devon4net configuration files

To start usinf devon4net in your .net core application add this configuration in your appsettings.json file:

```

"devonfw": {
    "UseDetailedErrorsKey": true,
    "UseIIS": false,
    "UseSwagger": true,
    "Environment": "Development",
    "KillSwitch": {
        "killSwitchSettingsFile": "killswitch.appsettings.json"
    },
    "Kestrel": {
        "UseHttps": true,
        "HttpProtocol": "Http2", //Http1, Http2, Http1AndHttp2, none
        "ApplicationPort": 8082,
        "KeepAliveTimeout": 120, //in seconds
        "MaxConcurrentConnections": 100,
        "MaxConcurrentUpgradedConnections": 100,
        "MaxRequestBodySize": 28.6, //In MB. The default maximum request body size is
        30,000,000 bytes, which is approximately 28.6 MB
        "Http2MaxStreamsPerConnection": 100,
        "Http2InitialConnectionWindowSize": 131072, // From 65,535 and less than 2^31 (2,147,483,648)
        "Http2InitialStreamWindowSize": 98304, // From 65,535 and less than 2^31 (2,147,483,648)
        "AllowSynchronousIO": true,
        "SslProtocol": "Tls12", //Tls, Tls11,Tls12, Tls13, Ssl2, Ssl3, none. For Https2
        Tls12 is needed
        "ServerCertificate": {
            "Certificate": "localhost.pfx",
            "CertificatePassword": "localhost"
        },
        "ClientCertificate": {
            "DisableClientCertificateCheck": true,
            "RequireClientCertificate": false,
            "CheckCertificateRevocation": true,
            "ClientCertificates": {
                "Whitelist": [
                    "3A87A49460E8FE0E2A198E63D408DC58435BC501"
                ],
                "DisableClientCertificateCheck": false
            }
        }
    },
    "IIS": {
        "ForwardClientCertificate": true,
        "AutomaticAuthentication": true,
        "AuthenticationDisplayName" : ""
    }
}

```

Also, for start using the devon4net components, you should add the next json options in your appsettings.json or appsettings.Development.json file:

```
{
  "ExtraSettingsFiles": [
    "Put a directory path (relative/absolute/linux-like) like /run/secrets/global
    where there are many settings/secret files to load",
    "Put a specific file name (with/without path) like /app-configs/app/extra-
    settings.json"
  ],
  "ConnectionStrings": {
    "Default": "Todos",
    "Employee": "Employee",
    "RabbitMqBackup": "Add your database connection string here for messaging backup",
    "MediatRBackup": "Add your database connection string here for messaging backup"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  },
  "Swagger": {
    "Version": "v1",
    "Title": "devon4net API",
    "Description": "devon4net API Contract",
    "Terms": "https://www.devonfw.com/terms-of-use/",
    "Contact": {
      "Name": "devonfw",
      "Email": "sample@mail.com",
      "Url": "https://www.devonfw.com"
    },
    "License": {
      "Name": "devonfw - Terms of Use",
      "Url": "https://www.devonfw.com/terms-of-use/"
    },
    "Endpoint": {
      "Name": "V1 Docs",
      "Url": "/swagger/v1/swagger.json",
      "UrlUi": "swagger",
      "RouteTemplate": "swagger/v1/{documentName}/swagger.json"
    }
  },
  "JWT": {
    "Audience": "devon4Net",
    "Issuer": "devon4Net",
    "TokenExpirationTime": 60,
    "ValidateIssuerSigningKey": true,
    "ValidateLifetime": true,
    "ClockSkew": 5,
    "Security": {
      "SecretKeyLengthAlgorithm": ""
    }
  }
}
```

```

    "SecretKeyEncryptionAlgorithm": "",
    "SecretKey": "",
    "Certificate": "",
    "CertificatePassword": "",
    "CertificateEncryptionAlgorithm": ""
  },
},
"Cors": [],
//[
//  {
//    "CorsPolicy": "CorsPolicy1",
//    "Origins": "http://example.com,http://www.contoso.com",
//    "Headers": "accept,content-type,origin,x-custom-header",
//    "Methods": "GET,POST,HEAD",
//    "AllowCredentials": true
//  },
//  {
//    "CorsPolicy": "CorsPolicy2",
//    "Origins": "http://example.com,http://www.contoso.com",
//    "Headers": "accept,content-type,origin,x-custom-header",
//    "Methods": "GET,POST,HEAD",
//    "AllowCredentials": true
//  }
//]
,
"CircuitBreaker": {
  "CheckCertificate": false,
  "Endpoints": [
    {
      "Name": "AnsibleTower",
      "BaseAddress": "PUT THE IP ADDRESS HERE",
      "Headers": {},
      "WaitAndRetrySeconds": [
        0.0001,
        0.0005,
        0.001
      ],
      "DurationOfBreak": 0.0005,
      "UseCertificate": false,
      "Certificate": "localhost.pfx",
      "CertificatePassword": "localhost",
      "SslProtocol": "3072" //TLS12
    },
    {
      "Name": "CyberArk",
      "BaseAddress": "PUT THE IP ADDRESS HERE",
      "Headers": {},
      "WaitAndRetrySeconds": [
        0.0001,
        0.0005,
        0.001
      ],
      "DurationOfBreak": 0.0005,
      "UseCertificate": false,
      "Certificate": "localhost.pfx",
      "CertificatePassword": "localhost",
      "SslProtocol": "3072" //TLS12
    }
  ]
}

```

```

        0.0005,
        0.001
    ],
    "DurationOfBreak": 0.0005,
    "UseCertificate": false,
    "Certificate": "localhost.pfx",
    "CertificatePassword": "localhost",
    "SslProtocol": "3072" //TLS12
},
{
    "Name": "SmaxHcm",
    "BaseAddress": "PUT THE IP ADDRESS HERE",
    "Headers": {
    },
    "WaitAndRetrySeconds": [
        0.0001,
        0.0005,
        0.001
    ],
    "DurationOfBreak": 0.0005,
    "UseCertificate": false,
    "Certificate": "localhost.pfx",
    "CertificatePassword": "localhost",
    "SslProtocol": "3072" //TLS12
}
]
},
"Headers": {
    "AccessControlExposeHeader": "Authorization",
    "StrictTransportSecurityHeader": "",
    "XFrameOptionsHeader": "DENY",
    "XssProtectionHeader": "1;mode=block",
    "XContentTypeOptionsHeader": "nosniff",
    "ContentSecurityPolicyHeader": "",
    "PermittedCrossDomainPoliciesHeader": "",
    "ReferrerPolicyHeader": ""
},
"Log": {
    "UseAOPTrace": false,
    "LogLevel": "Debug",
    "SqliteDatabase": "logs/log.db",
    "LogFile": "logs/{0}_devonfw.log",
    "SeqLogServerHost": "http://127.0.0.1:5341",
    "GrayLog": {
        "GrayLogHost": "127.0.0.1",
        "GrayLogPort": "12201",
        "GrayLogProtocol": "UDP",
        "UseSecureConnection": true,
        "UseAsyncLogging": true,
        "RetryCount": 5,
        "RetryIntervalMs": 15,
    }
}
]
}

```

```
        "MaxUdpMessageSize": 8192
    }
},
"RabbitMq": {
    "EnableRabbitMq": false,
    "Hosts": [
        {
            "Host": "127.0.0.1",
            "Port": 5672,
            "Ssl": false,
            "SslServerName": "localhost",
            "SslCertPath": "localhost.pfx",
            "SslCertPassPhrase": "localhost",
            "SslPolicyErrors": "RemoteCertificateNotAvailable" //None, RemoteCertificateNotAvailable, RemoteCertificateNameMismatch, RemoteCertificateChainErrors
        }
    ],
    "VirtualHost": "/",
    "UserName": "admin",
    "Password": "password",
    "Product": "devon4net",
    "RequestedHeartbeat": 10, //Set to zero for no heartbeat
    "PrefetchCount": 50,
    "PublisherConfirms": false,
    "PersistentMessages": true,
    "Platform": "localhost",
    "Timeout": 10,
    "Backup": {
        "UseLocalBackup": false,
        "DatabaseName": "devon4netMessageBackup.db"
    }
},
"MediatR": {
    "EnableMediatR": false,
    "Backup": {
        "UseLocalBackup": false,
        "DatabaseName": "devon4netMessageBackup.db"
    }
},
"LiteDb": {
    "DatabaseLocation": "devon4net.db"
},
"AnsibleTower": {
    "EnableAnsible": false,
    "Name": "AnsibleTower",
    "CircuitBreakerName": "AnsibleTower",
    "ApiUrlBase": "/api/v2/?format=json",
    "Version": "1.0.5.29",
    "Username": "",
    "Password": ""
}
```

```

},
"CyberArk": {
  "EnableCyberArk": false,
  "Username": "",
  "Password": "",
  "CircuitBreakerName": "CyberArk"
},
"SmaxHcm": {
  "EnableSmax": false,
  "Username": "",
  "Password": "",
  "TenantId": "",
  "CircuitBreakerName": "SmaxHcm",
  "ProviderId": ""
},
{
  "Kafka": {
    "EnableKafka": true,
    "Administration": [
      {
        "AdminId": "Admin1",
        "Servers": "127.0.0.1:9092"
      }
    ],
    "Producers": [
      {
        "ProducerId": "Producer1", // devon identifier
        "Servers": "127.0.0.1:9092", // Initial list of brokers as a CSV list of broker host or host:port. The application may also use `rd_kafka_brokers_add()` to add brokers during runtime
        "ClientId": "client1", //Client identifier
        "Topic": "devonfw", // topics to deliver the message
        "MessageMaxBytes": 1000000, //Maximum Kafka protocol request message size. Due to differing framing overhead between protocol versions the producer is unable to reliably enforce a strict max message limit at produce time and may exceed the maximum size by one message in protocol ProduceRequests, the broker will enforce the the topic's `max.message.bytes` limit (see Apache Kafka documentation)
        "CompressionLevel": -1, // [0-9] for gzip; [0-12] for lz4; only 0 for snappy; -1 = codec-dependent default compression level
        "CompressionType": "None", // None, Gzip, Snappy, Lz4, Zstd
        "ReceiveMessageMaxBytes": 10000000,
        "EnableSslCertificateVerification": false,
        "CancellationDelayMaxMs": 100, // The maximum length of time (in milliseconds) before a cancellation request is acted on. Low values may result in measurably higher CPU usage
        "Ack": "None", //Zero=Broker does not send any response/ack to client, One=The leader will write the record to its local log but will respond without awaiting full acknowledgement from all followers. All=Broker will block until message is committed by all in sync replicas (ISRs). If there are less than min.insync.replicas (broker configuration) in the ISR set the produce request will fail
        "Debug": "", //A comma-separated list of debug contexts to enable. Detailed Producer debugging: broker,topic,msg. Consumer: consumer,cgrp,topic,fetch
      }
    ]
  }
}

```

```

    "BrokerAddressTtl": 1000, //How long to cache the broker address resolving results (milliseconds)
    "BatchNumMessages": 1000000, // Maximum size (in bytes) of all messages batched in one MessageSet, including protocol framing overhead. This limit is applied after the first message has been added to the batch, regardless of the first message's size, this is to ensure that messages that exceed batch.size are produced. The total MessageSet size is also limited by batch.num.messages and message.max.bytes
    "EnableIdempotence": false, //When set to 'true', the producer will ensure that messages are successfully produced exactly once and in the original produce order. The following configuration properties are adjusted automatically (if not modified by the user) when idempotence is enabled: 'max.in.flight.requests.per.connection=5' (must be less than or equal to 5), 'retries=INT32_MAX' (must be greater than 0), 'acks=all', 'queuing.strategy=fifo'. Producer instantiation will fail if user-supplied configuration is incompatible
        "MaxInFlight": 5,
        "MessageSendMaxRetries": 5,
        "BatchSize": 10000000 // Maximum size (in bytes) of all messages batched in one MessageSet, including protocol framing overhead. This limit is applied after the first message has been added to the batch, regardless of the first message's size, this is to ensure that messages that exceed batch.size are produced. The total MessageSet size is also limited by batch.num.messages and message.max.bytes
    }
],
"Consumers": [
{
    "ConsumerId": "Consumer1", // devon identifier
    "Servers": "127.0.0.1:9092",
    "GroupId": "group1",
    "Topics": "devonfw", // Comma separated topics to subscribe
    "AutoCommit": true, //Automatically and periodically commit offsets in the background. Note: setting this to false does not prevent the consumer from fetching previously committed start offsets. To circumvent this behaviour set specific start offsets per partition in the call to assign()
    "StatisticsIntervalMs": 0, //librdkafka statistics emit interval. The application also needs to register a stats callback using 'rd_kafka_conf_set_stats_cb()'. The granularity is 1000ms. A value of 0 disables statistics
    "SessionTimeoutMs": 10000, //Client group session and failure detection timeout. The consumer sends periodic heartbeats (heartbeat.interval.ms) to indicate its liveness to the broker. If no hearts are received by the broker for a group member within the session timeout, the broker will remove the consumer from the group and trigger a rebalance. The allowed range is configured with the **broker** configuration properties 'group.min.session.timeout.ms' and 'group.max.session.timeout.ms'. Also see 'max.poll.interval.ms'
    "AutoOffsetReset": "Largest", //Action to take when there is no initial offset in offset store or the desired offset is out of range: 'smallest','earliest' - automatically reset the offset to the smallest offset, 'largest','latest' - automatically reset the offset to the largest offset, 'error' - trigger an error which is retrieved by consuming messages and checking 'message->err'
    "EnablePartitionEof": true, //Verify CRC32 of consumed messages, ensuring no on-the-wire or on-disk corruption to the messages occurred. This check comes at slightly increased CPU usage
}
]
}

```

```

    "IsolationLevel": "ReadCommitted", //Controls how to read messages written transactionally: 'ReadCommitted' - only return transactional messages which have been committed. 'ReadUncommitted' - return all messages, even transactional messages which have been aborted.
    "EnableSslCertificateVerification": false,
    "Debug": "" //A comma-separated list of debug contexts to enable. Detailed Producer debugging: broker,topic,msg. Consumer: consumer,cgrp,topic,fetch
  }
]
}
}

```

24.2.2. devon4net Cobigen Guide

Overview

In this guide we will explain how to generate a new WebApi project from an OpenAPI 3.0.0 specification. This means that we are going to use a “contract first” strategy. This is going to be possible due to these type of files that contain all the information about entities, operations, etc...

In order to make it work we are using [CobiGen](#), a powerful tool for generating source code. CobiGen allows users to generate all the structure and code of the components, helping to save a lot of time otherwise wasted on repetitive tasks.

Getting things ready

devonfw Distribution

The devonfw distributions can be obtained from the [TeamForge releases library](#) and are packaged in zips files that include all the needed tools, software and configurations.

It is not necessary to install nor configure anything. Just extracting the zip content is enough to have a fully functional devonfw. The only thing you have to do is run **create-or-update-workspace.bat** and then **update-all-workspaces.bat** to set up all the needed tools.

devon4net Templates

We are going to use the template of devon4net as a base to generate all the code, so what we have to do now is to download said template using the following steps.

First of all you have to set up all the environment for .NET, you can do this using [the following tutorial](#). Next we are going to create a new folder where we want to have the WebAPI project, lastly we are going to open the terminal there.

Type the following:

```
dotnet new -i Devon4Net.WebAPI.Template
```

and then:

```
dotnet new Devon4NetAPI
```

OpenAPI File

In order to let CobiGen generate all the files, we first have to make some modifications to our OpenAPI file.

It is obligatory to put the “*x-rootpackage*” tag to indicate where CobiGen will place the generated files as well as the “*x-component*” tags for each component, keep in mind that due to CobiGen’s limitations each component **must** have its own entity.

You can read more information about how to configure your OpenAPI file and a working example [here](#).

Generating files

Cobigen allow us to generate the files in two different ways. One of them is using Eclipse which it can be done by using the its grafical interface. The other way to generate the code is using the Cobigen CLI tool.

Generating files through Eclipse

In order to generate the files using Eclipse we need to follow some simple steps.

First we are going to import our basic devon4net WebAPI Project into Eclipse. to do so open Eclipse with the “eclipse-main.bat” file that can be found in the devon distribution root folder. Once we are inside of Eclipse we go to **File > Open projects from file system...** and, under "Directory", search for your project.

Next we copy our OpenAPI file into the root folder of the project.

And then we right click on OpenAPI file and then select **CobiGen > Generate...** It will display a window like this:

To select all .NET features choose **CRUD devon4net Server** otherwise you can select only those that interest you.

Ones you select all the files that you want to generate, click on the “*Finish*” button to generate all the source code.

Generating files through Cobigen CLI

In ordet to generate the files using the Cobigen CLI it is needed to do the following steps:

1. Go to devonfw distribution folder
2. Run **console.bat**, this will open a console.

3. Go to the folder you downloaded the **devon4net template** and your **yml** file.

4. Run the command:

```
cobigen generate {yourOpenAPIFile}.yml
```

5. A list of increments will be printed so that you can start the generation. It has to be selected **CRUD devon4net Server** increment.

Configuration

Dependency Injection configuration

At this point it is needed to make some modifications in the code in order to configure correctly the server. To do so it is needed to locate the services and the repositories files that were created in **Devon4Net.WebAPI.Implementation**

Services location:

Repositories location:

```
<a href="="images.Repositories.png">[cobigen]</a> | <em>images.Repositories.png</em>
```

Now, we are going to open the following file **Devon4Net.WebAPI.Implementation\Configure\DevonConfiguration.cs**. In there we have to add the Dependency Inyection for the services and the repositories that Cobigen has generated. The following image is an example of what is needed to add.

Moreover it is needed to remove the last line in order to be able to run the application:

```
throw new NotImplementedException(...);
```

Configure data base

Cobigen is generating an empty context that has to be filled with manaully in order to be able to work with the database. The context can be found in **[Project_Name]\Devon4Net.WebAPI.Implementation\Domain\Database\CobigenContext.cs**.

Configure services

In order to finish the configuration of the services it is needed to go to each service file of the managements generated.

In there we will see some "NotImplementedExceptions", so it is needed to read carefuely each coment inside of each exception in order to be able to use the service. It can be shown an example of the service with its NotImplementedExceptions comments:

Run the application

After doing all the steps defined above, open a terminal in path: **[Project_Name]/Devon4Net.Application.WebAPI** and then type:

```
dotnet run
```

This will deploy our application in our localhost with the port 8081, so when you click [here](#) (<https://localhost:8082/swagger>) you can see, in swagger, all the services and the data model.

24.2.3. Use HTTP2 protocol

You can specify the HTTP protocol to be used on your devon4net application modifying some node values at *devonfw* node in your appsettings configuration file.

HttpProtocol

The supported protocols are:

Protocol	Description
Http1	Http1 protocol
Http2	Http2 Protocol
Http1AndHttp2	Both supported

SSL

To activate the HTTP2, the *SslProtocol* node must be set to *Tls12* value.

The SSL protocol supported version values are:

- Tls
- Tls11
- Tls12
- Tls13
- Ssl2
- Ssl3

24.2.4. Create a certificate for development purposes

In order to create a valid certificate for development purposes the Open SSL tools are needed.

Certificate authority (CA)

Run the next commands in a shell:

```

1. openssl req -x509 -nodes -new -sha256 -days 1024 -newkey rsa:2048 -keyout
RootCA.key -out RootCA.pem -subj
"/C=ES/ST=Valencia/L=Valencia/O=Certificates/CN=localhost.local"

2. openssl x509 -outform pem -in RootCA.pem -out RootCA.crt

```

If you want to convert your certificate run the command:

```
openssl pkcs12 -export -out localhost.pfx -inkey RootCA.key -in RootCA.crt
```

Domain name certificate

Run the next commands in a shell:

```

1. openssl req -new -nodes -newkey rsa:2048 -keyout localhost.key -out localhost.csr
-subj "/C=ES/ST=Valencia/L=Valencia/O=Certificates/CN=localhost.local"

2. openssl x509 -req -sha256 -days 1024 -in localhost.csr -CA RootCA.pem -CAkey
RootCA.key -CAcreateserial -extfile domains.ext -out localhost.crt

```

Where the *domains.ext* file should contain:

```

authorityKeyIdentifier=keyid,issuer
basicConstraints=CA:FALSE
keyUsage = digitalSignature, nonRepudiation, keyEncipherment, dataEncipherment
subjectAltName = @alt_names
[alt_names]
DNS.1 = localhost
DNS.2 = localhost.local
DNS.3 = 127.0.0.1
DNS.4 = fake1.local
DNS.5 = fake2.local

```

If you want to convert your certificate run the command:

```
openssl pkcs12 -export -out localhost.pfx -inkey localhost.key -in localhost.crt
```

24.2.5. Setup the database driver

Add the database connection on the `SetupDatabase` method at `Startup.cs`

```
private void SetupDatabase(IServiceCollection services)
{
    services.SetupDatabase<TodoContext>(Configuration, "Default",
WebAPI.Configuration.Enums.DatabaseType.InMemory);
}
```

Where:

Param	Description
TodoContext	Is the database context definition
Default	Is the connection string defined at <i>ConnectionString</i> node at the appsettings configuration file
WebAPI.Configuration.Enums.DatabaseType.InMemory	Is the database driver selection. In this case InMemory data base is chosen

The supported databases are:

- SqlServer
- Sqlite
- InMemory
- Cosmos
- PostgreSQL
- MySql
- MariaDb
- FireBird
- Oracle
- MSAccess

24.2.6. Change the JWT encryption algorithm

In the appsettings.json configuration file, you can use the next values on the *SecretKeyLengthAlgorithm* and *SecretKeyEncryptionAlgorithm* nodes at JWT configuration:

Algorithm	Description
Aes128Encryption	" http://www.w3.org/2001/04/xmlenc#aes128-cbc "
Aes192Encryption	" http://www.w3.org/2001/04/xmlenc#aes192-cbc "
Aes256Encryption	" http://www.w3.org/2001/04/xmlenc#aes256-cbc "
DesEncryption	" http://www.w3.org/2001/04/xmlenc#des-cbc "
Aes128KeyWrap	" http://www.w3.org/2001/04/xmlenc#kw-aes128 "

Algorithm	Description
Aes192KeyWrap	"http://www.w3.org/2001/04/xmlenc#kw-aes192"
Aes256KeyWrap	"http://www.w3.org/2001/04/xmlenc#kw-aes256"
RsaV15KeyWrap	"http://www.w3.org/2001/04/xmlenc#rsa-1_5"
Ripemd160Digest	"http://www.w3.org/2001/04/xmlenc#ripemd160"
RsaOaepKeyWrap	"http://www.w3.org/2001/04/xmlenc#rsa-oaep"
Aes128KW	"A128KW"
Aes256KW	"A256KW"
RsaPKCS1	"RSA1_5"
RsaOAEP	"RSA-OAEP"
ExclusiveC14n	"http://www.w3.org/2001/10/xml-exc-c14n#"
ExclusiveC14nWithComments	"http://www.w3.org/2001/10/xml-exc-c14n#WithComments"
EnvelopedSignature	"http://www.w3.org/2000/09/xmldsig#enveloped-signature"
Sha256Digest	"http://www.w3.org/2001/04/xmlenc#sha256"
Sha384Digest	"http://www.w3.org/2001/04/xmldsig-more#sha384"
Sha512Digest	"http://www.w3.org/2001/04/xmlenc#sha512"
Sha256	"SHA256"
Sha384	"SHA384"
Sha512	"SHA512"
EcdsaSha256Signature	"http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256"
EcdsaSha384Signature	"http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384"
EcdsaSha512Signature	"http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512"
HmacSha256Signature	"http://www.w3.org/2001/04/xmldsig-more#hmac-sha256"
HmacSha384Signature	"http://www.w3.org/2001/04/xmldsig-more#hmac-sha384"
HmacSha512Signature	"http://www.w3.org/2001/04/xmldsig-more#hmac-sha512"
RsaSha256Signature	"http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"

Algorithm	Description
RsaSha384Signature	"http://www.w3.org/2001/04/xmldsig-more#rsa-sha384"
RsaSha512Signature	"http://www.w3.org/2001/04/xmldsig-more#rsa-sha512"
RsaSsaPssSha256Signature	"http://www.w3.org/2007/05/xmldsig-more#sha256-rsa-MGF1"
RsaSsaPssSha384Signature	"http://www.w3.org/2007/05/xmldsig-more#sha384-rsa-MGF1"
RsaSsaPssSha512Signature	"http://www.w3.org/2007/05/xmldsig-more#sha512-rsa-MGF1"
EcdsaSha256	"ES256"
EcdsaSha384	"ES384"
EcdsaSha512	"ES512"
HmacSha256	"HS256"
HmacSha384	"HS384"
HmacSha512	"HS512"
None	"none"
RsaSha256	"RS256"
RsaSha384	"RS384"
RsaSha512	"RS512"
RsaSsaPssSha256	"PS256"
RsaSsaPssSha384	"PS384"
RsaSsaPssSha512	"PS512"
Aes128CbcHmacSha256	"A128CBC-HS256"
Aes192CbcHmacSha384	"A192CBC-HS384"
Aes256CbcHmacSha512	"A256CBC-HS512"

25. Cobigen guide

25.1. devon4net Cobigen Guide

Overview

In this guide we will explain how to generate a new WebApi project from an OpenAPI 3.0.0 specification. This means that we are going to use a “contract first” strategy. This is going to be possible due to these type of files that contain all the information about entities, operations, etc...

In order to make it work we are using [CobiGen](#), a powerful tool for generating source code. CobiGen allows users to generate all the structure and code of the components, helping to save a lot of time otherwise wasted on repetitive tasks.

Getting things ready

devonfw Distribution

The devonfw distributions can be obtained from the [TeamForge releases library](#) and are packaged in zips files that include all the needed tools, software and configurations.

It is not necessary to install nor configure anything. Just extracting the zip content is enough to have a fully functional devonfw. The only thing you have to do is run **create-or-update-workspace.bat** and then **update-all-workspaces.bat** to set up all the needed tools.

devon4net Templates

We are going to use the template of devon4net as a base to generate all the code, so what we have to do now is to download said template using the following steps.

First of all you have to set up all the environment for .NET, you can do this using [the following tutorial](#). Next we are going to create a new folder where we want to have the WebAPI project, lastly we are going to open the terminal there.

Type the following:

```
dotnet new -i Devon4Net.WebAPI.Template
```

and then:

```
dotnet new Devon4NetAPI
```

OpenAPI File

In order to let CobiGen generate all the files, we first have to make some modifications to our OpenAPI file.

It is obligatory to put the “*x-rootpackage*” tag to indicate where CobiGen will place the generated

files as well as the "*x-component*" tags for each component, keep in mind that due to CobiGen's limitations each component **must** have its own entity.

You can read more information about how to configure your OpenAPI file and a working example [here](#).

Generating files

Cobigen allow us to generate the files in two different ways. One of them is using Eclipse which it can be done by using the its grafical interface. The other way to generate the code is using the Cobigen CLI tool.

Generating files through Eclipse

In order to generate the files using Eclipse we need to follow some simple steps.

First we are going to import our basic devon4net WebAPI Project into Eclipse. to do so open Eclipse with the "eclipse-main.bat" file that can be found in the devon distribution root folder. Once we are inside of Eclipse we go to **File > Open projects from file system...** and, under "Directory", search for your project.

Next we copy our OpenAPI file into the root folder of the project.

And then we right click on OpenAPI file and then select **CobiGen > Generate...** It will display a window like this:

To select all .NET features choose **CRUD devon4net Server** otherwise you can select only those that interest you.

Once you select all the files that you want to generate, click on the "*Finish*" button to generate all the source code.

Generating files through Cobigen CLI

In order to generate the files using the Cobigen CLI it is needed to do the following steps:

1. Go to devonfw distribution folder
2. Run **console.bat**, this will open a console.
3. Go to the folder you downloaded the **devon4net template** and your **yml** file.
4. Run the command:

```
cobigen generate {yourOpenAPIfile}.yml
```

5. A list of increments will be printed so that you can start the generation. It has to be selected **CRUD devon4net Server** increment.

Configuration

Dependency Injection configuration

At this point it is needed to make some modifications in the code in order to configure correctly the server. To do so it is needed to locate the services and the repositories files that were created in **Devon4Net.WebAPI.Implementation**

Services location:

Repositories location:

```
<a href="="images/Repositories.png">[cobigen]</a> | <em>images/Repositories.png</em>
```

Now, we are going to open the following file **Devon4Net.WebAPI.Implementation\Configure\DevonConfiguration.cs**. In there we have to add the Dependency Injection for the services and the repositories that Cobigen has generated. The following image is an example of what is needed to add.

Moreover it is needed to remove the last line in order to be able to run the application:

```
throw new NotImplementedException(...);
```

Configure data base

Cobigen is generating an empty context that has to be filled with manually in order to be able to work with the database. The context can be found in **[Project_Name]\Devon4Net.WebAPI.Implementation\Domain\Database/CobigenContext.cs**.

Configure services

In order to finish the configuration of the services it is needed to go to each service file of the managements generated.

In there we will see some "NotImplementedExceptions", so it is needed to read carefully each coment inside of each exception in order to be able to use the service. It can be shown an example of the service with its NotImplementedExceptions comments:

Run the application

After doing all the steps defined above, open a terminal in path: **[Project_Name]\Devon4Net.Application.WebAPI** and then type:

```
dotnet run
```

This will deploy our application in our localhost with the port 8081, so when you click [here](#)

(<https://localhost:8082/swagger>) you can see, in swagger, all the services and the data model.

26. Coding conventions

26.1. Code conventions

26.1.1. Introduction

This document covers .NET Coding Standards and is recommended to be read by team leaders/sw architects and developing teams operating in the Microsoft .NET environment.

“All the code in the system looks as if it was written by a single – very competent – individual” (K. Beck)

26.1.2. Capitalization Conventions

Terminology

Camel Case (camelCase)

Each word or abbreviation in the middle of the phrase begins with a capital letter, with no intervening spaces or punctuation.

The camelCasing convention, used only for parameter names, capitalizes the first character of each word except the first word, as shown in the following examples. As the example also shows, two-letter acronyms that begin a camel-cased identifier are both lowercase.

[= fa thumbs o up] use camelCasing for parameter names.

Pascal Case (PascalCase)

The first letter of each concatenated word is capitalized. No other characters are used to separate the words, like hyphens or underscores.

The PascalCasing convention, used for all identifiers except parameter names, capitalizes the first character of each word (including acronyms over two letters in length).

[= fa thumbs o up] use PascalCasing for all public member, type, and namespace names consisting of multiple words.

Underscore Prefix (_ underScore)

For underscore (_), the word after _ use camelCase terminology.

26.1.3. General Naming Conventions

[= fa thumbs o up] choose easily readable identifier names.

[= fa thumbs o up] favor readability over brevity.

- e.g.: GetLength is a better name than GetInt.
- Aim for the “ubiquitous language” (E. Evans): A language distilled from the domain language, which helps the team clarifying domain concepts and communicating with domain experts.

[= fa thumbs o up] prefer adding a suffix rather than a prefix to indicate a new version of an existing API.

[= fa thumbs o up] use a numeric suffix to indicate a new version of an existing API, particularly if the existing name of the API is the only name that makes sense (i.e., if it is an industry standard) and if adding any meaningful suffix (or changing the name) is not an appropriate option.

[= fa thumbs o down] do not use underscores, hyphens, or any other non-alphanumeric characters.

[= fa thumbs o down] do not use Hungarian notation.

[= fa thumbs o down] avoid using identifiers that conflict with keywords of widely used programming languages.

[= fa thumbs o down] do not use abbreviations or contractions as part of identifier names.

[= fa thumbs o down] do not use any acronyms that are not widely accepted, and even if they are, only when necessary.

[= fa thumbs o down] do not use the "Ex" (or a similar) suffix for an identifier to distinguish it from an earlier version of the same API.

[= fa thumbs o down] do not use C# reserved words as names.

[= fa thumbs o down] do not use Hungarian notation. Hungarian notation is the practice of including a prefix in identifiers to encode some metadata about the parameter, such as the data type of the identifier.

- e.g.: iNumberOfClients, sClientName

26.1.4. Names of Assemblies and DLLs

An assembly is the unit of deployment and identity for managed code programs. Although assemblies can span one or more files, typically an assembly maps one-to-one with a DLL. Therefore, this section describes only DLL naming conventions, which then can be mapped to assembly naming conventions.

[= fa thumbs o up] choose names for your assembly DLLs that suggest large chunks of functionality, such as System.Data.

Assembly and DLL names don't have to correspond to namespace names, but it is reasonable to follow the namespace name when naming assemblies. A good rule of thumb is to name the DLL based on the common prefix of the assemblies contained in the assembly. For example, an assembly

with two namespaces, MyCompany.MyTechnology.FirstFeature and MyCompany.MyTechnology.SecondFeature, could be called MyCompany.MyTechnology.dll.

[= fa thumbs up] consider naming DLLs according to the following pattern:
 <Company>.<Component>.dll where <Component> contains one or more dot-separated clauses.

For example: Litware.Controls.dll.

26.1.5. General coding style

- Source files: One Namespace per file and one class per file.
- Braces: On new line. Always use braces when optional.
- Indention: Use tabs with size of 4.
- Comments: Use // for simple comment or /// for summaries. Do not /* ... */ and do not flower box.
- Use Use built-in C# native data types vs .NET CTS types (string instead of String)
- Avoid changing default type in Enums.
- Use *base* or *this* only in constructors or within an override.
- Always check for null before invoking events.
- Avoid using *Finalize*. Use C# Destructors and do not create Finalize() method.
- Suggestion: Use blank lines, to make it much more readable by dividing it into small, easy-to-digest sections:
 - Use a single blank line to separate logical groups of code, such as control structures.
 - Use two blank lines to separate method definitions

Case	Convention
Source File	Pascal case. Match class name and file name
Namespace	Pascal case
Class	Pascal case
Interface	Pascal case
Generics	Single capital letter (T or K)
Methods	Pascal case (use a Verb or Verb+Object)
Public field	Pascal case
Private field	Camel case with underscore (_) prefix
Static field	Pascal case
Property	Pascal case. Try to use get and set convention {get;set;}
Constant	Pascal case

Case	Convention
Enum	Pascal case
Variable (inline)	Camel case
Param	Camel case

26.1.6. Use of Region guideline

Regions can be used to collapse code inside Visual Studio .NET. Regions are ideal candidates to hide boiler plate style code that adds little value to the reader on your code. Regions can then be expanded to provide progressive disclosure of the underlying details of the class or method.

- Do Not regionalise entire type definitions that are of an important nature. Types such as enums (which tend to be fairly static in their nature) can be regionalised – their permissible values show up in Intellisense anyway.
- Do Not regionalise an entire file. When another developer opens the file, all they will see is a single line in the code editor pane.
- Do regionalise boiler plate type code.

26.1.7. Use of Comment guideline

Code is the only completely reliable documentation: write “good code” first!

Avoid Unnecessary comments

- Choosing good names for fields, methods, parameters, etc. “let the code speak” (K. Beck) by itself reducing the need for comments and documentation
- Avoid “repeating the code” and commenting the obvious
- Avoid commenting “tricky code”: rewrite it! If there’s no time at present to refactor a tricky section, mark it with a TODO and schedule time to take care of it as soon as possible.

Effective comments

- Use comments to summarize a section of code
- Use comments to clarify sensitive pieces of code
- Use comments to clarify the intent of the code
- Bad written or out-of-date comments are more damaging than helpful:
- Write clear and effective comments
- Pay attention to pre-existing comments when modifying code or copying&pasting code

26.1.8. External links

[Naming guidelines](#)

[General naming conventions](#)

Capitalization conventions

Assembly and Name Spaces conventions

27. Environment

27.1. Environment

27.1.1. Overview

27.1.2. Required software

[Visual Studio Code](#)

[C# Extension for VS Code](#)

[.Net Core SDK](#)

27.1.3. Setting up the environment

1. Download and install [Visual Studio Code](#)
2. Download and install [.Net Core SDK](#)
3. [Install the extension Omnisharp](#) in Visual Studio Code

Hello world

1. Open a project:
 - Open Visual Studio Code.
 - Click on the Explorer icon on the left menu and then click **Open Folder**.
 - Select the folder you want your C# project to be in and click **Select Folder**. For our example, we'll create a folder for our project named 'HelloWorld'.
2. Initialize a C# project:
 - Open the Integrated Terminal from Visual Studio Code by typing CTRL+` (backtick). Alternatively, you can select **View > Integrated Terminal** from the main menu.
 - In the terminal window, type `dotnet new console`.
 - This creates a `Program.cs` file in your folder with a simple "Hello World" program already written, along with a C# project file named `HelloWorld.csproj`.
3. Resolve the build assets:
 - For **.NET Core 2.0**, this step is optional. The `dotnet restore` command executes automatically when a new project is created.
4. Run the "Hello World" program:
 - Type `dotnet run`.

Debug

1. Open `Program.cs` by clicking on it. The first time you open a C# file in Visual Studio Code, OmniSharp will load in the editor.

2. Visual Studio Code will prompt you to add the missing assets to build and debug your app. Select Yes.
3. To open the Debug view, click on the Debugging icon on the left side menu.
4. Locate the green arrow at the top of the pane. Make sure the drop-down next to it has **.NET Core Launch (console)** selected.
5. Add a breakpoint to your project by clicking on the **editor margin** (the space on the left of the line numbers in the editor).
6. Select F5 or the green arrow to start debugging. The debugger stops execution of your program when it reaches the breakpoint you set in the previous step.
 - While debugging you can view your local variables in the top left pane or use the debug console.
7. Select the green arrow at the top to continue debugging, or select the red square at the top to stop.



For more information and troubleshooting tips on .NET Core debugging with OmniSharp in Visual Studio Code, see [Instructions for setting up the .NET Core debugger](#).

27.1.4. External links

[.Net Core](#)

[Using .NET Core in Visual Studio Code](#)

[.Net Core in Visual Studio Code tutorial](#)

28. Packages

28.1. Packages

28.1.1. Packages overview



devon4Net is composed by a number of packages that increases the functionality and boosts time development. Each package has it's own configuration to make them work properly. In *appsettings.json* set up your environment. On *appsettings.{environment}.json* you can configure each component.

28.1.2. The packages

You can get the *devon4Net* packages on [nuget.org](https://www.nuget.org).

Devon4Net.Application.WebAPI.Configuration

Description

The *devon4Net* web API configuration core.

Configuration

- Install package on your solution:

```
PM> Install-Package Devon4Net.Application.WebAPI.Configuration
```

Default configuration values

```

"devonfw": {
  "UseDetailedErrorsKey": true,
  "UseIIS": false,
  "UseSwagger": true,
  "Environment": "Development",
  "KillSwitch": {
    "killSwitchSettingsFile": "killswitch.appsettings.json"
  },
  "Kestrel": {
    "UseHttps": true,
    "HttpProtocol": "Http2", //Http1, Http2, Http1AndHttp2, none
    "ApplicationPort": 8082,
    "KeepAliveTimeout": 120, //in seconds
    "MaxConcurrentConnections": 100,
    "MaxConcurrentUpgradedConnections": 100,
    "MaxRequestBodySize": 28.6, //In MB. The default maximum request body size is
30,000,000 bytes, which is approximately 28.6 MB
    "Http2MaxStreamsPerConnection": 100,
    "Http2InitialWindowSize": 131072, // From 65,535 and less than 2^31 (2,147,483,648)
    "Http2InitialStreamWindowSize": 98304, // From 65,535 and less than 2^31 (2,147,483,648)
    "AllowSynchronousIO": true,
    "SslProtocol": "Tls12", //Tls, Tls11,Tls12, Tls13, Ssl2, Ssl3, none. For Https2
Tls12 is needed
    "ServerCertificate": {
      "Certificate": "localhost.pfx",
      "CertificatePassword": "localhost"
    },
    "ClientCertificate": {
      "DisableClientCertificateCheck": true,
      "RequireClientCertificate": false,
      "CheckCertificateRevocation": true,
      "ClientCertificates": {
        "Whitelist": [
          "3A87A49460E8FE0E2A198E63D408DC58435BC501"
        ],
        "DisableClientCertificateCheck": false
      }
    }
  },
  "IIS": {
    "ForwardClientCertificate": true,
    "AutomaticAuthentication": true,
    "AuthenticationDisplayName" : ""
  }
}

```

Devon4Net.Infrastructure.CircuitBreaker

Description

The Devon4Net.Infrastructure.CircuitBreaker component implements the retry pattern for HTTP/HTTPS calls.

Configuration

- Install package on your solution:

```
PM> Install-Package Devon4Net.Infrastructure.CircuitBreaker
```

Default configuration values

```
"CircuitBreaker": {
  "CheckCertificate": true,
  "Endpoints": [
    {
      "Name": "SampleService",
      "BaseAddress": "https://localhost:5001/",
      "Headers": {},
      "WaitAndRetrySeconds": [
        0.0001,
        0.0005,
        0.001
      ],
      "DurationOfBreak": 0.0005,
      "UseCertificate": true,
      "Certificate": "localhost.pfx",
      "CertificatePassword": "localhost",
      "SslProtocol": "3072" //TLS12
    }
  ]
}
```

Property	Description
CheckCertificate	True if HTTPS is required. This is useful when developing an API Gateway needs a secured HTTP, disabling this on development we can use communications with a valid server certificate
Endpoints	Array with predefined sites to connect with
Name	The name key to identify the destination URL
Headers	Not ready yet

Property	Description
WaitAndRetrySeconds	Array which determines the number of retries and the lapse period between each retry. The value is in milliseconds.
Certificate	Certificate client to use to perform the HTTP call
SslProtocol	The secure protocol to use on the call

Protocols

Protocol	Key	Description
SSL3	48	Specifies the Secure Socket Layer (SSL) 3.0 security protocol. SSL 3.0 has been superseded by the Transport Layer Security (TLS) protocol and is provided for backward compatibility only.
TLS	192	Specifies the Transport Layer Security (TLS) 1.0 security protocol. The TLS 1.0 protocol is defined in IETF RFC 2246.
TLS11	768	Specifies the Transport Layer Security (TLS) 1.1 security protocol. The TLS 1.1 protocol is defined in IETF RFC 4346. On Windows systems, this value is supported starting with Windows 7.
TLS12	3072	Specifies the Transport Layer Security (TLS) 1.2 security protocol. The TLS 1.2 protocol is defined in IETF RFC 5246. On Windows systems, this value is supported starting with Windows 7.
TLS13	12288	Specifies the TLS 1.3 security protocol. The TLS protocol is defined in IETF RFC 8446.

Usage

Add via Dependency Injection the circuit breaker instance. PE:

```

public class FooService : Service<TodosContext>, ILoginService
{
    public FooService(IUnitOfWork<AUTContext> uow, ICircuitBreakerHttpClient
circuitBreakerClient,
        ILogger<LoginService> logger) : base(uow)
    {
        ...
    }
}

```

At this point you can use the circuit breaker functionality in your code.

To perform a POST call you should use your circuit breaker instance as follows:

```

await circuitBreakerClient.PostAsync<YourOutputClass>(NameOftheService, EndPoint,
InputData, MediaType.ApplicationJson).ConfigureAwait(false);

```

Where:

Property	Description
YourOutputClass	The type of the class that you are expecting to retrieve from the POST call
NameOftheService	The key name of the endpoint provided in the appsettings.json file at Endpoints[] node
EndPoint	Part of the url to use with the base address. PE: /validate
InputData	Your instance of the class with values that you want to use in the POST call
MediaType.ApplicationJson	The media type flag for the POST call

devon4Net.Domain.UnitOfWork

Description

Unit of work implementation for devon4net solution. This unit of work provides the different methods to access the data layer with an atomic context. Sync and Async repository operations are provided. Customized Eager Loading method also provided for custom entity properties.



This component will move on next releases to Infrastructure instead of being part of Domain components

Configuration

- Install package on your solution:

```
PM> Install-Package devon4Net.Domain.UnitOfWork
```

- Adding the database connection information:

Add the database connection on the SetupDatabase method at Startup.cs

```
private void SetupDatabase(IServiceCollection services)
{
    services.SetupDatabase<TodoContext>(Configuration, "Default",
WebAPI.Configuration.Enums.DatabaseType.InMemory);
}
```

Where:

Param	Description
TodoContext	Is the database context definition
Default	Is the connection string defined at <i>ConnectionString</i> node at the appsettings configuration file
WebAPI.Configuration.Enums.DatabaseType.InMemory	Is the database driver selection. In this case InMemory data base is chosen

The supported databases are:

- SqlServer
- Sqlite
- InMemory
- Cosmos
- PostgreSQL
- MySql
- MariaDb
- FireBird
- Oracle
- MSAccess

Notes

Now you can use the unit of work via dependency injection on your classes:

Figure 82. Use of Unit of work via dependency injection

As you can see in the image, you can use Unit Of Work class with your defined ModelContext

classes.

Predicate expression builder

- Use this expression builder to generate lambda expressions dynamically.

```
var predicate = PredicateBuilder.True<T>();
```

Where T is a class. At this moment, you can build your expression and apply it to obtain your results in a efficient way and not retrieving data each time you apply an expression.

- Example from My Thai Star .Net Core implementation:

```

public async Task<PaginationResult<Dish>> GetpagedDishListFromFilter(int currentPage,
int pageSize, bool isFav, decimal maxPrice, int minLikes, string searchBy, IList<long>
categoryIdList, long userId)
{
    var includeList = new
List<string>{"DishCategory","DishCategory.IdCategoryNavigation",
"DishIngredient","DishIngredient.IdIngredientNavigation","IdImageNavigation"};

    //Here we create our predicate builder
    var dishPredicate = PredicateBuilder.True<Dish>();

    //Now we start applying the different criteria:
    if (!string.IsNullOrEmpty(searchBy))
    {
        var criteria = searchBy.ToLower();
        dishPredicate = dishPredicate.And(d => d.Name.ToLower().Contains(criteria) ||
d.Description.ToLower().Contains(criteria));
    }

    if (maxPrice > 0) dishPredicate = dishPredicate.And(d=>d.Price<=maxPrice);

    if (categoryIdList.Any())
    {
        dishPredicate = dishPredicate.And(r => r.DishCategory.Any(a =>
categoryIdList.Contains(a.IdCategory)));
    }

    if (isFav && userId >= 0)
    {
        var favourites = await UoW.Repository<UserFavourite>().GetAllAsync(w=>w.IdUser
== userId);
        var dishes = favourites.Select(s => s.IdDish);
        dishPredicate = dishPredicate.And(r=> dishes.Contains(r.Id));
    }

    // Now we can use the predicate to retrieve data from database with just one call
    return await UoW.Repository<Dish>().GetAllIncludePagedAsync(currentpage, pageSize,
includeList, dishPredicate);
}

```

devon4Net.Infrastructure.Extensions

Description

Miscellaneous extension library which contains : - Predicate expression builder - DateTime formatter - HttpClient - HttpContext (Middleware support)

Configuration

- Install package on your solution:

```
PM> Install-Package devon4Net.Infrastructure.Extensions
```

HttpContext

- TryAddHeader method is used on *devon4Net.Infrastructure.Middleware* component to add automatically response header options such authorization.

devon4Net.Infrastructure.JWT

Description

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

— What is JSON Web Token?, <https://jwt.io/introduction/>

- devon4Net component to manage JWT standard to provide security to .Net API applications.

Configuration

- Install package on your solution:

```
PM> devon4Net.Infrastructure.JWT
```

Default configuration values

```

"JWT": {
    "Audience": "devon4Net",
    "Issuer": "devon4Net",
    "TokenExpirationTime": 60,
    "ValidateIssuerSigningKey": true,
    "ValidateLifetime": true,
    "ClockSkew": 5,
    "Security": {
        "SecretKeyLengthAlgorithm": "",
        "SecretKeyEncryptionAlgorithm": "",
        "SecretKey": "",
        "Certificate": "",
        "CertificatePassword": "",
        "CertificateEncryptionAlgorithm": ""
    }
}

```

- *ClockSkew* indicates the token expiration time in minutes
- *Certificate* you can specify the name of your certificate (if it is on the same path) or the full path of the certificate. If the certificate does not exists an exception will be raised.
- *SecretKeyLengthAlgorithm*, *SecretKeyEncryptionAlgorithm* and *CertificateEncryptionAlgorithm* supported algorithms are:

Algorithm	Description
Aes128Encryption	"http://www.w3.org/2001/04/xmlenc#aes128-cbc"
Aes192Encryption	"http://www.w3.org/2001/04/xmlenc#aes192-cbc"
Aes256Encryption	"http://www.w3.org/2001/04/xmlenc#aes256-cbc"
DesEncryption	"http://www.w3.org/2001/04/xmlenc#des-cbc"
Aes128KeyWrap	"http://www.w3.org/2001/04/xmlenc#kw-aes128"
Aes192KeyWrap	"http://www.w3.org/2001/04/xmlenc#kw-aes192"
Aes256KeyWrap	"http://www.w3.org/2001/04/xmlenc#kw-aes256"
RsaV15KeyWrap	"http://www.w3.org/2001/04/xmlenc#rsa-1_5"
Ripemd160Digest	"http://www.w3.org/2001/04/xmlenc#ripemd160"
RsaOaepKeyWrap	"http://www.w3.org/2001/04/xmlenc#rsa-oaep"
Aes128KW	"A128KW"
Aes256KW	"A256KW"
RsaPKCS1	"RSA1_5"
RsaOAEP	"RSA-OAEP"
ExclusiveC14n	"http://www.w3.org/2001/10/xml-exc-c14n#"

Algorithm	Description
ExclusiveC14nWithComments	"http://www.w3.org/2001/10/xml-exc-c14n#WithComments"
EnvelopedSignature	"http://www.w3.org/2000/09/xmldsig#enveloped-signature"
Sha256Digest	"http://www.w3.org/2001/04/xmlenc#sha256"
Sha384Digest	"http://www.w3.org/2001/04/xmldsig-more#sha384"
Sha512Digest	"http://www.w3.org/2001/04/xmlenc#sha512"
Sha256	"SHA256"
Sha384	"SHA384"
Sha512	"SHA512"
EcdsaSha256Signature	"http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256"
EcdsaSha384Signature	"http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha384"
EcdsaSha512Signature	"http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha512"
HmacSha256Signature	"http://www.w3.org/2001/04/xmldsig-more#hmac-sha256"
HmacSha384Signature	"http://www.w3.org/2001/04/xmldsig-more#hmac-sha384"
HmacSha512Signature	"http://www.w3.org/2001/04/xmldsig-more#hmac-sha512"
RsaSha256Signature	"http://www.w3.org/2001/04/xmldsig-more#rsa-sha256"
RsaSha384Signature	"http://www.w3.org/2001/04/xmldsig-more#rsa-sha384"
RsaSha512Signature	"http://www.w3.org/2001/04/xmldsig-more#rsa-sha512"
RsaSsaPssSha256Signature	"http://www.w3.org/2007/05/xmldsig-more#sha256-rsa-MGF1"
RsaSsaPssSha384Signature	"http://www.w3.org/2007/05/xmldsig-more#sha384-rsa-MGF1"
RsaSsaPssSha512Signature	"http://www.w3.org/2007/05/xmldsig-more#sha512-rsa-MGF1"
EcdsaSha256	"ES256"
EcdsaSha384	"ES384"

Algorithm	Description
EcdsaSha512	"ES512"
HmacSha256	"HS256"
HmacSha384	"HS384"
HmacSha512	"HS512"
None	"none"
RsaSha256	"RS256"
RsaSha384	"RS384"
RsaSha512	"RS512"
RsaSsaPssSha256	"PS256"
RsaSsaPssSha384	"PS384"
RsaSsaPssSha512	"PS512"
Aes128CbcHmacSha256	"A128CBC-HS256"
Aes192CbcHmacSha384	"A192CBC-HS384"
Aes256CbcHmacSha512	"A256CBC-HS512"



Please check [Microsoft documentation](#) to get the lastest updates on supported encryption algorithms

- Add this line of code (only if you use this component stand alone):

```
services.AddBusinessCommonJwtPolicy();
```

On

Startup.cs

or on:

```
devon4Net.Application.Configuration.Startup/JwtApplicationConfiguration/ConfigureJwtPolicy method.
```

- Inside the *AddBusinessCommonJwtPolicy* method you can add your JWT Policy like in My Thai Star application sample:

```
services.ConfigureJwtAddPolicy("MTSWaiterPolicy", "role", "waiter");
```

Notes

- The certificate will be used to generate the key to encrypt the json web token.

devon4Net.Infrastructure.Middleware

Description

- devon4Net support for middleware classes.
- In ASP.NET Core, middleware classes can handle an HTTP request or response. Middleware can either:
 - Handle an incoming HTTP request by generating an HTTP response.
 - Process an incoming HTTP request, modify it, and pass it on to another piece of middleware.
 - Process an outgoing HTTP response, modify it, and pass it on to either another piece of middleware, or the ASP.NET Core web server.
- devon4Net supports the following automatic response headers:
 - AccessControlExposeHeader
 - StrictTransportSecurityHeader
 - XFrameOptionsHeader
 - XSSProtectionHeader
 - XContentTypeOptionsHeader
 - ContentSecurityPolicyHeader
 - PermittedCrossDomainPoliciesHeader
 - ReferrerPolicyHeader:toc: macro

Configuration

- Install package on your solution:

```
PM> Install-Package devon4Net.Infrastructure.Middleware
```

- You can configure your Middleware configuration on *appsettings.{environment}.json*:

```

"Middleware": {
  "Headers": {
    "AccessControlExposeHeader": "Authorization",
    "StrictTransportSecurityHeader": "",
    "XFrameOptionsHeader": "DENY",
    "XssProtectionHeader": "1;mode=block",
    "XContentTypeOptionsHeader": "nosniff",
    "ContentSecurityPolicyHeader": "",
    "PermittedCrossDomainPoliciesHeader": "",
    "ReferrerPolicyHeader": ""
  }
}

```

- On the above sample, the server application will add to response header the AccessControlExposeHeader, XFrameOptionsHeader, XssProtectionHeader and XContentTypeOptionsHeader headers.
- If the header response type does not have a value, it will not be added to the response headers.

devon4Net.Infrastructure.Swagger

Description

- devon4net Swagger abstraction to provide full externalized easy configuration.
- Swagger offers the easiest to use tools to take full advantage of all the capabilities of the OpenAPI Specification (OAS).

Configuration

- Install package on your solution:

```
PM> devon4Net.Infrastructure.Swagger
```

- You can configure your Swagger configuration on *appsettings.{environment}.json*:

```

"Swagger": {
    "Version": "v1",
    "Title": "devon4net API",
    "Description": "devon4net API Contract",
    "Terms": "https://www.devonfw.com/terms-of-use/",
    "Contact": {
        "Name": "devonfw",
        "Email": "sample@mail.com",
        "Url": "https://www.devonfw.com"
    },
    "License": {
        "Name": "devonfw - Terms of Use",
        "Url": "https://www.devonfw.com/terms-of-use/"
    },
    "Endpoint": {
        "Name": "V1 Docs",
        "Url": "/swagger/v1/swagger.json",
        "UrlUi": "swagger",
        "RouteTemplate": "swagger/v1/{documentName}/swagger.json"
    }
}

```

- Add this line of code (only if you use this component stand alone):

```
services.ConfigureSwaggerService();
```

On

```
Startup.cs
```

- Also add this line of code (only if you use this component stand alone):

```
app.ConfigureSwaggerApplication();
```

On

```
Startup.cs/Configure(IApplicationBuilder app, IHostingEnvironment env)
```

- Ensure your API actions and non-route parameters are decorated with explicit "Http" and "From" bindings.

Notes

- To access to swagger UI launch your API project and type in your html browser the url <http://localhost:yourPort/swagger>.

- In order to generate the documentation annotate your actions with summary, remarks and response tags:

```

/// <summary>
/// Method to make a reservation with potential guests. The method returns the
reservation token with the format:
{(CB_|GB_)}{now.Year}{now.Month:00}{now.Day:00}{_}{MD5({Host/Guest-
email}{now.Year}{now.Month:00}{now.Day:00}{now.Hour:00}{now.Minute:00}{now.Second:00})
}
/// </summary>
/// <param name="bookingDto"></param>
/// <response code="201">Ok.</response>
/// <response code="400">Bad request. Parser data error.</response>
/// <response code="401">Unauthorized. Authentication fail.</response>
/// <response code="403">Forbidden. Authorization error.</response>
/// <response code="500">Internal Server Error. The search process ended with
error.</response>
[HttpPost]
[HttpOptions]
[Route("/mythaistar/services/rest/bookingmanagement/v1/booking")]
[AllowAnonymous]
[EnableCors("CorsPolicy")]
public async Task<IActionResult> BookingBooking([FromBody]BookingDto bookingDto)
{
    try
    {
        ...
    }
}

```

- Ensure that your project has the *generate XML documentation file* check active on build menu:

Figure 83. Swagger documentation

- Ensure that your XML files has the attribute copy always to true:

Figure 84. Swagger documentation

devon4Net.Infrastructure.Test

Description

devon4Net Base classes to create unit tests and integration tests with Moq and xUnit.

Configuration

- Load the template: > dotnet new -i devon4Net.Test.Template > dotnet new devon4NetTest

Notes

- At this point you can find this classes:

- BaseManagementTest
- DatabaseManagementTest<T> (Where T is a *devon4NetBaseContext* class)
- For unit testing, inherit a class from *BaseManagementTest*.
- For integration tests, inherit a class from *DatabaseManagementTest*.
- The recommended databases in integration test are *in memory database* or *SQLite database*.
- Please check *My thai Star* test project.

28.1.3. Deperecated packages

devon4Net.Domain.Context

Description

devon4Net.Domain.Context contains the extended class *devon4NetBaseContext* in order to make easier the process of having a model context configured against different database engines. This configuration allows an easier testing configuration against local and in memory databases.

Configuration

- Install package on your solution:

```
PM> Install-Package devon4Net.Domain.Context
```

- Add to *appsettings.{environment}.json* file your database connections:

```
"ConnectionStrings":  
{  
  "DefaultConnection":  
    "Server=localhost;Database=MyThaiStar;User  
    Id=sa;Password=sa;MultipleActiveResultSets=True",  
  
  "AuthConnection":  
    "Server=(localdb)\\mssqllocaldb;Database=aspnet-DualAuthCore-5E206A0B-D4DA-4E71-92D3-  
    87FD6B120C5E;Trusted_Connection=True;MultipleActiveResultSets=true",  
  
  "SqliteConnection": "Data Source=c:\\tmp\\membership.db;"  
}
```

- On Startup.cs :

```
void ConfigureServices(IServiceCollection services)
```

- Add your database connections defined on previous point:

```
services.ConfigureDataBase(
    new Dictionary<string, string> {
        {ConfigurationConst.DefaultConnection,
        Configuration.GetConnectionString(ConfigurationConst.DefaultConnection) }});
});
```

- On `devon4Net.Application.Configuration.Startup/ DataBaseConfiguration/ConfigureDataBase` configure your connections.

devon4Net.Infrastructure ApplicationUser

Description

devon4Net Application user classes to implement basic Microsoft's basic authentication in order to be used on authentication methodologies such Jason Web Token (JWT).

Configuration

- Install package on your solution:

```
PM> devon4Net.Infrastructure ApplicationUser
```

- Add the database connection string for user management on `appsettings.{environment}.json`:

```
".ConnectionStrings":
{
    "AuthConnection":
        "Server=(localdb)\mssqllocaldb;Database=aspnet-DualAuthCore-5E206A0B-D4DA-4E71-92D3-
        87FD6B120C5E;Trusted_Connection=True;MultipleActiveResultSets=true"
}
```

- Add the following line of code

```
services.AddApplicationUserDependencyInjection();
```

On

```
Startup.cs/ConfigureServices(IServiceCollection services)
```

or on:

```
devon4Net.Application.Configuration.Startup/ DependencyInjectionConfiguration/Configure
DependencyInjectionService method.
```

- Add the data seeder on Configure method on start.cs class:

```

public void Configure(IApplicationBuilder app, IHostingEnvironment env, DataSeeder
seeder)
{
    ...
    app.UseAuthentication();
    seeder.SeedAsync().Wait();
    ...
}

```

Notes

- You can use the following methods to set up the database configuration:

```
public static void AddApplicationDbContextInMemoryService(this IServiceCollection
services)
```

```
public static void AddApplicationDbContextSqliteService(this IServiceCollection
services, string connectionString)
```

```
public static void AddApplicationDbContextSqlServerService(this IServiceCollection
services, string connectionString)
```

- The method *AddApplicationDbContextInMemoryService* uses the *AuthContext* connection string name to set up the database.
- This component is used with the components *devon4Net.Infrastructure.JWT* and *devon4Net.Infrastructure.JWT.MVC*.

devon4Net.Infrastructure.Communication

Description

Basic client classes to invoke GET/POST methods asynchronously. This component has the minimal classes to send basic data. For more complex operations please use *ASP4Net.Infrastructure.Extensions*.

Configuration

- Install package on your solution:

```
PM> devon4Net.Infrastructure.Communication
```

- Create an instance of *RestManagementService* class.
- Use next methods to use GET/POST basic options:

```
public Task<string> CallGetMethod(string url);
public Task<Stream> CallGetMethodAsStream(string url);
public Task<string> CallPostMethod<T>(string url, T dataToSend);
public Task<string> CallPutMethod<T>(string url, T dataToSend);
```

Notes

- Example:

```
private async Task RestManagementServiceSample(EmailDto dataToSend)
{
    var url = Configuration["EmailServiceUrl"];
    var restManagementService = new RestManagementService();
    await restManagementService.CallPostMethod(url, dataToSend);
}
```

devon4Net.Infrastructure.JWT.MVC

Description

- devon4Net Extended controller to interact with JWT features

Configuration

- Extend your `_ Microsoft.AspNetCore.Mvc.Controller_` class with `devon4NetJWTController` class:

```
public class LoginController : devon4NetJWTController
{
    private readonly ILoginService _loginService;

    public LoginController(ILoginService loginService, SignInManager<ApplicationUser>
signInManager, UserManager<ApplicationUser> userManager, ILogger<LoginController>
logger, IMapper mapper) : base(logger, mapper)
    {
        _loginService = loginService;
    }

    ....
```

Notes

- In order to generate a JWT, you should implement the JWT generation on user login. For example, in My Thai Star is created as follows:

```

public async Task<IActionResult> Login([FromBody]LoginDto loginDto)
{
    try
    {
        if (loginDto == null) return Ok();
        var logged = await _loginService.LoginAsync(loginDto.UserName,
loginDto.Password);

        if (logged)
        {
            var user = await _loginService.GetUserByUserNameAsync(loginDto.UserName);

            var encodedJwt = new
JwtClientToken().CreateClientToken(_loginService.GetUserClaimsAsync(user));

            Response.Headers.Add("Access-Control-Expose-Headers", "Authorization");

            Response.Headers.Add("Authorization",
"${JwtBearerDefaults.AuthenticationScheme} {encodedJwt}");

            return Ok(encodedJwt);
        }
        else
        {
            Response.Headers.Clear();
            return StatusCode((int) HttpStatusCode.Unauthorized, "Login Error");
        }
    }
    catch (Exception ex)
    {
        return StatusCode((int) HttpStatusCode.InternalServerError, $"{ex.Message} : {ex.InnerException}");
    }
}

```

- In My Thai Star the JWT will contain the user information such id, roles...
- Once you extend your controller with *devon4NetJWTController* you will have available these methods to simplify user management:

```

public interface Idevon4NetJWTController
{
    // Gets the current user
    JwtSecurityToken GetCurrentUser();

    // Gets an specific assigned claim of current user
    Claim GetUserClaim(string claimName, JwtSecurityToken jwtUser = null);

    // Gets all the assigned claims of current user
    IEnumerable<Claim> GetUserClaims(JwtSecurityToken jwtUser = null);
}

```

devon4Net.Infrastructure.MVC

Description

Common classes to extend controller functionality on API. Also provides support for paged results in devon4Net applications and automapper injected class.

Configuration

- Install package on your solution:

```
PM> devon4Net.Infrastructure.MVC
```

Notes

- The generic class *ResultObjectDto<T>* provides a typed result object with pagination.
- The extended class provides the following methods:

```

ResultObjectDto<T> GenerateResultDto<T>(int? page, int? size, int? total);
ResultObjectDto<T> GenerateResultDto<T>(List<T> result, int? page = null, int?
size = null);

```

- *GenerateResultDto* provides typed *ResultObjectDto* object or a list of typed *ResultObjectDto* object. The aim of this methods is to provide a clean management for result objects and not repeating code through the different controller classes.
- The following sample from *My Thai Star* shows how to use it:

```

public async Task<IActionResult> Search([FromBody] FilterDtoSearchObject filterDto)
{
    if (filterDto == null) filterDto = new FilterDtoSearchObject();

    try
    {
        var dishList = await _dishService.GetDishListFromFilter(false,
filterDto.GetMaxPrice(), filterDto.GetMinLikes(),
filterDto.GetSearchBy(), filterDto.GetCategories(), -1);

        return new OkObjectResult(GenerateResultDto(dishList).ToJson());
    }
    catch (Exception ex)
    {
        return StatusCode((int) HttpStatusCode.InternalServerError, $"'{ex.Message}' : {ex.InnerException}");
    }
}

```

devon4Net.Infrastructure.AOP

Description

Simple AOP Exception handler for .Net Controller classes integrated with Serilog.

Configuration

- Install package on your solution:

```
PM> Install-Package devon4Net.Domain.AOP
```

Add this line of code on ConfigureServices method on Startup.cs

```
services.AddAopAttributeService();
```

Notes

Now automatically your exposed API methods exposed on controller classes will be tracked on the methods:

- OnActionExecuting
- OnActionExecuted
- OnResultExecuting
- OnResultExecuted

If an exception occurs, a message will be displayed on log with the stack trace.

devon4Net.Infrastructure.Cors

Description

Enables CORS configuration for devon4Net application. Multiple domains can be configured from configuration. Mandatory to web clients (p.e. Angular) to prevent making AJAX requests to another domain.

Cross-Origin Resource Sharing (CORS) is a mechanism that uses additional HTTP headers to tell a browser to let a web application running at one origin (domain) have permission to access selected resources from a server at a different origin. A web application makes a cross-origin HTTP request when it requests a resource that has a different origin (domain, protocol, and port) than its own origin.

Please refer to [this link](#) to get more information about CORS and .Net core.

Configuration

- Install package on your solution:

```
PM> devon4Net.Infrastructure.Cors
```

- You can configure your Cors configuration on *appsettings.{environment}.json*:

CorsPolicy: indicates the name of the policy. You can use this name to add security headers on your API exposed methods.

Origins: The allowed domains

Headers: The allowed headers such `accept`,`content-type`,`origin`,`x-custom-header`

- If you specify the cors configuration as empty array, a default cors-policy will be used with all origins enabled:

```
"Cors": []
```

- On the other hand, you can specify different Cors policies in your solution as follows:

```

"Cors": []
[
  {
    "CorsPolicy": "CorsPolicy1",
    "Origins": "http://example.com,http://www.contoso.com",
    "Headers": "accept,content-type,origin,x-custom-header",
    "Methods": "GET,POST,HEAD",
    "AllowCredentials": true
  },
  {
    "CorsPolicy": "CorsPolicy2",
    "Origins": "http://example.com,http://www.contoso.com",
    "Headers": "accept,content-type,origin,x-custom-header",
    "Methods": "GET,POST,HEAD",
    "AllowCredentials": true
  }
]

```

Notes

- To use CORS in your API methods, use the next notation:

```

[EnableCors("YourCorsPolicy")]
public IActionResult Index() {
    return View();
}

```

- if you want to disable the CORS check use the following annotation:

```

[DisableCors]
public IActionResult Index() {
    return View();
}

```

28.2. Required software

[Visual Studio Code](#)

[C# Extension for VS Code](#)

[.Net Core SDK](#)

[CORS in .Net Core](#)

29. Templates

29.1. Templates

29.1.1. Overview

The .Net Core and .Net Framework given templates allows to start coding an application with the following functionality ready to use:

Please refer to [User guide](#) in order to start developing.

29.1.2. Net Core 3.0

The *.Net Core 3.0* template allows you to start developing an n-layer server application to provide the latest features. The template can be used in Visual Studio Code and Visual Studio 2019.

The application result can be deployed as a console application, microservice or web page.

To start developing with devon4Net template, please follow this instructions:

Using devon4Net template

Option 1

1. Open your favourite terminal (Win/Linux/iOS)
2. Go to future project's path
3. Type `dotnet new --install Devon4Net.WebAPI.Template`
4. Type `dotnet new Devon4NetAPI`
5. Go to project's path
6. You are ready to start developing with *devon4Net*

Option 2

1. Create a new dotnet API project from scratch
2. Add the nuget package reference to your project
3. Type `dotnet new --install Devon4Net.WebAPI.Template`

29.1.3. Net Core 2.1.x

The *.Net Core 2.1.x* template allows you to start developing an n-layer server application to provide the latest features. The template can be used in Visual Studio Code and Visual Studio 2017.

The application result can be deployed as a console application, microservice or web page.

To start developing with devon4Net template, please follow this instructions:

Using devon4Net template

1. Open your favourite terminal (Win/Linux/iOS)
2. Go to future project's path
3. Type `dotnet new --install Devon4Net.WebAPI.Template::1.0.8`
4. Type `dotnet new Devon4NetAPI`
5. Go to project's path
6. You are ready to start developing with *devon4Net*



For the latest updates on references packages, please get the sources from [Github](#)

29.1.4. Links

[= fa floppy o] [.Net templates](#)

30. Samples

30.1. Samples

30.1.1. My Thai Star Restaurant

[= fa floppy o] [My Thai Star \(.Net Core Server + Angular client\)](#)

► /home/travis/build/devonfw/devonfw-guide/target/generated-docs/videos/mts_startup.mp4 (video)

Angular requirements

- [Node](#)
- [Angular CLI](#)
- [Yarn](#)

Angular client

1. Install Node.js LTS version
2. Install Angular CLI from command line:
 - `npm install -g @angular/cli`
3. Install Yarn
4. Go to Angular client from command line
5. Execute :*yarn install*
6. Launch the app from command line: *ng serve* and check <http://localhost:4200>
7. You are ready

.Net Core server

Basic architecture details

Following the OASP conventions the .Net Core 2.0 My Thai Star backend is going to be developed dividing the application in *Components* and using a n-layer architecture.

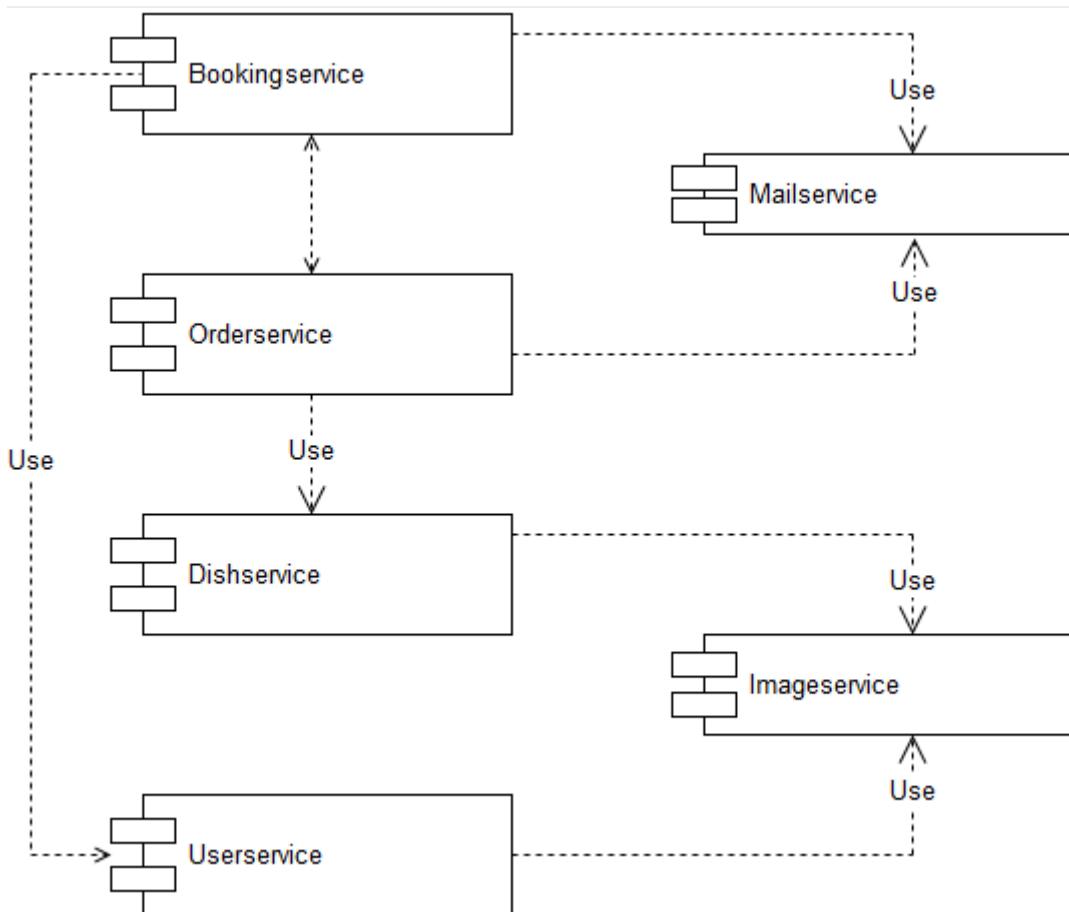
MTS.CORE

- ◀ Application
 - ▶ Resources
 - ▶ WebApi
- ◀ Architecture
 - ▶ Oasp4Net.Architecture.Common
- ◀ Business
 - ▶ Oasp4Net.Business.Common
 - ▶ Oasp4Net.Business.Controller
 - ▶ Oasp4Net.Business.CoreViews
- ◀ DataAccess
 - ▶ DataAccessLayer
 - ▶ Repositories
- ◀ Infrastructure
 - ▶ Oasp4Net.Infrastructure.AOP
 - ▶ Oasp4Net.Infrastructure.JWT
 - ▶ Oasp4Net.Infrastructure.Mail
- ◀ Service
 - ▶ bin
 - ▶ BookingService
 - ▶ DishService
 - ▶ InvitedGuestService
 - ▶ LoginService
 - ▶ obj
 - ▶ OrderDishExtraIngredientService
 - ▶ OrderLineService
 - ▶ OrderService
 - RSS Oasp4Net.Business.Service.csproj

≡ Traiectum.Oasp4NetCore.sln

Components

The application is going to be divided in different components to encapsulate the different domains of the application functionalities.



As *main components* we will find:

- *_BookingService*: Manages the bookings part of the application. With this component the users (anonymous/logged in) can create new bookings or cancel an existing booking. The users with waiter role can see all scheduled bookings.
- *OrderService*: This component handles the process to order dishes (related to bookings). A user (as a host or as a guest) can create orders (that contain dishes) or cancel an existing one. The users with waiter role can see all ordered orders.
- *DishService*: This component groups the logic related to the menu (dishes) view. Its main feature is to provide the client with the data of the available dishes but also can be used by other components (Ordermanagement) as a data provider in some processes.
- *UserService*: Takes care of the User Profile management, allowing to create and update the data profiles.

As *common components* (that don't exactly represent an application's area but provide functionalities that can be used by the *main components*):

- *Mailservice*: with this service we will provide the functionality for sending email notifications. This is a shared service between different app components such as *bookingmanagement* or *ordercomponent*.

Other components:

- Security (will manage the access to the *private* part of the application using a [jwt](#) implementation).

- Twitter integration: planned as a *Microservice* will provide the twitter integration needed for some specific functionalities of the application.

Layers

Introduction

The .Net Core backend for My Thai Star application is going to be based on:

- **OASP4NET** as the .Net Core framework
- **VSCode** as the Development environment
- **TOBAGO** as code generation tool

Application layer

This layer will expose the REST api to exchange information with the client applications.

The application will expose the services on port 8081 and it can be launched as a self host console application (microservice approach) and as a Web Api application hosted on IIS/IIS Express.

Business layer

This layer will define the controllers which will be used on the application layer to expose the different services. Also, will define the swagger contract making use of summary comments and framework attributes.

This layer also includes the object response classes in order to interact with external clients.

Service layer

The layer in charge of hosting the business logic of the application. Also orchestrates the object conversion between object response and entity objects defined in *Data layer*.

Data layer

The layer to communicate with the data base.

Data layer makes use of *Entity Framework*. The Database context is defined on *DataAccessLayer* assembly (ModelContext).

This layer makes use of the *Repository pattern* and *Unit of work* in order to encapsulate the complexity. Making use of this combined patterns we ensure an organized and easy work model.

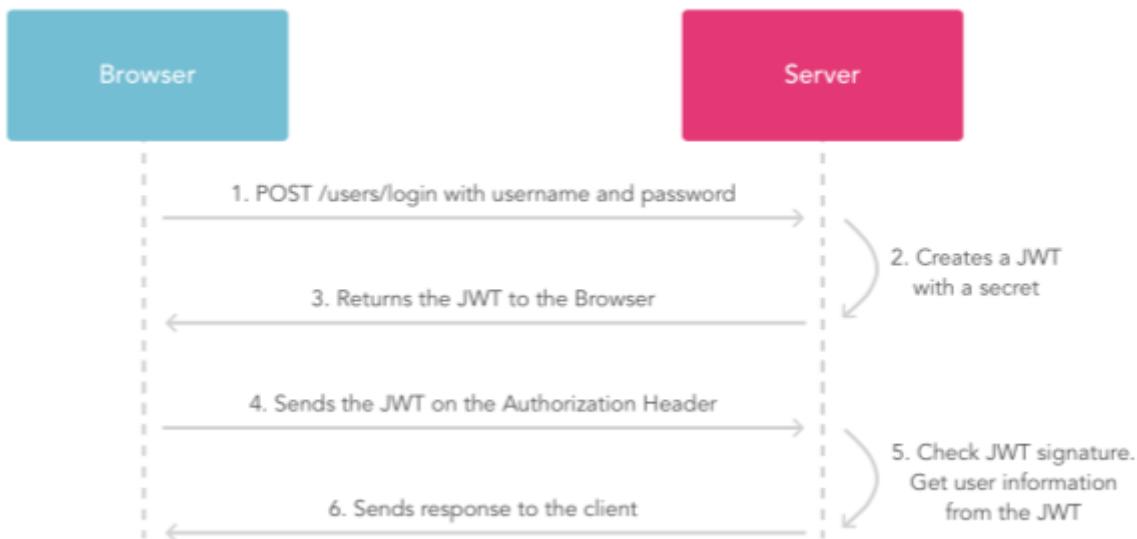
As in the previous layers, the *data access* layer will have both *interface* and *implementation* tiers. However, in this case, the implementation will be slightly different due to the use of *generics*.

Cross-Cutting concerns

the layer to make use of transversal components such JWT and mailing.

Jwt basics

- A user will provide a username / password combination to our auth server.
- The auth server will try to identify the user and, if the credentials match, will issue a token.
- The user will send the token as the *Authorization* header to access resources on server protected by JWT Authentication.



Jwt implementation details

The *Json Web Token* pattern will be implemented based on the [jwt on .net core](#) framework that is provided by default in the *Oasp4Net* projects.

Authentication

Based on *Microsoft* approach, we will implement a class to define the security *entry point* and filters. Also, as *My Thai Star* is a mainly *public* application, we will define here the resources that won't be secured.

On *Oasp4Net.Infrastructure.JWT* assembly is defined a subset of *Microsoft's authorization schema*. Database. It is started up the first time the application launches.

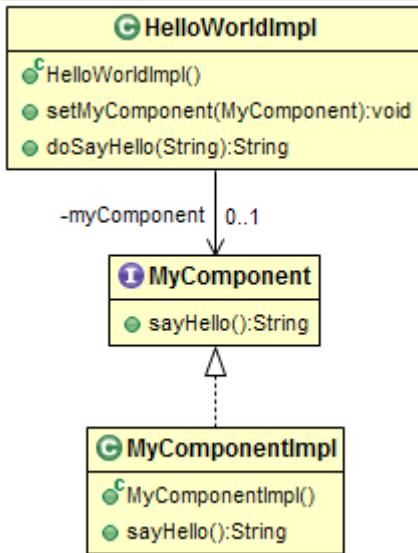
You can read more about *_Authorization* on:

[Authorization in ASP.NET Core](#)

[Claim based authorization](#)

Dependency injection

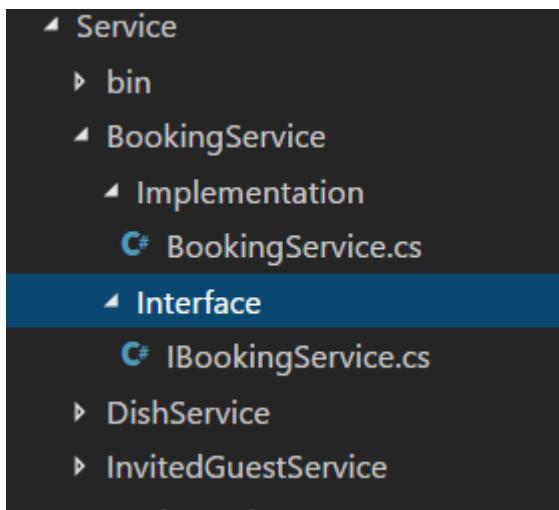
As it is explained in the [Microsoft documentation](#) we are going to implement the *dependency injection* pattern basing our solution on *.Net Core*.



- Separation of API and implementation: Inside each layer we will separate the elements in different tiers: *interface* and *implementation*. The *interface* tier will store the *interface* with the methods definition and inside the *implementation* we will store the class that implements the *interface*.

Layer communication method

The connection between layers, to access to the functionalities of each one, will be solved using the *dependency injection*.



Connection BookingService - Logic

```

public class BookingService : EntityService<Booking>, IBookingService
{
    private readonly IBookingRepository _bookingRepository;
    private readonly IRepository<Order> _orderRepository;
    private readonly IRepository<InvitedGuest> _invitedGuestRepository;
    private readonly IOrderLineRepository _orderLineRepository;
    private readonly IUnitOfWork _unitOfWork;

    public BookingService(IUnitOfWork unitOfWork,
        IBookingRepository repository,
        IRepository<Order> orderRepository,
        IRepository<InvitedGuest> invitedGuestRepository,
        IOrderLineRepository orderLineRepository) : base(unitOfWork, repository)
    {
        _unitOfWork = unitOfWork;
        _bookingRepository = repository;
        _orderRepository = orderRepository;
        _invitedGuestRepository = invitedGuestRepository;
        _orderLineRepository = orderLineRepository;
    }
}

```

To give service to the defined *User Stories* we will need to implement the following services:

- provide all available dishes.
- save a booking.
- save an order.
- provide a list of bookings (only for waiters) and allow filtering.
- provide a list of orders (only for waiters) and allow filtering.
- login service (see the *Security* section).
- provide the *current user* data (see the *Security* section)

Following the [naming conventions] proposed for *Oasp4Net* applications we will define the following *end points* for the listed services.

- (POST) [/mythaistar/services/rest/dishmanagement/v1/dish/search](#).
- (POST) [/mythaistar/services/rest/bookingmanagement/v1/booking](#).
- (POST) [/mythaistar/services/rest/ordermanagement/v1/order](#).
- (POST) [/mythaistar/services/rest/bookingmanagement/v1/booking/search](#).
- (POST) [/mythaistar/services/rest/ordermanagement/v1/order/search](#).
- (POST) [/mythaistar/services/rest/ordermanagement/v1/order/filter](#) (to filter with fields that does not belong to the Order entity).
- (POST) [/mythaistar/login](#).
- (GET) [/mythaistar/services/rest/security/v1/currentuser/](#).

You can find all the details for the services implementation in the [Swagger definition](#) included in the My Thai Star project on Github.

Api Exposed

The *Oasp4Net.Business.Controller* assembly in the *business* layer of a *component* will store the definition of the service by a *interface*. In this definition of the service we will set-up the *endpoints* of the service, the type of data expected and returned, the *HTTP* method for each endpoint of the service and other configurations if needed.

```

    /// <summary>
    /// Method to make a reservation with potential guests. The method returns the
    reservation token with the format:
    {(CB_|GB_)}{now.Year}{now.Month:00}{now.Day:00}{_}{MD5({Host/Guest-
    email}{now.Year}{now.Month:00}{now.Day:00}{now.Hour:00}{now.Minute:00}{now.Second:00})
}
    /// </summary>

    /// <param name="bookingView"></param>
    /// <response code="201">Ok.</response>
    /// <response code="400">Bad request. Parser data error.</response>
    /// <response code="401">Unauthorized. Authentication fail.</response>
    /// <response code="403">Forbidden. Authorization error.</response>
    /// <response code="500">Internal Server Error. The search process ended with
    error.</response>
    [HttpPost]
    [HttpOptions]
    [Route("/mythaistar/services/rest/bookingmanagement/v1/booking")]
    [AllowAnonymous]
    [EnableCors("CorsPolicy")]
    public IActionResult BookingBooking([FromBody]BookingView bookingView)
    {
        ...
    }
}

```

Using the summary annotations and attributes will tell to swagger the contract via the XML doc generated on compiling time. This doc will be stored in *XmlDocumentation* folder.

The Api methods will be exposed on the application layer.

30.1.2. Google Mail API Consumer

[= fa floppy o] [Google Mail API Consumer](#)

Application	MyThaiStarEmailService.exe
Config file	MyThaiStarEmailService.exe.Config
Default port	8080

Overview

1. Execute MyThaiStarEmailService.exe.
2. The first time google will ask you for credentials (just one time) in your default browser:
 - Account: mythaistarrestaurant@gmail.com
 - Password: mythaistarrestaurant2501
3. Visit the url: <http://localhost:8080/swagger>
4. Your server is ready!



My Thai Star Email Service

Email Show/Hide | List Operations | Expand Operations

POST /api>Email

[BASE URL: , API VERSION: v1]

Figure 85. GMail Server Swagger contract page

JSON Example

This is the JSON example to test with swagger client. Please read the swagger documentation.

```
{
  "EmailFrom": "mythaistarrestaurant@gmail.com",
  "EmailAndTokenTo": {
    "MD5Token1": "Email_Here!@gmail.com",
    "MD5Token2": "Email_Here!@gmail.com"
  },
  "EmailType": 0,
  "DetailMenu": [
    "Thai Spicy Basil Fried Rice x2",
    "Thai green chicken curry x2"
  ],
  "BookingDate": "2017-05-31T12:53:39.7864723+02:00",
  "Assistants": 2,
  "BookingToken": "MD5Booking",
  "Price": 20.0,
  "ButtonActionList": {
    "http://accept.url": "Accept",
    "http://cancel.url": "Cancel"
  },
  "Host": {
    "Email_Here!@gmail.com": "José Manuel"
  }
}
```

Configure the service port

If you want to change the default port, please edit the config file and change the next entry in appSettings node:

```
<appSettings>
  <add key="LocalListenPort" value="8080" />
</appSettings>
```

External links

[Google API Account Configuration](#)

[About Scopes](#)

30.1.3. Downloads

[= fa floppy o] [My Thai Star \(.Net Core Server + Angular client\)](#)

[= fa floppy o] [Google Mail API Consumer](#)

Part VI: devon4node

devonfw is a platform which provides solutions to building business applications which combine best-in-class frameworks and libraries as well as industry proven practices and code conventions. devonfw is 100% Open Source (Apache License version 2.0) since the beginning of 2018.

devon4node is the NodeJS stack of devonfw. It allows you to build business applications (backends) using NodeJS technology in standardized way based on established best-practices.

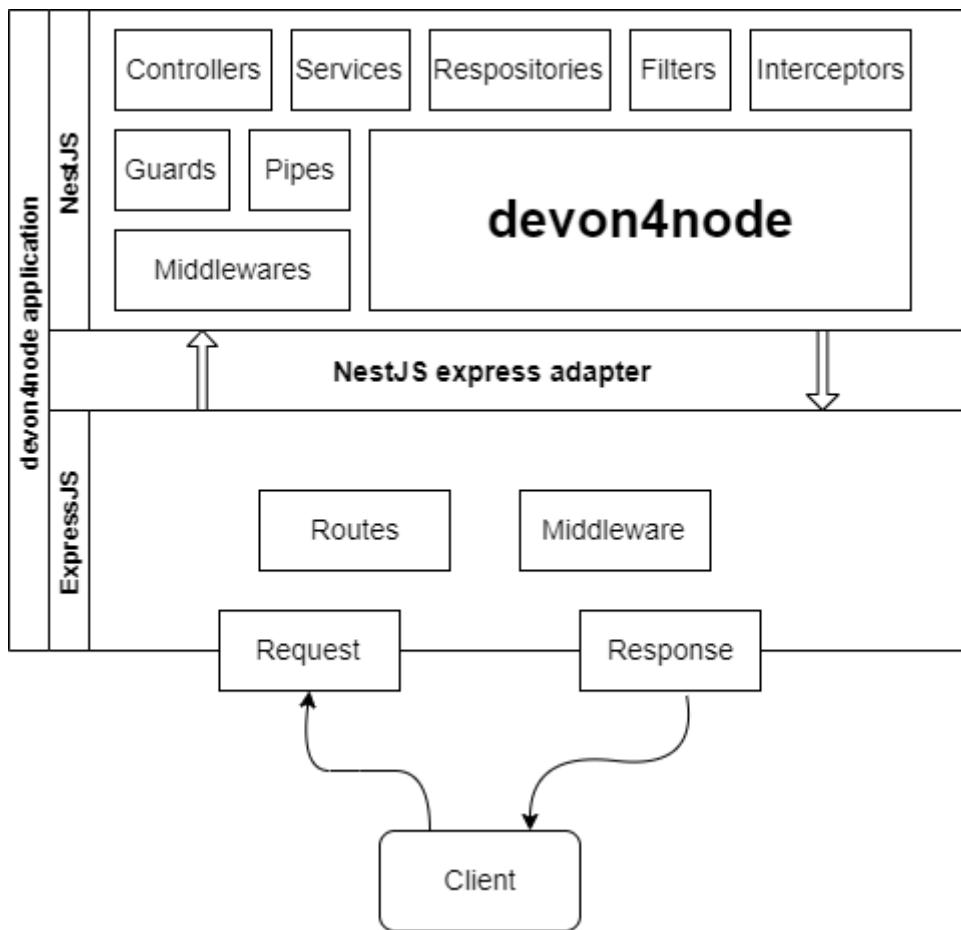
devon4node is based on [NestJS](#). Nest (NestJS) is a framework for building efficient, scalable Node.js server-side applications. It uses progressive TypeScript and combines elements of OOP (Object Oriented Programming), FP (Functional Programming), and FRP (Functional Reactive Programming).

31. devon4node Architecture

As we mention in the introduction, devon4node is based on [NestJS](#). Nest (NestJS) is a framework for building efficient, scalable Node.js server-side applications.

31.1. HTTP layer

By using [NestJS](#), devon4node is a platform-agnostic framework. NestJS focuses only on the logical layer, and delegates the transport layer to another framework, such as ExpressJS. You can see it in the following diagram:



As you can see, NestJS does not listen directly for incoming requests. It has an adapter to communicate with ExpressJS and ExpressJS is the responsible for that. ExpressJS is only one of the frameworks that NestJS can work with. We have also another adapter available out-of-the-box: the [Fastify](#) adapter. With that, you can replace ExpressJS for Fastify. But you can still use all your NestJS components. You can also create your own adapter to make NestJS work with other HTTP frameworks.

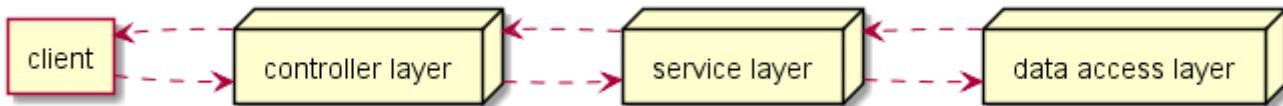
At this point, you may think: why is NestJS (and devon4node) using ExpressJS by default instead of Fastify? Because, as you can see in the previous diagram, there is a component that is dependent on the HTTP framework: the middleware. As ExpressJS is the most widely used framework, there exists a lot of middleware for it, so, in order to reuse them in our NestJS applications, NestJS uses ExpressJS by default. Anyway, you may think which HTTP framework best fits your requirements.

31.2. devon4node layers

As other devonfw technologies, devon4node separates the application into layers.

Those layers are:

- [Controller layer](#)
- [Service layer](#)
- [Data Access layer](#)



31.3. devon4node application structure

Although there are many frameworks to create backend applications in NodeJS, none of them effectively solve the main problem of - Architecture. This is the main reason we have chosen NestJS for the devon4node applications. Besides, NestJS is highly inspired by [Angular](#), therefore a developer who knows Angular can use his already acquired knowledge to write devon4node applications.

NestJS adopts various Angular concepts, such as dependency injection, piping, interceptors and modularity, among others. By using modularity we can reuse some of our modules between applications. One example that devon4node provide is the [mailer module](#).

31.3.1. Modules

Create a application module is simple, you only need to create an empty class with the decorator [Module](#):

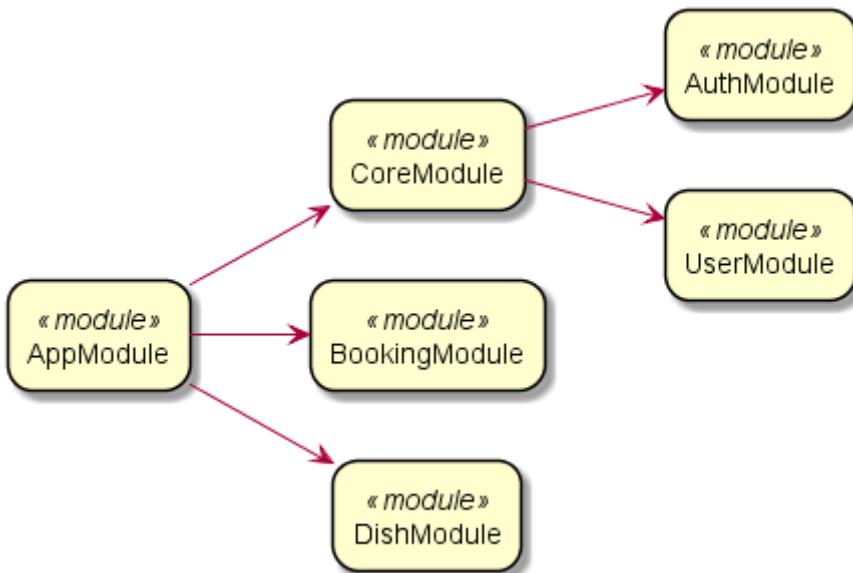
```
@Module({})
export class AppModule {}
```

In the module you can define:

- Imports: the list of imported modules that export the providers which are required in this module
- Controllers: the set of controllers defined in this module which have to be instantiated
- Providers: the providers that will be instantiated by the Nest injector and that may be shared at least across this module
- Exports: the subset of providers that are provided by this module and should be available in other modules which import this module

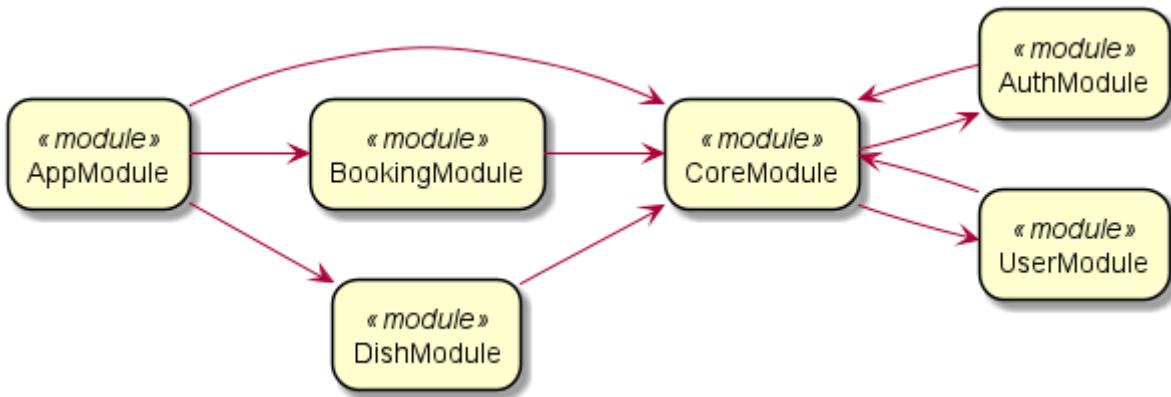
The main difference between Angular and NestJS is NestJS modules encapsulates providers by default. This means that it's impossible to inject providers that are neither directly part of the

current module nor exported from the imported modules. Thus, you may consider the exported providers from a module as the module's public interface, or API. Example of modules graph:



In devon4node we have three different kind of modules:

- **AppModule:** this is the root module. Everything that our application need must be imported here.
- **Global Modules:** this is a special kind of modules. When you make a module global, it's accessible for every module in your application. Your can see it in the next diagram. It's the same as the previous one, but now the CoreModule is global:



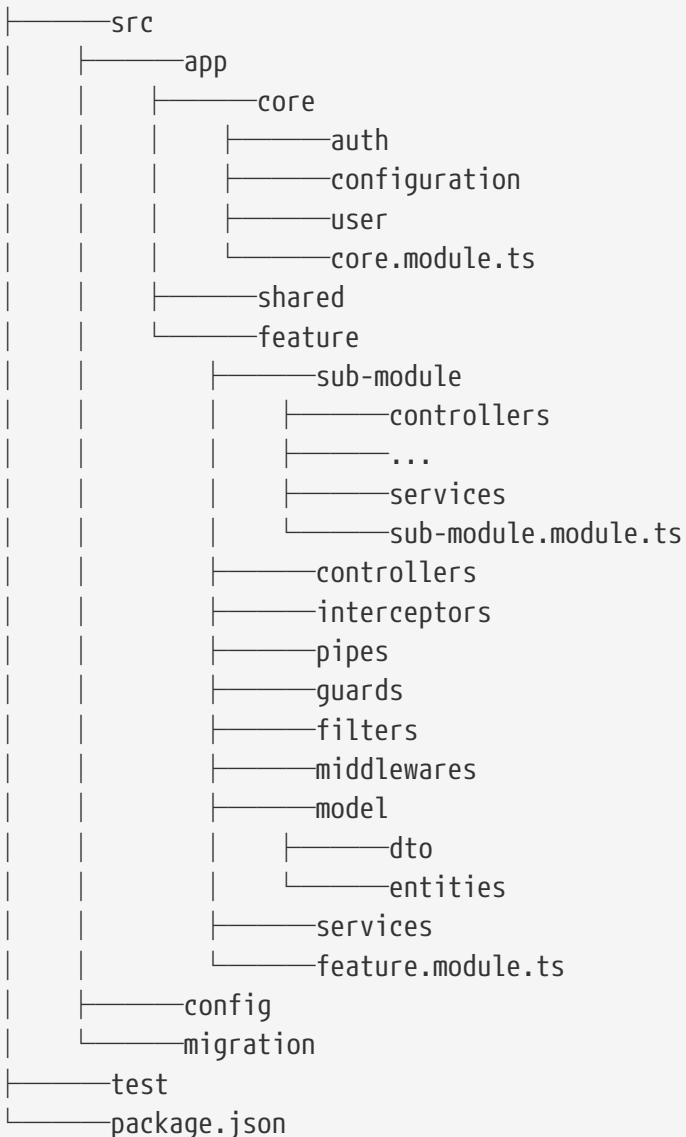
One example of global module is the CoreModule. In the CoreModule you must import every module which have providers that needs to be accessible in all modules of your application

- **Feature (or application) modules:** modules which contains the logic of our application. We must import it in the AppModule.

For more information about modules, see [NestJS documentation page](#)

31.3.2. Folder structure

devon4node defines a folder structure that every devon4node application must follow. The folder structure is:



[devon4node schematics](#) ensures this folder structure so, please, do not create files by your own, use the [devon4node schematics](#).

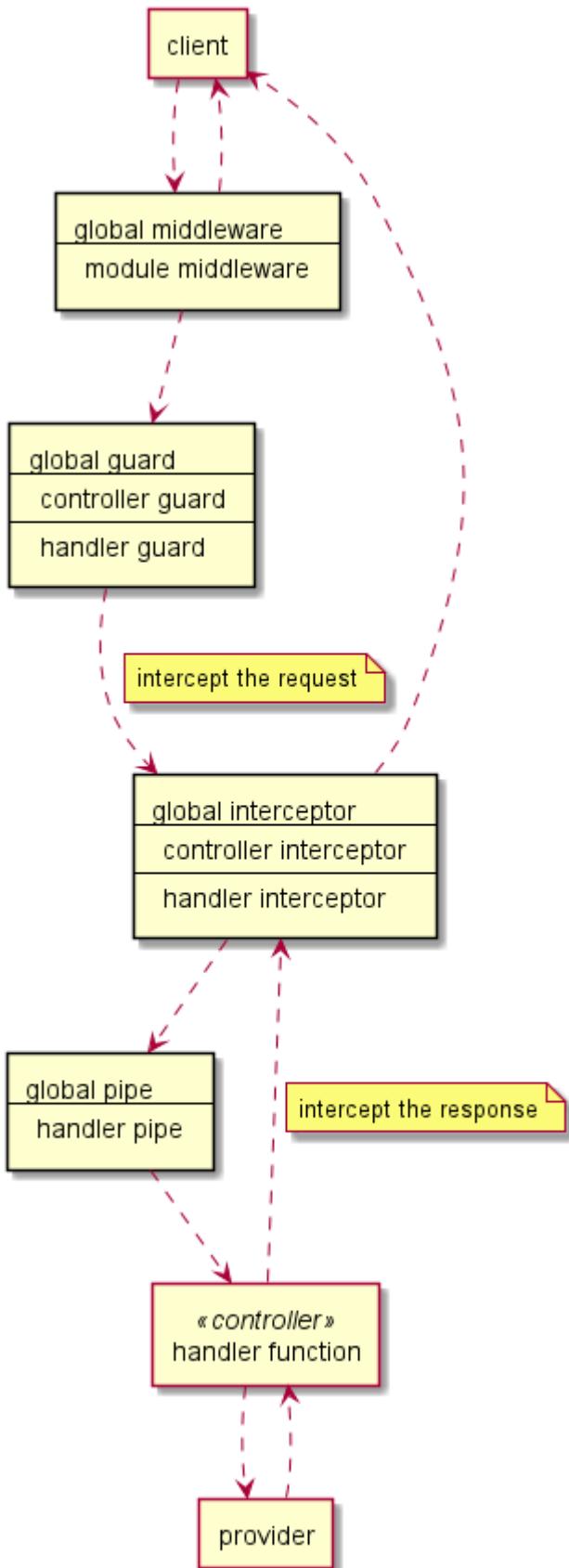
31.3.3. NestJS components

NestJS provides several components that you can use in your application:

- [Controllers](#)
- [Providers](#)
- [Middleware](#)
- [Guards](#)
- [Interceptors](#)
- [Pipes](#)
- [Exception filters](#)

In the [NestJS documentation](#) you can find all information about each component. But, something that is missing in the documentation is the execution order. Every component can be defined in

different levels: globally, in the controller or in the handler. As middleware is part of the HTTP server we can define it in a different way: globally or in the module.



It is not necessary to have defined components in every level. For example, you can have defined a interceptor globally but you do not have any other in the controller or handler level. If nothing is defined in some level, the request will continue to the next component.

As you can see in the previous image, the first component which receive the request is the global defined middleware. Then, it send the request to the module middleware. Each of them can return a response to the client, without passing the request to the next level.

Then, the request continue to the guards: first the global guard, next to controller guard and finally to the handler guard. At this point, we can throw an exception in all components and the exception filter will catch it and send a proper error message to the client. We do not paint the filters in the graphic in order to simplify it.

After the guards, is time to interceptors: global interceptors, controller interceptors and handler interceptors. And last, before arrive to the handler inside the controller, the request pass through the pipes.

When the handler has the response ready to send to the client, it does not go directly to the client. It come again to the interceptors, so we can also intercept the response. The order this time is the reverse: handler interceptors, controller interceptors and global interceptors. After that, we can finally send the response to the client.

Now, with this in mind, you are able to create the components in a better way.

32. Layers

32.1. Controller Layer

The controller layer is responsible for handling the requests/responses to the client. This layer knows everything about the endpoints exposed, the expected input (and also [validate](#) it), the response schema, the HTTP codes for the response and the HTTP errors that every endpoint can send.

32.1.1. How to implement the controller layer

This layer is implemented by the [NestJS controllers](#). Let's see how it works with an example:

```
@Controller('coffee/coffees')
export class CoffeeController {
    constructor(private readonly coffeeService: CoffeeService) {}

    @Post('search')
    @HttpCode(200)
    async searchCoffees(@Body() search: CoffeeSearch): Promise<Array<Coffee>> {
        try {
            return await this.coffeeService.searchCoffees(search);
        } catch (error) {
            throw new BadRequestException(error.message, error);
        }
    }
}
```

As you can see in the example, to create a controller you only need to decorate a class with the [Controller](#) decorator. This example is handling all request to [coffee/coffees](#).

Also, you have defined one handler. This handler is listening to POST request for the route [coffee/coffees/search](#). In addition, this handler is waiting for a CoffeeSearch object and returns an array of Coffee. In order to keep it simple, that's all that you need in order to define one route.

One important thing that can be observed in this example is that there is no business logic. It delegates to the service layer and return the response to the client. At this point, transformations from the value that you receive from the service layer to the desired return type are also allowed.

By default, every POST handler return an HTTP 204 response with the returned value as body, but you can change it in a easy way by using decorators. As you can see in the example, the handler will return a HTTP 200 response ([@HttpCode\(200\)](#)).

Finally, if the service layer throws an error, this handler will catch it and return a HTTP 400 Bad Request response. The controller layer is the only one that knows about the answers to the client, therefore it is the only one that knows which error codes should be sent.

32.1.2. Validation

In order to do not propagate errors in the incoming payload, we need to validate all data in the controller layer. See the [validation guide](#) for more information.

32.1.3. Error handling

In the previous example, we catch all errors using the try/catch statement. This is not the usual implementation. In order to catch properly the errors you must use the [exception filters](#). Example:

```
@Controller('coffee/coffees')
export class CoffeeController {
    constructor(private readonly coffeeService: CoffeeService) {}

    @Post('search')
    @HttpCode(200)
    @UseFilters(CaffeExceptionFilter)
    async searchCoffees(@Body() search: CoffeeSearch): Promise<Array<Coffee>> {
        return await this.coffeeService.searchCoffees(search);
    }
}
```

32.2. Service Layer

The logic layer is the heart of the application and contains the main business logic. It knows everything about the business logic, but it does not know about the response to the client and the HTTP errors. That's why this layer is separated from the controller layer.

32.2.1. How to implement the service layer

This layer is implemented by services, a specific kind of [providers](#). Let's see one example:

```
@Injectable()
export class CoffeeService {
    constructor(private readonly coffeeService: CoffeeService) {}

    async searchCoffees(@InjectRepository(Coffee) coffeeRepository: Repository<Coffee>): Promise<Array<Coffee>> {
        const coffees = this.coffeeRepository.find();

        return doSomeBusinessLogic(coffees);
    }
}
```

This is the CoffeeService that we inject in the example of [controller layer](#). As you can see, a service is a regular class with the [Injectable](#) decorator. Also, it inject as dependency the data access layer (in this specific case, the `Repository<Coffee>`).

The services expose methods in order to transform the input from the controllers by applying some business logic. They can also request data from the data access layer. And that's all.

32.3. Data Access Layer

The data access layer is responsible for all outgoing connections to access and process data. This is mainly about accessing data from a persistent data-store but also about invoking external services.

This layer is implemented using providers. Those providers could be: services, repositories and others. Although services can be used for this layer, they should not be confused with the service layer. Services in this layer are responsible for data access, while services in the service layer are responsible for business logic.

32.3.1. Database

We strongly recommend [TypeORM](#) for database management in devon4node applications. Although services can be used for this layer, they should not be confused with the service layer. Services in this layer are responsible for data access, while services in the service layer are responsible for business logic. TypeORM supports the most commonly used relational databases, like Oracle, MySQL, MariaDB, PostgreSQL, SQLite, MSSQL and others. Also, it supports no-relational databases like MongoDB.

TypeORM supports [Active Record](#) and Repository patterns. We recommend to use the Repository pattern. This pattern allows you to separate the data objects from the methods to manipulate the database.

32.3.2. External APIs

In order to manage the data in a external API, you need to create a service for that purpose. In order to manage the connections with the external API, we strongly recommend the [NestJS HTTP module](#)

33. Guides

33.1. Key Principles

devon4node is built following some basic principles like:

- [SOLID](#)
- [Patterns](#)
- [Open](#)

But key principles that best define devon4node (and are inherited from NestJS) are:

- Simplicity (aka [KISS](#))
- Reusability
- Productivity

33.1.1. Simplicity

In devon4node we tried to do everything as simple as possible. Following this principle we will be able to do easy to maintain applications.

For example, in order to expose all CRUD operations for an entity, you only need to create a controller like:

```
@Crud({
  model: {
    type: Employee,
  },
})
@CrudType(Employee)
@Controller('employee/employees')
export class EmployeeCrudController {
  constructor(public service: EmployeeCrudService) {}
}
```

You can find this code in the [employee example](#). Only with this code your exposing the full CRUD operations for the employee entity. As you can see, it's an empty class with some decorators and the [EmployeeCrudService](#) injected as dependency. Simple, isn't it? The [EmployeeCrudService](#) is also simple:

```
@Injectable()
export class EmployeeCrudService extends TypeOrmCrudService<Employee> {
  constructor(@InjectRepository(Employee) repo: Repository<Employee>) {
    super(repo);
  }
}
```

Another empty class which extends from `TypeOrmCrudService<Employee>` and injects the Employee Repository as dependency. Nothing else.

With these examples you can get an idea of how simple it can be to code a devon4node application .

33.1.2. Reusability

NestJS (and devon4node) applications are designed in a modular way. This allows you to isolate some functionality in a module, and then reuse it in every application that you need. This is the same behaviour that Angular has. You can see it in the NestJS modules like [TypeORM](#), [Swagger](#) and others. Also, in devon4node we have the [Mailer module](#).

In your applications, you only need to import those modules and then you will be able to use the functionality that they implement. Example

```
@Module({
  imports: [ AuthModule, ConfigurationModule ],
})
export class SomeModule {}
```

33.1.3. Productivity

devon4node is designed to create secure enterprise applications. But also, it allow you to do it in a fast way. To increase the productivity devon4node, devon4node provide schematics in order to generate some boilerplate code.

For example, to create a module you need to create a new file for a module (or copy it) and write the code, then you need to import it in the AppModule. This is a easy example, but you can introduce some errors: forget to import it in the AppModule, introduce errors with the copy/paste and so on. By using the command `nest g module --name <module-name>` it will do everything for you. Just a simple command. In this specific case probably you do not see any advantage, but there are other complex cases where you can generate more complex code with nest and devon4node schematics command.

See [code generation](#) in order to know how to increase your productivity creating devon4node applications.

33.2. Code Generation

As we mention in the page [key principles](#), one of our key principles is Productivity. In order to provide that productivity, we have some tools to generate code. These tools will help you generate the common parts of the application so that you can focus only on the specific functionality.

Those tools are:

- [devon4node schematics through Nest CLI](#)
- [CobiGen](#)

33.2.1. Nest CLI and Devon4node schematics

We are going to use the Nest CLI to generate code of our application, you can know more about NodeJs CLI in the official [documentation](#).

Install devon4node schematics

First of all, you need to install Nest CLI

Execute the command `yarn global add @nestjs/cli`. You can also use npm: `npm install -g @nestjs/cli`

And then Devon4node schematics globally with the following command:

`yarn global add @devon4node/schematics` or `npm install -g @devon4node/schematics`



If you get an error trying execute any devon4node schematic related to collection not found, try to reinstall devon4node/schematics on the project folder or be sure that schematics folder is inside `@devon4node` in `node_modules`. `yarn add @devon4node/schematics`

Generate new devon4node application

To start creating a devon4node application, execute the command:

`nest g -c @devon4node/schematics application [application-name]`

If you do not put a name, the command line will ask you for one.

Generate code for Typeorm

Initialize typeorm into your current project in a correct way.

`nest g -c @devon4node/schematics typeorm`

Then, you will be asked about which DB you want to use.

```
$ nest g -c @devon4node/schematics typeorm
? What kind of database do you want to use? (Use arrow keys)
> postgres
cockroachdb
mariadb
mysql
sqlite
oracle
mssql
(Move up and down to reveal more choices)
```

Generate CRUD

Generate CRUD methods for a entity. Requires TypeORM installed in the project.

It will add the `@nestjs/crud` module as a project dependency. Then, generates an entity, a CRUD controller and a CRUD service. It also register the entity, controller and service in the module.

Execute `nest g -c @devon4node/schematics crud` and then you will need to write a name for the crud.

```
$ nest g -c @devon4node/schematics crud
? Introduce the crud name todo
CREATE src/app/model/entities/todo.entity.ts (165 bytes)
CREATE src/app/controllers/todo.crud.controller.ts (427 bytes)
CREATE src/app/services/todo.crud.service.ts (415 bytes)
UPDATE src/app/app.module.ts (618 bytes)
UPDATE package.json (2407 bytes)
```

Generate Typeorm entity

Add a TypeOrm entity to your project. Requires TypeORM installed in the project.

Execute `nest g -c @devon4node/schematics entity` and you will be asked for an entity name.

Add config-module

Add the config module to the project.

It will add the `@devon4node/common` module as a project dependency. Then, it will generate the configuration module into your project and add it in the core module. Also, it generates the config files for the most common environments.

The command to execute will be `nest g -c @devon4node/schematics config-module`

Add mailer module

Add `@devon4node/mailer` module to project.

It will add the `@devon4node/mailer` module as a project dependency. Also, it will add it to the core module and it will generate some email template examples.

Write the command `nest g -c @devon4node/schematics mailer`

Add swagger module

Add swagger module to project.

It will add the `@nestjs/swagger` module as a project dependency. Also, it will update the `main.ts` file in order to expose the endpoint for swagger. The default endpoint is: `/v1/api`

Execute the command `nest g -c @devon4node/schematics swagger`

Add auth-jwt module

Add the auth JWT module to the project.

It will add to your project the auth-jwt and user module. Also, it will import those modules into the core module.

Execute `nest g -c @devon4node/schematics auth-jwt`

Add security

Add cors and helmet to your project.

It will add helmet package as project dependency and update the main.ts file in order to enable the cors and helmet in your application.

Execute `nest g -c @devon4node/schematics security`

Generate database migrations

1. Generate database migrations

- a. In order to create migration scripts with typeorm, you need to install ts-node: `yarn global add ts-node` or `npm i -g ts-node`
- b. Generate the tables creation migration: `yarn run typeorm migration:generate -n CreateTables`

```
PS C:\devonfw-ide-scripts-2020.08.001\workspaces\employee> yarn run typeorm migration:generate -n InsertData
yarn run v1.22.4
$ node --require ts-node/register ./node_modules/typeorm/cli.js migration:generate -n InsertData
Migration C:\devonfw-ide-scripts-2020.08.001\workspaces\employee\src\migration\1604057459677-InsertData.ts has been generated successfully.
Done in 25.22s.
```

It will connect to the database, read all entities and then it will generate a migration file with all sql queries need to transform the current status of the database to the status defined by the entities. If the database is empty, it will generate all sql queries need to create all tables defined in the entities. You can find a example in the todo example

As typeorm is the tool used for DB. You can check official documentation for more information. See [typeorm CLI documentation](#).

33.2.2. CobiGen

Currently, we do not have templates to generate devon4node code (we have planned to do that in the future). Instead, we have templates that read the code of a devon4node application and generate a devon4ng application. Visit the [CobiGen](#) page for more information.

33.3. Coding Conventions

devon4node defines some coding conventions in order to improve the readability, reduce the merge conflicts and be able to develop applications in an industrialized way.

In order to ensure that you are following the devon4node coding conventions, you can use the

following tools:

- **ESLint**: ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code, with the goal of making code more consistent and avoiding bugs. We recommend to use the [ESLint VSCode extension](#) (included in the devonfw Platform Extension Pack) in order to be able to see the linting errors while you are developing.
- **Prettier**: Prettier is a code formatter. We recommend to use the Prettier VSCode extension (included in the devonfw Platform Extension Pack) and enable the `editor.formatOnSave` option.
- **devon4node application schematic**: this tool will generate code following the devon4node coding conventions. Also, when you generate a new project using the devon4node application schematic, it generates the configuration files for TSLint and Prettier that satisfy the devon4node coding conventions.

When you combine all tools, you can be sure that you follow the devon4node coding conventions.

33.3.1. Detailed devon4node Coding Conventions

Here we will detail some of most important devon4node coding conventions. To be sure that you follows all devon4node coding conventions use the tools described before.

Indentation

All devon4node code files must be indented using spaces. The indentation width must be 2 spaces.

White space

In order to improve the readability of your code, you must introduce whitespaces. Example:

```
if(condition){
```

must be

```
if (condition) {
```

Naming conventions

File naming

The file name must follow the pattern: (name in kebab case).(kind of component).(extension) The test file name must follow the pattern: (name in kebab case).(kind of component).spec.(extension)

Example:

```
auth-jwt.service.ts  
auth-jwt.service.spec.ts
```

Interface naming

The interface names must be in pascal case, and must start with I. There is some controversy in starting the interface names with an I, but we decided to do it because in most of cases you will have an interface and a class with the same name, so, to differentiate them, we decided to start the interfaces with I. Other devonfw stacks solves it by adding the suffix `Impl` in the class implementations.

Example:

```
interface ICoffee {}
```

Class naming

The class names must be in pascal case.

Example:

```
class Coffee {}
```

Variable naming

All variable names must be in camel case.

```
const coffeeList: Coffe[];
```

Declarations

For all variable declarations we must use `const` or `let`. `var` is forbidden. We prefer to use `const` when possible.

Programming practices

Trailing comma

All statements must end with a trailing comma. Example:

```
{
  one: 'one',
  two: 'two' // bad
}
{
  one: 'one',
  two: 'two', // good
}
```

Arrow functions

All anonymous functions must be defined with the arrow function notation. In most of cases it's not a problem, but sometimes, when you do not want to bind `this` when you define the function, you can use the other function definition. In this special cases you must disable the linter for those sentence.

Comments

Comments must start with a whitespace. Example:

```
//This is a bad comment
// This is OK
```

Quotemarks

For string definitions, we must use single quotes.

if statements

In all if statements you always must use brackets. Example:

```
// Bad if statement
if (condition)
    return true;

// Good if statement
if (condition) {
    return true;
}
```

33.3.2. Pre-commit hooks

In order to ensure that your new code follows the coding conventions, devon4node uses by default husky. Husky is a tool that allows you to configure git hooks easily in your project. When you make a `git commit` in your devon4node project, it will execute two actions:

- Prettify the staged files
- Execute the linter in the staged files

If any action fails, you won't be able to commit your new changes.



If you want to skip the git hooks, you can do a commit passing the `--no-verify` flag.

33.4. Dependency Injection

The [dependency injection](#) is a well-known common design pattern applied by frameworks in all languages, like [Spring](#) in Java, [Angular](#) and others. The intention of this page is not to explain how

dependency injection works, but instead how it is addressed by NestJS.

NestJS resolve the dependency injection in their modules. When you define a provider in a module, it can be injected in all components of the module. By default, those providers are only available in the module where it is defined. The only way to export a module provider to other modules which import it is adding those provider to the export array. You can also reexport modules.

33.4.1. Inject dependencies in NestJS

In order to inject a dependency in a NestJS component, you need to declare it in the component constructor. Example:

```
export class CoffeeController {  
    constructor(public readonly coffeeService: CoffeeService) {}  
}
```

NestJS can resolve all dependencies that are defined in the module as provider, and also the dependencies exported by the modules imported. Example:

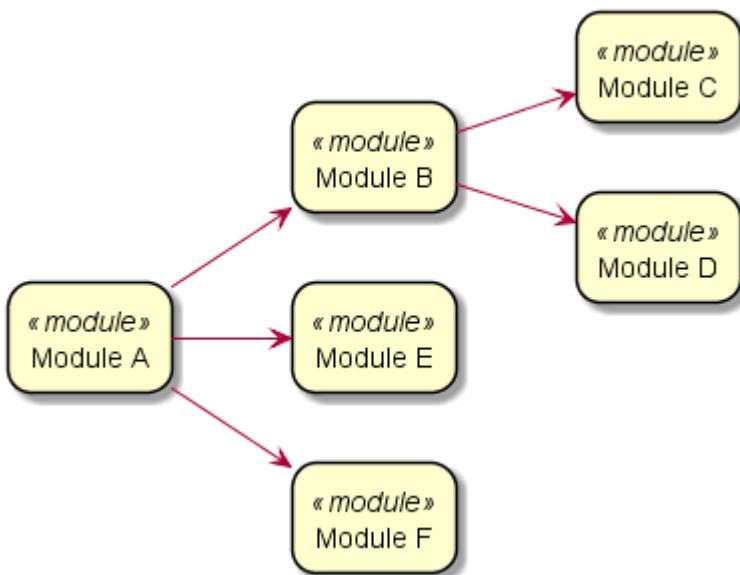
```
@Module({  
    controllers: [CoffeeController],  
    providers: [CoffeeService],  
})  
export class CoffeeModule {}
```

Inject dependencies in the constructor is the preferred choice, but, sometimes it is not possible. For example, when you are extending another class and want to keep the constructor definition. In this specific cases we can inject dependencies in the class properties. Example:

```
export class CoffeeController {  
    @Inject(CoffeeService)  
    private readonly coffeeService: CoffeeService;  
}
```

33.4.2. Dependency Graph

Dependency Injection

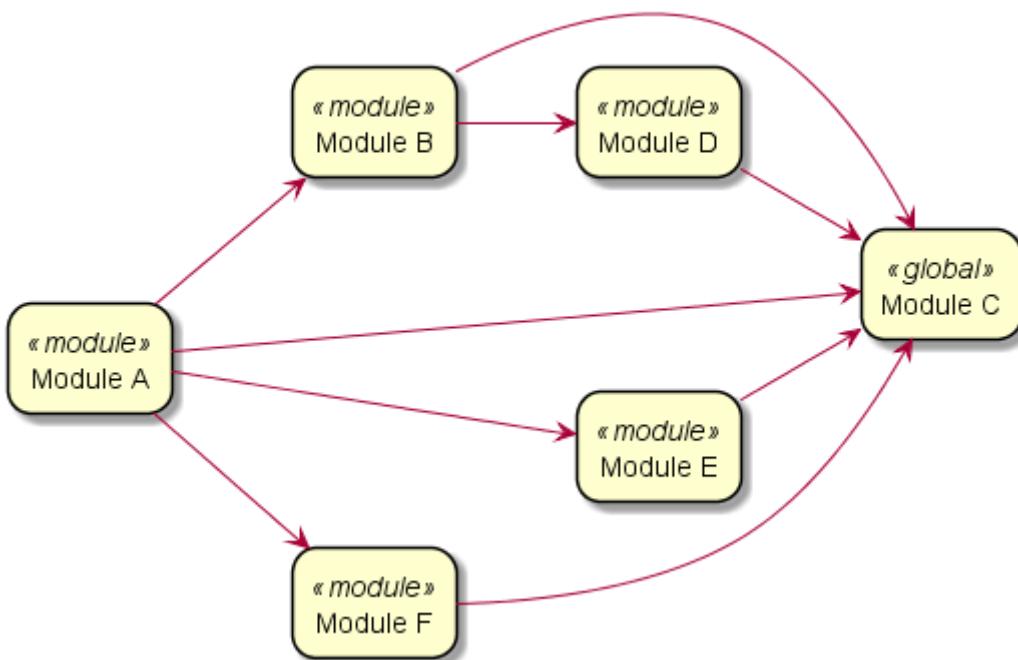


In the previous image, the Module A can inject dependencies exported by Module B, Module E and Module F. If module B reexport Module C and Module D, they are also accesible by Module A.

If there is a conflict with the injection token, it resolves the provider with less distance with the module. For example: if the modules C and F exports a `UserService` provider, the Module A will resolve the `UserService` exported by the Module F, because the distance from Module A to Module F is 1, and the distance from Module A to Module C is 2.

When you define a module as global, the dependency injection system is the same. The only difference is now all modules as a link to the global module. For example, if we make the Module C as global the dependency graph will be:

Dependency Injection with global module



33.4.3. Custom providers

When you want to change the provider name, you can use a NestJS feature called [custom providers](#). For example, if you want to define a provider called `MockUserService` with the provider token `UserService` you can define it like:

```
@Module({
  providers: [{ 
    provide: UserService,
    useValue: MockUserService,
  }],
})
```

With this, when you inject want to inject `UserService` as dependency, the `MockUserService` will be injected.

Custom provider token can be also a string:

```
@Module({
  providers: [{ 
    provide: 'USER_SERVICE',
    useValue: MockUserService,
  }],
})
```

but now, when you want to inject it as dependency you need to use the `@Inject` decorator.

```
constructor(@Inject('USER_SERVICE') userService: any) {}
```

33.5. Configuration Module

devon4node provides a way to generate a configuration module inside your application. To generate it you only need to execute the command `nest g -c @devon4node/schematics config-module`. This command will generate inside your application:

- Configuration module inside the core module.
- config folder where all environment configuration are stored.
 - default configuration: configuration for your local development environment.
 - develop environment configuration for the develop environment.
 - uat environment configuration for the uat environment.
 - production environment configuration for the production environment.
 - production environment configuration for the production environment.
 - test environment configuration used by test.



some code generators will add some properties to this module, so, be sure that the config module is the first module that you generate in your application.

33.5.1. Use the configuration service

To use the configuration service, you only need to inject it as dependency. As configuration module is defined in the core module, it will be available everywhere in your application. Example:

```
export class MyProvider {
    constructor(public readonly configService: ConfigurationService) {}

    myMethod() {
        return this.configService.isDev;
    }
}
```

33.5.2. Choose an environment file

By default, when you use the configuration service it will take the properties defined in the default.ts file. If you want to change the configuration file, you only need to set the NODE_ENV environment property with the name of the desired environment. Examples: in windows execute `set NODE_ENV=develop` before executing the application, in linux execute `NODE_ENV=develop` before executing the application or `NODE_ENV=develop yarn start`.

33.5.3. Override configuration properties

Sometimes, you want to keep some configuration property secure, and you do not want to publish it to the repository, or you want to reuse some configuration file but you need to change some properties. For those scenarios, you can override configuration properties by defining a environment variable with the same name. For example, if you want to override the property host, you can do: `set host="newhost"`. It also works with objects. For example, if you want to change the value of secret in the property jwtConfig for [this example](#), you can set a environment variable like this: `set jwtConfig="{\"secret\": \"newsecret\"}"`. As you can see, this environment variable has a JSON value. It will take object and merge the jwtConfig property with the properties defined inside the environment variable. It other properties maintain their value. The behaviour is the same for the nested objects.

33.5.4. Add a configuration property

In order to add a new property to the configuration module, you need to follow some steps:

- Add the property to IConfig interface in `src/app/core/configuration/types.ts` file. With this, we can ensure that the `ConfigurationService` and the environment files has those property at compiling time.
- Add the new property getter to `ConfigurationService`. You must use the `get` method of `ConfigurationService` to ensure that the property will be loaded from the desired config file. You can also add extra logic if needed.

- Add the property to all config files inside the `src/config` folder.

Example:

We want to add the property `devonfwUrl` to our ConfigurationService, so:

We add the following code in IConfig interface:

```
devonfwUrl: string;
```

Then, we add the getter in the `ConfigurationService`:

```
get devonfwUrl(): string {
  return this.get('devonfwUrl')!;
}
```

Finally, we add the definition in all config files:

```
devonfwUrl: 'https://devonfw.com',
```

33.6. Auth JWT module

devon4node provides a way to generate a default authentication module using JWT (JSON Web Token). It uses the [@nestjs/passport](#) library described [here](#).

To generate the devon4node auth-jwt module you only need to execute the command: `nest generate -c @devon4node/schematics auth-jwt`. We generate this module inside the applications instead of distributing a npm package because this module is prone to be modified depending on the requirements. It also generates a basic user module.

In this page we will explain the default implementation provided by devon4node. For more information about authentication, JWT, passport and other you can see:

- [JWT](#)
- [NestJS authentication](#)
- [Passport](#)
- [Passport JWT](#)

33.6.1. Auth JWT endpoints

In order to execute authentication operations, the auth-jwt module exposes the following endpoints:

- `POST /auth/login`: receive an username and a password and return the token in the header if the combination of username and password is correct.

- `POST /auth/register`: register a new user.
- `GET /auth/currentuser`: return the user data if he is authenticated.

33.6.2. Protect endpoints with auth-jwt

In order to protect your endpoints with auth-jwt module you only need to add the `AuthGuard()` in the `UseGuards` decorator. Example:

```
@Get('currentuser')
@UseGuards(AuthGuard())
currentUser(@Request() req: UserRequest) {
  return req.user;
}
```

Now, all request to currentuser are protected by the AuthGuard.

33.6.3. Role based Access Control

The auth-jwt module provides also a way to control the access to some endpoints by using roles. For example, if you want to grant access to a endpoint only to admins, you only need to add the `Roles` decorator to those endpoints with the roles allowed. Example:

```
@Get('currentuser')
@UseGuards(AuthGuard())
@Roles(roles.ADMIN)
currentUser(@Request() req: UserRequest) {
  return req.user;
}
```

33.7. Swagger

We can use swagger (OpenAPI) in order to describe the endpoints that our application exposes.

NestJS provides a module which will read the code of our application and will expose one endpoint where we can see the swagger.

Add swagger to a devon4node application is simple, you only need to execute the command `nest g -c @devon4node/schematics swagger` and it will do everything for you. The next time that you start your application, you will be able to see the swagger at `/v1/api` endpoint.

The swagger module can read your code in order to create the swagger definition, but sometimes you need to help him by decorating your handlers.

For more information about decorators and other behaviours about swagger module, you can see the [NestJS swagger documentation page](#)



the OpenAPI specification that this module supports is v2.0. The OpenAPI v3.0 is not available yet by using this module.

33.8. TypeORM

TypeORM is the default ORM provided by devon4node. It supports MySQL, MariaDB, Postgres, CockroachDB, SQLite, Microsoft SQL Server, Oracle, sql.js relational databases and also supports MongoDB NoSQL database.

Add TypeORM support to a devon4node application is very easy: you only need to execute the command `nest g -c @devon4node/schematics typeorm` and it will add all required dependencies to the project and also imports the `@nestjs/typeorm` module.

For more information about TypeORM and the integration with NestJS you can visit [TypeORM webpage](#), [TypeORM GitHub repository](#) and [NestJS TypeORM documentation page](#)

33.8.1. Configuration

When you have the configuration module, the typeorm generator will add one property in order to be able to configure the database depending on the environment. Example:

```
database: {
  type: 'sqlite',
  database: ':memory:',
  synchronize: false,
  migrationsRun: true,
  logging: true,
  entities: ['dist/**/*.entity.js'],
  migrations: ['dist/migration/**/*.js'],
  subscribers: ['dist/subscriber/**/*.js'],
  cli: {
    entitiesDir: 'src/entity',
    migrationsDir: 'src/migration',
    subscribersDir: 'src/subscriber',
  },
},
```

This object is a TypeORM ConnectionOptions. For fore information about it visit the [TypeORM Connection Options page](#).

There is also a special case: the default configuration. As the devon4node CLI need the database configuration when you use the `devon4node db` command, we also provide the `ormconfig.json` file. In this file you must put the configuration for you local environment. In order to do not have duplicated the configuration for local environment, in the default config file the database property is setted like:

```
database: require('../ormconfig.json'),
```

So, you only need to maintain the `ormconfig.json` file for the local environment.

33.8.2. Entity

Entity is a class that maps to a database table. The `devon4node` schematics has a generator to create new entities. You only need to execute the command `nest g -c @devon4node/schematics entity <entity-name>` and it generate the entity.

In the entity, you must define all columns, relations, primary keys of your database table. By default, `devon4node` provides a class named `BaseEntity`. All entities created with the `devon4node` schematics will extends the `BaseEntity`. This entity provides you some common columns:

- `id`: the primary key of you table
- `version`: the version of the entry (used for auditing purposes)
- `createdAt`: creation date of the entry (used for auditing purposes)
- `updatedAt`: last update date of the entry (used for auditing purposes)

For more information about Entities, please visit the [TypeORM entities page](#)

33.8.3. Repository

With repositories, you can manage (insert, update, delete, load, etc.) a concrete entity. Using this pattern, we have separated the data (Entities) from the methods to manage it (Repositories).

To use a repository you only need to:

- Import it in the module as follows:

```
@Module({
  imports: [TypeOrmModule.forFeature([Employee])],
})
```



if you generate the entities with the `devon4node` schematic, this step is not neccesary, `devon4node` schematic will do it for you.

- Inject the repository as dependency in your service:

```
constructor(@InjectRepository(Employee) employeeRepository: Repository<Employee>)
```

You can see more details in the [NestJS database](#) and [NestJS TypeORM](#) documentation pages.

33.9. Serializer

Serialization is the process of translating data structures or object state into a format that can be

transmitted across network and reconstructed later.

NestJS by default serialize all data to JSON (`JSON.stringify`). Sometimes this is not enough. In some situations you need to exclude some property (e.g password). Instead doing it manually, devon4node provides an interceptor (`ClassSerializerInterceptor`) that will do it for you. You only need to return a class instance as always and the interceptor will transform those class to the expected data.

The `ClassSerializerInterceptor` takes the [class-transformer](#) decorators in order to know how to transform the class and then send the result to the client.

Some of class-transformer decorators are:

- Expose
- Exclude
- Type
- Transform

And methods to transform data:

- `plainToClass`
- `plainToClassFromExist`
- `classToPlain`
- `classToClass`
- `serialize`
- `deserialize`
- `deserializeArray`

See the [class-transformer](#) page for more information.

See [NestJS serialization page](#) for more information about `ClassSerializerInterceptor`.

33.10. Validation

To be sure that your application will works well, you must validate any input data. devon4node by default provides a `ValidationPipe`. This `ValidationPipe` is the responsible of validate the request input and, if the input do not pass the validation process, it returns a `400 Bad Request` error.

33.10.1. Defining Validators

The `ValidationPipe` needs to know how to validate the input. For that purpose we use the `class-validator` package. This package allows you to define the validation of a class by using decorators.

For example:

```
export class Coffee {
  @IsDefined()
  @IsString()
  @MaxLength(255)
  name: string;

  @IsDefined()
  @IsString()
  @MaxLength(25)
  type: string;

  @IsDefined()
  @IsNumber()
  quantity: number;
}
```

As you can see in the previous example, we used some decorators in order to define the validators for every property of the Coffee class. You can find all decorators in the [class-validator github repository](#).

Now, when you want to receive a Coffee as input in some endpoint, it will execute the validations before executing the handler function.



In order to be able to use the class-validator package, you must use classes instead of interfaces. As you know interfaces disappear at compiling time, and class-validator need to know the metadata of the properties in order to be able to validate.



The **ValidationPipe** only works if you put a specific type in the handler definition. For example, if you define a handler like `getCoffee(@Body() coffee: any): Coffee {}` the ValidationPipe will not do anything. You must specify the type of the input: `getCoffee(@Body() coffee: Coffee): Coffee {}`

33.11. Logger

When you create a new devon4node application, it already has a logger: `src/app/shared/logger/winston.logger.ts`. This logger provide the methods `log`, `error` and `warn`. All of those methods will write a log message, but with a different log level.

The winston logger has two transports: one to log everything inside the file logs/general.log and the other to log only the error logs inside the file logs/error.log. In addition, it uses the default NestJS logger in order to show the logs in the console.

As you can see it is a simple example about how to use logger in a devon4node application. It will be update to a complex one in the next versions.

33.11.1. How to use logger

In order to use the logger you only need to inject the logger as a dependency:

```
constructor(logger: WinstonLogger){}
```

and then use it

```
async getAll() {
  this.service.getAll();
  this.logger.log('Returning all data');
}
```

33.12. Mailer Module

This module enables you to send emails in devon4node. It also provides a template engine using Handlebars.

It is a NestJS module that inject into your application a MailerService, which is the responsible to send the emails using the nodemailer library.

33.12.1. Installing

Execute the following command in a devon4node project:

```
yarn add @devon4node/mailer
```

33.12.2. Configuring

To configure the mailer module, you only need to import it in your application into another module. Example:

```
@Module({
  ...
  imports: [
    MailerModule.forRoot(),
  ],
  ...
})
```

Your must pass the configuration using the forRoot or forRootAsync methods.

forRoot()

The forRoot method receives an MailerModuleOptions object as parameter. It configures the

MailerModule using the input MailerModuleOptions object.

The structure of MailerModuleOptions is:

```
{
  hbsOptions?: {
    templatesDir: string;
    extension?: string;
    partialsDir?: string;
    helpers?: IHelperFunction[];
    compilerOptions?: ICompileOptions;
  },
  mailOptions?: nodemailerSmtpTransportOptions;
  emailFrom: string;
}
```

Here, you need to specify the [Handlebars compile options](#), the [nodemailer transport options](#) and the email address which will send the emails. Then, you need to call to forRoot function in the module imports. Example:

```
@Module({
  ...
  imports: [
    MailerModule.forRoot({
      mailOptions: {
        host: 'localhost',
        port: 1025,
        secure: false,
        tls: {
          rejectUnauthorized: false,
        },
      },
      emailFrom: 'noreply@capgemini.com',
      hbsOptions: {
        templatesDir: join(__dirname, '../../', 'templates/views'),
        partialsDir: join(__dirname, '../../', 'templates/partials'),
        helpers: [
          { name: 'fullname',
            func: person => `${person.name} ${person.surname}` },
        ],
      },
    }),
    ...
  })
})
```

forRootAsync()

The method forRootAsync enables you to get the mailer configuration in a asynchronous way. It is

useful when you need to get the configuration using, for example, a service (e.g. ConfigurationService).

Example:

```
@Module({
  ...
  imports: [
    MailerModule.forRootAsync({
      imports: [ConfigurationModule],
      useFactory: (config: ConfigurationService) => {
        return config.mailerConfig;
      },
      inject: [ConfigurationService],
    }),
    ...
  ]
})
```

In this example, we use the ConfigurationService in order to get the MailerModuleOptions (the same as forRoot)

33.12.3. Usage

In order to use, you only need to inject using the dependency injection the MailerService.

Example:

```
@Injectable()
export class CatsService {
  constructor(private readonly mailer: MailerService) {}
}
```

Then, you only need to use the methods provided by the MailerService in your service. Take into account that you can inject it in every place that support NestJS dependency injection.

MailerService methods

sendPlainMail

The method sendPlainMail receive a string sends a email.

The method signatures are:

```
sendPlainMail(emailOptions: SendMailOptions): Promise<SentMessageInfo>;
sendPlainMail(to: string, subject: string, mail: string): Promise<SentMessageInfo>;
```

Examples:

```
this.mailer.sendPlainMail({
  to: 'example@example.com',
  subject: 'This is a subject',
  html: '<h1>Hello world</h1>'
});
this.mailer.sendPlainMail('example@example.com', 'This is a subject', '<h1>Hello
world</h1>');
```

sendTemplateMail

The method `sendTemplateMail` sends a email based on a Handlebars template. The templates are registered using the `templatesDir` option or using the `addTemplate` method. The template name is the name of the template (without extension) or the first parameter of the method `addTemplate`.

The method signatures are:

```
sendTemplateMail(emailOptions: SendMailOptions, templateName: string, emailData: any,
hbsOptions?: RuntimeOptions): Promise<SentMessageInfo>;
sendTemplateMail(to: string, subject: string, templateName: string, emailData: any,
hbsOptions?: RuntimeOptions): Promise<SentMessageInfo>;
```

Examples:

```
this.mailer.sendTemplateMail({
  to: 'example@example.com',
  subject: 'This is a subject',
  html: '<h1>Hello world</h1>'
}, 'template1', { person: {name: 'Dario', surname: 'Rodriguez'}});
this.mailer.sendTemplateMail('example@example.com', 'This is a subject', 'template1',
{ person: {name: 'Dario', surname: 'Rodriguez'}});
```

addTemplate

Adds a new template to the MailerService.

Method signature:

```
addTemplate(name: string, template: string, options?: CompileOptions): void;
```

Example:

```
this.mailer.addTemplate('newTemplate',
'<html><head></head><body>{{>partial1}}</body></html>')
```

registerPartial

Register a new partial in Handlebars.

Method signature:

```
registerPartial(name: string, partial: Handlebars.Template<any>): void;
```

Example:

```
this.mailer.registerPartial('partial', '<h1>Hello World</h1>')
```

registerHelper

Register a new helper in Handlebars.

Method signature:

```
registerHelper(name: string, helper: Handlebars.HelperDelegate): void;
```

Example:

```
this.mailer.registerHelper('fullname', person => `${person.name} ${person.surname}`)
```

33.12.4. Handlebars templates

As mentioned above, this module allow you to use Handlebars as template engine, but it is optional. If you do not need the Handlebars, you just need to keep the hbsOptions undefined.

In order to get the templates form the file system, you can specify the template folder, the partials folder and the helpers. At the moment of module initialization, it will read the content of the template folder, and will register every file with the name (without extension) and the content as Handlebars template. It will do the same for the partials.

You can specify the extension of template files using the [extension](#) parameter. The default value is [.handlebars](#)

33.12.5. Local development

If you want to work with this module but you don't have a SMTP server, you can use the [streamTransport](#). Example:

```
{
  mailOptions: {
    streamTransport: true,
    newline: 'windows',
  },
  emailFrom: ...
  hbsOptions: ...
}
```

Then, you need to get the sendPlainMail or sendTemplateMail result, and print the email to the standard output (STDOUT). Example:

```
const mail = await this.mailer.sendTemplateMail(...);

mail.message.pipe(process.stdout);
```

33.13. Importing your ESLint reports into SonarQube

This guide covers the import of ESLint reports into SonarQube instances in CI environments, as this is the recommended way of using ESLint and SonarQube for devon4node projects. The prerequisites for this process are a CI environment, preferably a [Production Line](#) instance, and the [ESLint CLI](#), which is already included when generating a new devon4node project.

Configuring the ESLint analysis

You can configure the ESLint analysis parameters in the `.eslintrc.js` file inside the top-level directory of your project. If you created your node project using the devon4node application schematic, this file will already exist. If you want to make further adjustments to it, have a look at the [ESLint documentation](#).

The ESLint analysis script `lint` is already configured in the `scripts` part of your `package.json`. Simply add `-f json > report.json`, so that the output of the analysis is saved in a `json` file. Additional information to customization options for the ESLint CLI can be found [here](#).

To run the analysis, execute the script with `npm run lint` inside the base directory of your project.

Configuring SonarQube

If you haven't already generated your CICD-related files, follow the tutorial on the [devon4node schematic](#) of our CICDGEN project, as you will need a `Jenkinsfile` configured in your project to proceed.

Inside the script for the SonarQube code analysis in your `Jenkinsfile`, add the parameter `-Dsonar.eslint.reportPaths=report.json`. Now, whenever a SonarQube analysis is triggered by your CI environment, the generated report will be loaded into your SonarQube instance. To avoid duplicated issues, you can associate an empty TypeScript quality profile with your project in its server configurations.

34. devon4node applications

34.1. devon4node Samples

In the folder `/samples`, you can find some devon4node examples that could be useful for you in order to understand better the framework.

The samples are:

- [Todo](#)
- [Employee](#)
- [Components example](#)

Also, we have another realistic example in the [My Thai Star repository](#). This example is the implementation of My Thai Star backend, which is compatible with the frontend made with Angular. To do that, this node implementation exposes the same API as Java backend. Take care with this example, as we need to follow the Java API, some components do not follow the devon4node patterns and code conventions.

34.1.1. Todo example

This example is the backend part of an TO-DO application. It exposes and API where you can create, read, update and delete a TO-DO list.

In order to start the application, run the following commands in the todo folder:

```
$ yarn  
$ yarn build  
$ yarn start
```

Now, you can access to the application using the url <http://localhost:3000/v1/todo/todos>. If you want to know all endpoints exposed, you can see the swagger at: <http://localhost:3000/v1/api>.

Also, in this example we show you how to control the access to your application by implementing an authentication mechanism using JWT and role based strategy. In order to access to the list of todos (<http://localhost:3000/v1/todo/todos>), first you need to call to [POST http://localhost:3000/v1/auth/login](#) and in the body you need to send the user information:

```
{  
  "username": "user",  
  "password": "password"  
}
```

It will return a JWT token for the user `user`. The role of this user is `USER`, so you can only access to the methods `GET`, `POST` and `DELETE` of the endpoint <http://localhost:3000/v1/todo/todos>. If you login with the user `admin/admin`, you will be able to access to the methods `UPDATE` and `PATCH`.

34.1.2. Employee example

This is an example of employee management application. With the application you can create, read, update and delete employees.

In order to start the application, run the following commands in the todo folder:

```
$ yarn  
$ yarn build  
$ yarn start
```

Now, you can access to the application using the url <http://localhost:8081/v1/employee/employees>. If you want to know all endpoints exposed, you can see the swagger at: <http://localhost:8081/v1/api>.

This is a simple example without authentication. With this example you can learn how to work with database migrations. You can find them in the folder /src/migrations. The TypeORM is configured in order to execute the migrations every time that you start this application (`"migrationsRun": true` at `ormconfig.json`). You can also execute the migration manually by typing the command `devon4node db migration:run`, or revert executing `devon4node db migration:revert`. Take into account that the database that this application is using is an in-memory sqlite, so every time that you stop the application all data is lost.

34.1.3. Components example

This example allow you to understand better the execution order of the components of a devon4node application (guards, pipes, interceptors, filters, middleware).

In order to start the application, run the following commands in the todo folder:

```
$ yarn  
$ yarn build  
$ yarn start
```

In order to see the execution order, you can call to <http://localhost:3000/v1>. It will show you the execution order of all components except the filters. If you want to know the execution order while a filter is applied, call to the endpoint with the following queries: `?hello=error`, `?hello=controller`, `?hello=global`.

34.2. Create the employee sample step by step

34.2.1. Application requisites

The employee application needs:

- A configuration module
- A SQLite in memory database

- Security: CORS
- Swagger support
- Authentication using JWT
- CRUD for manage employees. The employees will have the following properties:
 - name
 - surname
 - email

34.2.2. Create the application

1. Install Nest CLI

Execute the command `yarn global add @nestjs/cli`

2. Install devon4node schematics

3. Execute the command `yarn global add @devon4node/schematics`

4. Create the new application

Execute the command `nest g -c @devon4node/schematics application employee`

5. Then, we need to add some components, go inside the project folder and execute the following commands:

Go inside project folder: `cd employee`.

Config module: `nest g -c @devon4node/schematics config-module`.

Typeorm database, choose sqlite DB when asked `nest g -c @devon4node/schematics typeorm`.

Add security: `nest g -c @devon4node/schematics security`.

Swagger module: `nest g -c @devon4node/schematics swagger`.

Auth-jwt authentication: `nest g -c @devon4node/schematics auth-jwt`.

Add an application module: `nest g -c @devon4node/schematics module employee`.

Add CRUD component: `nest g -c @devon4node/schematics crud employee/employee`.

With this, you will generate the following files:

```
/employee/.prettierrc
/employee/nest-cli.json
/employee/package.json
/employee/README.md
/employee/tsconfig.build.json
/employee/tsconfig.json
/employee/tslint.json
```

```
/employee/src/main.ts
/employee/test/app.e2e-spec.ts
/employee/test/jest-e2e.json
/employee/src/app/app.controller.spec.ts
/employee/src/app/app.controller.ts
/employee/src/app/app.module.ts
/employee/src/app/app.service.ts
/employee/src/app/core/core.module.ts
/employee/src/app/shared/logger/winston.logger.ts
/employee/src/app/core/configuration/configuration.module.ts
/employee/src/app/core/configuration/model/index.ts
/employee/src/app/core/configuration/model/types.ts
/employee/src/app/core/configuration/services/configuration.service.spec.ts
/employee/src/app/core/configuration/services/configuration.service.ts
/employee/src/app/core/configuration/services/index.ts
/employee/src/config/default.ts
/employee/src/config/develop.ts
/employee/src/config/production.ts
/employee/src/config/test.ts
/employee/src/config/uat.ts
/employee/docker-compose.yml
/employee/ormconfig.json
/employee/src/app/shared/model/entities/base-entity.entity.ts
/employee/src/app/core/auth/auth.module.ts
/employee/src/app/core/auth/controllers/auth.controller.spec.ts
/employee/src/app/core/auth/controllers/auth.controller.ts
/employee/src/app/core/auth/controllers/index.ts
/employee/src/app/core/auth/decorators/index.ts
/employee/src/app/core/auth/decorators/roles.decorator.spec.ts
/employee/src/app/core/auth/decorators/roles.decorator.ts
/employee/src/app/core/auth/guards/index.ts
/employee/src/app/core/auth/guards/roles.guard.spec.ts
/employee/src/app/core/auth/guards/roles.guard.ts
/employee/src/app/core/auth/model/index.ts
/employee/src/app/core/auth/model/roles.enum.ts
/employee/src/app/core/auth/model/user-request.interface.ts
/employee/src/app/core/auth/services/auth.service.spec.ts
/employee/src/app/core/auth/services/auth.service.ts
/employee/src/app/core/auth/services/index.ts
/employee/src/app/core/auth/strategies/index.ts
/employee/src/app/core/auth/strategies/jwt.strategy.spec.ts
/employee/src/app/core/auth/strategies/jwt.strategy.ts
/employee/src/app/core/user/user.module.ts
/employee/src/app/core/user/model/index.ts
/employee/src/app/core/user/model/dto/user-payload.dto.ts
/employee/src/app/core/user/model/entities/user.entity.ts
/employee/src/app/core/user/services/index.ts
/employee/src/app/core/user/services/user.service.spec.ts
/employee/src/app/core/user/services/user.service.ts
/employee/test/auth/auth.service.mock.ts
/employee/test/user/user.repository.mock.ts
```

```
/employee/src/app/employee/employee.module.ts  
/employee/src/app/employee/model/entities/employee.entity.ts  
/employee/src/app/employee/model/index.ts  
/employee/src/app/employee/controllers/employee.crud.controller.ts  
/employee/src/app/employee/services/employee.crud.service.ts  
/employee/src/app/employee/services/index.ts  
/employee/src/app/employee/controllers/index.ts
```

6. Open the VSCode

Execute the commands:

```
yarn install  
code .
```

7. Fill in the entity: src/app/employee/model/entities/employee.entity.ts

a. Add the columns

```
@Entity()  
export class Employee extends BaseEntity {  
    @Column('varchar', { length: 255, nullable: true })  
    name?: string;  
  
    @Column('varchar', { length: 255, nullable: true })  
    surname?: string;  
  
    @Column('varchar', { length: 255, nullable: true })  
    email?: string;  
}
```

b. Add the validations

```

@Entity()
export class Employee extends BaseEntity {
    @IsDefined({ groups: [CrudValidationGroups.CREATE] })
    @IsOptional({ groups: [CrudValidationGroups.UPDATE] })
    @MaxLength(255)
    @Column('varchar', { length: 255, nullable: true })
    name?: string;

    @IsDefined({ groups: [CrudValidationGroups.CREATE] })
    @IsOptional({ groups: [CrudValidationGroups.UPDATE] })
    @MaxLength(255)
    @Column('varchar', { length: 255, nullable: true })
    surname?: string;

    @IsDefined({ groups: [CrudValidationGroups.CREATE] })
    @IsOptional({ groups: [CrudValidationGroups.UPDATE] })
    @MaxLength(255)
    @IsEmail()
    @Column('varchar', { length: 255, nullable: true })
    email?: string;
}

```

c. Add the transformations

In this specific case, we will not transform any property, but you can see an example in the [src/app/shared/model/entities/base-entity.entity.ts](#) file.

```

export abstract class BaseEntity {
    @PrimaryGeneratedColumn('increment')
    id!: number;

    @VersionColumn({ default: 1 })
    @Exclude({ toPlainOnly: true })
    version!: number;

    @CreateDateColumn()
    @Exclude({ toPlainOnly: true })
    createdAt!: string;

    @UpdateDateColumn()
    @Exclude({ toPlainOnly: true })
    updatedAt!: string;
}

```

d. Add swagger metadata

```

@Entity()
export class Employee extends BaseEntity {
    @ApiPropertyOptional()
    @IsDefined({ groups: [CrudValidationGroups.CREATE] })
    @IsOptional({ groups: [CrudValidationGroups.UPDATE] })
    @MaxLength(255)
    @Column('varchar', { length: 255, nullable: true })
    name?: string;

    @ApiPropertyOptional()
    @IsDefined({ groups: [CrudValidationGroups.CREATE] })
    @IsOptional({ groups: [CrudValidationGroups.UPDATE] })
    @MaxLength(255)
    @Column('varchar', { length: 255, nullable: true })
    surname?: string;

    @ApiPropertyOptional()
    @IsDefined({ groups: [CrudValidationGroups.CREATE] })
    @IsOptional({ groups: [CrudValidationGroups.UPDATE] })
    @MaxLength(255)
    @IsEmail()
    @Column('varchar', { length: 255, nullable: true })
    email?: string;
}

```

8. Add swagger metadata to `src/app/employee/controllers/employee.crud.controller.ts`

```
@ApiTags('employee')
```

9. Generate database migrations

- Build the application: `yarn build`
- In order to create migration scripts with typeorm, you need to install ts-node: `yarn global add ts-node`
- Generate the tables creation migration: `yarn run typeorm migration:generate -n CreateTables`

[generate migrations] | *images/sample/generate-migrations.PNG*

The output will be something similar to:

```
export class CreateTables1572480273012 implements MigrationInterface {
    name = 'CreateTables1572480273012';

    public async up(queryRunner: QueryRunner): Promise<any> {
        await queryRunner.query(
            'CREATE TABLE "user" ("id" integer PRIMARY KEY AUTOINCREMENT NOT NULL,
"version" integer NOT NULL DEFAULT (1), "createdAt" datetime NOT NULL DEFAULT
(datetime('now')), "updatedAt" datetime NOT NULL DEFAULT (datetime('now')),
"username" varchar(255) NOT NULL, "password" varchar(255) NOT NULL, "role"
integer NOT NULL DEFAULT (0))',
            undefined,
        );
        await queryRunner.query(
            'CREATE TABLE "employee" ("id" integer PRIMARY KEY AUTOINCREMENT NOT NULL,
"version" integer NOT NULL DEFAULT (1), "createdAt" datetime NOT NULL DEFAULT
(datetime('now')), "updatedAt" datetime NOT NULL DEFAULT (datetime('now')),
"name" varchar(255), "surname" varchar(255), "email" varchar(255))',
            undefined,
        );
    }

    public async down(queryRunner: QueryRunner): Promise<any> {
        await queryRunner.query(`DROP TABLE "employee"`, undefined);
        await queryRunner.query(`DROP TABLE "user"`, undefined);
    }
}
```

The number in the name is a timestamp, so may change in your application.

d. Create a migration to insert data: `yarn run typeorm migration:generate -n InsertData`

[insert data] | *images/sample/insert-data.PNG*

and fill in with the following code:

```
export class InsertData1572480830290 implements MigrationInterface {
  public async up(queryRunner: QueryRunner): Promise<any> {
    await queryRunner.query(
      `INSERT INTO EMPLOYEE(id, name, surname, email) VALUES(1, 'Santiago',
      'Fowler', 'Santiago.Fowler@example.com');`,
    );
    await queryRunner.query(
      `INSERT INTO EMPLOYEE(id, name, surname, email) VALUES(2, 'Clinton',
      'Thornton', 'Clinton.Thornton@example.com');`,
    );
    await queryRunner.query(
      `INSERT INTO EMPLOYEE(id, name, surname, email) VALUES(3, 'Lisa',
      'Rodriquez', 'Lisa.Rodriquez@example.com');`,
    );
    await queryRunner.query(
      `INSERT INTO EMPLOYEE(id, name, surname, email) VALUES(4, 'Calvin',
      'Becker', 'Calvin.Becker@example.com');`,
    );
    await queryRunner.query(`INSERT INTO USER(id, username, password, role)
    VALUES(?, ?, ?, ?);`, [
      1,
      'user',
      await hash('password', await genSalt(12)),
      roles.USER,
    ]);
    await queryRunner.query(`INSERT INTO USER(id, username, password, role)
    VALUES(?, ?, ?, ?);`, [
      2,
      'admin',
      await hash('admin', await genSalt(12)),
      roles.ADMIN,
    ]);
  }
}

public async down(queryRunner: QueryRunner): Promise<any> {
  await queryRunner.query(`DELETE FROM EMPLOYEE`);
  await queryRunner.query(`DELETE FROM USER`);
}
}
```

10. Start the application: `yarn start:dev`

```
[Nest] 90288 - 2019-10-31 1:21:42 AM [NestFactory] Starting Nest application...
[Nest] 90288 - [Nest] 71428 - 2019-10-31 1:22:28 AM [NestFactory] Starting Nest application...
[Nest] 71428 - 2019-10-31 1:22:31 AM [InstanceLoader] PassportModule dependencies initialized
[Nest] 71428 - 2019-10-31 1:22:33 AM [InstanceLoader] ConfigurationModule dependencies initialized
[Nest] 71428 - 2019-10-31 1:22:33 AM [InstanceLoader] TypeOrmModule dependencies initialized
[Nest] 71428 - 2019-10-31 1:22:33 AM [InstanceLoader] CoreModule dependencies initialized
[Nest] 71428 - 2019-10-31 1:22:34 AM [InstanceLoader] AppModule dependencies initialized
[Nest] 71428 - 2019-10-31 1:22:34 AM [InstanceLoader] JwtModule dependencies initialized
query: SELECT * FROM "sqlite_master" WHERE "type" = 'table' AND "name" = "migrations"
query: CREATE TABLE "migrations" ("id" integer PRIMARY KEY AUTOINCREMENT NOT NULL, "timestamp" bigint NOT NULL, "name" varchar NOT NULL)
query: SELECT * FROM "migrations" "migrations" ORDER BY id DESC
query: BEGIN TRANSACTION
query: CREATE TABLE "user" ("id" integer PRIMARY KEY AUTOINCREMENT NOT NULL, "version" integer NOT NULL DEFAULT (1), "createdAt" datetime NOT NULL DEFAULT (datetime('now')), "updatedAt" datetime NOT NULL DEFAULT (datetime('now')), "username" varchar(255) NOT NULL, "password" varchar(255) NOT NULL, "role" integer NOT NULL DEFAULT (0))
query: CREATE TABLE "employee" ("id" integer PRIMARY KEY AUTOINCREMENT NOT NULL, "version" integer NOT NULL DEFAULT (1), "createdAt" datetime NOT NULL DEFAULT (datetime('now')), "updatedAt" datetime NOT NULL DEFAULT (datetime('now')), "name" varchar(255), "surname" varchar(255), "email" varchar(255))
query: INSERT INTO "migrations"("timestamp", "name") VALUES (?, ?); -- PARAMETERS: [1572488273012, "CreateTables1572488273012"]
query: INSERT INTO EMPLOYEE(id, name, surname, email) VALUES(1, 'Stefano', 'Rossini', 'stefano.rossini@capgemini.com');
query: INSERT INTO EMPLOYEE(id, name, surname, email) VALUES(2, 'Angelo', 'Muresu', 'angelo.muresu@capgemini.com');
query: INSERT INTO EMPLOYEE(id, name, surname, email) VALUES(3, 'Jaime', 'Gonzalez', 'jaime.diaz-gonzalez@capgemini.com');
query: INSERT INTO EMPLOYEE(id, name, surname, email) VALUES(4, 'Dario', 'Rodriguez', 'dario.rodriguez-gonzalez@capgemini.com');
query: INSERT INTO USER(id, username, password, role) VALUES(?, ?, ?, ?); -- PARAMETERS: [1, "user", "$2b$12$Qk9QlqUpeElrezMB.v6ujoJRMfOeyfmY10eoVcBAADIR4L7kZfa", 0]
query: INSERT INTO USER(id, username, password, role) VALUES(?, ?, ?, ?); -- PARAMETERS: [2, "admin", "$2b$12$AhaalSdKQgiogwYttelJbwB8RZlk707J5k1KeCQ97pDr/fPs2", 1]
query: INSERT INTO "migrations"("timestamp", "name") VALUES (?, ?); -- PARAMETERS: [15724880630290, "InsertData15724880630290"]
query: COMMIT
[Nest] 71428 - 2019-10-31 1:23:01 AM [InstanceLoader] TypeOrmCoreModule dependencies initialized
[Nest] 71428 - 2019-10-31 1:23:01 AM [InstanceLoader] TypeOrmModule dependencies initialized
[Nest] 71428 - 2019-10-31 1:23:01 AM [InstanceLoader] TypeOrmModule dependencies initialized
[Nest] 71428 - 2019-10-31 1:23:02 AM [InstanceLoader] UserModule dependencies initialized
[Nest] 71428 - 2019-10-31 1:23:02 AM [InstanceLoader] EmployeeModule dependencies initialized
[Nest] 71428 - 2019-10-31 1:23:02 AM [InstanceLoader] AuthModule dependencies initialized
[Nest] 71428 - 2019-10-31 1:23:02 AM [RoutesResolver] AppController (/v1/):
[Nest] 71428 - 2019-10-31 1:23:02 AM [RouterExplorer] Mapped (/, GET) route
[Nest] 71428 - 2019-10-31 1:23:02 AM [RouterExplorer] AuthController (/v1/auth):
[Nest] 71428 - 2019-10-31 1:23:02 AM [RouterExplorer] Mapped (/login, POST) route
[Nest] 71428 - 2019-10-31 1:23:04 AM [RouterExplorer] Mapped (/register, POST) route
[Nest] 71428 - 2019-10-31 1:23:06 AM [RouterExplorer] Mapped (/currentuser, GET) route
[Nest] 71428 - 2019-10-31 1:23:07 AM [RoutesResolver] EmployeeCrudController (/v1/employee/employees):
[Nest] 71428 - 2019-10-31 1:23:07 AM [RouterExplorer] Mapped (/, GET) route
[Nest] 71428 - 2019-10-31 1:23:08 AM [RouterExplorer] Mapped (/:id, GET) route
[Nest] 71428 - 2019-10-31 1:23:08 AM [RouterExplorer] Mapped (/:id, POST) route
[Nest] 71428 - 2019-10-31 1:23:08 AM [RouterExplorer] Mapped (/:id, PUT) route
[Nest] 71428 - 2019-10-31 1:23:08 AM [RouterExplorer] Mapped (/:id, PATCH) route
[Nest] 71428 - 2019-10-31 1:23:10 AM [RouterExplorer] Mapped (/:id, DELETE) route
[Nest] 71428 - 2019-10-31 1:23:10 AM [RouterExplorer] Mapped (/:id, DELETE) route
[Nest] 71428 - 2019-10-31 1:23:10 AM [NestApplication] Nest application successfully started
```

11. Check the swagger endpoint: <http://localhost:3000/v1/api>

The screenshot shows the Swagger UI interface for the Nest application. The 'default' section contains three endpoints: 'auth/login' (POST), 'auth/register' (POST), and 'auth/currentuser' (GET). The 'employee' section contains several endpoints: 'employee/employees' (GET, POST), 'employee/employees/{id}' (GET, PATCH, PUT, DELETE), and 'employee/employees/bulk' (POST).

12. Make petitions to the employee CRUD: <http://localhost:3000/v1/employee/employees>

```

JSON Datos sin procesar Cabeceras
Guardar Copiar Contrar todo Expander todo Filtrar JSON
▼ 0:
  id: 1
  name: "Stefano"
  surname: "Rossini"
  email: "stefano.rossini@capgemini.com"
▼ 1:
  id: 2
  name: "Angelo"
  surname: "Muresu"
  email: "angelo.muresu@capgemini.com"
▼ 2:
  id: 3
  name: "Jaime"
  surname: "Gonzalez"
  email: "jaime.diaz-gonzalez@capgemini.com"
▼ 3:
  id: 4
  name: "Dario"
  surname: "Rodriguez"
  email: "dario.rodriguez-gonzalez@capgemini.com"

```

13. Write the tests

As we do not create any method, only add some properties to the entity, all application must be tested by the autogenerated code. As we add some modules, you need to uncomment some lines in the [src/app/core/configuration/services/configuration.service.spec.ts](#):

```

describe('ConfigurationService', () => {
  const configService: ConfigurationService = new ConfigurationService();

  it('should return the values of test config file', () => {
    expect(configService.isDev).toStrictEqual(def.isDev);
    expect(configService.host).toStrictEqual(def.host);
    expect(configService.port).toStrictEqual(def.port);
    expect(configService.clientUrl).toStrictEqual(def.clientUrl);
    expect(configService.globalPrefix).toStrictEqual(def.globalPrefix);
    // Remove comments if you add those modules
    expect(configService.database).toStrictEqual(def.database);
    expect(configService.swaggerConfig).toStrictEqual(def.swaggerConfig);
    expect(configService.jwtConfig).toStrictEqual(def.jwtConfig);
    // expect(configService.mailerConfig).toStrictEqual(def.mailerConfig);
  });
  it('should take the value of environment variable if defined', () => {
    process.env.isDev = 'true';
    process.env.host = 'notlocalhost';
    process.env.port = '123456';
    process.env.clientUrl = 'http://theclienturl.net';
    process.env.globalPrefix = 'v2';
    process.env.swaggerConfig = JSON.stringify({
      swaggerTitle: 'Test Application',
    });
    process.env.database = JSON.stringify({
      type: 'oracle',
      cli: { entitiesDir: 'src/notentitiesdir' },
    });
    process.env.jwtConfig = JSON.stringify({ secret: 'NOTSECRET' });
    // process.env.mailerConfig = JSON.stringify({ mailOptions: { host:
  });
}

```

```
'notlocalhost' });

expect(configService.isDev).toBe(true);
expect(configService.host).toBe('notlocalhost');
expect(configService.port).toBe(123456);
expect(configService.clientUrl).toBe('http://theclienturl.net');
expect(configService.globalPrefix).toBe('v2');
const database: any = { ...def.database, type: 'oracle' };
database.cli.entitiesDir = 'src/notentitiesdir';
expect(configService.database).toStrictEqual(database);
expect(configService.swaggerConfig).toStrictEqual({
  ...def.swaggerConfig,
  swaggerTitle: 'Test Application',
});
expect(configService.jwtConfig).toStrictEqual({
  ...def.jwtConfig,
  secret: 'NOTSECRET',
});
// const mail: any = { ...def.mailerConfig };
// mail.mailOptions.host = 'notlocalhost';
// expect(configService.mailerConfig).toStrictEqual(mail);

process.env.isDev = undefined;
process.env.host = undefined;
process.env.port = undefined;
process.env.clientUrl = undefined;
process.env.globalPrefix = undefined;
process.env.database = undefined;
process.env.swaggerConfig = undefined;
process.env.jwtConfig = undefined;
// process.env.mailerConfig = undefined;
});
});
```

And the output should be:

```

λ yarn test:cov
yarn run v1.19.1
$ jest --coverage
PASS  src/app/core/configuration/services/configuration.service.spec.ts (66.748s)
PASS  src/app/core/auth/guards/roles.guard.spec.ts (10.912s)
PASS  src/app/core/auth/controllers/auth.controller.spec.ts (91.655s)
PASS  src/app/core/auth/services/auth.service.spec.ts (96.123s)
PASS  src/app/core/auth/decorators/roles.decorator.spec.ts (6.346s)
PASS  src/app/core/user/services/user.service.spec.ts (24.786s)
PASS  src/app/app.controller.spec.ts (6.016s)
PASS  src/app/core/auth:strategies/jwt.strategy.spec.ts (7.399s)

```

File	% Stmt	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
app	100	100	100	100	
app.controller.ts	100	100	100	100	
app.service.ts	100	100	100	100	
app/core/auth/controllers	100	100	100	100	
auth.controller.ts	100	100	100	100	
app/core/auth/decorators	100	100	100	100	
roles.decorator.ts	100	100	100	100	
app/core/auth/guards	100	100	100	100	
roles.guard.ts	100	100	100	100	
app/core/auth/model	100	100	100	100	
index.ts	100	100	100	100	
roles.enum.ts	100	100	100	100	
app/core/auth/services	100	100	100	100	
auth.service.ts	100	100	100	100	
index.ts	100	100	100	100	
app/core/auth:strategies	100	100	100	100	
jwt.strategy.ts	100	100	100	100	
app/core/configuration	100	100	100	100	
configuration.module.ts	100	100	100	100	
app/core/configuration/services	100	100	100	100	
configuration.service.ts	100	100	100	100	
index.ts	100	100	100	100	
app/core/user/model	100	100	100	100	
index.ts	100	100	100	100	
app/core/user/model/dto	100	100	100	100	
user-payload.dto.ts	100	100	100	100	
app/core/user/model/entities	100	100	100	100	
user.entity.ts	100	100	100	100	
app/core/user/services	100	100	100	100	
index.ts	100	100	100	100	
user.service.ts	100	100	100	100	
app/shared/model/entities	100	100	100	100	
base-entity.entity.ts	100	100	100	100	
config	100	100	100	100	
default.ts	100	100	100	100	
test.ts	100	100	100	100	

```

Test Suites: 8 passed, 8 total
Tests:    26 passed, 26 total
Snapshots: 0 total
Time:    118.109s
Ran all test suites.
Done in 132.98s.

```

Part VII: Choosing your Database

35. Database

For your business application with devonfw you need to choose the right database. In devonfw we are not biased for a particular product so you have the freedom of choice.

35.1. RDBMS

The classical and well-established form of a database is a relational database management system (RDBMS). In devonfw we recommend to use an RDBMS unless you have specific need. However, in case you have the need for big data, graph-data, BLOB focus, or schema-less dynamic data you can have a look at [NoSQL](#) options but be aware that these may be experimental and are not fully supported by devonfw.

35.1.1. Options

In devonfw we are not biased for a particular RDBMS so you have the freedom of choice. Here are the most common options:

- [SAP Hana](#) (high performance in-memory, many advanced features)
- [Oracle](#) (most established, well featured for enterprise)
- [PostgreSQL](#) (great open-source RDBMS)
- [MariaDB](#) (true OSS successor of MySQL)
- [MS SQL Server](#) (best choice for Microsoft and Windows dominated IT landscapes)

Please click on any of the above choices and go to the according guide to find specific details such as client/driver.

35.2. NoSQL

While not (yet) officially supported and recommended there are also interesting NoSQL (Not Only SQL) databases that could be an interesting alternative. Please be aware that you will typically not be able to use [JPA](#) (and hibernate). Further before choosing a NoSQL database you should check the following aspects:

- Is the database of choice reliable and mature enough for your project?
- Can the operators of your product support the database of choice properly (provisioning, administration, backup, scaling & clustering, monitoring, etc.)?
- Does the database of choice meet the requirements of your project (ACID vs. eventual consistency, CAP theorem)?

There are good reasons to choose a particular NoSQL database in specific cases (e.g. extreme demand for big-data, throughput or scaling). But as indicated by the questions above you need to be fully aware of what you are doing. NoSQL databases can be *schemaless* (untyped, dynamic & flexible) and/or *schemaful* (typed, structured & strict). Furthermore, there are different types of NoSQL databases that are discussed in the following sub-sections:

35.2.1. Column DB

Column NoSQL databases are more related to a regular [RDBMS](#) with their tables and columns. However, they typically do not offer relational support with joins to the same level as you expect from an RDBMS. Therefore, you have to carefully design your data-model upfront with the all the knowledge how you later want to query your data.

The most prominent options are:

- [Cassandra](#) (high-performance, schema-based DB)
- [HBase](#) (distributed, big-data Hadoop database)

35.2.2. Key-Value DB

As indicated by the name, a key-value database stores objects as key/value pairs similar to [Properties](#) or [Map](#) in Java.

The most prominent options are:

- [Redis](#) (in-memory key/value store, especially used as cache or message broker)
- [aerospike](#)

35.2.3. Document DB

A document database is similar to a key-value database, but it stores objects in standard structured formats such as XML, JSON, or BSON. Therefore not only flat key/value pairs but even trees of hierarchical data can be stored, retrieved and queried.

The most prominent options are:

- [MongoDB](#)
- [CouchDB](#)
- [RavenDB](#)

35.2.4. Graph DB

If the connections (links/relations) between your data is key and an [RDBMS](#) is just not flexible or fast enough for your plans, then a graph database can help you. They are very strong on storing and querying complex connections between entities. For queries there are even specific standards and languages like [Gremlin](#).

The most prominent options are:

- [neo4j](#)
- [blazegraph](#)

35.2.5. Hybrid DB

In addition to the above types there are some NoSQL databases that are hybrid and combine the features and aspects of these types. While as an architect and developer you might love the idea to get all in one, you have to be careful with your choice. If you do not exactly know your problem, you are not ready to make the right choice for your database. Further, you might still be best-off with an good old [RDBMS](#) if you need to address multiple aspects together. Anyhow, for experiments, PoCs, or small microservices with little risk it might be a great idea to choose a hybrid NoSQL database. If you have collected very positive, profound and productive experience with such product you can grow on it.

The most prominent options are:

- [OrientDB](#) (object-oriented, hyper-flexible, column- and graph-based)

36. SAP HANA

This section contains hints for those who use [SAP HANA](#), a very powerful and fast RDBMS. Besides general hints about the driver there are tips for more tight integration with other SAP features or products.

36.1. Driver

SAP Hana is a commercial and professional product. However, the hana JDBC driver is available in Maven Central what makes it easy to integrate. All you need is the following maven dependency:

```
<dependency>
    <groupId>com.sap.cloud.db.jdbc</groupId>
    <artifactId>ngdbc</artifactId>
    <version>${hana.driver.version}</version>
</dependency>
```

Of course the version (`${hana.driver.version}`) needs to be adopted to your needs (Hana installtion in production, e.g. [2.4.64](#)). For an overview of available driver versions see [here](#).

36.2. Developer Usage

For your local development environment you will love the free [SAP HANA, Express Edition](#).

You can run HANA in several ways:

- On-premise
 - Via a [Docker image](#) (Linux only)
 - Via a pre-configured [virtual machine](#) (Windows, Linux, OS X)
 - Installed natively on your [local machine](#) (Linux only)
- In the cloud
 - Via a pre-configured machine on the [Google Cloud Platform](#)
 - Via a pre-configured machine in the [Microsoft Azure Cloud](#)
 - Via a pre-configured machine on [Amazon Web Services](#)

To get started with SAP HANA, Express Edition you can check out the [tutorials](#) at the [SAP Developer Center](#).

36.3. Pooling

See [Overriding the JDBC Connection Pool Settings](#).

36.4. Fuzzy Search

See <https://blogs.sap.com/2015/08/28/dynamism-of-fuzzy-search-in-sap-hana/> or the SAP HANA Search Developer Guide

37. Oracle RDBMS

This section contains hints for those who use Oracle RDBMS. Besides general hints about the driver there are tips for more tight integration with other Oracle features or products. However, if you work for a project where Oracle RDBMS is settled and not going to be replaced (you are in a vendor lock-in anyway), you might want to use even more from Oracle technology to take advantage from a closer integration.

37.1. XE

For local development you should setup Oracle XE (eXpress Edition). You need an oracle account, then you can download it from [here](#).

The most comfortable way to run it as needed is using docker. You can build your own docker image from the downloaded RPM using the [instructions and dockerfile from oracle](#). The following commands will build and start Oracle XE **18.4.0** on your machine:

```
git clone https://github.com/oracle/docker-images.git
cd docker-images/OracleDatabase/SingleInstance/dockerfiles
./buildDockerImage.sh -x -v 18.4.0
docker run -d -p 1521:1521 --name=oracle-xe --restart=always -e ORACLE_PWD=<>my-sys-
pwd<> oracle/database:18.4.0-xe
```

Please note that the `buildDockerImage.sh` will take a long time. Further after `docker run` has passed you need to give time for your new container to startup and setup the Oracle XE DB. So be patient and give it some time. (In case the build of the docker-image [fails reproducibly](#) and you want to give up with the Dockerfiles from Oracle you can also try this unofficial [docker-oracle-xe](#) solution. However, this is not recommended and may lead to other problems.).

Starting with XE 18c you need to be aware that oracle introduced a [multi-tenant architecture](#). Hence `xe` refers to the root `CDB` while you typically want to connect to the `PDB` (pluggable database) and XE ships with exactly one of this called `xepdb1`. To connect to your local XE database you need to use `xepdb1` as the `Service Name` (typically in `SQL Developer`). The `hostname` should be `localhost` and the `port` is by default `1521` if you did not remap it with docker to something else. In order to create schema users, use `sys` as `Username` and change `Role` to `SYSDBA`.

Hint: If you happen to end up connected to `xe` instead of `xepdb1` in some case (e.g. in `sqlplus`), you may switch using this statement:

```
ALTER SESSION SET CONTAINER = XEPDB1;
```

The JDBC URL for your Oracle XE Database is:

```
jdbc:oracle:thin:@//localhost:1521/xepdb1
```

To locally connect as sysdba without password use the following command (`connect / as sysdba` is not working anymore):

```
sqlplus sys/Oracle18@localhost/XE as sysdba
```

37.2. Driver

The oracle JDBC driver is [available in maven central](#). Oracle JDBC drivers usually are backward and forward compatible so you should be able to use an older driver with a newer Oracle DB, etc. Your dependency for the oracle driver should look as follows:

```
<dependency>
  <groupId>com.oracle.database.jdbc</groupId>
  <artifactId>ojdbc10</artifactId>
  <version>${oracle.driver.version}</version>
</dependency>
```

For the most recent Oracle DB 19 the property `oracle.driver.version` should be `19.8.0.0`. The number in the `artifactId` correlates to the minimum Java Version so for Java8 `artifactId` should be `ojdbc8` instead. It is fine to use `ojdbc10` with Java11 or higher.

37.3. Pooling

In order to boost performance JDBC connections should be pooled and reused. If you are using Oracle RDBMS and do not plan to change that you can use the Oracle specific connection pool "Universal Connection Pool (UCP)" that is perfectly integrated with the Oracle driver. According to the documentation, UCP can even be used to [manage third party data sources](#). Like the JDBC driver also the UCP is available in maven central. The dependency should look like this:

```
<dependency>
  <groupId>com.oracle.database.jdbc</groupId>
  <artifactId>ucp</artifactId>
  <version>${oracle.ucp.version}</version>
</dependency>
```

with property `oracle.ucp.version` analogue to `oracle.driver.version`.

Configuration is done via application.properties like this (example):

```

#Oracle UCP
# Datasource for accessing the database
spring.datasource.url=jdbc:oracle:thin:@192.168.58.2:1521:xe
spring.jpa.database-platform=org.hibernate.dialect.Oracle12cDialect
spring.datasource.user=MyUser
spring.datasource.password=ThisIsMyPassword
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
spring.datasource.schema=MySchema

spring.datasource.type=oracle.ucp.jdbc.PoolDataSourceImpl
spring.datasource.factory=oracle.ucp.jdbc.PoolDataSourceFactory
spring.datasource.factory-method=getPoolDataSource
spring.datasource.connectionFactoryClassName=oracle.jdbc.pool.OracleDataSource
spring.datasource.validateConnectionOnBorrow=true
spring.datasource.connectionPoolName=MyPool
spring.datasource.jmx-enabled=true

# Optional: Set the log level to INTERNAL_ERROR, SEVERE, WARNING, INFO, CONFIG, FINE,
TRACE_10, FINER, TRACE_20, TRACE_30, or FINEST
# logging.level.oracle.ucp=INTERNAL_ERROR
# Optional: activate tracing
# logging.level.oracle.ucp.jdbc.oracle.OracleUniversalPooledConnection=TRACE

#Optional: Configures pool size manually
#spring.datasource.minPoolSize=10
#spring.datasource.maxPoolSize=40
#spring.datasource.initialPoolSize=20

```

Resources: [FAQ](#), [developer's guide](#), [Java API Reference](#). For an in-depth discussion on how to use JDBC and UCP, see the Oracle documentation [Connection Management Strategies for Java Applications using JDBC and UCP](#).

Note: there is a bug in UCP 12.1.0.2 that results in the creation of thousands of `java.lang.Timer` threads over hours or days of system uptime (see [article on stackoverflow](#)). Also, Oracle has a strange bug fixing / patching policy: instead of producing a fixed version 12.1.0.3 or 12.1.0.2.x, Oracle publishes collections of *.class files that must be manually patched into the ucp.jar! Therefore, use the newest versions only.

37.4. Messaging

In case you want to do messaging based on JMS you might consider the [Oracle JMS](#) also called Oracle Streams Advanced Queuing, or Oracle Advanced Queuing, or OAQ or AQ for short. OAQ is a JMS provider based on the Oracle RDBMS and included in the DB product for no extra fee. OAQ has some features that exceed the JMS standard like a retention time (i.e. a built-in backup mechanism that allows to make messages "unread" within a configurable period of time so that these messages do not have to be resent by the sending application). Also, OAQ messages are stored in relational tables so they can easily be observed by a test driver in a system test scenario. Capgemini has used the [Spring Data JDBC Extension](#) in order to process OAQ messages within **the same technical**

transaction as the resulting Oracle RDBMS data changes **without** using 2PC and an XA-compliant transaction manager - which is not available out of the box in Tomcat. This is possible only due to the fact that OAQ queues and RDBMS tables actually reside in the same database. However, this is higher magic and should only be tried if high transaction rates must be achieved by avoiding 2PC.

37.5. General Notes on the use of Oracle products

Oracle sells commercial products and receives licence fees for them. This includes access to a support organization. Therefore, at an early stage of your project, prepare for contacting [oracle support](#) in case of technical problems. You will need the Oracle support ID **of your customer** [i.e. the legal entity who pays the licence fee and runs the RDBMS] and your customer must grant you permission to use it in a service request - it is not legal to use a your own support ID in a customer-related project. Your customer pays for that service anyway, so use it in case of a problem!

Software components like the JDBC driver or the UCP may be available without a registration or fee but they are protected by the Oracle Technology Network (OTN) License Agreement. The most important aspect of this licence agreement is the fact that an IT service provider is not allowed to simply download the Oracle software component, bundle it in a software artefact and deliver it to the customer. Instead, the Oracle software component must be (from a legal point of view) provided by the owner of the Oracle DB licence (i.e. your customer). This can be achieved in two ways: Advise your customer to install the Oracle software component in the application server as a library that can be used by your custom built system. Or, in cases where this is not feasible, e.g. in a OpenShift environment where the IT service provider delivers complete Docker images, you must advise your customer to (legally, i.e. documented in a written form) provide the Oracle software component to you, i.e. you don't download the software component from the Oracle site but receive it from your customer.

37.6. Fix for TNS-Listener issues

When switching networks (e.g. due to VPN) you might end up that your local Oracle XE stopps working with this error:

```
Listener refused the connection with the following error:  
ORA-12505, TNS:listener does not currently know of SID given in connect descriptor
```

While a reboot resolves this problem, it is a huge pain to reboot every time this error occurs as this wastes a lot of time. Therefore we suggest the following fix:

- Go to your oracle installation and open the folder `product/«version»/dbhomeXE/network/admin`.
- Edit the file `listener.ora` and change the value of the property `HOST` from your qualified hostname to `localhost (HOST = localhost)`.
- Edit the file `tnsnames.ora` and change the value of the `HOST` properties (two occurrences) from your qualified hostname to `localhost (HOST = localhost)`.
- Reboot your machine or (on windows) restart the service `OracleServiceXE` via `services.msc`.
- Now this problem should be gone forever and you can continue your work.

On older XE versions until 11g you could run the following SQL (`sqlplus / as sysdba @reset_tns_listener.sql`):

```
WHENEVER SQLERROR EXIT;
ALTER SYSTEM SET local_listener = '(ADDRESS = (PROTOCOL = TCP)(HOST = 127.0.0.1)(PORT
= 1521))';
ALTER SYSTEM REGISTER;
EXIT;
```

38. MS-SQL-Server

This section gives guidance and hints for those who use [Microsoft SQL Server](#) as [RDBMS](#).

38.1. Driver

Microsoft SQL Server is a commercial and professional product. However, the JDBC driver is MIT licensed and available in Maven Central what makes it easy to integrate. Your dependency for the driver should look as following:

```
<dependency>
    <groupId>com.microsoft.sqlserver</groupId>
    <artifactId>mssql-jdbc</artifactId>
    <version>${mssqlserver.driver.version}</version>
</dependency>
```

Of course the version (`${mssqlserver.driver.version}`) needs to be adopted to your needs (SQL Server installation in production and JDK version, e.g. [7.4.1.jre8](#)). For an overview of available driver versions see [here](#).

39. PostgreSQL

This section gives guidance and hints for those who use [PostgreSQL](#) as [RDBMS](#).

39.1. Driver

PostgreSQL is fully open-source. The driver is therefore available in maven central. Your dependency for the driver should look as following:

```
<dependency>
    <groupId>postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>${postgresql.driver.version}</version>
</dependency>
```

Of course the version (`${postgresql.driver.version}`) needs to be adopted to your needs (PostgreSQL installation in production and JDBC level suitable for your JDK, e.g. [9.1-901-1.jdbc4](#)). For an overview of available driver versions see [here](#).

40. MariaDB

This section gives guidance and hints for those who use [MariaDB](#) as [RDBMS](#).

40.1. Driver

MariaDB is fully open-source. The driver is therefore available in maven central. Your dependency for the driver should look as following:

```
<dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
    <version>${mariadb.driver.version}</version>
</dependency>
```

Of course the version (`${mariadb.driver.version}`) needs to be adopted to your needs (MariaDB installation in production and JDK version, e.g. [2.5.1](#)). For an overview of available driver versions see [here](#).

41. Database

For your business application with devonfw you need to choose the right database. In devonfw we are not biased for a particular product so you have the freedom of choice.

41.1. RDBMS

The classical and well-established form of a database is a relational database management system (RDBMS). In devonfw we recommend to use an RDBMS unless you have specific need. However, in case you have the need for big data, graph-data, BLOB focus, or schema-less dynamic data you can have a look at [NoSQL](#) options but be aware that these may be experimental and are not fully supported by devonfw.

41.1.1. Options

In devonfw we are not biased for a particular RDBMS so you have the freedom of choice. Here are the most common options:

- [SAP Hana](#) (high performance in-memory, many advanced features)
- [Oracle](#) (most established, well featured for enterprise)
- [PostgreSQL](#) (great open-source RDBMS)
- [MariaDB](#) (true OSS successor of MySQL)
- [MS SQL Server](#) (best choice for Microsoft and Windows dominated IT landscapes)

Please click on any of the above choices and go to the according guide to find specific details such as client/driver.

41.2. NoSQL

While not (yet) officially supported and recommended there are also interesting NoSQL (Not Only SQL) databases that could be an interesting alternative. Please be aware that you will typically not be able to use [JPA](#) (and hibernate). Further before choosing a NoSQL database you should check the following aspects:

- Is the database of choice reliable and mature enough for your project?
- Can the operators of your product support the database of choice properly (provisioning, administration, backup, scaling & clustering, monitoring, etc.)?
- Does the database of choice meet the requirements of your project (ACID vs. eventual consistency, CAP theorem)?

There are good reasons to choose a particular NoSQL database in specific cases (e.g. extreme demand for big-data, throughput or scaling). But as indicated by the questions above you need to be fully aware of what you are doing. NoSQL databases can be *schemaless* (untyped, dynamic & flexible) and/or *schemaful* (typed, structured & strict). Further, there are different types of NoSQL databases that are discussed in the following sub-sections:

41.2.1. Column DB

Column NoSQL databases are more related to a regular [RDBMS](#) with their tables and columns. However, they typically do not offer relational support with joins to the same level as you expect from an RDBMS. Therefore, you have to carefully design your data-model upfront with the all the knowledge how you later want to query your data.

The most prominent options are:

- [Cassandra](#) (high-performance, schema-based DB)
- [HBase](#) (distributed, big-data Hadoop database)

41.2.2. Key-Value DB

As indicated by the name, a key-value database stores objects as key/value pairs similar to [Properties](#) or [Map](#) in Java.

The most prominent options are:

- [Redis](#) (in-memory key/value store, especially used as cache or message broker)
- [aerospike](#)

41.2.3. Document DB

A document database is similar to a key-value database, but it stores objects in standard structured formats such as XML, JSON, or BSON. Therefore not only flat key/value pairs but even trees of hierarchical data can be stored, retrieved and queried.

The most prominent options are:

- [MongoDB](#)
- [CouchDB](#)
- [RavenDB](#)

41.2.4. Graph DB

If the connections (links/relations) between your data is key and an [RDBMS](#) is just not flexible or fast enough for your plans, then a graph database can help you. They are very strong on storing and querying complex connections between entities. For queries there are even specific standards and languages like [Gremlin](#).

The most prominent options are:

- [neo4j](#)
- [blazegraph](#)

41.2.5. Hybrid DB

In addition to the above types there are some NoSQL databases that are hybrid and combine the features and aspects of these types. While as an architect and developer you might love the idea to get all in one, you have to be careful with your choice. If you do not exactly know your problem, you are not ready to make the right choice for your database. Further, you might still be best-off with an good old [RDBMS](#) if you need to address multiple aspects together. Anyhow, for experiments, PoCs, or small microservices with little risk it might be a great idea to choose a hybrid NoSQL database. If you have collected very positive, profound and productive experience with such product you can grow on it.

The most prominent options are:

- [OrientDB](#) (object-oriented, hyper-flexible, column- and graph-based)

42. Cassandra

This section is the place to share experience for those who use [Cassandra](#) as [NoSQL database](#).

42.1. Attention



In devonfw we do not properly support this. You are free to use this database but do not expect advanced integration, CobiGen templates, etc.

42.2. Driver

Please be aware that there is not a regular JDBC driver in case you are using Java (devon4j). For driver options see [here](#).

42.3. Spring-Data

There is spring-data support available for cassandra via [spring-data-cassandra](#).



Please note that some time ago we had feedback from projects that had issues with spring-data-cassandra and switched back to using the driver natively. We assume the issues are meanwhile resolved. TODO: collect more feedback and update this guide.

43. neo4j

This section is the place to share experience for those who use [neo4j](#) as [NoSQL database](#).

43.1. Attention



In devonfw we do not properly support this. You are free to use this database but do not expect advanced integration, CobiGen templates, etc.

43.2. Driver

Please be aware that there is not a regular JDBC driver in case you are using Java (devon4j). For driver options see [here](#).

43.3. Spring-Data

There is spring-data integration available via [spring-data-neo4j](#).

44. MongoDB

This section is the place to share experience for those who use [MongoDB](#) as [NoSQL database](#).

44.1. Attention



In devonfw we do not properly support this. You are free to use this database but do not expect advanced integration, CobiGen templates, etc.

44.2. Driver

Please be aware that there is not a regular JDBC driver in case you are using Java (devon4j). For driver options see [here](#).

45. CouchDB

This section is the place to share experience for those who use [CouchDB](#) as [NoSQL database](#).

45.1. Attention



In devonfw we do not properly support this. You are free to use this database but do not expect advanced integration, CobiGen templates, etc.

45.2. Driver

Please be aware that there is not a regular JDBC driver in case you are using Java (devon4j). For driver options see [here](#).

46. Redis

This section is the place to share experience for those who use [Redis](#) as [NoSQL database](#).

46.1. Attention



In devonfw we do not properly support this. You are free to use this database but do not expect advanced integration, CobiGen templates, etc.

46.2. Driver

Please be aware that there is not a regular JDBC driver in case you are using Java (devon4j). For driver options see [here](#).

47. OrientDB

This section is the place to share experience for those who use [OrientDB](#) (see also [Open-Source community edition](#)) as [NoSQL database](#).

47.1. Attention



In devonfw we do not properly support this. You are free to use this database but do not expect advanced integration, CobiGen templates, etc.

47.2. Driver

For driver options see [here](#).

47.3. Administration

OrientDB comes with a powerful, impressive admin interface for your web-browser called [Studio](#).

48. Blazegraph

This section is the place to share experience for those who use [Blazegraph](#) as [NoSQL database](#).

48.1. Attention



In devonfw we do not properly support this. You are free to use this database but do not expect advanced integration, CobiGen templates, etc.

48.2. Driver

Please be aware that there is not a regular JDBC driver in case you are using Java (devon4j). For driver options see [here](#).

49. HBase

This section is the place to share experience for those who use [HBase](#) as [NoSQL database](#).

49.1. Attention



In devonfw we do not properly support this. You are free to use this database but do not expect advanced integration, CobiGen templates, etc.

49.2. Driver

Please be aware that there is not a regular JDBC driver in case you are using Java (devon4j). For driver options see [here](#) and [hbase-java-api tutorial](#).

50. RavenDB

This section is the place to share experience for those who use [RavenDB](#) as [NoSQL database](#).

50.1. Attention



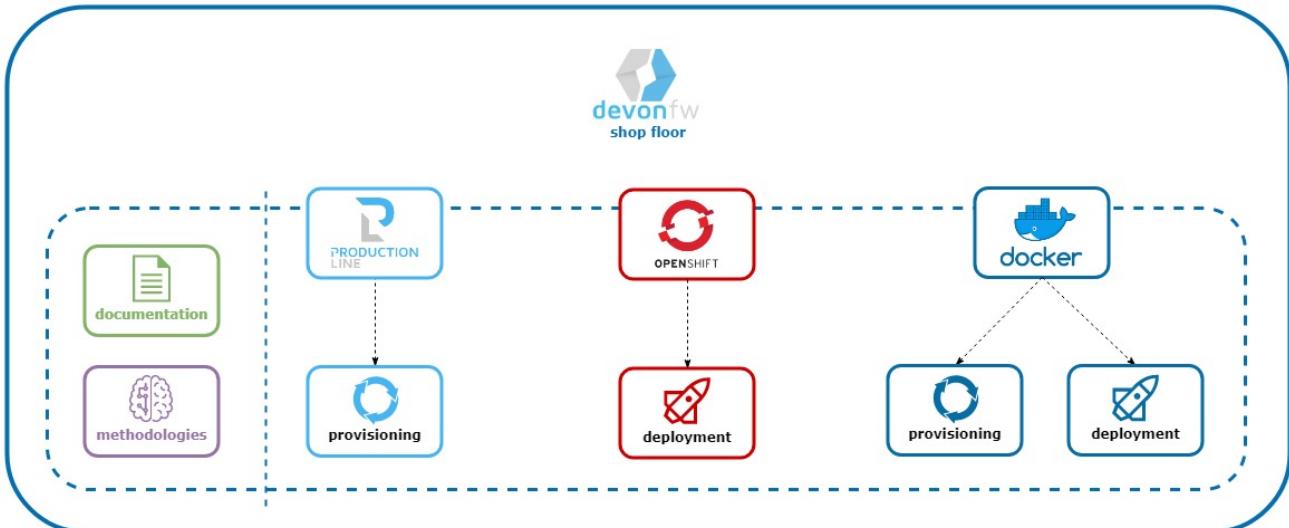
In devonfw we do not properly support this. You are free to use this database but do not expect advanced integration, CobiGen templates, etc.

50.2. Driver

Please be aware that there is not a regular JDBC driver in case you are using Java (devon4j). For driver options see [ravendb-jvm-client](#) and [Java Client Features](#).

Part VIII: devonfw shop floor

51. What is devonfw shop floor?



devonfw shop floor is a platform to industrialize continuous delivery and continuous integration processes.

devonfw shop floor is a set of documentation, tools and methodologies used to configure the provisioning, development and uat environments used in your projects. devonfw shop floor allows the administrators of those environments to apply CI/CD operations and enables automated application deployment.

devonfw shop floor is mainly oriented to configure the provisioning environment provided by Production Line and deploy applications on an OpenShift cluster. In the cases where Production Line or OpenShift cluster are not available, there will be alternatives to achieve similar goals.

The **devonfw shop floor 4 OpenShift** is a solution based on the experience of priming devonfw for OpenShift by RedHat.



**RED HAT® OPENSHIFT
PRIMED**

Let's start.

52. How to use it

This is the documentation about shop floor and its different tools. Here you are going to learn how to create new projects, so that they can include continuous integration and continuous delivery processes, and be deployed automatically in different environments.

52.1. Prerequisites - Provisioning environment

To start working you need to have some services running in your provisioning environment, such as Jenkins (automation server), GitLab (git repository), SonarQube (program analysis), Nexus (software repository) or similar.

To host those services we recommend to have a Production Line instance but you can use other platforms. Here is the list for the different options:

- [Production Line](#).
- [dsf4docker](#).

52.2. Step 1 - Configuration and services integration

The first step is configuring your services and integrate them with jenkins. Here you have an example about how to manually configure the next services:

- [Nexus](#).
- [SonarQube](#).

52.3. Step 2 - Create the project

52.3.1. Create and integrate git repository

The second is create or git repository and integrate it with Jenkins.

Here you can find a manual guide about how it:

- [GitLab](#) new project.

52.3.2. Start new devonfw project

It is time to create your devonfw project:

You can find all that you need about how to create a [new devonfw project](#)

52.3.3. cicd configuration

Now you need to add cicd files in your project.

Manual configuration

Jenkinsfile

Here you can find all that you need to know to do your [Jenkinsfile](#).

Dockerfile

Here you can find all that you need to know to do your [Dockerfile](#).

Automatic configuration

cicdgen

If you are using production line for provisioning you could use cicdgen to configure automatically almost everything explained in the manual configuration. To do it see the [cicdgen](#) documentation.

52.4. Step 3 - Deployment

The third is configure our deployment environment. Here is the list for the different options:

- [dsf4openshift](#).

52.5. Step 4 - Monitoring

Here you can find information about tools for monitoring:

- [build monitor view](#) for Jenkins. With this tool you will be able to see in real time what is the state of your Jenkins pipelines.

53. Provisioning environments

53.1. Production Line provisioning environment



The Production Line Project is a set of server-side collaboration tools for Capgemini engagements. It has been developed for supporting project engagements with individual tools like issue tracking, continuous integration, continuous deployment, documentation, binary storage and much more!

For additional information use the [official documentation](#).

53.1.1. How to obtain your Production Line

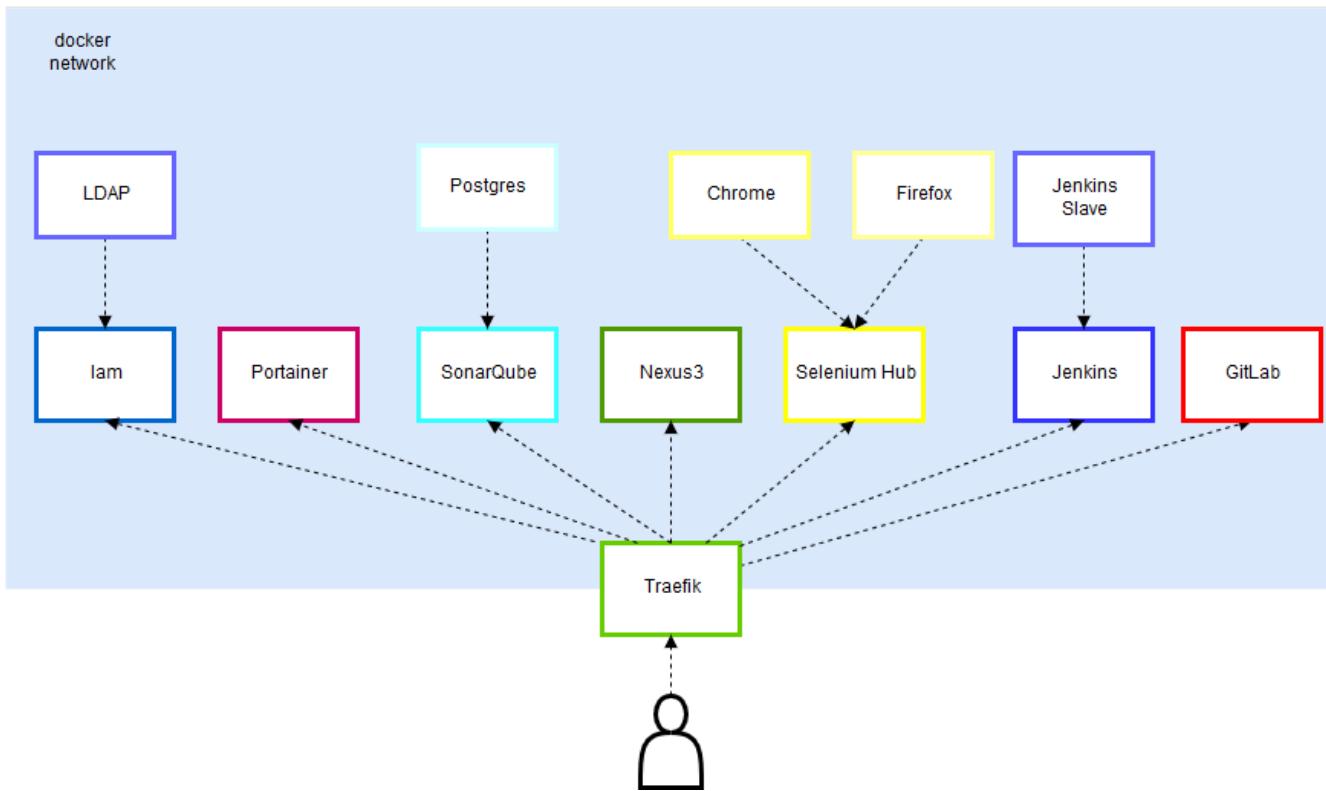
You can order your Production Line environment instance following the [official guide](#). Remember that you need to order at least the next tools: * Jenkins * GitLab * SonarQube * Nexus

[Back.](#)

53.2. dsf4docker provisioning environment



53.2.1. Architecture overview



53.2.2. Prerequisite

To use dsf4docker provisioning environment you need a remote server and you must clone or download devonfw shop floor.

53.2.3. How to use it

Navigate to `./devonfw-shop-floor/dsf4docker/environment` and here you can find one scripts to install it, and another one to uninstall it.

Install devonfw shop floor 4 Docker

There is an installation script to do so, so the complete installation should be completed by running it. Make sure this script has execution permissions in the Docker Host:

```
chmod +x dsf4docker-install.sh  
sudo ./dsf4docker-install.sh
```

This script, besides the container "installation" itself, will also adapt the `docker-compose.yml` file to your host (using `sed` to replace the `IP_ADDRESS` word of the file for your real Docker Host's IP address).

Uninstall devonfw shop floor 4 Docker

As well as for the installation, if we want to remove everything concerning **devonfw shop floor 4 Docker** from our Docker Host, we'll run this script:

```
chmod +x dsf4docker-uninstall.sh  
sudo ./dsf4docker-uninstall.sh
```

53.2.4. Troubleshooting

When trying to execute the install or uninstall .sh there may be some problems related to the windows/linux format file, so if you see this error log while executing the script:

```
./dsf4docker-install.sh: line 16: $'\r': command not found
```

You need to do a file conversion with this command:

```
dos2unix dsf4docker-install.sh
```

or

```
dos2unix dsf4docker-uninstall.sh
```

53.2.5. A little history

The **Docker** part of the shop floor is created based on the experience of the environment setup of the project **Mirabaud Advisory**, and intended to be updated to latest versions. Mirabaud Advisory is a web service developed with devonfw (Java) that, alongside its own implementation, it needed an environment both for the team to follow CICD rules through their 1-week-long sprints and for the client (Mirabaud) to check the already done work.

There is a practical experience about the [Mirabaud Case](#).

[Back](#).

54. Configuration and services integration

54.1. Nexus Configuration

In this document you will see how you can configure Nexus repository and how to integrate it with Jenkins.

54.1.1. Login in Nexus

The first time you enter in Nexus you need to log in with the user 'admin' and the password that is inside the path: /volumes/nexus/nexus-data Then you can change that password and create a new one.

54.1.2. Prerequisites

Repositories

You need to have one repository for snapshots, another for releases and another one for release-candidates. Normally you use maven2 (hosted) repositories and if you are going to use a docker registry, you need docker (hosted) too.

To create a repository in Nexus go to the administration clicking on the gear icon at top menu bar. Then on the left menu click on Repositories and press the **Create repository** button.

Name ↑	Type	Format	Status
docker-group	group	docker	Online
docker-hub	proxy	docker	Online - Re

Now you must choose the type of the repository and configure it. This is an example for Snapshot:



Repositories / Select Recipe / Create Repository: maven2 (hosted)

Name: A unique identifier for this repository

Online: If checked, the repository accepts incoming requests

Maven 2

Version policy:

What type of artifacts does this repository store?

Layout policy:

Validate that all paths are Maven artifact or metadata paths

Storage

Blob store:

Blob store used to store asset contents

Strict Content Type Validation:

Validate that all content uploaded to this repository is of a MIME type appropriate for the repository format

Hosted

Deployment policy:

Controls if deployments of and updates to artifacts are allowed

Create repository

Cancel

54.1.3. Create user to upload/download content

Once you have the repositories, you need a user to upload/download content. To do it go to the administration clicking on the gear icon at top menu bar. Then on the left menu click on Users and press the **Create local** user button.

Sonatype Nexus Repository Manager
OSS 3.12.1-01

Administration

- IQ Server
- Server
- Security
 - Privileges
 - Roles
 - Users**
 - Anonymous

Users Manage users

Create local user | Source: Local

User ID ↑	Realm
admin	default
anonymous	default
bptDev	default

Now you need to fill a form like this:

 **Users** /  Create User

ID:
This will be used as the username

First name:

Last name:

Email:
Used for notifications

Password:

Confirm password:

Status:

Roles:

Available	Granted
<input style="width: 100%; height: 20px; border: 1px solid #ccc; border-radius: 4px; padding: 2px 5px; font-size: 12px; color: #333; background-color: #f9f9f9; margin-bottom: 5px;" type="text" value="Filter"/> <div style="border: 1px solid #ccc; padding: 5px; height: 150px; overflow-y: auto;"> <ul style="list-style-type: none"> nx-anonymous Others Users </div>	<input style="width: 100%; height: 20px; border: 1px solid #ccc; border-radius: 4px; padding: 2px 5px; font-size: 12px; color: #333; background-color: #f9f9f9; margin-bottom: 5px;" type="text" value="Admins"/> <div style="border: 1px solid #ccc; padding: 5px; height: 150px; overflow-y: auto;"> <ul style="list-style-type: none"> Admins <li style="background-color: #e0e0e0;">nx-admin </div>
<input style="border: 1px solid #ccc; border-radius: 4px; padding: 2px 10px; font-size: 12px; color: #333; background-color: #f9f9f9; margin-right: 5px;" type="button" value=">"/> <input style="border: 1px solid #ccc; border-radius: 4px; padding: 2px 10px; font-size: 12px; color: #333; background-color: #f9f9f9; margin-left: 5px;" type="button" value="<"/>	

54.1.4. Jenkins integration

To use Nexus in our pipelines you need to configure Jenkins.

Customize jenkins

The first time you enter jenkins, you are asked for the pluggins to be installed. We select *install suggested plugins* and later we can add the plugins that we need depending on the project necessities.

Getting Started X

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

Then we need to create our first admin user, we can do it like this:

Getting Started

Create First Admin User

Username:

Password:

Confirm password:

Full name:

E-mail address:

The next step is the jenkins URL:

Jenkins URL: <http://jenkins.10.36.39.36.nip.io/>

Your jenkins setup is ready!

Add nexus user credentials

First of all you need to add the user created in the step before to Jenkins. To do it (on the left menu) click on Credentials, then on System. Now you could access to **Global credentials (unrestricted)**.

The screenshot shows the Jenkins System page. On the left, there is a sidebar with various links: New Item, People, Build History, Project Relationship, Check File Fingerprint, Selenium Grid, Manage Jenkins, My Views, Disk Usage, Job Config History, Open Blue Ocean, Credentials, System, and Add domain. The 'Credentials' link is highlighted with a yellow box. The main content area has a title 'System' with a house icon. Below it, there is a table with one row labeled 'Domain'. The row contains an icon of a castle, the text 'Global credentials (unrestricted)', and a 'Credentials' button. Below the table, it says 'Icon: S M L'.

Enter on it and you could see a button on the left to **Add credentials**. Click on it and fill a form like this:

The screenshot shows the 'Add Credentials' form for a 'Username with password' credential. The form fields are: Kind (Username with password), Scope (Global (Jenkins, nodes, items, all child items, etc)), Username (nexus-api), Password (*****), ID (nexus), and Description (Nexus credentials.). There is an 'OK' button at the bottom.

Add the nexus user to maven global settings

In order to do this, you will need the **Config File Provider** plugin so we need to download it. Go to Jenkins → Manage Jenkins → Manage plugins and "available" tab and search for it:

The screenshot shows the Jenkins Manage Plugins page. The 'Available' tab is selected. A search bar at the top right is set to 'Filter: config file prov'. A table lists the 'Config File Provider' plugin, which is checked for installation. The table columns are Name, Version, and Action buttons. The 'Config File Provider' row has a note: 'Ability to provide configuration files (e.g. settings.xml for maven, XML, groovy, custom files,...) loaded through the UI which will be copied to the job workspace.' At the bottom, there are buttons for 'Install without restart', 'Download now and install after restart', and 'Check now'.

Click on "Download now and install after restart".

Now you need to go to Manage Jenkins clicking on left menu and enter in **Managed files**.

Click on **Add a new config/Global Maven settings.xml**, change the id for a new one more readable:

 Manage Jenkins
 Config Files
 Add a new Config

Type

Select the file type you want to create

Global Maven settings.xml

A global maven settings.xml which can be referenced within Apache Maven jobs.
 Use it within maven projects or maven builder and reference credentials for a server authentication from here: [credentials](#)

Maven settings.xml

A settings.xml which can be referenced within Apache Maven jobs.
 Use it within maven projects or maven builder and reference credentials for a server authentication from here: [credentials](#)

Properties file

a Properties file [credentials](#)

Json file

a Json file

Maven toolchains.xml

a toolchains.xml which can be referenced within Apache Maven jobs

Simple XML file

a general xml file

Groovy file

a reusable groovy script

Custom file

a custom file (e.g. text or any other not yet available format)

Extended Email Publisher Groovy Template

A Groovy template used by the Extended Email Publisher plugin to generate emails.

Extended Email Publisher Jelly Template

A Jelly template used by the Extended Email Publisher plugin to generate emails.

ID

ID of the config file

Submit

Then click on "**Submit**"

You can edit or remove your configuration files

Global Maven settings.xml



MyGlobalSettings

global settings

Edit the Global Maven settings.xml to add your nexus repositories credentials(the ones you created before) as you could see in the next image:

The configuration	
ID	<input type="text" value="MavenSettings"/>
Name	<input type="text" value="MyGlobalSettings"/>
Comment	<input type="text" value="global settings"/>
Replace All	<input checked="" type="checkbox"/>
Server Credentials	ServerId: <input type="text" value="pl-nexus"/> ? Credentials: <input type="text" value="nexus-api***** (nexus-api)"/> Delete ? Add ▾

And you are done.

54.2. SonarQube Configuration

To use SonarQube you need to use a **token** to connect, and to know the results of the analysis you need a **webhook**. Also, you need to install and configure SonarQube in Jenkins.

54.2.1. Generate user token

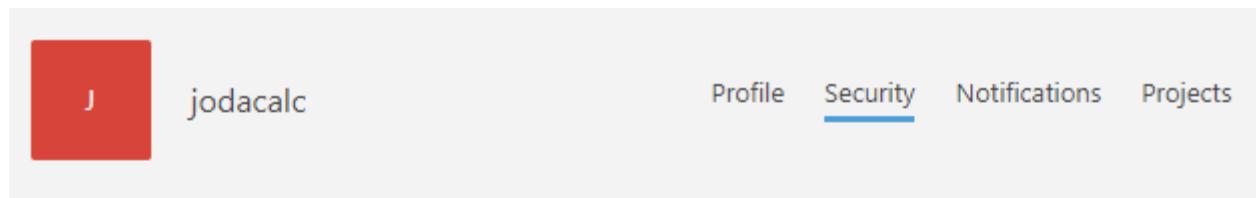
To generate the user token, go to your account clicking in the left icon on the top menu bar.



If you don't have any account, you can use the admin/admin user/pass

The screenshot shows the SonarQube administration interface. At the top, there's a navigation bar with links for Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and a search bar. On the right side of the header, there's a user profile for 'jodacalc' with an email 'jodacalc@capgemini.com'. Below the header, the main content area has a sidebar with 'Administration' and 'Configuration' sections. In the 'General Settings' section, there's a note about editing global settings for the instance. On the right, there's a 'My Account' dropdown menu with options like 'Log out'.

Go to security tab and generate the token.



The screenshot shows the SonarQube user profile for 'jodacalc'. At the top, there's a red square icon with a white letter 'J', the name 'jodacalc', and tabs for Profile, Security (which is underlined), Notifications, and Projects. The 'Security' tab is active, showing a table of tokens.

Tokens

If you want to enforce security by not providing credentials of a real SonarQube user to run your code scan or to invoke web services, you can provide a User Token as a replacement of the user login. This will increase the security of your installation by not letting your analysis user's password going through your network.

Name	Created	
jenkins	October 10, 2018	Revoke
Generate New Token	<input type="text" value="Enter Token Name"/>	Generate

54.2.2. Webhook

When you execute our SonarQube scanner in our pipeline job, you need to ask SonarQube if the quality gate has been passed. To do it you need to create a webhook.

Go to administration clicking the option on the top bar menu and select the tab for Configuration.

Then search in the left menu to go to webhook section and create your webhook.

An example for Production Line:

The screenshot shows the SonarQube Administration interface. The top navigation bar includes links for sonarqube, Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration, and a search bar. Below the navigation is a secondary menu with Configuration, Security, Projects, System, and Marketplace.

The main content area is titled "General Settings" with the sub-instruction "Edit global settings for this SonarQube instance." On the left, a sidebar lists "Analysis Scope" options: Webhooks, Dependency-Check, Flex, General, Java, JavaScript, PHP, and Python. The "Webhooks" option is selected and expanded, showing its configuration details.

The "Webhooks" configuration section includes a "Name" field (set to "jenkins") and a "URL" field (set to "http://jenkins-core:8080/jenkins/sonarqube-webhook/"). There is also a "Reset" button and a note stating "Default: <no value>".

54.2.3. Jenkins integration

To use SonarQube in our pipelines you need to configure Jenkins to integrate SonarQube.

SonarQube Scanner

First, you need to configure the scanner. Go to Manage Jenkins clicking on left menu and enter in ***Global Tool Configuration***.

Go to SonarQube Scanner section and add a new SonarQube scanner like this.

The screenshot shows the Jenkins Global Tool Configuration page under the "Manage Jenkins" section. The left sidebar has a "SonarQube Scanner installations" link. The main content area shows a "SonarQube Scanner" configuration card.

The card includes a "Name" field set to "SonarQube-scanner", a checked "Install automatically" checkbox, and an "Install from Maven Central" section with a dropdown set to "SonarQube Scanner 3.2.0.1227". There are "Delete Installer" and "Delete SonarQube Scanner" buttons at the bottom right.

Below the card is a "Add Installer" dropdown and a "Add SonarQube Scanner" button. A note at the bottom states "List of SonarQube Scanner installations on this system".

SonarQube Server

Now you need to configure where is our SonarQube server using the user token that you create before. Go to Manage Jenkins clicking on left menu and enter in ***Configure System***.

For example, in Production Line the server is the next:

SonarQube servers

Environment variables

Enable injection of SonarQube server configuration as build environment variables

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

SonarQube installations

Name

SharesSonar

Server URL

http://sonarqube-core:9000/sonarqube

Default is http://localhost:9000

Server authentication token

.....

SonarQube authentication token. Mandatory when anonymous access is disabled.

Advanced...**Delete SonarQube****Add SonarQube**

List of SonarQube installations



Remember, the token was created at the beginning of this SonarQube configuration.

54.2.4. SonarQube configuration

Now is time to configure your sonar in order to measure the quality of your code. To do it, please follow the official documentation about our plugins and Quality Gates and Profiles [here](#).

How to ignore files

Usually the developers need to ignore some files from Sonar analysis. To do that, they must add the next line as a parameter of the sonar execution to their Jenkinsfile in the SonarQube code analysis step.

```
-Dsonar.exclusions='**/*.spec.ts, **/*.model.ts, **/*mock.ts'
```

55. Create project

55.1. Create and integrate git repository

55.1.1. GitLab Configuration

Create new repository

To create a new project in GitLab, go to your dashboard and click the green *New project* button or use the plus icon in the navigation bar.

The screenshot shows the top navigation bar of the GitLab interface. It includes a logo, a search bar, and several menu items like 'Projects', 'Groups', 'Activity', 'Milestones', and 'Snippets'. A prominent green button labeled '+ New project' is located in the top right corner. Below the navigation bar, there's a main content area with tabs for 'Your projects', 'Starred projects', and 'Explore projects'. On the right side of this area, there are search and filter options, including a dropdown for 'Last updated' and a green 'New project' button.

This opens the New project page. Choose your group and fill the name of your project, the description and the visibility level in the next form:

The screenshot shows the 'New project' form. On the left, there's a sidebar with information about what a project is and how features are enabled. The main form has three tabs: 'Blank project' (selected), 'Create from template', and 'Import project'. The 'Blank project' tab contains fields for 'Project path' (set to 'https://shared-services.pls2-eu.capgemini.com/gitlab/jodacalc'), 'Project name' ('my-awesome-project'), and 'Project description (optional)'. Below these are sections for 'Description format' (with a dropdown menu showing 'Groups' highlighted) and 'Visibility Level' (with radio buttons for 'Private', 'Internal' (selected), and 'Public'). At the bottom are 'Create project' and 'Cancel' buttons.



more information about how to create projects in [GitLab in the official documentation](#)

Service integration

To learn how to configure the integration between GitLab and Jenkins see the next [example](#)

55.2. start new devonfw project

It is time to create your devonfw project:

55.2.1. How to create new devonfw project

Here you can find the official guides to start new devonfw projects:

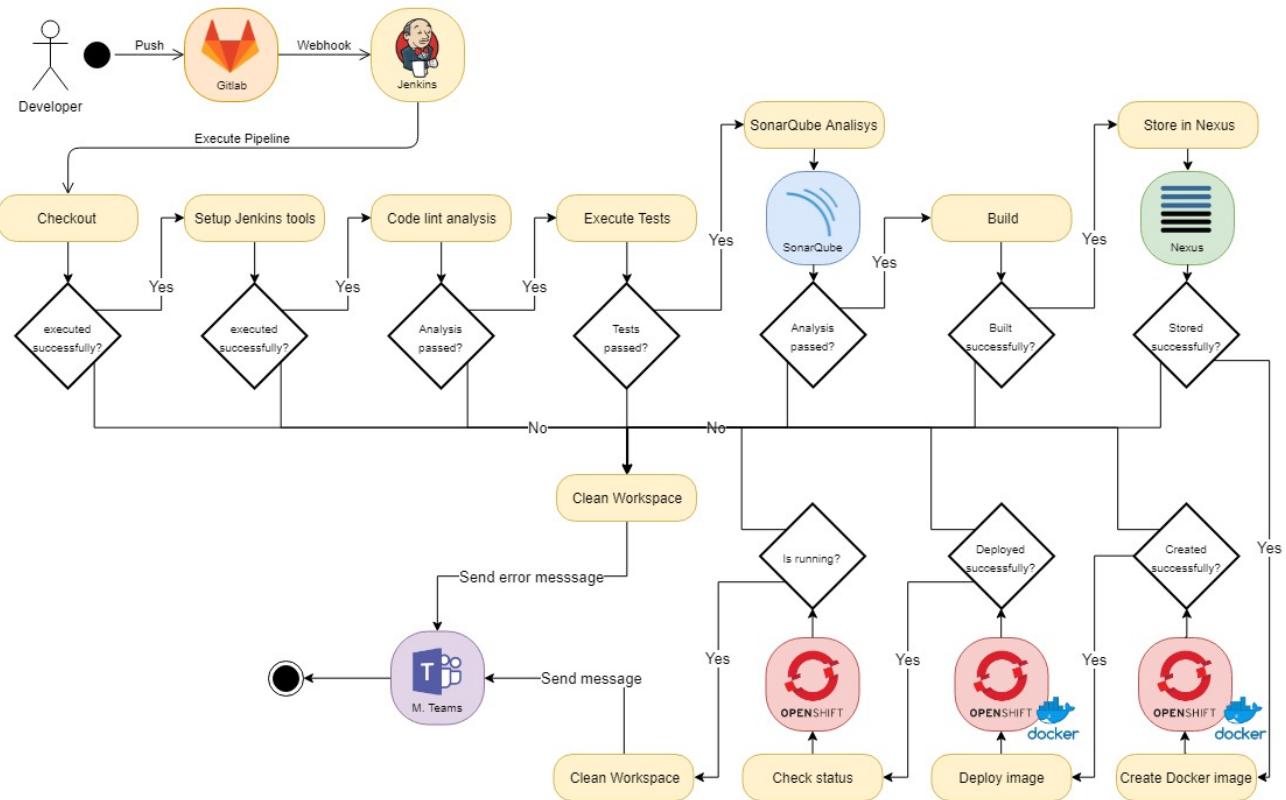
- visit our [devon4ng](#) guide.
- visit our [devon4j](#) guide.

55.3. cicd configuration

55.3.1. Manual configuration

Jenkinsfile

Introduction



Here you are going to learn how you should configure the jenkinsfile of your project to apply CI/CD operations and enables automated application deployment.

Here you can find examples of the Jenkinsfile generated by cicdgen:

- [devon4](#)
- [devon4ng](#)
- [devon4node](#)

Next you could find an explanation about what is done in these Jenkinsfiles.

Environment values

At the top of the pipeline you should add the environment variables. in this tutorial you need the next variables:

```
// sonarQube
// Name of the sonarQube tool
sonarTool = 'SonarQube'
// Name of the sonarQube environment
sonarEnv = "SonarQube"

// Nexus
// Artifact groupId
groupId = '<%= groupid %>'
// Nexus repository ID
repositoryId = 'pl-nexus'
// Nexus internal URL
repositoryUrl = 'http://nexus3-core:8081/nexus3/repository/'
// Maven global settings configuration ID
globalSettingsId = 'MavenSettings'
// Maven tool id
mavenInstallation = 'Maven3'

// Docker registry
dockerRegistry = 'docker-registry-<%= plurl %>'
dockerRegistryCredentials = 'nexus-docker'
dockerTool = 'docker-global'

// OpenShift
openshiftUrl = '<%= ocurl %>'
openShiftCredentials = 'openshift'
openShiftNamespace = '<%= ocn %>'
```

Stages

The pipeline consists of stages, and at the beginning of each stage it is declared for which branches the step will be executed.

```
stages {  
  
    stage ('Setup Jenkins Tools & install dependencies') {  
        when {  
            anyOf {  
                branch 'master'  
                branch 'develop'  
                branch 'release/*'  
            }  
        }  
    }  
}
```

Now it is time to create the stages.

Setup Jenkins tools

The first stage is one of the most dangerous, because in it on one hand the tools are added to the pipeline and to the path and on other hand the values are tagged depending on the branch that is being executed. If you are going to create a ci/cd for a new branch or you are going to modify something, be very careful with everything that this first step declares.

This is an example of this stage:

```

script {
    tool yarn
    tool Chrome-stable
    tool dockerTool

    if (env.BRANCH_NAME.startsWith('release')) {
        dockerTag = "release"
        repositoryName = 'maven-releases'
        dockerEnvironment = "_uat"
        openShiftNamespace += "-uat"
        sonarProjectKey = '-release'
    }

    if (env.BRANCH_NAME == 'develop') {
        dockerTag = "latest"
        repositoryName = 'maven-snapshots'
        dockerEnvironment = "_dev"
        openShiftNamespace += "-dev"
        sonarProjectKey = '-develop'
    }

    if (env.BRANCH_NAME == 'master') {
        dockerTag = "production"
        repositoryName = 'maven-releases'
        dockerEnvironment = '_prod'
        openShiftNamespace += "-prod"
        sonarProjectKey = ''
    }

    sh "yarn"
}

```

Code lint analysis

The next stage is to analyze the code making a lint analysis. To do it your project should have a tslint file with the configuration (*tslint.json*).

analyzing the code in your pipeline is as simple as executing the following command:

```
sh """yarn lint"""
```



Your project need to have an script with tslint configuration (*tslint.json*).

Execute tests

To test you application first of all your application should have created the tests and you should use one of the next two options:

Execute test with maven (It should be used by devon4j).

```
withMaven(globalMavenSettingsConfig: globalSettingsId, maven: mavenInstallation) {
    sh "mvn clean test"
}
```

Execute test with yarn (It should be used by devon4ng or devon4node).

```
sh """yarn test:ci"""
```



Remember that your project should have the tests created and in case of do it with yarn or npm, you package.json should have the script declared. This is an example
`"test:ci": "ng test --browsers ChromeHeadless --watch=false".`

SonarQube Analisys

It is time to see if your application complies the requirements of the sonar analysis.

To do it you could use one of the next two options:

Execute Sonar with sonarTool (It should be used by devon4ng or devon4node).

```
script {
    def scannerHome = tool sonarTool
    def props = readJSON file: 'package.json'
    withSonarQubeEnv(sonarEnv) {
        sh """
            ${scannerHome}/bin/sonar-scanner \
                -Dsonar.projectKey=${props.name}${sonarProjectKey} \
                -Dsonar.projectName=${props.name}${sonarProjectKey} \
                -Dsonar.projectVersion=${props.version} \
                -Dsonar.sources=${srcDir} \
                -Dsonar.typescript.lcov.reportPaths=coverage/lcov.info
        """
    }
    timeout(time: 1, unit: 'HOURS') {
        def qg = waitForQualityGate()
        if (qg.status != 'OK') {
            error "Pipeline aborted due to quality gate failure: ${qg.status}"
        }
    }
}
```

Execute Sonar with maven (It should be used by devon4j).

```

script {
    withMaven(globalMavenSettingsConfig: globalSettingsId, maven: mavenInstallation) {
        withSonarQubeEnv(sonarEnv) {
            // Change the project name (in order to simulate branches with the free
version)
            sh "cp pom.xml pom.xml.bak"
            sh "cp api/pom.xml api/pom.xml.bak"
            sh "cp core/pom.xml core/pom.xml.bak"
            sh "cp server/pom.xml server/pom.xml.bak"

            def pom = readMavenPom file: './pom.xml';
            pom.artifactId = "${pom.artifactId}${sonarProjectKey}"
            writeMavenPom model: pom, file: 'pom.xml'

            def apiPom = readMavenPom file: 'api/pom.xml'
            apiPom.parent.artifactId = pom.artifactId
            apiPom.artifactId = "${pom.artifactId}-api"
            writeMavenPom model: apiPom, file: 'api/pom.xml'

            def corePom = readMavenPom file: 'core/pom.xml'
            corePom.parent.artifactId = pom.artifactId
            corePom.artifactId = "${pom.artifactId}-core"
            writeMavenPom model: corePom, file: 'core/pom.xml'

            def serverPom = readMavenPom file: 'server/pom.xml'
            serverPom.parent.artifactId = pom.artifactId
            serverPom.artifactId = "${pom.artifactId}-server"
            writeMavenPom model: serverPom, file: 'server/pom.xml'

            sh "mvn sonar:sonar"

            sh "mv pom.xml.bak pom.xml"
            sh "mv api/pom.xml.bak api/pom.xml"
            sh "mv core/pom.xml.bak core/pom.xml"
            sh "mv server/pom.xml.bak server/pom.xml"
        }
    }
    timeout(time: 1, unit: 'HOURS') {
        def qg = waitForQualityGate()
        if (qg.status != 'OK') {
            error "Pipeline aborted due to quality gate failure: ${qg.status}"
        }
    }
}

```

Build

If SonarQube is passed, you could build your application. To do it, if you are using devon4ng or devon4node you only need to add the next command:

sh """/yarn build""""



If you are using devon4j this and the next step *Store in Nexus* are making together using `mvn deploy`.

Store in Nexus

One time the application has been built the code of the application you could find the artifacts stored in the dist folder. You should push these artifacts to store them in Nexus.

You can do it following one of the next options:

Use maven deploy config of your project (It should be used by devon4j).

```
withMaven(globalMavenSettingsConfig: globalSettingsId, maven: mavenInstallation) {  
    sh "mvn deploy -Dmaven.test.skip=true"  
}
```

Configure maven deploy in your pipeline (It should be used by devon4ng and devon4node).

```

script {
    def props = readJSON file: 'package.json'
    zip dir: 'dist/', zipFile: """${props.name}.zip"""
    version = props.version
    if (!version.endsWith("-SNAPSHOT") && env.BRANCH_NAME == 'develop') {
        version = "${version}-SNAPSHOT"
        version = version.replace("-RC", "")
    }

    if (!version.endsWith("-RC") && env.BRANCH_NAME.startsWith('release')) {
        version = "${version}-RC"
        version = version.replace("-SNAPSHOT", "")
    }

    if (env.BRANCH_NAME == 'master' && (version.endsWith("-RC") || version.endsWith("-SNAPSHOT"))){
        version = version.replace("-RC", "")
        version = version.replace("-SNAPSHOT", "")
    }

    withMaven(globalMavenSettingsConfig: globalSettingsId, maven: mavenInstallation) {
        sh """
            mvn deploy:deploy-file \
                -DgroupId=${groupId} \
                -DartifactId=${props.name} \
                -Dversion=${version} \
                -Dpackaging=zip \
                -Dfile=${props.name}.zip \
                -DrepositoryId=${repositoryId} \
                -Durl=${repositoryUrl}${repositoryName}
        """
    }
}

```

Create docker image

Now we need to use this artifacts to create a Docker image. To create the docker image you need an external server to do it. You could do it using one of the next:

Create docker image using OpenShift cluster

To create the docker image with this option you need to configure your OpenShift. You could read how to configure it [here](#).

```

props = readJSON file: 'package.json'
withCredentials([usernamePassword(credentialsId: "${openShiftCredentials}",
passwordVariable: 'pass', usernameVariable: 'user')]) {
    sh "oc login -u ${user} -p ${pass} ${openshiftUrl} --insecure-skip-tls-verify"
    try {
        sh "oc start-build ${props.name} --namespace=${openShiftNamespace} --from
-dir=dist --wait"
        sh "oc import-image ${props.name} --namespace=${openShiftNamespace}
--from=${dockerRegistry}/${props.name}:${dockerTag} --confirm"
    } catch (e) {
        sh """
            oc logs \$(oc get builds -l build=${props.name}
--namespace=${openShiftNamespace} --sort-by=.metadata.creationTimestamp -o name | tail
-n 1) --namespace=${namespace}
            throw e
        """
    }
}

```

 if your project is a maven project you should read the `pom.xml` file instead of the `package.json`, you could do it with the next command `def pom = readMavenPom file: 'pom.xml'`. Due to the fact that there are different variable names between those two files, remember to modify `${props.name}` for `${pom.artifactId}` in the code.

Create docker image using docker server

To create the docker image with this option you need to install docker and configure where is the docker host in your jenkins.

```

docker.withRegistry("""${dockerRegistryProtocol}${dockerRegistry}""",
dockerRegistryCredentials) {
    def props = readJSON file: 'package.json'
    def customImage = docker.build("${props.name}:${props.version}", "-f
${dockerFileName} .")
    customImage.push()
    customImage.push(dockerTag);
}

```

here

 if your project is a maven project you should read the `pom.xml` file instead of the `package.json`, you could do it with the next command `def pom = readMavenPom file: 'pom.xml'`. Due to the fact that there are different variable names between those two files, remember to modify `${props.name}` for `${pom.artifactId}` and `${props.version}` for `${pom.version}` in the code.

Deploy docker image

Once you have the docker image in the registry we only need to import it into your deployment environment. We can do it executing one of the next commands:

Deploy docker image in OpenShift cluster

To deploy the docker image with this option you need to configure your OpenShift. You could read how to configure it [here](#).

```
script {
    props = readJSON file: 'package.json'
    withCredentials([usernamePassword(credentialsId: "${openShiftCredentials}", passwordVariable: 'pass', usernameVariable: 'user')]) {
        sh "oc login -u ${user} -p ${pass} ${openshiftUrl} --insecure-skip-tls-verify"
        try {
            sh "oc import-image ${props.name} --namespace=${openShiftNamespace} --from=${dockerRegistry}/${props.name}:${dockerTag} --confirm"
        } catch (e) {
            sh """
                oc logs \$(oc get builds -l build=${props.name} --namespace=${openShiftNamespace} --sort-by=.metadata.creationTimestamp -o name | tail -n 1) --namespace=${openShiftNamespace}
                throw e
            """
        }
    }
}
```



if your project is a maven project you should read the `pom.xml` file instead of the `package.json`, you could do it with the next command `def pom = readMavenPom file: 'pom.xml'`. Due to the fact that there are different variable names between those two files, remember to modify `/${props.name}` for `/${pom.artifactId}` in the code.

Deploy docker image using docker server

To deploy the docker image with this option you need to install docker and configure your docker server and also integrate it with Jenkins.

```

script {
    docker.withRegistry("""${dockerRegistryProtocol}${dockerRegistry}""",
    dockerRegistryCredentials) {
        def props = readJSON file: 'package.json'
        docker.image("${props.name}:${props.version}").pull()

        def containerId = sh returnStdout: true, script: """docker ps -aqf
"name=${containerName}${dockerEnvironment}" """
        if (containerId?.trim()) {
            sh "docker rm -f ${containerId.trim()}"
        }

        println """docker run -d --name ${containerName}${dockerEnvironment}
--network=${networkName} ${dockerRegistry}/${props.name}:${props.version}"""
        sh """docker run -d --name ${containerName}${dockerEnvironment}
--network=${networkName} ${dockerRegistry}/${props.name}:${props.version}"""
    }
}

```



if your project is a maven project you should read the *pom.xml* file instead of the *package.json*, you could do it with the next command `def pom = readMavenPom file: 'pom.xml'`. Due to the fact that there are different variable names between those two files, remember to modify `${props.name}` for `${pom.artifactId}` and `${props.version}` for `${pom.version}` in the code.

Check status

Now is time to check if your pods are running ok.

To check if your pods are ok in OpenShift you should add the next code to your pipeline:

```

script {
    props = readJSON file: 'package.json'
    sleep 30
    withCredentials([usernamePassword(credentialsId: "${openShiftCredentials}",
passwordVariable: 'pass', usernameVariable: 'user')]) {
        sh "oc login -u ${user} -p ${pass} ${openshiftUrl} --insecure-skip-tls-verify"
        sh "oc project ${openShiftNamespace}"

        def oldRetry = -1;
        def oldState = "";

        sh "oc get pods -l app=${props.name} > out"
        def status = sh (
            script: "sed 's/[\\t ]*[\\t ]*/ /g' < out | sed '2q;d' | cut -d' ' -f3",
            returnStdout: true
        ).trim()

        def retry = sh (
            script: "sed 's/[\\t ]*[\\t ]*/ /g' < out | sed '2q;d' | cut -d' ' -f4",
            returnStdout: true
        ).trim().toInteger();

        while (retry < 5 && (oldRetry != retry || oldState != status)) {
            sleep 30
            oldRetry = retry
            oldState = status

            sh """oc get pods -l app=${props.name} > out"""
            status = sh (
                script: "sed 's/[\\t ]*[\\t ]*/ /g' < out | sed '2q;d' | cut -d' ' -f3",
                returnStdout: true
            ).trim()

            retry = sh (
                script: "sed 's/[\\t ]*[\\t ]*/ /g' < out | sed '2q;d' | cut -d' ' -f4",
                returnStdout: true
            ).trim().toInteger();
        }

        if(status != "Running"){
            try {
                sh """oc logs \$(oc get pods -l app=${props.name} --sort
-by=.metadata.creationTimestamp -o name | tail -n 1)"""
            } catch (e) {
                sh "echo error reading logs"
            }
            error("The pod is not running, cause: " + status)
        }
    }
}

```

Post operations

When all its finish, remember to clean your workspace.

```
post { cleanup { cleanWs() } }
```



You could also delete your dir adding the next command `deleteDir()`.

Dockerfile

You have examples of dockerfiles in cicdgen repository.

inside these folders you could find all the files that you need to use those dockerfiles. Two dockerfiles are provided, *Dockerfile* and *Dockerfile.ci*, the first one is to compile the code and create the docker image used normally in local, and *Dockerfile.ci* is to use in Jenkins or similar, after building the application.

- visit our [devon4ng Dockerfiles](#).
- visit our [devon4j Dockerfiles](#).
- visit our [devon4node Dockerfiles](#).



Dockerfile.ci should be copied to de artifacts and renamed as *Dockerfile* to work. In the case of *devon4ng* and *devon4node* this is the `dist` folder, in case of *devon4ng* is on `server/target` folder.

55.3.2. Automatic configuration

cicdgen

If you are using production line for provisioning you could use cicdgen to configure automatically almost everything explained in the manual configuration. To do it see the [cicdgen](#) documentation.

56. Deployment environments

56.1. OpenShift

56.1.1. dsf4openshift deployment environment

In this section you will see how you can create a new environment instance in OpenShift and the things that you must add to the Jenkinsfiles of your repository to deploy a branch in this new environment. To conclude you are going to see how to add config files for environment in the source code of the applications.

Configure your OpenShift to deploy your devonfw projects

Prerequisites

OpenShift Cluster

To have your deployment environment with OpenShift you need to have an OpenShift Cluster.

Manual configuration

Here you can find all that you need to know to [configure OpenShift](#) manually.

Automatic configuration

Here you can find all that you need to know to [configure OpenShift](#) automatically.

Service integration with jenkins

Prerequisites

To integrate it, you need to have installed the plugin OpenShift Client. To install it go to Manage Jenkins clicking on left menu and enter in **Manage Plugins**. Go to Available tab and search it using the filter textbox in the top right corner and install it.

Configuration

Second, you need to configure the OC Client. Go to Manage Jenkins clicking on left menu and enter in **Global Tool Configuration**.

Go to OpenShift Client Tools section and add a new one like this.

OpenShift Client Tools

OpenShift Client Tools installations

Name	OpenShiftv3.11.0	(?)
<input checked="" type="checkbox"/> Install automatically	(?)	
Run Shell Command		
Label	(?)	
Command	<pre>if [! -f /opt/oc3.11.0/oc]; then echo "oc3.11.0 does not exist." wget https://github.com/openshift/origin/releases/download/v3.11.0/openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit.tar.gz tar -xvfz openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit.tar.gz ls ls openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit sudo mv openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit /opt/oc3.11.0 else echo "oc3.11.0 exist." fi</pre>	(?)
Tool Home	/opt/oc3.11.0	(?)

Upgrade your Jenkinsfile

Now it is time to add/upgrade the next stages in to your Jenkinsfile:

Add [create docker image](#) stage.

Add [deploy docker image](#) stage.

Add [check status](#) stage.

Upgrade [Setup Jenkins tools](#) stage.



Remember to upgrade your parameters to difference which environment is used per branch.

56.1.2. OpenShift deployment environment manual configuration

In this section you will see how you can create a new environment instance in your OpenShift cluster to deploy devonfw projects using docker images.

Prerequisites

devonfw project

You need to have a devonfw project in a git repository or a docker image uploaded to a docker registry.

Communication between components

Openshift must have access to docker registry.

Download OpenShift Client Tools

First of all you need to download the OpenShift client, you can find it [here](#).

Remember that what you need to download **oc Client Tools** and not *OKD Server*.



This tutorial has been made with the version 3.10.0 of the client, it is recommended to use the most current client, but if it does not work, it is possible that the instructions have become obsolete or that the OpenShift used needs another older/newer version of the client. To download a specific version of the client you can find here the [older versions](#) and the [version 3.10.0](#).

Add oc client to path

Once you have downloaded the client you have to add it to the **PATH** environment variable.

Log into OpenShift with admin account

You can log using a terminal and executing the next instructions:

```
oc login $OpenShiftUrl
```



You need a valid user to log in.

Select the project where you are going to create the environment

```
oc project $projectName
```

Add all the secrets that you need

For example, to create a secret for a nexus repository you should execute the next commands:

```
oc create secret docker-registry $nameForSecret --docker-server=${dockerRegistry}
--docker-username=${user} --docker-password=${pass} --docker-email=no-reply@email.com
```

Configure OpenShift

Configure builds to create docker image using OpenShift

If you need to create docker images of your projects you could use OpenShift to do it (*Of course only if you have enough rights*).

To do it, follow the next steps.

Create new builds configs

The first thing you need to do for create a new environment is prepare the buildconfigs for the front and for the middleware and rise default memory limits for the middleware. You can do it using a terminal and executing the next instructions:

These are a summary about the parameters used in our commands:

- **\${dockerRegistry}**: The url of the docker repository.

- **\${props.name}** : The name of the project (for example could be find on package.json)
- **\${dockerTag}** : The tag of the image



From now on you will refer to the name that you are going to give to the environment as **\$environment** . Remember to modify it for the correct value in all instructions.

devon4ng build config

You need to create nginx build config with docker.

```
oc new-build --strategy docker --binary --docker-image nginx:alpine-perl
--name=${props.name}-$environment --to=${dockerRegistry}/${props.name}:${dockerTag}
--to-docker=true
```



You need nginx:alpine-perl to read the environment config file in openshift, if you are not going to use it, you could use nginx:latest instead.

devon4node build config

```
oc new-build --strategy docker --binary --docker-image node:lts --name=${props.name}
-$environment --to=${dockerRegistry}/${props.name}:${dockerTag} --to-docker=true
```

devon4j build config

```
oc new-build --strategy docker --binary --docker-image openjdk:<version>
--name=${props.name}-$environment --to=${dockerRegistry}/${props.name}:${dockerTag}
--to-docker=true
```



You need to specify the <version> of java used for your project. Also you can use the -alpine image. This image is based on the popular [Alpine Linux project](#). Alpine Linux is much smaller than most distribution base images (~5MB), and thus leads to much slimmer images in general. More information on [docker hub](#).

How to use the build

In this step is where you will build a docker image from a compiled application.

Prerequisite

To build the source in OpenShift, first of all you need to compile your source and **obtain the artifacts "dist folder"** or download it from a repository. Normally the artifacts have been built on Jenkins and have been stored in Nexus.

To download it, you can access to your registry, select the last version and download the ".tar". The

next image shows an example of where is the link to download it, marked in yellow:

The screenshot shows the Sonatype Nexus Repository Manager interface. On the left, there's a navigation sidebar with 'Browse', 'Search', 'Upload', and 'Logout'. The main area shows a hierarchical file tree under 'com/capgemini/insereng/api'. A yellow box highlights the download link for a file named '12.25.3-32-excluded.tar'.

Summary

Repository	snapshots
Format	Maven2
Component Group	com.capgemini.insereng
Component Name	Insereng-api
Component Version	0.6.0-20190115.12253.32
Path	com/capgemini/insereng/api/0.6.0-SNAPSHOT/Insereng-api-0.6.0-20190115.12253.32/12.25.3-32-excluded.tar

Content type: application/x-tar
File size: 11 MB
Blob created: Tue Jan 15 2019 12:22:54 GMT+0100 (hora estàndard de Europa central)
Blob updated: Tue Jan 15 2019 12:22:54 GMT+0100 (hora estàndard de Europa central)
Last 30 days: 0 downloads
Last downloaded: Tue Jan 15 2019
Locally cached: true
Blob reference: daf5fb27702ff6f24f3ad9b-20f3e206-f476fd0-27fd7fec-4efabcc47a1-47
Containing repo: snapshots
Uploader: datplayer
Uploader's IP Address: 172.16.201
Most popular version: 0.6.0-20190115.12253.32
Age: 0 days
Popularity: 0

Attributes

Build in OpenShift

When you have the artifacts, you can send them to your openshift and build them using your buildconfig that you created on the previous step. This is going to create a new docker image and push it to your registry.

If your docker registry need credentials you should use a secret. You could add it to your buildconfig using the next command:

```
oc set build-secret --push bc/${props.name}-$environment ${nameForSecret}
```

Now you can use your build config and push the docker image to your registry. To do it you need to use a terminal and execute the following:

```
oc start-build ${props.name}-$environment --from-dir=${artifactsPath} --follow
```



`\${artifactsPath}` is the path where you have the artifacts of the prerequisite (On jenkins is the dist folder generated by the build).



Maybe you need to [raise your memory or CPU limits](#).

Configure new environment

Now it is time to configure the environment.

Prerequisite

You need a docker image of your application. You could create it using OpenShift as you see in the

last step.

Create new app on OpenShift

To create new app you need to use the next command.

```
oc new-app --docker-image=${artifactsPath} --name=${props.name}-$environment --source -secret=${nameForSecret}
```



You could add environment variables using `-e $name=$value`



If you do not need to use a secret remove the end part of the command `--source -secret=${nameForSecret}`

Create routes

Finally, you need add a route to access the service.

Add http route

If you want to create an http route execute the following command in a terminal:

```
oc expose svc/${props.name}-$environment
```

Add https route

If you want to create an https route you can do it executing the following command:

```
oc create route edge --service=${props.name}-$environment
```

If you want to change the default route path you can use the command `--hostname=$url`. For example:

```
oc expose svc/${props.name}-$environment --hostname=$url
```

```
oc create route edge --service=${props.name}-$environment --hostname=$url
```

Import new images from registry

When you have new images in the registry you must import them to OpenShift. You could do it executing the next commands:

```
oc import-image ${props.name}-$environment  
--from=${dockerRegistry}/${props.name}:${dockerTag} --confirm
```



Maybe you need to raise your memory or CPU limits. It is explained below.

Raise/decrease memory or CPU limits

If you need to raise (or decrease) the memory or CPU limits that you need you could do it for your deployments and builders configurations following the next steps.

For deployments

You could do it in OpenShift using the user interface. To do it you should enter in OpenShift and go to deployments.

The screenshot shows the left-hand navigation sidebar of the OpenShift web interface. The sidebar has several sections: Overview, Applications (selected), Builds, Resources, Storage, Monitoring, and Catalog. Under the Applications section, there are sub-options: Deployments (selected), Stateful Sets, Pods, Services, and Routes. The Deployments option is highlighted with a blue background and white text.

At the right top, you could see a drop down actions, click on it and you could edit the resource limits of the container.

The screenshot shows the deployment details page for 'maildev'. At the top, it says 'Deployments > maildev' and 'maildev created 2 months ago'. Below that is a navigation bar with tabs: History (selected), Configuration, Environment, and Events. A message indicates 'Deployment #1 is active. View Log' was created 2 months ago. On the right, there is a 'Actions' dropdown menu with options: Deploy, Edit, Pause Rollouts, Add Storage, Add Autoscaler, Edit Resource Limits (highlighted with a yellow box), Edit Health Checks, Edit YAML (highlighted with a light blue box), and Delete. Below the actions is a table showing deployment details:

Deployment	Status	Created	Trigger
#1 (latest)	Active, 1 replica	2 months ago	Config change

Resource Limits: maildev

Resource limits control how much CPU and memory a container will consume on a node.

[Learn More ↗](#)

CPU 10 millicores min to 4 cores max

Request

100

millicores

The minimum amount of CPU the container is guaranteed.

Limit

100

millicores

The maximum amount of CPU the container is allowed to use when running.

[What are millicores?](#)

Memory 5 MiB min to 32 GiB max

Request

100

MiB

The minimum amount of memory the container is guaranteed.

Limit

100

MiB

The maximum amount of memory the container is allowed to use when running.

[What are MiB?](#) Pause rollouts for this deployment config

Pausing lets you make changes without triggering a rollout. You can resume rollouts at any time. If unchecked, a new rollout will start on save.

[Save](#)[Cancel](#)

Maybe you should modify the resource limits of the pod too. To do it you should click on drop down actions and go to edit YAML. Then you could see something like the next image.

Edit Deployment Config maildev

```

15  spec:
16    replicas: 1
17    selector:
18      app: maildev
19      deploymentconfig: maildev
20    strategy:
21      activeDeadlineSeconds: 21600
22      resources: {}
23    rollingParams:
24      intervalSeconds: 1
25      maxSurge: 25%
26      maxUnavailable: 25%
27      timeoutSeconds: 600
28      updatePeriodSeconds: 1
29      type: Rolling
30    template:
31      metadata:
32        annotations:
33          openshift.io/generated-by: OpenShiftWebConsole
34        creationTimestamp: null
35        labels:
36          app: maildev
37          deploymentconfig: maildev
38      spec:
39        containers:
40          - image: dario RodrigueZ/maildev@sha256:c986ea3de0da115d6f1aae1e60b1a70cea33c9ff080702ec47e2fc77f97fed
41            imagePullPolicy: Always
42            name: maildev
43            ports:
44              - containerPort: 2525
45                protocol: TCP
46              - containerPort: 8080
47                protocol: TCP
48        resources:
49          limits:
50            memory: 300Mi
51          requests:
52            memory: 300Mi
53        terminationMessagePath: /dev/termination-log
54        terminationMessagePolicy: File
55        dnsPolicy: ClusterFirst

```

[Save](#)[Cancel](#)

In the image, you could see that appear resources two times. One at the bottom of the image, this are the container resources that you modified on the previous paragraph and another one at the top of the image. The resources of the top are for the pod, you should give to it at least the same of the sum for all containers that the pod use.

Also you could do it using command line interface and executing the next command:

To modify pod limits

```
oc patch dc/boat-frontend-test --patch
'{"spec": {"strategy": {"resources": {"limits": {"cpu": "100m", "memory": "100Mi"}, "requests": {"cpu": "100m", "memory": "100Mi"} }}}'
```

To modify container limits

When this guide was written Openshift have a bug and you cannot do it from command line interface.



If that command did not work and you received an error like this `error: unable to parse '{spec:...': yaml: found unexpected end of stream`, try to use the patch using `'''` instead of `'`. It looks like this: `--patch "{\"spec\":...\"}'''`

For builders

You could do it using command line interface and executing the next command:

```
oc patch bc/${props.name}${APP_NAME_SUFFIX} --patch
'{"spec": {"resources": {"limits": {"cpu": "125m", "memory": "400Mi"}, "requests": {"cpu": "125m", "memory": "400Mi"} }}}'
```



If that command did not work and you received an error like this `error: unable to parse '{spec:...': yaml: found unexpected end of stream`, try to use the patch using `'''` instead of `'`. It looks like this: `--patch "{\"spec\":...\"}'''`

56.1.3. OpenShift deployment environment automatic configuration

In this section you will see how you can create a new environment instance in your OpenShift cluster to deploy devonfw projects using docker images.

Prerequisites

Add OpenShift Client to Jenkins

To integrate it, you need to have installed the plugin OpenShift Client. To install it go to Manage Jenkins clicking on left menu and enter in ***Manage Plugins***. Go to Available tab and search it using the filter textbox in the top right corner and install it.

Configuration OpenShift Client in Jenkins

Second, you need to configure the OC Client. Go to Manage Jenkins clicking on left menu and enter in ***Global Tool Configuration***.

Go to OpenShift Client Tools section and add a new one like this.

OpenShift Client Tools

OpenShift Client Tools installations

	Name	OpenShiftv3.11.0	
<input checked="" type="checkbox"/> Install automatically			
	Label		
<pre>Command: if [! -f /opt/oc3.11.0/oc]; then echo "oc3.11.0 does not exist." wget https://github.com/openshift/origin/releases/download/v3.11.0/openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit.tar.gz tar -xvfz openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit.tar.gz ls ls openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit sudo mv openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit /opt/oc3.11.0 else echo "oc3.11.0 exist." fi</pre>			
Tool Home		/opt/oc3.11.0	

devonfw project

You need to have a devonfw project in a git repository or a docker image uploaded to a docker registry.

Communication between components

Jenkins must have access to git, docker registry and OpenShift.

Openshift must have access to docker registry.

Jenkinsfiles to Configure OpenShift

You can find one Jenkinsfile per devonfw technology in [devonfw shop floor](#) repository to configure automatically your OpenShift cluster.

How to use it

To use it you need to follow the next steps

Create a new pipeline

You need to create a new pipeline in your repository and point it to Jenkinsfile in devonfw shop floor repository.

Pipeline

Definition Pipeline script from SCM

SCM Git

Repositories

Repository URL https://github.com/devonfw/devonfw-shop-floor.git

Credentials - none - Add Advanced... Add Repository

Branches to build

Branch Specifier (blank for 'any') */develop

Add Branch

Repository browser (Auto)

Additional Behaviours Add

Script Path dsf4openshift/configure-environments/devon4ng/Jenkinsfile

Lightweight checkout

[Pipeline Syntax](#)

Note: In the script path section you should use the Jenkinsfile of the technology that you need.

Build with parameters

The first time that you execute the pipeline is going to fail because Jenkins does not know that this pipeline needs parameters to execute. The better that you can do is stop it manually when *Declarative: Checkout SCM* is over.

Then you could see a button to Build with Parameters, click on it and fill the next form, these are the parameters:

Docker registry credentials for OpenShift

CREATE_SECRET: This option allows you to add the credentials of your docker registry in your OpenShift and stored it as a secret called docker-registry + registry_secret_name_suffix value.

Remember that you only need one secret to connect with your registry per namespace, if you are going to add more than one application in the same namespace that use the same registry, use the same name suffix and please do not create more than one secret in the same namespace. The namespace is the OpenShift project when you are going to deploy your application.

You can see your secrets stored in OpenShift going to OpenShift and click on the left menu:

The screenshot shows the OpenShift Container Platform interface. At the top, it says "OPENSHIFT CONTAINER PLATFORM". Below that is a navigation bar with a menu icon, the project name "s2portaldev", and a dropdown arrow. The left sidebar has several sections: "Overview", "Applications" (with a right arrow), "Builds" (with a right arrow), "Resources" (which is highlighted with a yellow background and has a right arrow), "Storage", "Monitoring", and "Catalog". The main content area is titled "Resources" and contains "Quota", "Membership", "Config Maps", and "Secrets" (which is highlighted with a yellow background). Below that is a section titled "Other Resources".



If the secret exists, you should uncheck the checkbox and fill the name suffix to use it.

REGISTRY_SECRET_NAME_SUFFIX: This is the suffix of the name for your docker registry credentials stored in OpenShift as a secret. The name is going to be docker-registry + this suffix, if you use more than one docker-registry in the same namespace you need to add a suffix. For example you could add the name of your project, then to have the name as docker-registry-myprojectname you should use -myprojectname value.

Build your docker image using OpenShift and store it in your docker registry

CREATE_DOCKER_BUILDER: This option allows you to create a build configuration in your OpenShift to create the docker images of your project and store them in your docker registry. If you are going to create the builder, your application is needed, you need to specify where is your git repository and which is the branch and credentials to use it.

The following parameters of this section are only necessary if a builder is to be created.

GIT_REPOSITORY: This is the url of your git repository.

i If you are using production line, remember to use the internal rout of your repository, to use it you must change the base url of your production line for the internal route <http://gitlab-core:80/gitlab>. For example, if your production line repository is for example <https://shared-services.pl.s2-eu.capgemini.com/gitlab/boat/boat-frontend.git> use <http://gitlab-core:80/gitlab/boat/boat-frontend.git>)

GIT_BRANCH: This is the branch that we are going to use for creating the first docker image. The next time that you are going to use the builder you could use another branches.

GIT_CREDENTIALS: This is the credentials id stored in your jenkins to download the code from your git repository.

BUILD_SCRIPT: In case of use devon4ng or devon4node you could specify which is the build script used to build and create the first docker image with this builder.

JAVA_VERSION In case of use devon4j this is the java version used for your docker image.

Docker registry information

DOCKER_REGISTRY: This is the url of your docker registry.

i If you are using production line, the url of your registry is docker-registry- + your production line url. For example, if your production line is shared-services.pl.s2-eu.capgemini.com your docker registry is docker-registry-shared-services.pl.s2-eu.capgemini.com.

If you cannot access to your docker registry, please open an incident in i4u.

DOCKER_REGISTRY_CREDENTIALS: This is the credentials id stored in your jenkins to download or upload docker images in your docker registry.

DOCKER_TAG: This is the tag that is going to be used for the builder to push the docker image and for the deployment config to pull and deploy it.

OpenShift cluster information

OPENSHIFT_URL: This is the url of your OpenShift cluster.

OPENSHIFT_CREDENTIALS: This is the credentials id stored in your jenkins to use OpenShift.

OPENSHIFT_NAMESPACE: This is the name of the project in your OpenShift where you are going to use. The name of the project in OpenShift is called namespace.

Take care because although you see at the top of your OpenShift interface the name of the project that you are using, this name is the display-name and not the value that you need. To obtain the correct value you must check your OpenShift url like you see in the next image:

The screenshot shows the OpenShift Web Console interface. The top navigation bar includes the 'OpenShift Web Console' logo, back and forward buttons, a refresh button, and a search bar with the URL <https://ocp.itaas.s2-eu.capgemini.com/console/project/s2portaldev/overview>. The main header says 'OPENSHIFT CONTAINER PLATFORM'. The left sidebar has a menu icon, the project name 's2portaldev', and links for Overview, Applications, Builds, Resources, Storage, Monitoring, and Catalog. The main content area displays an application named 'devon4ng-dev' under the heading 'APPLICATION'. Below it is a link for 'DEPLOYMENT CONFIG devon4ng-dev, #1'.

APP_NAME_SUFFIX: The name of all things created in your OpenShift project are going to be called as the configuration of your application says. Normally, our projects use a suffix that depends on the environment. You can see the values in the next list:

- For develop branch we use **-dev**
- For release branch we use **-uat**
- For master branch we use **-prod**

HOSTNAME: If you do not specify nothing, OpenShift is going to autogenerate a valid url for your application. You could modify the value by default but be sure that you configure everything to server your application in the route that you specify.

SECURED_PROTOCOL: If true, the protocol for the route will be https otherwise will be http.

Jenkins tools

All those parameters are the name of the tools in your Jenkinsfile.

To obtain it you need enter in your Jenkins and go to Manage Jenkins clicking on left menu and enter in **Global Tool Configuration** or in **Managed files**.

OPENSIFT_TOOL: Is located in Global tool configuration.

OpenShift Client Tools

OpenShift Client Tools installations

Add OpenShift Client Tools



OpenShift Client Tools

Name **OpenShiftv3.11.0** Install automatically

Run Shell Command



Label



Command

```
if [ ! -f /opt/oc3.11.0/oc ]; then
    echo "oc3.11.0 does not exist."
    wget https://github.com/openshift/origin/releases/download/v3.11.0/openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit.tar.gz
    tar -xvfz openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit.tar.gz
    ls
    ls openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit
    sudo mv openshift-origin-client-tools-v3.11.0-0cbc58b-linux-64bit /opt/oc3.11.0
else
    echo "oc3.11.0 exist."
fi
```



Tool Home

/opt/oc3.11.0

**Delete installer****NODEJS_TOOL:** Is located in Global tool configuration.

[jenkins openshift tool] | ./images/configuration/jenkins-openshift-tool.jpg

YARN_TOOL: Is located in Global tool configuration, inside the custom tools.

Custom tool

Name **Yarn**

Custom Tool Configuration...

 Install automatically

Run Shell Command



Label



Command

```
if ! which yarn > /dev/null; then
    curl -Ss https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
    echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee /etc/apt/sources.list.d/yarn.list
    sudo apt-get -y update && sudo apt-get -y install yarn
fi
```



Tool Home

/usr/lib/yarn

**Delete Installer****GLOBAL_SETTINGS_ID** Is located in Managed files. You need to click on edit button and take the id.

Config File Management

You can edit or remove your configuration files

Global Maven settings.xml

Global OASP Maven Settings



global oasp maven settings



MyGlobalSettings



global settings



Edit Configuration File

description

The configuration

ID

MavenSettings

Name

MyGlobalSettings

Comment

global settings

Replace All

MAVEN_INSTALLATION Is located in Global tool configuration.

Maven

Maven installations

Add Maven



Maven

Name: **Maven3**

Install automatically



Install from Apache

Version: 3.6.0 ▾

Delete Installer

57. Monitoring

57.1. Build monitor view

This tool you will be able to see in real time what is the state of your Jenkins pipelines.

57.1.1. Prerequisites

Add build monitor view plugin

To integrate it, you need to have installed the build monitor view. To install it go to Manage Jenkins clicking on left menu and enter in **Manage Plugins**. Go to Available tab and search it using the filter textbox in the top right corner and install it.

57.1.2. How to use it

When you have build monitor view installed, you could add a new view clicking on the **+** tab in the top bar.

S	New View	Name ↓
		devon4j
		devon4ng
		devon4node

Now you need to fill which is the name that you are going to give to your view and select *Build Monitor View* option.

Then you can see the configuration.

In **Job Filters** section you can specify which resources are going to be showed and whether subfolders should be included in the search.

In **Build Monitor - View Settings** you could specify which is the name at the top of the view and what is the ordering criterion.

In **Build Monitor - Widget Settings** you could specify if you want to show the committers and which is the field to display if it fails.

And this is the output:

You could limit the columns and the text scale clicking on the *gear button* at the right top corner.



58. Annexes

58.1. BitBucket

[Under construction]

The purpose of the present document is to provide the basic steps carried out to setup a BitBucket server in OpenShift.

Introduction

BitBucket is the Atlassian tool that extends the Git functionality, by adding integration with JIRA, Confluence, or Trello, as well as incorporates extra features for security or management of user accounts (See [BitBucket](#)).

BitBucket server is the Atlassian tool that runs the BitBucket services (See [BitBucket server](#)).

The followed approach has been not using command line, but OpenShift Web Console, by deploying the Docker image **atlassian/bitbucket-server** (available in [Docker Hub](#)) in the existing project **Deployment**.

The procedure below exposed consists basically in three main steps:

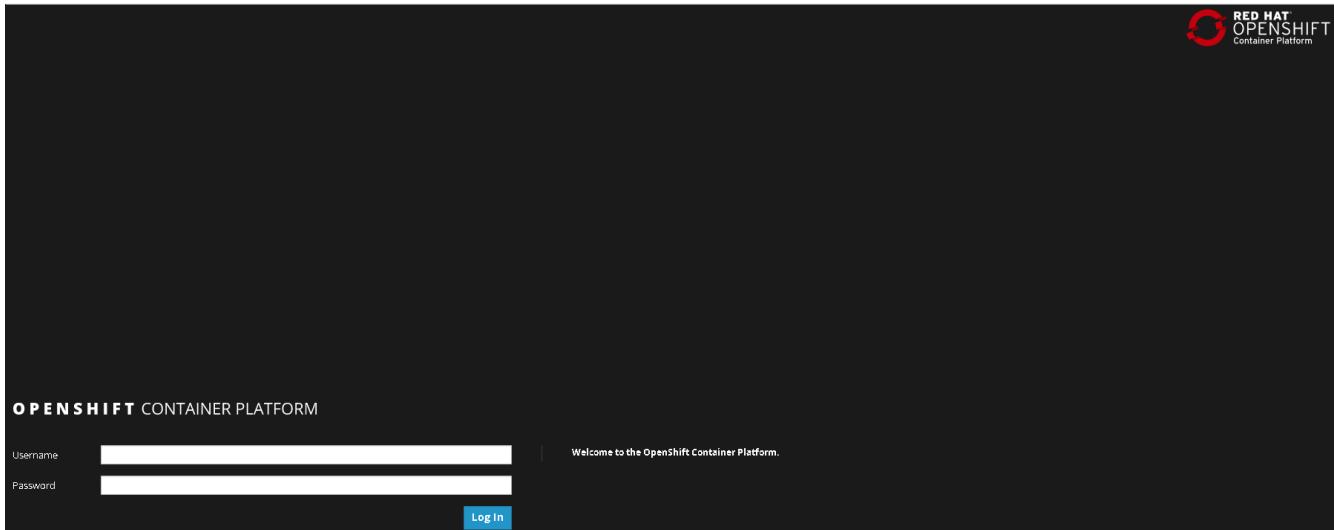
1. Deploy the BitBucket server image (from OpenShift web console)
2. Add a route for the external traffic (from OpenShift web console)
3. Configure the BitBucket server (from BitBucket server web console)

Prerequisites

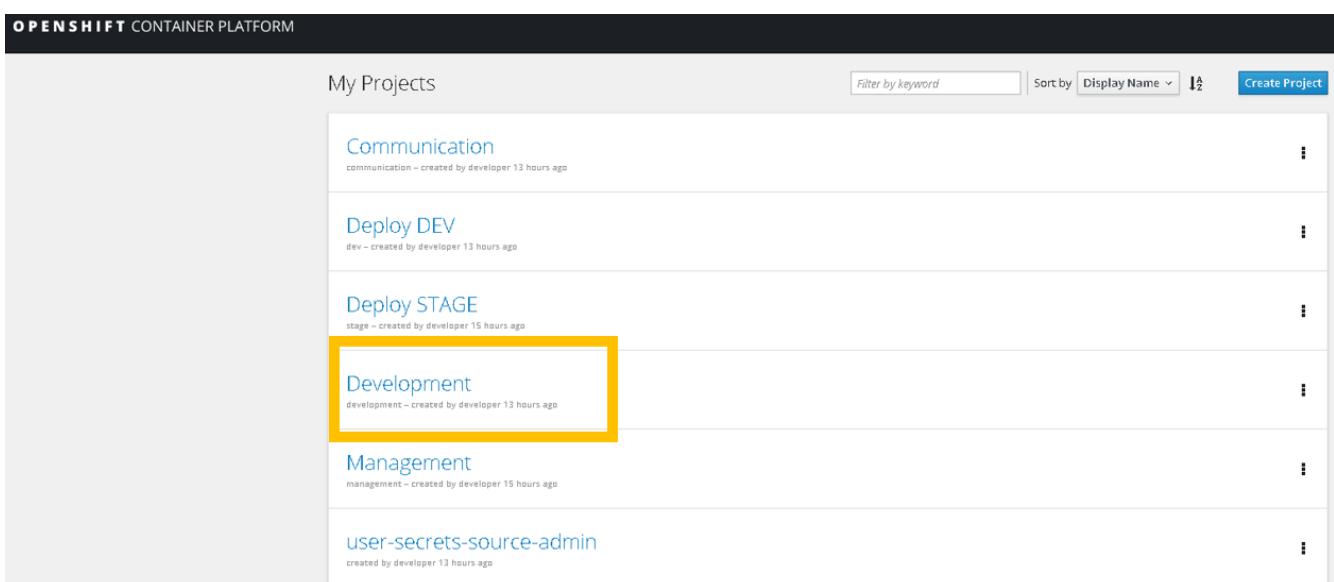
- OpenShift up & running
- Atlassian account (with personal account key). Not required for OpenShift, but for the initial BitBucket server configuration.

Procedure

] === Step 0: Log into our [OpenShift Web console](#)



Step 1: Get into Development project



OPENSHIFT CONTAINER PLATFORM

My Projects

Communication

Deploy DEV

Deploy STAGE

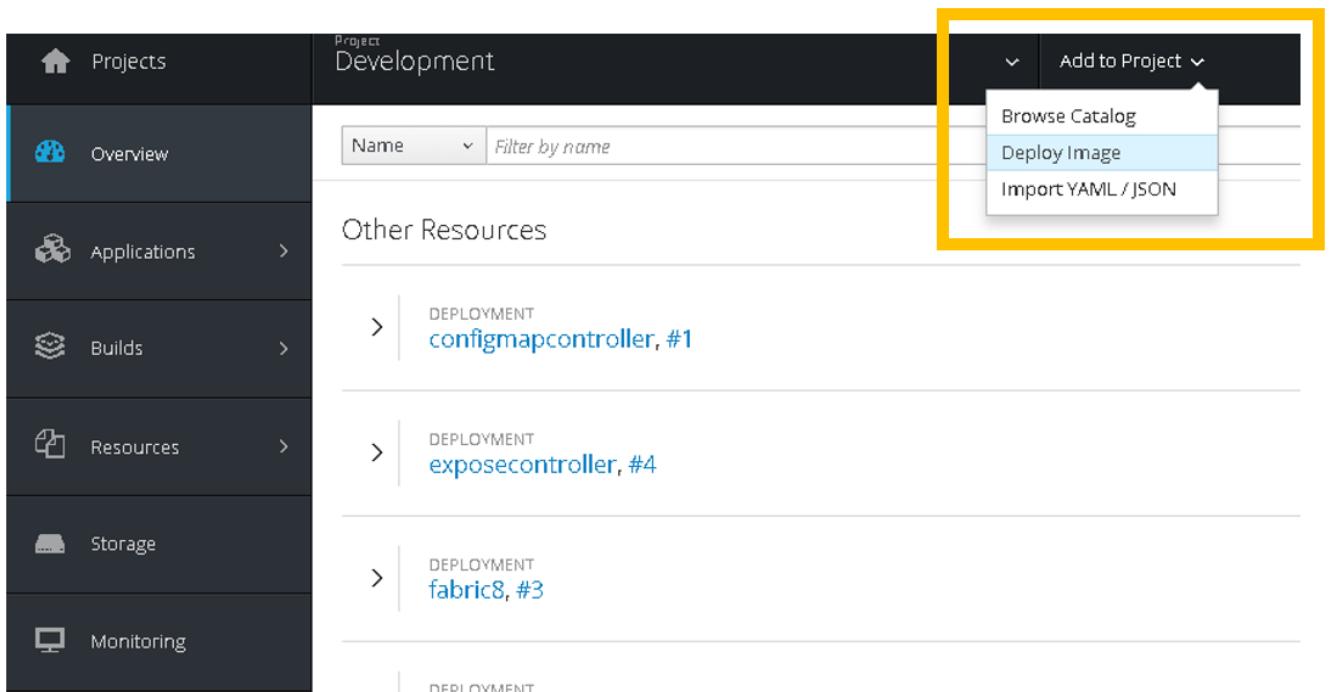
Development

Management

user-secrets-source-admin

Filter by keyword | Sort by Display Name | Create Project

Step 2.1: Deploy a new image to the project



Projects

Overview

Applications >

Builds >

Resources >

Storage

Monitoring

Project Development

Name Filter by name

Add to Project ▾

- Browse Catalog
- Deploy Image**
- Import YAML / JSON

Other Resources

DEPLOYMENT configmapcontroller, #1

DEPLOYMENT exposecontroller, #4

DEPLOYMENT fabric8, #3

Step 2.2: Introduce the image name (available in [Docker Hub](#)) and search

Image name: **atlassian/bitbucket-server**

The screenshot shows the 'Deploy Image' section of the OpenShift interface. The 'Image Name' input field is highlighted with a yellow box and contains the text 'atlassian/bitbucket-server'. To the right of the input field is a magnifying glass icon, also highlighted with a yellow box.

Step 2.3: Leave by the moment the default config. since it is enough for the basic setup. Press Create

The screenshot shows the 'Create New Image Stream' configuration page. It includes sections for 'Name' (set to 'bitbucket-server'), 'Pull Secret' (a dropdown menu), 'Environment Variables' (empty), 'Labels' (with an 'app' label set to 'bitbucket-server'), and a 'Create' button highlighted with a yellow box. Other UI elements like 'About Environment Variables' and 'About Labels' links are visible.

Step 2.4: Copy the oc commands in case it is required to work via command line, and Go to overview

The screenshot shows the 'Manage your app' overview page. It features a 'Completed' message with a 'Go to overview' link highlighted with a yellow box. Below this, there's information about managing the application via the web console or command line tools, including a code block of 'oc' commands and a note about CLI Reference and Basic CLI Operations.

Step 2.5: Wait until OpenShift deploys and starts up the image. All the info will be available.

Please notice that there are no pre-configured routes, hence the application is not accessible from

outside the cluster.

APPLICATION
bitbucket-server

DEPLOYMENT
[bitbucket-server, #1](#)

CONTAINER: BITBUCKET-SERVER
Image: atlassian/bitbucket-server 7f1a98b 318.0 MiB
Ports: 7990/TCP and 1 other

Networking

SERVICE Internal Traffic
[bitbucket-server](#)
7990/TCP (7990-tcp) → 7990 and 1 other

1 pod

ROUTES External Traffic
[Create Route](#)

Step 3: Create a route in order for the application to be accessible from outside the cluster (external traffic). Press Create

Please notice that there are different fields that can be specified (hostname, port). If required, the value of those fields can be modified later.

Hostname
www.example.com
Public hostname for the route. If not specified, a hostname is generated.
The hostname can't be changed after the route is created.

Path
/br/>Path that the router watches to route traffic to the service.

*** Service**
bitbucket-server
Service to route to.

Target Port
7990 → 7990 (TCP)
Target port for traffic.

Alternate Services
 Split traffic across multiple services
Routes can direct traffic to multiple services for A/B testing. Each service has a weight controlling how much traffic it gets.

Security
 Secure route
Routes can be secured using several TLS termination types for serving certificates.

Labels
[About Labels](#)
Labels for this route.

Name	Value	X
------	-------	---

[Add Label](#)
[Copy Service Labels](#)

Create **Cancel**

Leave by the moment the default config. as it is enough for the basic setup.

The route for external traffic is now available.

APPLICATION
bitbucket-server

DEPLOYMENT
bitbucket-server, #1

CONTAINER: BITBUCKET-SERVER
Image: atlassian/bitbucket-server 7f1a96b 318.0 MiB
Ports: 7990/TCP and 1 other

Networking
SERVICE Internal Traffic:
bitbucket-server
7990/TCP (7990-tcp) → 7990 and 1 other

ROUTES External Traffic
<http://bitbucket-server-development.apps.10.68.26.163.nip.io>
Route bitbucket-server, target port 7990-tcp

A yellow box highlights the external route information.

Now the BitBucket server container is up & running in our cluster.

The below steps correspond to the basic configuration of our BitBucket server.

Step 4.1: Click on the link of the external traffic route. This will open our BitBucket server setup wizard

Step 4.2: Leave by the moment the Internal database since it is enough for the basic setup (and it can be modified later), and click Next

Bitbucket

Bitbucket setup

Welcome

Language: English (United States)

Database: Internal
For evaluation and demo purposes only.
 External
Recommended for production use. See our [documentation](#) for more information.

Next

Step 4.3: Select the evaluation license, and click I have an account

Bitbucket setup

Licensing and settings

Application title

Base URL

All links created by Bitbucket (for emails, etc.) will be prefixed by this URL

Server ID **BS9D-12UM-9Q5X-FXMF**

License key I need an evaluation license

I have a Bitbucket license key

Generate license

Log in to your existing Atlassian account, or create a new one

[I have an account](#) [Create an account](#)

Step 4.4: Select the option Bitbucker (Server)

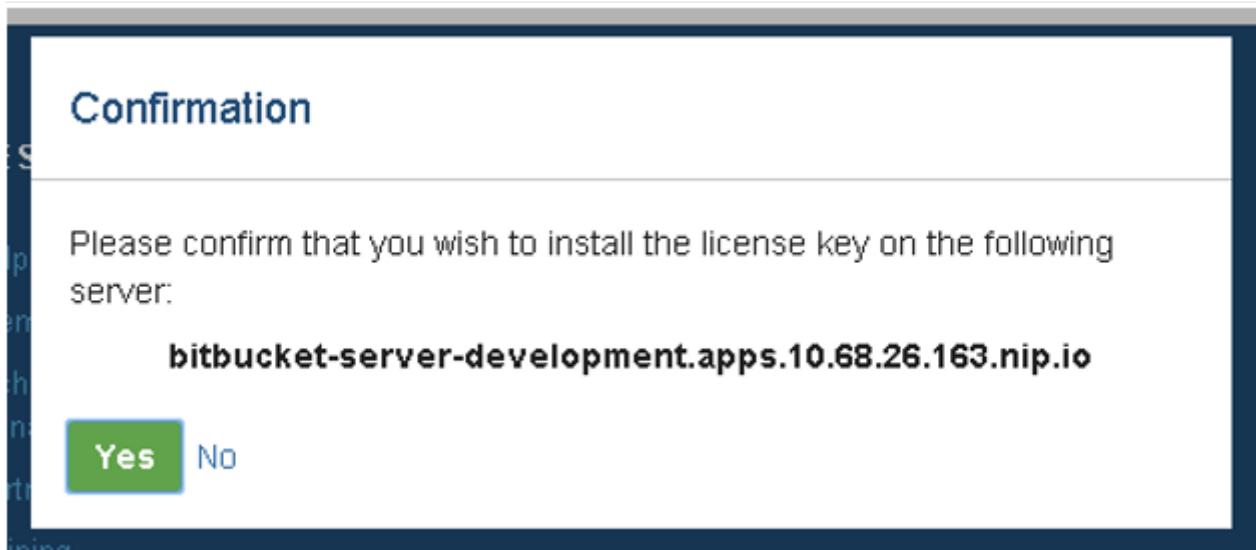
My Atlassian

New Evaluation License

Product	Bitbucket	
License type	<p>Bitbucket (Server)</p> <ul style="list-style-type: none">▪ Manage the entire application on your own servers or virtual machines.▪ Deployable to a single server. <div style="border: 2px solid yellow; padding: 5px; margin-top: 10px; text-align: center;"><input checked="" type="button" value=""/></div>	<p>Bitbucket (Data Center)</p> <p>Everything with server plus:</p> <ul style="list-style-type: none">▪ Active-active clustering for true high availability and uninterrupted access.▪ High performance under high load and at peak times▪ Disaster recovery. <div style="text-align: right; margin-top: 10px;"><input type="button" value="Select"/></div>
Organization		
<p>Your instance is:</p> <p><input checked="" type="radio"/> up and running</p> <p><input type="radio"/> not installed yet</p>		
Server ID	BS9D-12UM-9Q5X-FXMF	
<p>Please note we only provide evaluation support for 90 days per product.</p> <p>By clicking here you accept the Atlassian Customer Agreement.</p>		
<input type="button" value="Generate License"/> <input type="button" value="Cancel"/>		

Step 4.5: Introduce your organization (Capgemini), and click Generate License

Step 4.6: Confirm that you want to install the license on the BitBucket server



The license key will be automatically generated. Click **Next**

Step 4.7: Introduce the details of the Administration account.

Since our BitBucket server is not going to be integrated with JIRA, click on Go to Bitbucket. The integration with JIRA can be configured later.

The screenshot shows the Bitbucket setup page under the heading "Administrator account setup". It contains fields for "Username*", "Full name*", "Email address*", "Password*", and "Confirm password*". At the bottom right are two buttons: "Integrate with JIRA" and "Go to Bitbucket", with the latter being highlighted by a yellow box.

Step 4.8: Log in with the admin account that has been just created

DONE !!

The screenshot shows the Bitbucket dashboard. At the top, it says "Welcome to Bitbucket" and "Let's get started". Below that, there's a message: "Familiar with Bitbucket? [Get back to it.](#)". The main area shows a pull request titled "Bugfix/TIS-57 buttons need to be red". It includes tabs for "Overview", "Diff", and "Commits". The "Overview" tab shows a list of changes: "TIS-57 Changed button background to red for dialogs" and "TIS-57 Changed button background to red for forms". There are also sections for "Details", "Activity", and "Reviewers". A sidebar on the right says "User feedback is enabled" and "To help make Bitbucket better we'll occasionally ask users to give us feedback." It has buttons for "Ok, got it" and "Disable feedback".

[Under construction]

The purpose of the present document is to provide the basic steps carried out to improve the configuration of BitBucket server in OpenShift.

The improved configuration consists on:

- Persistent Volume Claims
- Health Checks (*pending to be completed*)

Persistent Volume Claims.

Please notice that the BitBucket server container does not use persistent volume claims by default, which means that the data (e.g.: BitBucket server config.) will be lost from one deployment to another.

The screenshot shows the OpenShift web interface for managing a deployment named 'bitbucket-server'. The 'Configuration' tab is active. In the 'Volumes' section, there is a table with one row. The row has a yellow border and contains the following data:

Type:	emptyDir {temporary directory destroyed with the pod}
Medium:	node's default

It is very important to create a persistent volume claim in order to prevent the mentioned loss of data.

Step 1: Add storage

bitbucket-server created 20 hours ago

[app](#) [bitbucket-server](#)[History](#) [Configuration](#) [Environment](#) [Events](#)**Details**

Selectors:	app=bitbucket-server deploymentconfig=bitbucket-server
Replicas:	1 replica
Strategy:	Rolling
Timeout:	10800 sec
Update Period:	1 sec
Interval:	1 sec
Max Unavailable:	25%
Max Surge:	25%

Template

Container bitbucket-server does not have health checks to ensure your application is running correctly.
[Add Health Checks](#)

Containers

CONTAINER: BITBUCKET-SERVER

Image: atlassian/bitbucket-server
Ports: 7990/TCP, 7999/TCP
Mount: bitbucket-server-storage → /var/atlassian/application-data/bitbucket read-write

Volumesbitbucket-server-storage [Remove](#)

Type:	persistent volume claim (reference to a persistent volume claim)
Claim name:	bitbucket-server-storage
Mode:	read-write

[Add Storage](#) | [Add Config Files](#)**Autoscaling**[Add Autoscaler](#)**Hooks** [Learn More](#)

none

Triggers**Manual (CLI):**[Learn More](#) **Change Of:**`oc rollout latest dc/bitbucket-server -n deve`

Config

Step 2: Select the appropriate storage, or create it from scratch if necessary

Add Storage

Add an existing persistent volume claim to the template of deployment config bitbucket-server.

* Storage

	CV		
<input checked="" type="radio"/> bitbucket-server-storage	100 GiB	(Read-Write-Once)	Bound to volume pv0067
<input type="radio"/> jenkins	100 GiB	(Read-Write-Once)	Bound to volume pv0075

Select storage to use or [create storage](#).

Volume

Specify details about how volumes are going to be mounted inside containers.

Mount Path

example: /data

Mount path for the volume inside the container. If not specified, the volume will not be mounted automatically.

Subpath

example: application/resources

Optional path within the volume from which it will be mounted into the container. Defaults to the volume's root.

Volume Name

(generated if empty)

Unique name used to identify this volume. If not specified, a volume name is generated.

Read only

Mount the volume as read-only.

Pause rollouts for this deployment config

Pausing lets you make changes without triggering a rollout. You can resume rollouts at any time. If unchecked, a new rollout will start on save.

Add

Cancel

Step 3: Introduce the required information

- **Path** as it is specified in the [BitBucket server Docker image](#) (/var/atlassian/application-data/bitbucket)
- **Volume name** with a unique name to clearly identify the volume

Add Storage

Add an existing persistent volume claim to the template of deployment config bitbucket-server.

* Storage

<input checked="" type="radio"/>	bitbucket-server-storage	100 GiB	(Read--Write-Once)	Bound to volume pv0067
<input type="radio"/>	jenkins	100 GiB	(Read-Write-Once)	Bound to volume pv0075

Select storage to use or [create storage](#).

Volume

Specify details about how volumes are going to be mounted inside containers.

Mount Path

[View](#)
/var/atlassian/application-data/bitbucket

Mount path for the volume inside the container. If not specified, the volume will not be mounted automatically.

Subpath

example: application/resources

Optional path within the volume from which it will be mounted into the container. Defaults to the volume's root.

Volume Name

[View](#)
bitbucket-server-volume

Unique name used to identify this volume. If not specified, a volume name is generated.

Read only

Mount the volume as read-only.

Pause rollouts for this deployment config

Pausing lets you make changes without triggering a rollout. You can resume rollouts at any time. If unchecked, a new rollout will start on save.

[Add](#)

[Cancel](#)

The change will be immediately applied

bitbucket-server created a day ago

[app](#) [bitbucket-server](#)

History Configuration Environment Events

Details

Selectors:	app=bitbucket-server deploymentconfig=bitbucket-server
Replicas:	1 replica
Strategy:	Rolling
Timeout:	600 sec
Update Period:	1 sec
Interval:	1 sec
Max Unavailable:	25%
Max Surge:	25%

Template

Container bitbucket-server does not have health checks to ensure your application is running correctly.
[Add Health Checks](#)

Containers

CONTAINER: BITBUCKET-SERVER
Image: atlassian/bitbucket-server 7f1a98b 318.0 MiB
Ports: 7990/TCP, 7999/TCP

Volumes

bitbucket-server-volume [Remove](#)

Type:	persistent volume claim (reference to a persistent volume claim)
Claim name:	bitbucket-server-storage
Mode:	read-write

Autoscaling

[Add Autoscaler](#)

Hooks [Learn More](#)

none

Triggers

Manual (CLI):

[Learn More](#)

Change Of:

Config

New Image For:

myproject/bitbucket-server:latest

58.2. Selenium Basic Grid

58.2.1. Basic Selenium Grid setup in OpenShift

[Under construction]

The purpose of the present document is to provide the basic steps carried out to setup a Selenium Grid (Hub + Nodes) in OpenShift.

Introduction

Selenium is a tool to automate web browser across many platforms. It allows the automation of the testing in many different browsers, operating systems, programming languages, or testing frameworks. (for further information please see [Selenium](#))

Selenium Grid is the platform provided by Selenium in order to perform the execution of tests in parallel and in a distributed way.

It basically consists on a Selenium Server (also known as hub or simply server) which redirects the requests it receives to the appropriate node (Firefox node, Chrome node, ...) depending on how the Selenium WebDriver is configured or implemented (See [Selenium Doc.](#))

Additional documentation:

- https://www.tutorialspoint.com/selenium/selenium_grids.htm
- <http://www.softwaretestinghelp.com/selenium-ide-download-and-installation-selenium-tutorial-2>
- <https://examples.javacodegeeks.com/enterprise-java/selenium/selenium-standalone-server-example>
- <https://tripleqa.com/2016/09/26/hello-world-selenium>
- <http://queirozf.com/entries/selenium-hello-world-style-tutorial>

Prerequisites

- OpenShift up & running

Procedure

The present procedure is divided into two different main parts:
* First part: Selenium Hub (server) installation
* Second part: Selenium node installation (Firefox & Chrome)
* Create persistent volumes for the hub and the node(s)

Selenium Hub installation

The followed approach consists on deploying new image from the OpenShift WenConsole.

The image as well as its documentation and details can be found at [Selenium Hub Docker Image](#)

Step 1: Deploy Image

The screenshot shows the devonfw UI interface. At the top, there's a navigation bar with 'Project' and 'My Project'. Below it is a search bar with 'Name' and 'Filter by name'. To the right of the search bar is a dropdown menu with 'Add to Project' and other options like 'Browse Catalog' and 'Import YAML/JSON'. A yellow box highlights the 'Deploy Image' button. Below the search bar, there's a section titled 'Deployments' with two entries: 'bitbucket-server, #11' and 'crud, #1'. The 'bitbucket-server' entry has a blue arrow icon to its left.

Step 2: Image Name

As it is specified in the [documentation](#) (`selenium/hub`)

(Please notice that, as it is described in the additional documentation of the above links, the server will run by default on 4444 port)

The screenshot shows the 'Deploy Image' configuration page. At the top, there are tabs for 'Browse Catalog', 'Deploy Image' (which is selected and highlighted with a blue underline), and 'Import YAML / JSON'. Below the tabs, there's a note: 'Deploy an existing image from an image stream tag or docker pull spec.' There are two radio button options: 'Image Stream Tag' (unchecked) and 'Image Name' (checked). The 'Image Name' field contains the value 'selenium/hub'. To the right of the input field is a magnifying glass icon inside a yellow box. The entire input field is also highlighted with a yellow box.

Step 3: Introduce the appropriate resource name

(`selenium-hub` in this case)

(No additional config. is required by the moment)

My Project > Add to Project

Browse Catalog Deploy Image Import YAML / JSON

Deploy an existing image from an image stream tag or docker pull spec.

Image Stream Tag

Namespace / Image Stream : Tag

Image Name

selenium/hub

 selenium/hub 5 days ago, 120.7 MB, 12 layers

- Image Stream **selenium-hub:latest** will track this image.
- This image will be deployed in Deployment Config **selenium-hub**.
- Port 4444/TCP will be load balanced by Service **selenium-hub**.

Other containers can access this service through the hostname **selenium-hub**.

* Name **selenium-hub**

Identifies the resources created for this image.

Pull Secret

Secret name

Secret for authentication when pulling images from a secured registry. [Learn More](#)

Create New Secret

Environment Variables

[About Environment Variables](#)

Name	Value

Add Environment Variable | Add Environment Variable Using a Config Map or Secret

Once the image is deployed, you will be able to check & review the config. of the container. Please notice by, by default, **no route is created for external traffic**, hence the application (the selenium server or hub) is not reachable from outside the cluster

APPLICATION
selenium-hub

DEPLOYMENT
selenium-hub, #1

CONTAINER: SELENIUM-HUB

Image: selenium/hub 61d075c 120.7 MB
Ports: 4444/TCP

1 pod

Networking

SERVICE Internal Traffic
selenium-hub
4444/TCP [4444-tcp] → 4444

ROUTES External Traffic
[Create Route](#)

Step 4: Create a route for external traffic

DEPLOYMENT
selenium-hub, #1

CONTAINER: SELENIUM-HUB

Image: selenium/hub 61d075c 120.7 MB
Ports: 4444/TCP

1 pod

Networking

SERVICE Internal Traffic
selenium-hub
4444/TCP [4444-tcp] → 4444

ROUTES External Traffic
[Create Route](#)

Step 5: Change the default config. if necessary

Create Route

Routing is a way to make your application publicly visible.

* Name

A unique name for the route within the project.

Hostname

Public hostname for the route. If not specified, a hostname is generated.

The hostname can't be changed after the route is created.

Path

Path that the router watches to route traffic to the service.

* Service

Service to route to.

Target Port

Target port for traffic.

DONE !!

The Selenium Server is now accesible from outside the cluster. Click on the link of the route and you will be able to see the server home page.

The screenshot shows the devonfw UI interface. At the top, it says "APPLICATION" and "selenium-hub". Below that, under "DEPLOYMENT", there is a section for "selenium-hub, #1" which includes details about the container image (selenium/hub 61d075c 120.7 MiB) and ports (4444/TCP). To the right of this, there is a circular icon with the number "1" and arrows pointing up and down, likely indicating the number of pods. Below the deployment, there is a "Networking" section showing a "SERVICE Internal Traffic" entry for "selenium-hub" with the port mapping "4444/TCP (4444/tcp) → 4444". On the far right, there is a "ROUTES External Traffic" section with a yellow border around it, containing the URL "http://selenium-hub-development.apps.10.68.26.163.nip.io" and the text "Route selenium-hub. target port 4444 tcp".

console/view config to see the default server config.

Please notice that the server is not detecting any node up & running, since we have not yet installed none of them.

Selenium Node Firefox installation

(Same steps apply for Selenium Node Chrome with the selenium/node-chrome Docker image)

The key point of the nodes installation is to specify the host name and port of the hub. If this step is not correctly done, the container will be setup but the application will not run.

The followed approach consists on deploying new image from the OpenShift WenConsole.

The image as well as its documentation and details can be found at [Selenium Hub Docker Image \(firefox node in this case\)](#)

Step 1: Deploy Image

Introduce the appropriate Docker Image name as it is specified in the [documentation \(selenium/node-firefox\)](#)

Step 2: Introduce the appropriate resource name

(selenium-node-firefox in this case)

Deploy an existing image from an image stream tag or docker pull spec.

Image Stream Tag

Namespace	/	Image Stream	:	Tag
-----------	---	--------------	---	-----

Image Name

selenium/node-firefox	<input type="button" value="Search"/>
-----------------------	---------------------------------------



selenium/node-firefox 5 days ago, 261.6 MiB, 16 layers

- Image Stream **selenium-node-firefox:latest** will track this image.
- This image will be deployed in Deployment Config **selenium-node-firefox**.

* Name

selenium-node-firefox

Identifies the resources created for this image.

Pull Secret

Secret name

Secret for authentication when pulling images from a secured registry. [Learn More](#)

[Create New Secret](#)

Step 3: Introduce, as environment variables, the host name and port of the selenium hub previously created

Env. var. for selenium hub host name

- Name: HUB_PORT_4444_TCP_ADDR
- Value: The Selenium hub host name. It's recommended to use the service name of the internal OpenShift service.

Env. var. for host selenium hub host port

- Name: HUB_PORT_4444_TCP_PORT
- Value: 4444 (*by default*), or the appropriate one if it was changed during the installation.



selenium/node-firefox 5 days ago, 261.6 MiB, 16 layers

- Image Stream **selenium-node-firefox:latest** will track this image.
- This image will be deployed in Deployment Config **selenium-node-firefox**.

* Name

selenium-node-firefox

Identifies the resources created for this image.

Pull Secret

Secret name

Secret for authentication when pulling images from a secured registry. [Learn More](#)

[Create New Secret](#)

Environment Variables

[About Environment Variables](#)

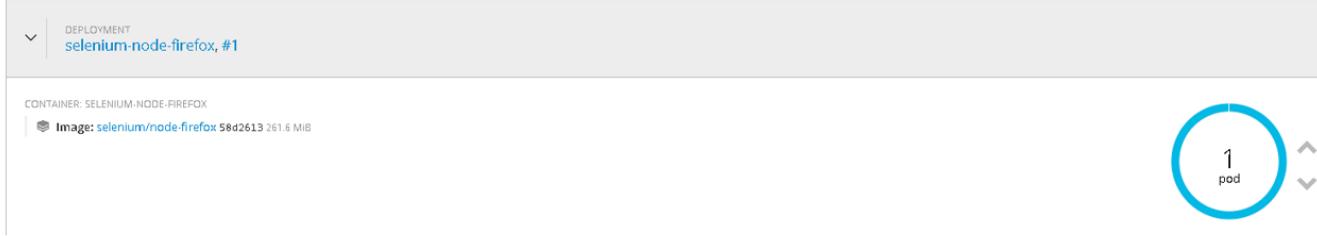
HUB_PORT_4444_TCP_ADDR	selenium-hub
------------------------	--------------

HUB_PORT_4444_TCP_PORT	4444
------------------------	------

[Add Environment Variable](#) | [Add Environment Variable Using a Config Map or Secret](#)

DONE !!

If the creation of the container was correct, we will be able to see our new selenium-node-firefox application up & running, as well as we will be able to see that the firefox node has correctly detected the selenium hub (*in the log of the POD*)

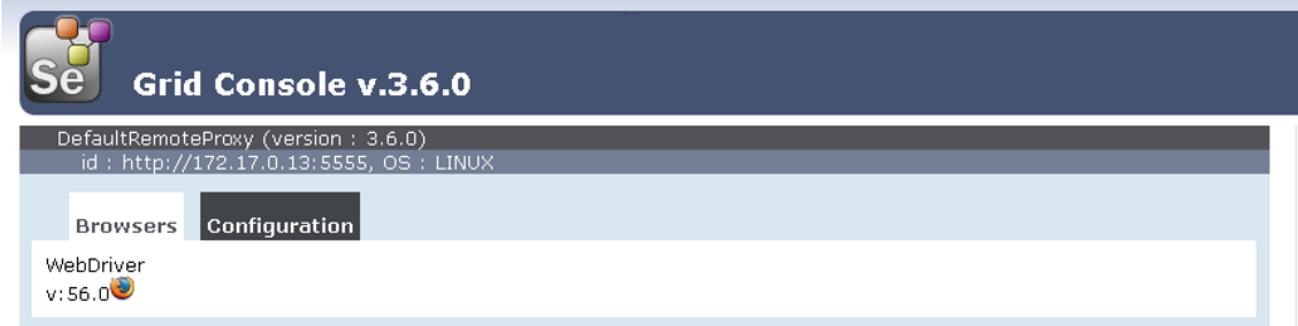


```

1 12:28:22.681 INFO - Selenium build info: version: '3.6.0', revision: '6fbf3ec767'
2 12:28:22.682 INFO - Launching a Selenium Grid node
3 2017-10-10 12:28:23.402:INFO::main: Logging initialized @1227ms to org.seleniumhq.jetty9.util.log.StderrLog
4 12:28:23.466 INFO - Driver class not found: com.opera.core.systems.OperaDriver
5 12:28:23.513 INFO - Driver provider class org.openqa.selenium.ie.InternetExplorerDriver registration is skipped:
6  registration capabilities Capabilities [{ensureCleanSession=true, browserName=internet explorer, version=, platform=WINDOWS}] does not match the current platform LINUX
7 12:28:23.514 INFO - Driver provider class org.openqa.selenium.edge.EdgeDriver registration is skipped:
8  registration capabilities Capabilities [{browserName=MicrosoftEdge, version=, platform=WINDOWS}] does not match the current platform LINUX
9 12:28:23.515 INFO - Driver provider class org.openqa.selenium.safari.SafariDriver registration is skipped:
10 registration capabilities Capabilities [{browserName=safari, version=, platform=MAC}] does not match the current platform LINUX
11 12:28:23.547 INFO - Using the passthrough mode handler
12 2017-10-10 12:28:23.580:INFO:osjs.Server:main: jetty-9.4.5.v20170502
13 2017-10-10 12:28:23.621:WARN:osjs.SecurityHandler:main: ServletContext@o.s.j.s.ServletContextHandler@2a3b5b47{/,,null,STARTING} has uncovered http methods for path: /
14 2017-10-10 12:28:23.628:INFO:osjs.ContextHandler:main: Started o.s.j.s.ServletContextHandler@2a3b5b47{/,,null,AVAILABLE}
15 2017-10-10 12:28:23.646:INFO:osjs.AbstractConnector:main: Started ServerConnector@6771beb3{HTTP/1.1}{[http/1.1]}{0.0.0.0:5555}
16 2017-10-10 12:28:23.646:INFO:osjs.Server:main: Started @1471ms
17 12:28:23.647 INFO - Selenium Grid node is up and ready to register to the hub
18 12:28:23.662 INFO - Starting auto registration thread. Will try to register every 5000 ms.
19 12:28:23.662 INFO - Registering the node to the hub: http://selenium-hub:4444/grid/register
20 12:28:23.723 INFO - The node is registered to the hub and ready to use

```

If we go back to the configuration of the SeleniumHub through the WebConsole, we also will be able to see the our new firefox node



[view config](#)

Persistent Volumes

Last part of the installation of the Selenium Grid consists on creating persistent volumes for both, the hub container and the node container.

Persistent Volumes can be easely created folling the the [BitBucket Extra server configuration](#)

58.3. Mirabaud Experience

58.3.1. Mirabaud CICD Environment Setup

Initial requirements:

- **OS:** RHEL 6.5

Remote setup in CI machine (located in the Netherlands)

- Jenkins
- Nexus
- GitLab
- Mattermost
- Atlassian Crucible
- SonarQube

1. Install Docker and Docker Compose in RHEL 6.5

Docker

Due to that OS version, the only way to have Docker running in the CI machine is by installing it from the **EPEL** repository (Extra Packages for Enterprise Linux).

1. Add EPEL

```
# rpm -iUvh http://dl.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-8.noarch.rpm
```

2. Install `docker.io` from that repository

```
# yum -y install docker-io
```

3. Start Docker daemon

```
# service docker start
```

4. Check the installation

```
# docker -v
Docker version 1.7.1, build 786b29d/1.7.1
```

Docker Compose

Download and install it via `curl`. It will use [this site](#).

```
# curl -L https://github.com/docker/compose/releases/download/1.5.0/docker-compose-  
'uname -s'-'uname -m' > /usr/local/bin/docker-compose  
  
# chmod +x /usr/local/bin/docker-compose
```

Add it to your **sudo** path:

1. Find out where it is:

```
# echo $PATH
```

2. Copy the **docker-compose** file from **/usr/local/bin/** to your **sudo** PATH.

```
# docker-compose -v  
docker-compose version 1.5.2, build 7240ff3
```

2. Directories structure

Several directories had been added to organize some files related to docker (like **docker-compose.yml**) and docker volumes for each service. Here's how it looks:

```

/home
  /[username]
    /jenkins
      /volumes
        /jenkins_home
  /sonarqube
    /volumes
      /conf
      /data
      /extensions
      /lib
        /bundled-plugins
/nexus
  /volumes
    /nexus-data
/crucible
  /volumes
  /
/gitlab
  docker-compose.yml
  /volumes
    /etc
      /gitlab
    /var
      /log
      /opt
/mattermost
  docker-compose.yml
  /volumes
    /db
      /var
        /lib
          /postgresql
          /data
    /app
      /mattermost
        /config
        /data
        /logs
  /web
    /cert

```

3. CICD Services with Docker

Some naming conventions had been followed as naming containers as `mirabaud_[service]`.

Several folders have been created to store each service's volumes, `docker-compose.yml`(s), extra configuration settings and so on:

Jenkins

Command

```
# docker run -d -p 8080:8080 -p 50000:50000 --name=mirabaud_jenkins \
-v /home/[username]/jenkins/volumes/jenkins_home:/var/jenkins_home \
jenkins
```

Generate keystore

```
keytool -importkeystore -srckeystore server.p12 -srcstoretype pkcs12 -srcalias 1
-destkeystore newserver.jks -deststoretype jks -destalias server
```

Start jenkins with SSL (TODO: make a docker-compose.yml for this):

```
sudo docker run -d --name mirabaud_jenkins -v /jenkins:/var/jenkins_home -p 8080:8443
jenkins --httpPort=-1 --httpsPort=8443
--httpsKeyStore=/var/jenkins_home/certs/keystore.jks
--httpsKeyStorePassword=Mirabaud2017
```

Volumes

```
volumes/jenkins_home:/var/jenkins_home
```

SonarQube

Command

```
# docker run -d -p 9000:9000 -p 9092:9092 --name=mirabaud_sonarqube \
-v /home/[username]/sonarqube/volumes/conf:/opt/sonarqube/conf \
-v /home/[username]/sonarqube/volumes/data:/opt/sonarqube/data \
-v /home/[username]/sonarqube/volumes/extensions:/opt/sonarqube/extensions \
-v /home/[username]/sonarqube/volumes/lib/bundled-
plugins:/opt/sonarqube//lib/bundled-plugins \
sonarqube
```

Volumes

```
volumes/conf:/opt/sonarqube/conf
volumes/data:/opt/sonarqube/data
volumes/extensions:/opt/sonarqube/extensions
volumes/lib/bundled-plugins:/opt/sonarqube/lib/bundled-plugins
```

Nexus

Command

```
# docker run -d -p 8081:8081 --name=mirabaud_nexus \
-v /home/[username]/nexus/nexus-data:/sonatype-work
sonatype/nexus
```

Volumes

```
volumes/nexus-data/:/sonatype-work
```

Atlassian Crucible

Command

```
# docker run -d -p 8084:8080 --name=mirabaud_crucible \
-v /home/[username]/crucible/volumes/data:/atlassian/data/crucible
mswinarski/atlassian-crucible:latest
```

Volumes

```
volumes/data:/atlassian/data/crucible
```

4. CICD Services with Docker Compose

Both Services had been deploying by using the `# docker-compose up -d` command from their root directories (`/gitlab` and `/mattermost`). The syntax of the two `docker-compose.yml` files is the one corresponding with the 1st version (due to the `docker-compose v1.5`).

GitLab

`docker-compose.yml`

```
mirabaud:
  image: 'gitlab/gitlab-ce:latest'
  restart: always
  ports:
    - '8888:80'
  volumes:
    - '/home/[username]/gitlab/volumes/etc/gitlab:/etc/gitlab'
    - '/home/[username]/gitlab/volumes/var/log:/var/log/gitlab'
    - '/home/[username]/gitlab/volumes/var/opt:/var/opt/gitlab'
```

Command (docker)

```
docker run -d -p 8888:80 --name=mirabaud_gitlab \
-v /home/[username]/gitlab/volumes/etc/gitlab:/etc/gitlab \
-v /home/[username]/gitlab/volumes/var/log:/var/log/gitlab \
-v /home/[username]/gitlab/volumes/var/opt:/var/opt/gitlab \
gitlab/gitlab-ce
```

Volumes

```
volumes/etc/gitlab:/etc/gitlab
volumes/var/opt:/var/log/gitlab
volumes/var/log:/var/log/gitlab
```

Mattermost

docker-compose.yml:

```

db:
  image: mattermost/mattermost-prod-db
  restart: unless-stopped
  volumes:
    - ./volumes/db/var/lib/postgresql/data:/var/lib/postgresql/data
    - /etc/localtime:/etc/localtime:ro
  environment:
    - POSTGRES_USER=mmuser
    - POSTGRES_PASSWORD=mmuser_password
    - POSTGRES_DB=mattermost

app:
  image: mattermost/mattermost-prod-app
  links:
    - db:db
  restart: unless-stopped
  volumes:
    - ./volumes/app/mattermost/config:/mattermost/config:rw
    - ./volumes/app/mattermost/data:/mattermost/data:rw
    - ./volumes/app/mattermost/logs:/mattermost/logs:rw
    - /etc/localtime:/etc/localtime:ro
  environment:
    - MM_USERNAME=mmuser
    - MM_PASSWORD=mmuser_password
    - MM_DBNAME=mattermost

web:
  image: mattermost/mattermost-prod-web
  ports:
    - "8088:80"
    - "8089:443"
  links:
    - app:app
  restart: unless-stopped
  volumes:
    - ./volumes/web/cert:/cert:ro
    - /etc/localtime:/etc/localtime:ro

```

SSL Certificate

How to generate the certificates:

Get the **crt** and **key** from CA or **generate a new one self-signed**. Then:

```
// 1. create the p12 keystore
# openssl pkcs12 -export -in cert.crt -inkey mycert.key -out certkeystore.p12

// 2. export the pem certificate with password
# openssl pkcs12 -in certkeystore.p12 -out cert.pem

// 3. export the pem certificate without password
# openssl rsa -in cert.pem -out key-no-password.pem
```

SSL:

Copy the cert and the key without password at:

[./volumes/web/cert/cert.pem](#)

and

[./volumes/web/cert/key-no-password.pem](#)

Restart the server and the SSL should be enabled at port **8089** using **HTTPS**.

Volumes

```
-- db --
volumes/db/var/lib/postgresql/data:/var/lib/postgresql/data
/etc/localtime:/etc/localtime:ro                                     # absolute path

-- app --
volumes/app/mattermost/config:/mattermost/config:rw
volumes/app/mattermost/data:/mattermost/data:rw
volumes/app/mattermost/logs:/mattermost/logs:rw
/etc/localtime:/etc/localtime:ro                                     # absolute path

-- web --
volumes/web/cert:/cert:ro
/etc/localtime:/etc/localtime:ro                                     # absolute path
```

5. Service Integration

All integrations had been done following **CICD Services Integration** guides:

- [Jenkins - Nexus integration](#)
- [Jenkins - GitLab integration](#)
- [Jenkins - SonarQube integration](#)



These guides may be obsolete. You can find here the [official configuration guides](#),

58.3.2. Jenkins - GitLab integration

The first step to have a Continuous Integration system for your development is to make sure that all your changes to your team's remote repository are evaluated by the time they are pushed. That usually implies the usage of so-called *webhooks*. You'll find a fancy explanation about what Webhooks are in [here](#).

To resume what we're doing here, we are going to prepare our Jenkins and our GitLab so when a developer pushes some changes to the GitLab repository, a pipeline in Jenkins gets triggered. Just like that, in an automatic way.

1. Jenkins GitLab plugin

As it usually happens, some Jenkins plug-in(s) must be installed. In this case, let's install those related with GitLab:

Filter:

Updates	Available	Installed	Advanced	
Enabled	Name ↓	Version	Previously installed version	Uninstall
<input checked="" type="checkbox"/>	bouncycastle API Plugin This plugin provides an stable API to Bouncy Castle related tasks.	2.16.2		<input type="button" value="Uninstall"/>
<input checked="" type="checkbox"/>	Credentials Plugin This plugin allows you to store credentials in Jenkins.	2.1.16		<input type="button" value="Uninstall"/>
<input checked="" type="checkbox"/>	Git client plugin Utility plugin for Git support in Jenkins	2.6.0	<input type="button" value="Downgrade to 2.5.0"/>	<input type="button" value="Uninstall"/>
<input checked="" type="checkbox"/>	Git plugin This plugin integrates Git with Jenkins.	3.6.3	<input type="button" value="Downgrade to 3.6.0"/>	<input type="button" value="Uninstall"/>
<input checked="" type="checkbox"/>	Gitlab Authentication plugin This is the an authentication plugin using gitlab OAuth.	1.0.9		<input type="button" value="Uninstall"/>
<input checked="" type="checkbox"/>	Gitlab Hook Plugin Enables Gitlab web hooks to be used to trigger SMC polling on Gitlab projects	1.4.2		<input type="button" value="Uninstall"/>
<input checked="" type="checkbox"/>	GitLab Logo Plugin Display GitLab Repository Icon on dashboard	1.0.3		<input type="button" value="Uninstall"/>
<input checked="" type="checkbox"/>	GitLab Merge Request Builder Integrates Jenkins with Gitlab to build Merge Requests	2.0.0		<input type="button" value="Uninstall"/>
<input checked="" type="checkbox"/>	GitLab Plugin This plugin integrates GitLab to Jenkins by faking a GitLab CI Server.	1.5.0	<input type="button" value="Downgrade to 1.4.8"/>	<input type="button" value="Uninstall"/>

2. GitLab API Token

To communicate with GitLab from Jenkins, we will need to create an authentication token from your GitLab user settings. A good practice for this would be to create it from a *machine user*. Something like (i.e.) `devonfw-ci/*****`.

The screenshot shows the GitLab User Settings interface. On the left sidebar, under 'Access Tokens', there is a sub-menu with options: User Settings, Profile, Account, Applications, Chat, Access Tokens (which is selected and highlighted in blue), Emails, Password, Notifications, SSH Keys, GPG Keys, Preferences, and Authentication log.

The main content area is titled 'Personal Access Tokens'. It contains two sections: 'Add a personal access Token' and 'Active Personal Access Tokens (2)'. The 'Add a personal access Token' section includes fields for 'Name' (with placeholder 'My App'), 'Expires at' (set to '2023-03-13'), and 'Scopes' (checkboxes for 'api', 'read_user', and 'read_registry'). A green 'Create personal access token' button is located below these fields. The 'Active Personal Access Tokens (2)' section lists two tokens:

Name	Created	Expires	Scopes	Action
Jenkins CI server	Nov 2, 2017	In 6 months	api, read_user, read_registry	Revoke
devonfw-ci-token	Oct 30, 2017	In over 2 years	api, read_user, read_registry	Revoke

Simply by adding a name to it and a date for it expire is enough:

Add a personal access Token

Pick a name for the application, and we'll give you a unique personal access Token.

Name

devonfw-ci

Expires at

2023-03-13

Scopes

- api** Access your API
- read_user** Read user information
- read_registry** Read Registry

Create personal access token

Your new personal access token has been created.

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

Your New Personal Access Token

zFczsZ4TC2ZM_Txu6rzo



Make sure you save it - you won't be able to access it again.

As GitLab said, you should make sure you don't lose your token. Otherwise you would need to create a new one.

This will allow Jenkins to connect with right permissions to our GitLab server.

3. Create "GitLab API" Token credentials

Those credentials will use that token already generated in GitLab to connect once we declare the GitLab server in the Global Jenkins configuration. Obviously, those credentials must be **GitLab API token**-like.

The screenshot shows the Jenkins 'Create New Credential' dialog. The 'Kind' dropdown menu is open, displaying several options: 'Username with password', 'Username with password', 'Docker Host Certificate Authentication', 'GitLab API token' (which is highlighted with a blue selection bar), 'SSH Username with private key', 'Secret file', 'Secret text', and 'Certificate'. Below the dropdown are fields for 'ID' and 'Description', both of which are empty. At the bottom left is a dark blue 'OK' button.

Then, we add the generated token in the **API token** field:

The screenshot shows the Jenkins 'Global Configuration' dialog. The 'Kind' dropdown menu is set to 'GitLab API token'. The 'Scope' dropdown menu is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'API token' field contains a series of dots ('.....') indicating a long string of characters. Below the API token field are fields for 'ID' and 'Description', both of which are empty. At the bottom left is a dark blue 'OK' button. A cursor arrow is visible at the bottom center of the screen.

Look in your Global credentials if they had been correctly created:

Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

Name	Kind	Description
 John Doe (Administrator)	Person object	Administrator for the system
 Alice Smith (Customer Support)	Person object	Customer support team
 Bob Johnson (Software Developer)	Person object	Software developer
 Charlie Brown (Marketing Manager)	Person object	Marketing manager
 Diana Lee (Customer Relations Manager)	Person object	Customer Relations Manager
 Eduardo (Software Engineer)	Person object	Software engineer
 Fiona (Customer Support)	Person object	Customer support
 Gina (Marketing Manager)	Person object	Marketing manager
 GitLab API token	GitLab API token	
 Hannah (Customer Support)	Person object	Customer support
 Ivan (Software Developer)	Person object	Software developer
 Jasmine (Customer Support)	Person object	Customer support
 Kiran (Software Engineer)	Person object	Software engineer

Icon: SMI

4. Create GitLab connection in Jenkins

Specify a GitLab connection in your Jenkins's [Manage Jenkins > Configure System](#) configuration. This will tell Jenkins where is our GitLab server, a user to access it from and so on.

You'll need to give it a name, for example, related with what this GitLab is dedicated for (specific clients, internal projects...). Then, the `Gitlab host URL` is just where your GitLab server is. If you have it locally, that field should look similar to:

- Connection name: `my-local-gitlab`
 - Gitlab host URL: `http://localhost:${PORT_NUMBER}`

Finally, we select our recently GitLab API token as credentials.

Gitlab

Enable authentication for '/project' end-point	<input checked="" type="checkbox"/>
GitLab connections	Connection name <input type="text" value="my-local-gitlab"/>
Gitlab host URL	A name for the connection <input type="text" value="http://gitlab.org"/>
Credentials	GitLab API token <input type="button" value="Add"/>
The complete URL to the Gitlab server (i.e. http://gitlab.org)	
API Token for accessing Gitlab	
<input type="button" value="Success"/> <input type="button" value="Advanced..."/> <input type="button" value="Test Connection"/> <input type="button" value="Delete"/>	
<input type="button" value="Add"/>	

5. Jenkins Pipeline changes

5.1 Choose GitLab connection in Pipeline's General configuration

First, our pipeline should allow us to add a GitLab connection to connect to (the already created one).

<input type="checkbox"/> GitHub project	
GitLab connection	<input type="text" value="my-local-gitlab"/>
GitLab Repository Name	<input type="text" value="myusername/webhook-test"/>

In the case of the local example, could be like this:

- GitLab connection: `my-local-gitlab`
- GitLab Repository Name: `myusername/webhook-test` (for example)

5.2 Create a Build Trigger

1. You should already see your GitLab project's URL (as you stated in the General settings of the Pipeline).
2. Write `.*build.*` in the comment for triggering a build
3. Specify or filter the branch of your repo you want use as target. That means, whenever a git action is done to that branch (for example, `master`), this Pipeline is going to be built.
4. Generate a Secret token (to be added in the yet-to-be-created GitLab webhook).

Build Triggers

Build after other projects are built

Build periodically

Build when a change is pushed to GitLab. GitLab CI Service URL: [https://gitlab.com/api/v4/projects/123456789/ci/services/1](#) 1

Enabled GitLab triggers

Push Events	<input checked="" type="checkbox"/>
Opened Merge Request Events	<input type="checkbox"/>
Accepted Merge Request Events	<input type="checkbox"/>
Closed Merge Request Events	<input type="checkbox"/>
Rebuild open Merge Requests	<input type="checkbox"/> Never
Comments	<input type="checkbox"/>
Comment (regex) for triggering a build	<input type="text"/> *build.* 2

Enable [ci-skip]

Ignore WIP Merge Requests

Set build description to build cause (eg. Merge request or Git Push.)

Build on successful pipeline events

Allowed branches

<input type="radio"/> Allow all branches to trigger this job	3
<input checked="" type="radio"/> Filter branches by name	4
Include <input type="text"/> ⚠ Cannot connect to GitLab to check whether selected branches exist. (show details)	
Exclude <input type="text"/>	
<input type="radio"/> Filter branches by regex	
<input type="checkbox"/> Filter merge request by label	

Secret token 5

6. GitLab Webhook

1. Go to your GitLab project's **Settings > Integration** section.
 2. Add the path to your Jenkins Pipeline. Make sure you add **project** instead of **job** in the path.
 3. Paste the generated Secret token of your Jenkins pipeline
 4. Select your git action that will trigger the build.

W webhook-test

Overview Repository Issues Merge Requests CI / CD Wiki Snippets

Settings

- General
- Members
- Integrations** 1
- Repository
- CI / CD

Integrations

Webhooks can be used for binding events when something is happening within the project.

URL
http://path/to/your/jenkins:[PORT_NUMBER]/project/name-of-the-pipeline 2

Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

Trigger

- Push events 4
This URL will be triggered by a push to the repository
- Tag push events
This URL will be triggered when a new tag is pushed to the repository
- Comments
This URL will be triggered when someone adds a comment
- Issues events
This URL will be triggered when an issue is created/updated/merged
- Confidential Issues events
This URL will be triggered when a confidential issue is created/updated/merged
- Merge Request events
This URL will be triggered when a merge request is created/updated/merged
- Job events
This URL will be triggered when the job status changes
- Pipeline events
This URL will be triggered when the pipeline status changes
- Wiki Page events
This URL will be triggered when a wiki page is created/updated

SSL verification

Enable SSL verification

Add webhook

7. Results

After all those steps you should have a result similar to this in your Pipeline:



[Open Blue Ocean](#)



[Full Stage View](#)



Build History trend —

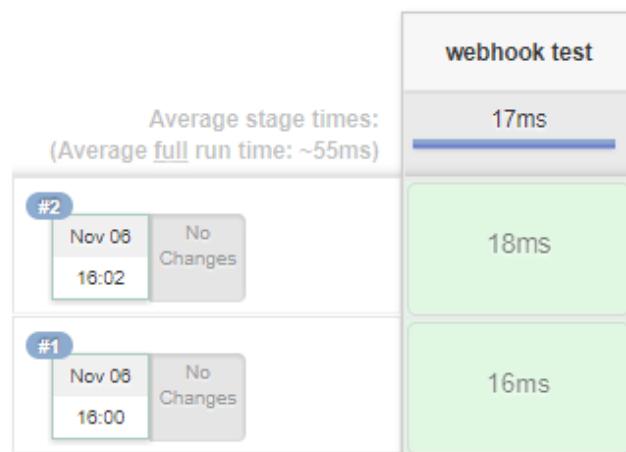
find

#2 Nov 6, 2017 3:02 PM Started by GitLab push by devonfw

#1 Nov 6, 2017 3:00 PM Started by GitLab push by devonfw

RSS for all RSS for failures

Stage View



Permalinks

- [Last build \(#2\), 42 min ago](#)
- [Last stable build \(#2\), 42 min ago](#)
- [Last successful build \(#2\), 42 min ago](#)
- [Last completed build \(#2\), 42 min ago](#)

Enjoy the Continuous Integration! :)

58.3.3. Jenkins - Nexus integration

Nexus is used to both host dependencies for devonfw projects to download (common Maven ones, custom ones such as `ojdb` and even devonfw so-far-IP modules). Moreover, it will host our projects' build artifacts (`.jar`, `.war`, ...) and expose them for us to download, wget and so on. A team should have a bidirectional relation with its Nexus repository.

1. Jenkins credentials to access Nexus

By default, when Nexus is installed, it contains 3 user credentials for different purposes. The admin ones look like this: `admin/admin123`. There are also other 2: `deployment/deployment123` and `TODO`.

```
// ADD USER TABLE IMAGE FROM NEXUS
```

In this case, let's use the ones with the greater permissions: `admin/admin123`.

Go to `Credentials > System` (left sidebar of Jenkins) then to `Global credentials (unrestricted)` on the page table and on the left sidebar again click on `Add Credentials`.

This should be shown in your Jenkins:

The screenshot shows the Jenkins global credentials configuration interface. The URL is `Jenkins > Credentials > System > Global credentials (unrestricted)`. On the left, there is a sidebar with a 'Back to credential domains' link and an 'Add Credentials' button. The main area has a 'Kind' dropdown set to 'Username with password'. It includes fields for 'Scope' (set to 'Global'), 'Username', 'Password', 'ID', and 'Description'. At the bottom is an 'OK' button.

Fill the form like this:

Kind: Username with password

Scope: Global (Jenkins, nodes, items, all child items, etc)

Username: admin

Password: *****

ID:

Description: Admin credentials to access Nexus

OK

And click in OK to create them. Check if the whole thing went as expected:

Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

Name	Kind	Description
jenkins-admin (jenkins-administrator)	Username with password	Administrator user for Jenkins
jenkins	Username with password	
Secret Token	Secret Token	Secret Token
user_jenkins-admin_digitaledgegroup	Username with password	Administrator user for the digital-edgegroup
jenkins	Username with password	
jenkins (jenkins-administrator)	Username with password	Administrator user for Jenkins
jenkins (jenkins-administrator)	Username with password	Administrator user for Jenkins
jenkins (jenkins-administrator)	Username with password	Administrator user for Jenkins
jenkins (jenkins-administrator)	Username with password	Administrator user for Jenkins
jenkins (jenkins-administrator)	Username with password	Administrator user for Jenkins
admin/***** (Admin credentials to access Nexus)	Username with password	Admin credentials to access Nexus

Icon: S M L

2. Jenkins Maven Settings

Those settings are also configured (or maybe not-yet-configured) in our **devonfw distributions** in:

```
/${devonfw-dist-path}
/software
/maven
/conf
    settings.xml
```

Go to **Manage Jenkins > Managed files** and select **Add a new Config** in the left sidebar.

 Manage Jenkins
 Config Files
 Add a new Config



Type

Select the file type you want to create

Global Maven settings.xml

A global maven settings.xml which can be referenced within Apache Maven jobs.
Use it within maven projects or maven builder and reference credentials for a server authentication from here: [credentials](#)

Maven settings.xml

A settings.xml which can be referenced within Apache Maven jobs.
Use it within maven projects or maven builder and reference credentials for a server authentication from here: [credentials](#)

Json file

a Json file

Maven toolchains.xml

a toolchains.xml which can be referenced within Apache Maven jobs

Simple XML file

a general xml file

Groovy file

a reusable groovy script

Custom file

a custom file (e.g. text or any other not yet available format)

Extended Email Publisher Groovy Template

A Groovy template used by the Extended Email Publisher plugin to generate emails.

Extended Email Publisher Jelly Template

A Jelly template used by the Extended Email Publisher plugin to generate emails.

Npm config file

a npmrc config file (an ini-formatted list of key = value parameters)

ID

ID of the config file

Submit

The ID field will get automatically filled with a unique value if you don't set it up. No problems about that. Click on **Submit** and let's create some Servers Credentials:

The configuration

ID	53d9e0a7-07c4-4fd9-a136-d76c271531c7
Name	MyGlobalSettings
Comment	global settings
Replace All	<input checked="" type="checkbox"/>
Server Credentials Add	

Content

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!--
4 Licensed to the Apache Software Foundation (ASF) under one
5 or more contributor license agreements. See the NOTICE file
6 distributed with this work for additional information
7 regarding copyright ownership. The ASF licenses this file
8 to you under the Apache License, Version 2.0 (the
9 "License"); you may not use this file except in compliance
10 with the License. You may obtain a copy of the License at
11
12 http://www.apache.org/licenses/LICENSE-2.0
13
14 Unless required by applicable law or agreed to in writing,
15 software distributed under the License is distributed on an
16 "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
17 KIND, either express or implied. See the License for the
18 specific language governing permissions and limitations
19 under the License.
20 -->
21

```

[Submit](#)

Those **Server Credentials** will allow Jenkins to access to the different repositories/servers that are going to be declared afterwards.

Let's create 4 server credentials.

- **my.nexus**: Will serve as general profile for **Maven**.
- **mynexus.releases**: When a `mvn deploy` process is executed, this will tell **Maven** where to push **releases** to.
- **mynexus.snapshots**: The same as before, but with **snapshots** instead.
- **mynexus.central**: Just in case we want to install an specific dependency that is not by default in the Maven Central repository (such as `ojdbc`), Maven will point to it instead.

Server Credentials	ServerId	my.nexus	
Credentials	admin***** (Admin credentials to access Nexus)		
Server Credentials	ServerId	mynexus.releases	
Credentials	admin***** (Admin credentials to access Nexus)		
Server Credentials	ServerId	mynexus.snapshots	
Credentials	admin***** (Admin credentials to access Nexus)		
Server Credentials	ServerId	mynexus.central	
Credentials	- current -		

A more or less complete Jenkins Maven settings would look like this:

```

<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">

    <mirrors>
        <mirror>
            <id>mynexus.central</id>
            <mirrorOf>central</mirrorOf>
            <name>central</name>
            <url>http://${URL-TO-YOUR-NEXUS-REPOS}/central</url>
        </mirror>
    </mirrors>

    <profiles>
        <profile>
            <id>my.nexus</id>
            <!-- 3 REPOS ARE DECLARED -->
            <repositories>
                <repository>
                    <id>mynexus.releases</id>
                    <name>mynexus Releases</name>
                    <url>http://${URL-TO-YOUR-NEXUS-REPOS}/releases</url>
                    <releases>
                        <enabled>true</enabled>
                        <updatePolicy>always</updatePolicy>
                    </releases>
                    <snapshots>
                        <enabled>false</enabled>
                        <updatePolicy>always</updatePolicy>
                    </snapshots>
                </repository>
            </repositories>
        </profile>
    </profiles>

```

```

        </snapshots>
    </repository>
    <repository>
        <id>mynexus.snapshots</id>
        <name>mynexus Snapshots</name>
        <url>http://${URL-TO-YOUR-NEXUS-REPOS}/snapshots</url>
        <releases>
            <enabled>false</enabled>
            <updatePolicy>always</updatePolicy>
        </releases>
        <snapshots>
            <enabled>true</enabled>
            <updatePolicy>always</updatePolicy>
        </snapshots>
    </repository>
</repositories>
<pluginRepositories>
    <pluginRepository>
        <id>public</id>
        <name>Public Repositories</name>
        <url>http://${URL-TO-YOUR-
NEXUS}/nexus/content/groups/public/</url>
        <releases>
            <enabled>true</enabled>
            <updatePolicy>always</updatePolicy>
        </releases>
        <snapshots>
            <enabled>true</enabled>
            <updatePolicy>always</updatePolicy>
        </snapshots>
    </pluginRepository>
</pluginRepositories>
</profile>
</profiles>
<!-- HERE IS WHERE WE TELL MAVEN TO CHOOSE THE my.nexus PROFILE -->
<activeProfiles>
    <activeProfile>my.nexus</activeProfile>
</activeProfiles>
</settings>

```

3. Use it in Jenkins Pipelines

58.3.4. Jenkins - SonarQube integration

First thing is installing both tools by, for example, Docker or Docker Compose. Then, we have to think about how they should collaborate to create a more efficient Continuous Integration process.

Once our project's pipeline is triggered (it could also be triggered in a fancy way, such as when a merge to the `develop` branch is done).

1. Jenkins SonarQube plugin

Typically in those integration cases, Jenkins plug-in installations become a **must**. Let's look for some available SonarQube plug-in(s) for Jenkins:

Enabled	Name	Version	Previously installed version	Uninstall
<input checked="" type="checkbox"/>	jQuery plugin This plugin provides an stable version of jQuery Javascript Library	1.12.4-0		Uninstall
<input checked="" type="checkbox"/>	Pipeline: Groovy Pipeline execution engine based on continuation passing style transformation of Groovy scripts.	2.41		Uninstall
<input checked="" type="checkbox"/>	SonarQube Scanner for Jenkins This plugin allows an easy integration of SonarQube , the open source platform for Continuous Inspection of code quality.	2.6.1		Uninstall

2. SonarQube token

Once installed let's create a **token** in SonarQube so that Jenkins can communicate with it to trigger their Jobs. Once we install SonarQube in our CI/CD machine (ideally a remote machine) let's login with **admin/admin** credentials:

Log In to SonarQube

[Log in](#)
[Cancel](#)

Afterwards, SonarQube itself asks you to create this token we talked about (the name is up to you):

Welcome to SonarQube!

Want to quickly analyze a first project? Follow these 2 easy steps.

1 Provide a token

Generate a token

Use existing token

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point of time in your user account.

Then a token is generated:

Welcome to SonarQube!

Want to quickly analyze a first project? Follow these 2 easy steps.

1 Provide a token

devonfw-ci-token: 

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point of time in your user account.

[Continue](#)

You click in "continue" and the token's generation is completed:

✓ devonfw-ci-token: 

3. Jenkins SonarQube Server setup

Now we need to tell Jenkins where is SonarQube and how to communicate with it. In [Manage Jenkins > Configure Settings](#). We add a name for the server (up to you), where it is located (URL), version and the Server authentication token created in point 2.

SonarQube servers

Environment variables

Enable injection of SonarQube server configuration as build environment variables

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

SonarQube installations

Name

SonarQube

Server URL

Default is <http://localhost:9000>

Server version

5.3 or higher

Configuration fields depend on the SonarQube server version.

Server authentication token

.....

4. Jenkins SonarQube Scanner

Install a SonarQube Scanner as a Global tool in Jenkins to be used in the project's pipeline.

SonarQube Scanner

SonarQube Scanner installations

SonarQube Scanner

Name

Install automatically

Install from Maven Central

Version

Delete Installer

5. Pipeline code

Last step is to add the SonarQube process in our project's Jenkins pipeline. The following code will trigger a SonarQube process that will evaluate our code's quality looking for bugs, duplications, and so on.

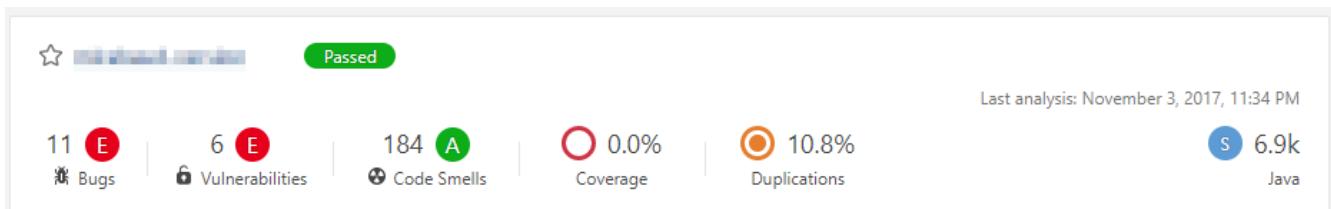
```
stage 'SonarQube Analysis'
def scannerHome = tool 'SonarQube scanner';
sh "${scannerHome}/bin/sonar-scanner \
-Dsonar.host.url=http://url-to-your-sq-server:9000/ \
-Dsonar.login=[SONAR_USER] -Dsonar.password=[SONAR_PASS] \
-Dsonar.projectKey=[PROJECT_KEY] \
-Dsonar.projectName=[PROJECT_NAME] \
-Dsonar.projectVersion=[PROJECT_VERSION] \
-Dsonar.sources=. -Dsonar.java.binaries=. \
-Dsonar.java.source=1.8 -Dsonar.language=java"
```

6. Results

After all this, you should end up having something like this in Jenkins:



And in SonarQube:



7. Changes in a devonfw project to execute SonarQube tests with Coverage

The plugin used to have Coverage reports in the SonarQube for devonfw projects is **Jacoco**. There are some changes in the project's parent **pom.xml** that are mandatory to use it.

Inside of the `<properties>` tag:

```

<properties>

(...)

<sonar.jacoco.version>3.8</sonar.jacoco.version>
<sonar.java.coveragePlugin>jacoco</sonar.java.coveragePlugin>
<sonar.core.codeCoveragePlugin>jacoco</sonar.core.codeCoveragePlugin>
<sonar.dynamicAnalysis>reuseReports</sonar.dynamicAnalysis>
<sonar.language>java</sonar.language>
<sonar.java.source>1.7</sonar.java.source>
<sonar.junit.reportPaths>target/surefire-reports</sonar.junit.reportPaths>
<sonar.jacoco.reportPaths>target/jacoco.exec</sonar.jacoco.reportPaths>
<sonar.sourceEncoding>UTF-8</sonar.sourceEncoding>
<sonar.exclusions>
  **/generated-sources/**/*,
  **io/oasp/mirabaud/general/**/*,
  **/*Dao.java,
  **/*Entity.java,
  **/*Cto.java,
  **/*Eto.java,
  **/*SearchCriteriaTo.java,
  **/*management.java,
  **/*SpringBootApp.java,
  **/*SpringBootBatchApp.java,
  **/*.xml,
  **/*.jsp
</sonar.exclusions>
<sonar.coverage.exclusions>
  **io/oasp/mirabaud/general/**/*,
  **/*Dao.java,
  **/*Entity.java,
  **/*Cto.java,
  **/*Eto.java,
  **/*SearchCriteriaTo.java,
  **/*management.java,
  **/*SpringBootApp.java,
  **/*SpringBootBatchApp.java,
  **/*.xml,
  **/*.jsp
</sonar.coverage.exclusions>
<sonar.host.url>http://${YOUR SONAR SERVER URL}</sonar.host.url>
<jacoco.version>0.7.9</jacoco.version>

<war.plugin.version>3.2.0</war.plugin.version>
<assembly.plugin.version>3.1.0</assembly.plugin.version>
</properties>

```

Of course, those `sonar` and `sonar.coverage` can/must be changed to fit with other projects.

Now add the **Jacoco Listener** as a dependency:

```
<dependencies>
  <dependency>
    <groupId>org.sonarsource.java</groupId>
    <artifactId>sonar-jacoco-listeners</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Plugin Management declarations:

```
<pluginManagement>
  <plugins>
    <plugin>
      <groupId>org.sonarsource.scanner.maven</groupId>
      <artifactId>sonar-maven-plugin</artifactId>
      <version>3.2</version>
    </plugin>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>${jacoco.version}</version>
    </plugin>
  </plugins>
<pluginManagement>
```

Plugins:

```
<plugins>
  (...)

  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.20.1</version>
    <configuration>
      <argLine>-XX:-UseSplitVerifier -Xmx2048m ${surefireArgLine}</argLine>
      <testFailureIgnore>false</testFailureIgnore>
      <useFile>false</useFile>
      <reportsDirectory>
        ${project.basedir}/${sonar.junit.reportPaths}</reportsDirectory>
        <argLine>${jacoco.agent.argLine}</argLine>
        <excludedGroups>${oasp.test.excluded.groups}</excludedGroups>
        <alwaysGenerateSurefireReport>true</alwaysGenerateSurefireReport>
        <aggregate>true</aggregate>
        <properties>
          <property>
            <name>listener</name>
```

```
<value>org.sonar.java.jacoco.JUnitListener</value>
</property>
</properties>
</configuration>
</plugin>
<plugin>
<groupId>org.jacoco</groupId>
<artifactId>jacoco-maven-plugin</artifactId>
<configuration>
<argLine>-Xmx128m</argLine>
<append>true</append>
<propertyName>jacoco.agent.argLine</propertyName>
<destFile>${sonar.jacoco.reportPath}</destFile>
<excludes>
<exclude>**/generated-sources/**/*,</exclude>
<exclude>**/io/oasp/${PROJECT_NAME}/general/**/*</exclude>
<exclude>**/*Dao.java</exclude>
<exclude>**/*Entity.java</exclude>
<exclude>**/*Cto.java</exclude>
<exclude>**/*Eto.java</exclude>
<exclude>**/*SearchCriteriaTo.java</exclude>
<exclude>**/*management.java</exclude>
<exclude>**/*SpringBootApp.java</exclude>
<exclude>**/*SpringBootBatchApp.java</exclude>
<exclude>**/*.class</exclude>
</excludes>
</configuration>
<executions>
<execution>
<id>prepare-agent</id>
<phase>initialize</phase>
<goals>
<goal>prepare-agent</goal>
</goals>
<configuration>
<destFile>${sonar.jacoco.reportPath}</destFile>
<append>true</append>
</configuration>
</execution>
<execution>
<id>report-aggregate</id>
<phase>verify</phase>
<goals>
<goal>report-aggregate</goal>
</goals>
</execution>
<execution>
<id>jacoco-site</id>
<phase>verify</phase>
<goals>
<goal>report</goal>
```

```

        </goals>
    </execution>
</executions>
</plugin>
</plugins>
```

Jenkins SonarQube execution

If the previous configuration is already setup, once Jenkins execute the sonar maven plugin, it will automatically execute coverage as well.

This is an example of a block of code from a devonfw project's [Jenkinsfile](#):

```

withMaven(globalMavenSettingsConfig: 'YOUR_GLOBAL_MAVEN_SETTINGS', jdk: 'OpenJDK
1.8', maven: 'Maven_3.3.9') {
    sh "mvn sonar:sonar -Dsonar.login=[USERNAME] -Dsonar.password=[PASSWORD]"
}
```

58.4. Azure DevOps

58.4.1. Connect to a Virtual Machine(VM) in Azure

Pre-requisites

Have a VM created and a private key in order to connect to it

Establish a connection

- 1- Open the client of your choice(putty,cmder,bash)
- 2- Ensure you have read-only access to the private key.

```
chmod 400 azureuser.pem
```

- 3- Run this command to connect to your VM

```
ssh -i <private key path> azureuser@51.103.78.61
```

note: To get the IP go to your azure portal, click on your VM, click on Networking and you will find the IP needed to establish the connection

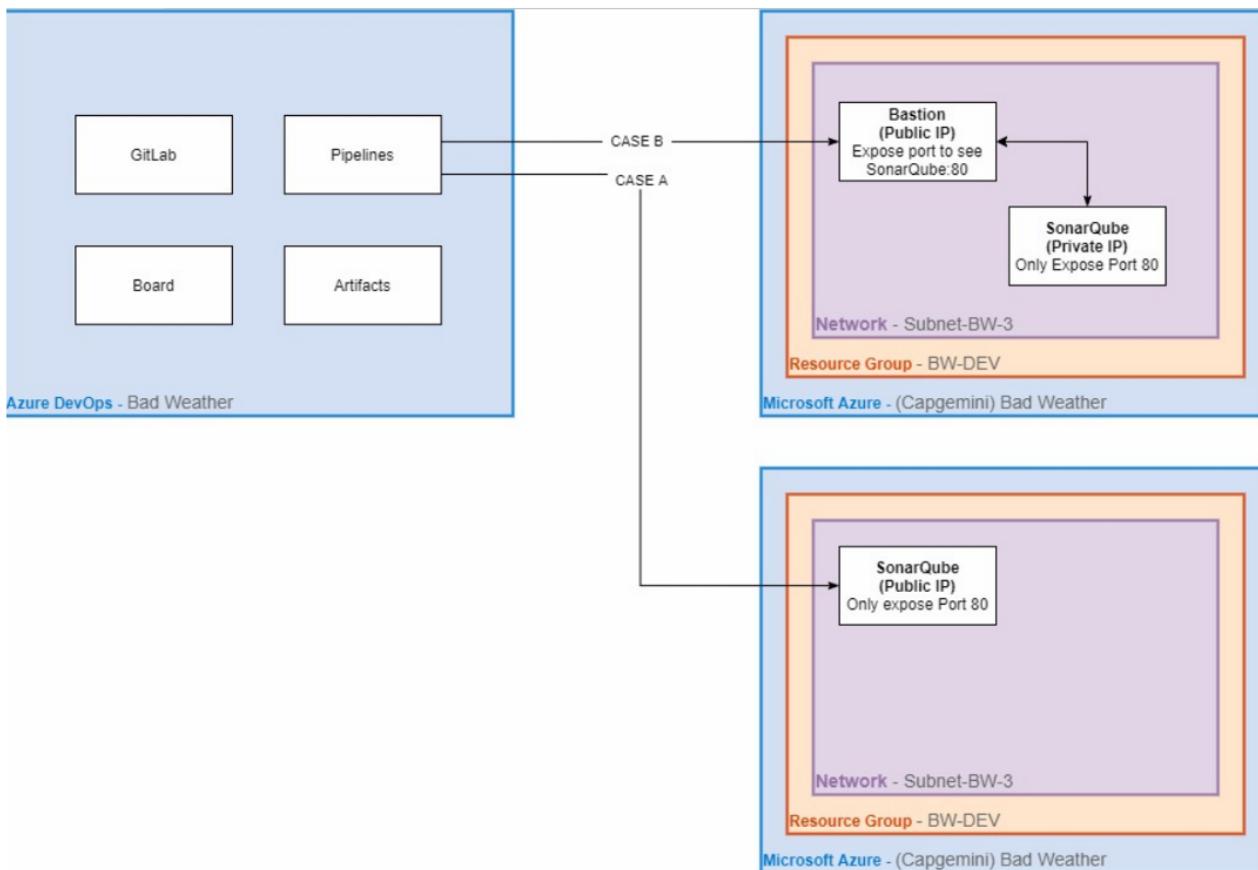
You are connected:

```
@LES007975 MINGW64 ~/Documents/Bad Weather/sonar
$ ssh -i Sonarkey.pem azureuser@51.103.78.61
The authenticity of host '51.103.78.61 (51.103.78.61)' can't be established.
ECDSA key fingerprint is SHA256:rMuCVUF2HqdDcoFG1/5hfWTjeOFZ1+HtTStJQJxU13E.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '51.103.78.61' (ECDSA) to the list of known hosts.
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Mon Nov  2 16:35:39 2020 from 90.68.91.45
```

58.4.2. Install Sonar using Docker and Docker-compose

As an example we will use the practical case of Bad Weather, a project where we were asked to install Sonar inside a VM in Azure portal



We had 2 possible scenarios, we went for the **case A** since no other service will be installed in this VM

Steps

- 1- Install docker and docker compose in the VM

```
sudo dnf config-manager --add-repo=https://download.docker.com/linux/centos/docker-ce.repo
sudo dnf list docker-ce
sudo dnf install docker-ce --nobest -y
sudo systemctl start docker
sudo systemctl enable docker
docker --version
sudo dnf install curl -y
sudo curl -L "https://github.com/docker/compose/releases/download/1.25.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
docker-compose --version
```

2- Deploy SonarQube and Postgress

2.1- Set necesary parameters for sonarqube

```
sudo sysctl -w vm.max_map_count=262144
sudo sysctl -w fs.file-max=65536
sudo ulimit -n 65536
sudo ulimit -u 4096
```

2.2- Use docker-compose with the next definition to deploy it:

```
vim /home/sonar/docker-compose.yaml
```

```

version: "3"

services:
  sonarqube:
    image: "sonarqube:7.9-community"
    networks:
      - sonar
    environment:
      - sonar.jdbc.username=user
      - sonar.jdbc.password=pass
      - sonar.jdbc.url=jdbc:postgresql://sonarqube-db:5432/sonar
    ports:
      - "80:9000"
    depends_on:
      - "sonarqube-db"
    volumes:
      - "$PWD/volumes/sonarqube/conf:/opt/sonarqube/conf"
      - "$PWD/volumes/sonarqube/data:/opt/sonarqube/data"
      - "$PWD/volumes/sonarqube/extensions:/opt/sonarqube/extensions"
      - "$PWD/volumes/sonarqube/logs:/opt/sonarqube/logs"
    ulimits:
      nofile:
        soft: 65536
        hard: 65536
  sonarqube-db:
    image: "postgres:12-alpine"
    networks:
      - sonar
    volumes:
      - "$PWD/volumes/sonarqube-db/data:/var/lib/postgresql/data"
    environment:
      - POSTGRES_USER=youruser
      - POSTGRES_PASSWORD=yourpass
      - POSTGRES_DB=sonar
      - PGDATA=/var/lib/postgresql/data

networks:
  sonar:
    driver: bridge

```

3- Update the start configuration to set automatically the correct values and run the docker-compose

```

vim /usr/local/sbin/start.sh

sysctl -w vm.max_map_count=262144
sysctl -w fs.file-max=65536
ulimit -n 65536
ulimit -u 4096

cd /home/sonar && docker-compose up -d

```

4- Add this to execute the docker-compose file every time the machine turns on

```

crontab -e
@reboot /usr/local/sbin/start.sh

vim /etc/sysctl.conf
vm.max_map_count=262144
fs.file-max=65536

```

Your Sonar is Up and running in your VM

58.4.3. Create an Azure pipeline from scratch

The following steps will allow you to create a basic pipeline in Azure Devops from scratch

In order to deploy in Azure, we've created an automatic pipeline in Azure Devops that will be executed automatically when developers make a push to the Azure repositories, the pipeline will compile the code, build the application and ensure with automatic tests that the build is not going to break the application, to ensure a good quality code the code will be analyzed by sonar as well as your code coverage and last but not least, your application will be deployed using Azure App Services.

58.4.4. Steps

1- Sign in to your Azure DevOps organization and navigate to your project.

2- Go to Pipelines, and then select New Pipeline.

3- Choose the location of your source code(Github, Bitbucket,Azure repos..etc), in this case we have our code in Azure Repos Git.

A list of your repositories will be shown here:

4- When the list of repositories appears, select your repository.

Depending on your project type(Java, .NET, Python or JavaScript) the following configuration will change, in this case our project is a .NET, for more type of projects please follow the [official documentation](#).

5- When the Configure tab appears, select ASP.NET Core(or the one according to your project)

New pipeline

Configure your pipeline

- ASP.NET** Build and test ASP.NET projects.
- .NET Core (.NET Framework)** Build and test ASP.NET Core projects targeting the full .NET Framework. Default path and default name
- .NET Desktop** Build and run tests for .NET Desktop or Windows classic desktop solutions.

6- A .yaml file in your ./ location will be generated with all the required steps to run your pipeline. The name of this .yaml file is 'azure-pipelines.yaml' which is the default name that will be used in your pipeline settings.

Note: If you change the name or the location, you will need to specify in the pipeline settings the new name or location:

Pipeline settings



Processing of new run requests

- Enabled
- Paused
- Disabled

Default path and default name

YAML file path

azure-pipelines.yml



The pipeline is created with the minimum required steps to run it which are the following:

TRIGGERS

Triggers that will activate the pipeline execution

trigger:

- master
- develop

VARIABLES

Variables that will be used in the next steps

```
variables:  
  solution: '**/*.sln'  
  buildPlatform: 'Any CPU'  
  buildConfiguration: 'Release'
```

TOOLS AND LIBRARIES

For .NET:

-NuGet Tool Installer task:

- task: NuGetToolInstaller@1

Use this task to find, download, and cache a specified version of NuGet and add it to the PATH.

-The NuGet command to run:

```
- task: NuGetCommand@2  
  inputs:  
    restoreSolution: '$(solution)'
```

The NuGet command to run.

For more info use the official [documentation](#).

BUILD

-Visual Studio Build task:

```
- task: VSBuild@1
  inputs:
    solution: '$(solution)'
    msbuildArgs: '/p:DeployOnBuild=true /p:WebPublishMethod=Package
/p:PackageAsSingleFile=true /p:SkipInvalidConfigurations=true
/p:DesktopBuildPackageLocation="$(build.artifactStagingDirectory)\WebApp.zip"
/p:DeployIisAppPath="Default Web Site"'
    platform: '$(buildPlatform)'
    configuration: '$(buildConfiguration)'
```

Use this task to build with MSBuild and set the Visual Studio version property.

For more info use the official [documentation](#)

TEST

-Visual Studio Test task:

```
- task: DotNetCoreCLI@2
  inputs:
    command: 'test'
    arguments: '/p:CollectCoverage=true /p:CoverletOutputFormat=opencover
/p:CoverletOutput=$(Agent.TempDirectory)/'
    projects: '$(solution)'
    publishTestResults: true
    continueOnError: false
    displayName: 'Dot Net Core CLI Test'
```

Use this task to run unit and functional tests (Selenium, Appium, Coded UI test, and more) using the Visual Studio Test Runner.

For more info use the official [documentation](#)

This steps are the ones generated when your pipeline is created, we can create the ones we need using the Azure Devops wizard in an easy way.

In our case, apart from build and test, we also need to deploy

DEPLOY

App Services

While deploying with App Services, 2 steps are required:

Step 1: Publish

Use this task in a pipeline to publish artifacts for the Azure Pipeline

```
- task: PublishPipelineArtifact@0
  inputs:
    artifactName: 'Bad_Weather_Backend'
    targetPath: '$(Build.ArtifactStagingDirectory)'
```

To know more about the use of predefined variables in azure take a look at the [documentation](#)

Step 2: Deployment

Use this task to deploy to a range of App Services on Azure

```
- task: AzureRmWebAppDeployment@4
  inputs:
    ConnectionType: 'AzureRM'
    azureSubscription: 'bad-weather-poc-rs-bw-dev'
    appType: 'webApp'
    WebAppName: 'bwbackendbe'
    packageForLinux: '$(build.artifactStagingDirectory)\WebApp.zip'
```

This task has 2 prerequisites:

1-App Service instance:

The task is used to deploy a Web App project or Azure Function project to an existing Azure App Service instance, which must exist before the task runs.

2-Azure Subscription:

In order to deploy to Azure, an Azure subscription must be [linked to the pipeline](#).

To know more about the input arguments for this task, make use of the official [documentation](#)

58.4.5. CONNECTION STRINGS

Once your database is created, you will need to connect that DB to your backend application, this can be made using [connection strings](#).

CREATE THE CONNECTION STRING

Go to the Azure portal, select the App Service that you want to connect with the DB, to be able to establish this connection, both your DB and your App Service must be under the same resource group.

P.E

The screenshot shows the Azure portal interface. At the top left, it says 'Home > SQL databases >'. Below that is a section titled 'SQL databases' with a 'Capgemini' logo. It has buttons for '+ Add', 'Reservations', and '...'. A tooltip says 'Try our new Azure SQL resource browser! This experience offers a...'. On the right, there's a detailed view for 'bw-sql-dev (bw-sql-srv-dev/bw-sql-dev)' which is a 'SQL database'. It has a search bar, 'Copy', 'Restore', and 'Export' buttons. Below that is an 'Overview' section and an 'Activity log'. To the right, under 'Essentials', it says 'Resource group (change) BW-dev'. Another red box highlights this 'BW-dev' entry. Below this is another resource, 'bwbackendbe', which is an 'App Service'. It has a search bar, 'Browse', 'Stop', 'Swap', 'Restart', 'Delete', 'Refresh', 'Get publish profile', and 'Reset publish pro'. It also has sections for 'Overview', 'Activity log', 'Access control (IAM)', and 'Tags'. To the right of 'bwbackendbe', it shows 'URL https://bwbackendbe.azurewebsites.net'. Under 'Essentials' for 'bwbackendbe', it also says 'Resource group (change) BW-dev', which is highlighted with a red box.

As we can see here, both the app service and the DB exist under the same resource group 'BW-dev'

Select your app service and go to 'settings > Configuration', scroll down looking for 'Connection strings' and click on "New connection string"

The screenshot shows the 'bwbackendbe' App Service configuration page. The left sidebar has 'Deployment' (selected), 'Settings' (highlighted with a red box), 'Configuration' (highlighted with a red box), 'Authentication / Authorization', and 'Application Insights'. The main area shows 'PBI_TenantId', 'PBI_WorkspaceId', 'PbiPassword', and 'PbiUsername' under 'App Config'. Below this is a 'Connection strings' section with a note: 'Connection strings are encrypted at rest and transmitted over an encrypted channel.' It has buttons for '+ New connection string', 'Show values', and 'Advanced edit', with '+ New connection string' highlighted with a red box. There's also a 'Filter connection strings' input field.

Put the name you want (we've put the name 'Context', this name will be used later in your appSettings.json) and select the DB type, and for fill the value box go to 'Home>SQL databases', click on the target DB and click on 'Show database connection strings', copy the value that appears there and paste it in the value box.

SQL databases

Capgemini

[+ Add](#)[Reservations](#)

...

i Try our new Azure SQL resource browser! This experience offers a unified view of all your SQL Server resources in Azure as well as improved sorting and filtering. Click here to go to the new experience.

Filter by name...

 Name ↑↓ [bw-sql-dev \(bw-sql-srv-dev/...\)](#) ...**bw-sql-dev (bw-sql-srv-dev/bw-sql-dev)**

SQL database

[Search \(Ctrl+/\)](#)[Copy](#)[Restore](#)[Export](#)[Set server firewall](#)[Delete](#)[Connect with...](#)

...

[JSON View](#)

Power Platform

Power BI (preview)

Power Apps (preview)

Power Automate (preview)

Settings

[Configure](#)[Geo-Replication](#)[Connection strings](#)[Sync to other databases](#)

Essentials

Resource group ([change](#))

BW-dev

Status

Online

Location

France Central

Subscription ([change](#))

Bad weather - PoC

Subscription ID

1cdec8f1-19c4-4c8c-86c9-84b6f046ea94

Taas ([change](#))

Server name

bw-sql-srv-dev.database.windows.net

Elastic pool

No elastic pool

Connection strings

[Show database connection strings](#)

Pricing tier

General Purpose: Gen5, 2 vCores

Earliest restore point

2020-12-10 00:00 UTC

bw-sql-dev (bw-sql-srv-dev/bw-sql-dev) | Connection strings

SQL database

[Search \(Ctrl+/\)](#)

Power Platform

Power BI (preview)

Power Apps (preview)

Power Automate (preview)

Settings

[Configure](#)[Geo-Replication](#)[Connection strings](#)

ADO.NET

JDBC

ODBC

PHP

Go

ADO.NET (SQL authentication)

```
Server=tcp:bw-sql-srv-dev.database.windows.net,1433;Initial Catalog=bw-sql-dev;Persist Security
Info=False;User ID=AzureUser;Password=
{your_password};MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection
Timeout=30;
```

[Copy all this text this is your
connection string value](#)

ADO.NET (Active Directory password authentication)

Paste the connection string in the 'value' box and click OK

Your connection string has been created.

USE THE CONNECTION STRING

Go to your project, open the file AppSettings.json and add the connection string

```
".ConnectionStrings": {
    "Context": "Source=(localdb)\\MSSQLLocalDB;Initial Catalog=my-db;Integrated
Security=True;"}
```

Context is the name that we choose for the connection string that we've created before and that value is only for local purposes.

When the application is deployed, the value for context will be replaced for the value of the connection string that we've created in the earlier steps, using this we avoid to put the user and the password into the code and we use them as secrets that will be replaced in the deployment.

58.4.6. Integrate the SonarQube plugin in an Azure DevOps pipeline

The purpose of this readme is that you can configure your Azure Devops pipeline in order to be able to run a code analysis, analyse the code coverage and publish the results through the Sonar plugin.

58.4.7. How to do it

Step 1: Create a service connection

The first thing to do is to declare your SonarQube server as a service endpoint in your Azure DevOps project settings.

Go to project settings → pipelines → service connections and **create** and choose 'SonarQube'.

Create service connection

Specify the server url and the connection name of your SonarQube server and the token Auth Go to your SonarQube server and log in as admin, once inside, go to administration → Security → Users → Administrator → Tokens → And **generate** the token. Copy the generated token(once created it will never appear again so don't lose it) and paste it and click on **save**.

Add SonarQube service connection

Connection name	SonarQube
Server Url	https://sonarqube-server:9000
Token
<input checked="" type="checkbox"/> Allow all pipelines to use this connection.	
<input type="button" value="OK"/> <input type="button" value="Close"/>	

The service connection has been created. Once this step is done your service creation will appear now in the *service connections* side bar.

For more info regarding the Authentication part please read the [official documentation](#)

Step 2: Add the required tasks in the azure pipeline

In order to integrate the SonarQube in the pipeline, 3 steps or **tasks** are required(Depending on the different solutions like .NET, Java, C..etc some of this tasks can be optional), this tasks are:

Prepare Analysis configuration Run Code Analysis Publish Quality Gate result

We can use the wizard to create this in an easy way, search "SonarQube" and let's configure the tasks one by one.

Prepare Analysis configuration:

Fill the required fields and click on **add**

The **prepare** task will be now shown in the pipeline code:

```
Settings
- task: SonarQubePrepare@4
  inputs:
    SonarQube: 'https://sonarqube'
    scannerMode: 'Scanner'
    projectKey: 'h'
    projectName: 'E'
```

Follow the [official documentation](#) if you have doubts while filling the fields:

Once the **prepare** is done, continue with the code analysis.

Run Code Analysis

Select this from the task assistant and just like happened with the first task, the code will appear in your pipeline.

```
- task: SonarQubeAnalyze@4
```

Now, let's publish the result of the analysis.

Publish quality gate result

Same as we did before, select in the display the publish extension and add it

```
Settings
- task: SonarQubePublish@4
  inputs:
    pollingTimeoutSec: '300' ]
```

Step 3: Run the pipeline

With this, all the required steps to integrate SonarQube in your Azure DevOps pipeline are done, the last thing you need to do is **run** your pipeline and your code will be analyzed and the results published.

58.4.8. Install and use custom sonar plugin in Azure Devops

By default, the sonar plugin is not capable to be used in every branch you want to, to do this you need to purchase a license or customize the current plugin in order to satisfy our needs.

How to costumize the plugin is not the purpose of this documentation, this documentation is for the intallment and use of it.

If you want to install a custom plugin, sign into your Azure Devops organization and once you are in, click on the marketplace icon:



Select *browse marketplace>publish extension

Choose the extension you want to install and click on the options

ADC-devonfw (ADC-devonfw)

Extensions Details Members + New extension search

Name ↑	Version	Updated	Availability	Rating	Installs	
devonfw-SonarQube	...	0.0.6	4 days ago	Private (shared with 1)	★★★★★ (0)	1

Important:

You need to choose the organization for which you are going to use the extension and **share it**, if not, you won't be able to install it.

devonfw-SonarQube ... 0.0.6

- [Reports](#)
- [View Extension](#)
- [Update](#)
- [Remove](#)
- [Share/Unshare](#)
- [Certificate](#)

Once you've done this click on View extension and 'Get it free', the extension will be downloaded and you will be able to use it in the next screen



Select an Azure DevOps organization

If there are no organizations you can see the possible causes [here](#).

Another cause might be that you forgot to share the extension.

Note: If the install button does not appear, it's possible that you don't have permissions to install it so you will need to talk with the owner of the org. Another possibility is that you can *request* an installation.

Once installed, in the pipeline wizard it will appear and you will be able to select it.

Tasks



sonar



devonfw Prepare Analysis Configuration
devonfw Prepare SonarQube analysis configuration



Prepare Analysis Configuration
Prepare SonarQube analysis configuration

We can see in the image the default plugin and the customized one.

58.5. OKD (OpenShift Origin)

58.6. OKD (*OpenShift Origin*)

58.6.1. What is OKD

OKD is a distribution of Kubernetes optimized for continuous application development and multi-tenant deployment. OKD is the upstream Kubernetes distribution embedded in Red Hat OpenShift.

OKD embeds Kubernetes and extends it with security and other integrated concepts. OKD is also referred to as Origin in github and in the documentation.

OKD provides a complete open source container application platform. If you are looking for enterprise-level support, or information on partner certification, Red Hat also offers [Red Hat OpenShift Container Platform](#).

Continue reading...

- [How to install Openshift Origin](#)
- [Initial setup](#)
 - [s2i](#)
 - [templates](#)
 - [Customize Openshift](#)
 - [Customize icons](#)
 - [Customize catalog](#)

58.6.2. OKD (*OpenShift Origin*)

What is OKD

OKD is a distribution of Kubernetes optimized for continuous application development and multi-tenant deployment. OKD is the upstream Kubernetes distribution embedded in Red Hat OpenShift.

OKD embeds Kubernetes and extends it with security and other integrated concepts. OKD is also referred to as Origin in github and in the documentation.

OKD provides a complete open source container application platform. If you are looking for enterprise-level support, or information on partner certification, Red Hat also offers [Red Hat OpenShift Container Platform](#).

Continue reading...

- [How to install Openshift Origin](#)
- [Initial setup](#)
 - [s2i](#)

- templates
- Customize Openshift
 - Customize icons
 - Customize catalog

58.6.3. Install OKD (*Openshift Origin*)

Pre-requisites

Install docker

<https://docs.docker.com/engine/installation/linux/docker-ce/debian/#set-up-the-repository>

```
$ sudo groupadd docker  
$ sudo usermod -aG docker $USER
```

Download Openshift Origin Client

Download Openshift Origin Client from [here](#)

When the download it's complete, only extract it on the directory that you want, for example `/home/administrador/oc`

Add oc to path

```
$ export PATH=$PATH:/home/administrador/oc
```

Install Openshift Cluster

Add the insecure registry

Create file `/etc/docker/daemon.json` with the next content:

```
{  
  "insecure-registries" : [ "172.30.0.0/16" ]  
}
```

Download docker images for openshift

```
$ oc cluster up
```

Install Oc Cluster Wrapper

To manage easier the cluster persistent, we are going to use oc cluster wrapper.

```
cd /home/administrador/oc
wget https://raw.githubusercontent.com/openshift-evangelists/oc-cluster-
wrapper/master/oc-cluster
```

```
oc-cluster up devonfw-shop-floor --public-hostname X.X.X.X
```

Configure iptables

We must create iptables rules to allow traffic from other machines.

- The next commands it's to let all traffic, don't do it on a real server.
 - \$ iptables -F
 - \$ iptables -X
 - \$ iptables -t nat -F
 - \$ iptables -t nat -X
 - \$ iptables -t mangle -F
 - \$ iptables -t mangle -X
 - \$ iptables -P INPUT ACCEPT
 - \$ iptables -P OUTPUT ACCEPT
 - \$ iptables -P FORWARD ACCEPT

58.6.4. How to use Oc Cluster Wrapper

With oc cluster wrapper we could have different clusters with different context.

Cluster up

```
$ oc-cluster up devonfw-shop-floor --public-hostname X.X.X.X
```

Cluster down

```
$ oc-cluster down
```

Use non-persistent cluster

```
oc cluster up --image openshift/origin --public-hostname X.X.X.X --routing-suffix
apps.X.X.X.X.nip.io
```

58.6.5. devonfw Openshift Origin Initial Setup

These are scripts to customize an Openshift cluster to be a devonfw Openshift.

How to use

Prerequisite: Customize Openshift

devonfw Openshift Origin use custom icons, and we need to add it to openshift. More information:

- [Customize Openshift](#)

Script initial-setup

Download [this](#) script and execute it.

More information about what this script does [here](#).

Known issues

Failed to push image

If you receive an error like this:

```
error: build error: Failed to push image: After retrying 6 times, Push image still failed due to error: Get http://172.30.1.1:5000/v2/: dial tcp 172.30.1.1:5000: getsockopt: connection refused
```

It's because the registry isn't working, go to openshift console and enter into the **default** project <https://x.x.x.x:8443/console/project/default/overview> and you must see two resources, **docker-registry** and **router** they must be running. If they don't work, try to deploy them and look at the logs what is happen.

58.6.6. s2i devonfw

This are the s2i source and templates to build an s2i images. It provides OpenShift builder images for components of the devonfw (at this moment only for angular and java).

This work is totally based on the implementation of [Michael Kuehl](#) from RedHat for Oasp s2i.

All this information is used as a part of the [initial setup](#) for openshift.

Previous setup

In order to build all of this, it will be necessary, first, to have a running OpenShift cluster. How to install it [here](#).

Usage

Before using the builder images, add them to the OpenShift cluster.

Deploy the Source-2-Image builder images

First, create a dedicated **devonfw** project as admin.

```
$ oc new-project devonfw --display-name='devonfw' --description='devonfw Application Standard Platform'
```

Now add the builder image configuration and start their build.

```
oc create -f https://raw.githubusercontent.com/devonfw/devonfw-shop-floor/master/dsf4openshift/openshift-devonfw-deployment/s2i/java/s2i-devonfw-java-imagestream.json --namespace=devonfw  
oc create -f https://raw.githubusercontent.com/devonfw/devonfw-shop-floor/master/dsf4openshift/openshift-devonfw-deployment/s2i/angular/s2i-devonfw-angular-imagestream.json --namespace=devonfw  
oc start-build s2i-devonfw-java --namespace=devonfw  
oc start-build s2i-devonfw-angular --namespace=devonfw
```

Make sure other projects can access the builder images:

```
oc policy add-role-to-group system:image-puller system:authenticated  
--namespace=devonfw
```

That's all!

Deploy devonfw templates

Now, it's time to create devonfw templates to use this s2i and add it to the browse catalog. More information [here](#).

Build All

Use [this](#) script to automatically install and build all image streams. The script also creates templates devonfw-angular and devonfw-java inside the project 'openshift' to be used by everyone.

1. Open a bash shell as Administrator
2. Execute shell file:

```
$ /PATH/TO/BUILD/FILE/initial-setup.sh
```

More information about what this script does [here](#).

Links & References

This is a list of useful articles, etc, that I found while creating the templates.

- [Template Icons](#)
- [Red Hat Cool Store Microservice Demo](#)
- [Openshift Web Console Customization](#)

58.6.7. devonfw templates

This are the devonfw templates to build devonfw apps for Openshift using the s2i images. They are based on the work of Mickuehl in Oasp templates/mythaistar for deploy My Thai Star.

- Inside the `example-mythaistar` we have an example to deploy My Thai Star application using devonfw templates.

All this information is used as a part of the [initial setup](#) for openshift.

How to use

Previous requirements

Deploy the Source-2-Image builder images

Remember that this templates need a build image from s2i-devonfw-angular and s2i-devonfw-java. More information:

- [Deploy the Source-2-Image builder images](#).

Customize Openshift

Remember that this templates also have custom icons, and to use it, we must modify the master-config.yml inside openshift. More information:

- [Customize Openshift](#).

Deploy devonfw templates

Now, it's time to create devonfw templates to use this s2i and add it to the browse catalog.

To let all user to use these templates in all openshift projects, we should create it in an openshift namespace. To do that, we must log in as an admin.

```
oc create -f https://raw.githubusercontent.com/devonfw/devonfw-shop-floor/master/dsf4openshift/openshift-devonfw-deployment/templates/devonfw-java-template.json --namespace=openshift
oc create -f https://raw.githubusercontent.com/devonfw/devonfw-shop-floor/master/dsf4openshift/openshift-devonfw-deployment/templates/devonfw-angular-template.json --namespace=openshift
```

When it finishes, remember to logout as an admin and enter with our normal user.

```
$ oc login
```

How to use devonfw templates in openshift

To use these templates with openshift, we can override any parameter values defined in the file by

adding the --param-file=paramfile option.

This file must be a list of <name>=<value> pairs. A parameter reference may appear in any text field inside the template items.

The parameters that we must override are the following

```
$ cat paramfile
APPLICATION_NAME=app-Name
APPLICATION_GROUP_NAME=group-Name
GIT_URI=Git uri
GIT_REF=master
CONTEXT_DIR=/context
```

The following parameters are optional

```
$ cat paramfile
APPLICATION_HOSTNAME=Custom hostname for service routes. Leave blank for default
hostname, e.g.: <application-name>.<project>.<default-domain-suffix>,
# Only for angular
REST_ENDPOINT_URL=The URL of the backend's REST API endpoint. This can be declared
after,
REST_ENDPOINT_PATTERN=The pattern URL of the backend's REST API endpoint that must
be modify by the REST_ENDPOINT_URL variable,
```

For example, to deploy My Thai Star Java

```
$ cat paramfile
APPLICATION_NAME="mythaistar-java"
APPLICATION_GROUP_NAME="My-Thai-Star"
GIT_URI="https://github.com/oasp/my-thai-star.git"
GIT_REF="develop"
CONTEXT_DIR="/java/mtsjs"

$ oc new-app --template=devonfw-java --namespace=mythaistar --param-file=paramfile
```

58.6.8. Customize Openshift Origin for devonfw

This is a guide to customize Openshift cluster.

Images Styles

The icons for templates must measure the same as below or the images don't show right:

- **Openshift logo:** 230px x 40px.
- **Template logo:** 50px x 50px.
- **Category logo:** 110px x 36px.

How to use

To use it, we need to enter in openshift as an admin and use the next command:

```
$ oc login
$ oc edit configmap/webconsole-config -n openshift-web-console
```

After this, we can see in our shell the webconsole-config.yaml, we only need to navigate until **extensions** and add the url for our own **css** in the **stylesheetURLs** and **javascript** in the **scriptURLs** section.

IMPORTANT: Scripts and stylesheets must be served with the correct content type or they will not be run by the browser. Scripts must be served with Content-Type: application/javascript and stylesheets with Content-Type: text/css.

In git repositories, the content type of raw is text/plain. You can use [rawgit](#) to convert a raw from a git repository to the correct content type.

Example:

```
webconsole-config.yaml: |
[...]
extensions:
  scriptURLs:
    - https://cdn.rawgit.com/devonfw/devonfw-shop-
floor/master/dsf4openshift/openshift-cluster-setup/initial-
setup/customizeOpenshift/scripts/catalog-categories.js
  stylesheetURLs:
    - https://cdn.rawgit.com/devonfw/devonfw-shop-
floor/master/dsf4openshift/openshift-cluster-setup/initial-
setup/customizeOpenshift/stylesheets/icon.css
[...]
```

More information

- [Customize icons](#) for Openshift.
- [Customize catalog](#) for Openshift.
- [Openshift docs](#) about customization.

Old versions

- Customize Openshift for [version 3.7](#).

How to add Custom Icons inside openshift

This is a guide to add custom icons into an Openshift cluster.

Here we can find an icons.css example to use the devonfw icons.

Images Styles

The icons for templates must measure the same as below or the images don't show right:

- **Openshift logo:** 230px x 40px.
- **Template logo:** 50px x 50px.
- **Category logo:** 110px x 36px.

Create a css

Custom logo for openshift cluster

For this example, we are going to call the css icons.css but you can call as you wish. Openshift cluster draw their icon by the id header-logo, then we only need to add to our icons.css the next Style Attribute ID

```
#header-logo {  
    background-image: url("https://raw.githubusercontent.com/devonfw/devonfw-shop-  
floor/master/dsf4openshift/openshift-cluster-setup/initial-  
setup/customizeOpenshift/images/devonfw-openshift.png");  
    width: 230px;  
    height: 40px;  
}
```

Custom icons for templates

To use a custom icon to a template openshift use a class name. Then, we need to insert inside our icons.css the next Style Class

```
.devonfw-logo {  
    background-image: url("https://raw.githubusercontent.com/devonfw/devonfw-shop-  
floor/master/dsf4openshift/openshift-cluster-setup/initial-  
setup/customizeOpenshift/images/devonfw.png");  
    width: 50px;  
    height: 50px;  
}
```

To show that custom icon on a template, we only need to write the name of our class in the tag "iconClass" of our template.

```
{
  ...
  "items": [
    {
      ...
      "metadata": {
        ...
        "annotations": {
          ...
          "iconClass": "devonfw-logo",
          ...
        }
      },
      ...
    }
  ]
}
```

Use our own css inside openshift

To do that, we need to enter in openshift as an admin and use the next command:

```
$ oc login
$ oc edit configmap/webconsole-config -n openshift-web-console
```

After this, we can see in our shell the webconsole-config.yaml, we only need to navigate until **extensions** and add the url for our own **css** in the **stylesheetURLs** section.

IMPORTANT: Scripts and stylesheets must be served with the correct content type or they will not be run by the browser. stylesheets must be served with Content-Type: text/css.

In git repositories, the content type of raw is text/plain. You can use [rawgit](#) to convert a raw from a git repository to the correct content type.

Example:

```
webconsole-config.yaml: |
  [...]
  extensions:
    stylesheetURLs:
      - https://cdn.rawgit.com/devonfw/devonfw-shop-
floor/master/dsf4openshift/openshift-cluster-setup/initial-
setup/customizeOpenshift/stylesheets/icons.css
  [...]
```

How to add custom catalog categories inside openshift

This is a guide to add custom [Catalog Categories](#) into an Openshift cluster.

[Here](#) we can find a catalog-categories.js example to use the devonfw catalog categories.

Create a script to add custom langaages and custom catalog categories

Custom language

For this example, we are going add a new language into the languages category. To do that we must create a script and we named as catalog-categories.js

```
// Find the Languages category.  
var category = _.find(window.OPENSHIFT_CONSTANTS.SERVICE_CATALOG_CATEGORIES,  
    { id: 'languages' });  
// Add Go as a new subcategory under Languages.  
category.subCategories.splice(2,0,{ // Insert at the third spot.  
    // Required. Must be unique.  
    id: "devonfw-languages",  
    // Required.  
    label: "devonfw",  
    // Optional. If specified, defines a unique icon for this item.  
    icon: "devonfw-logo-language",  
    // Required. Items matching any tag will appear in this subcategory.  
    tags: [  
        "devonfw",  
        "devonfw-angular",  
        "devonfw-java"  
    ]  
});
```

Custom category

For this example, we are going add a new category into the category tab. To do that we must create a script and we named as catalog-categories.js

```
// Add a Featured category as the first category tab.
window.OPENSHIFT_CONSTANTS.SERVICE_CATALOG_CATEGORIES.unshift({
  // Required. Must be unique.
  id: "devonfw-featured",
  // Required
  label: "devonfw",
  subCategories: [
    {
      // Required. Must be unique.
      id: "devonfw-languages",
      // Required.
      label: "devonfw",
      // Optional. If specified, defines a unique icon for this item.
      icon: "devonfw-logo-language",
      // Required. Items matching any tag will appear in this subcategory.
      tags: [
        "devonfw",
        "devonfw-angular",
        "devonfw-java"
      ]
    }
  ]
});
```

Use our own javascript inside openshift

To do that, we need to enter in openshift as an admin and use the next command:

```
$ oc login
$ oc edit configmap/webconsole-config -n openshift-web-console
```

After this, we can see in our shell the webconsole-config.yaml, we only need to navigate until **extensions** and add the url for our own **javascript** in the **scriptURLs** section.

IMPORTANT: Scripts and stylesheets must be served with the correct content type or they will not be run by the browser. Scripts must be served with Content-Type: application/javascript.

In git repositories, the content type of raw is text/plain. You can use [rawgit](#) to convert a raw from a git repository to the correct content type.

Example:

```
webconsole-config.yaml: |
  [...]
  extensions:
    scriptURLs:
      - https://cdn.rawgit.com/devonfw/devonfw-shop-
        floor/master/dsf4openshift/openshift-cluster-setup/initial-
        setup/customizeOpenshift/scripts/catalog-categories.js
  [...]
```

Customize Openshift Origin v3.7 for devonfw

This is a guide to customize Openshift cluster. For more information read the next:

- [Openshift docs customization](#) for the version 3.7.

Images Styles

The icons for templates must measure the same as below or the images don't show right:

- [Openshift logo](#): 230px x 40px.
- [Template logo](#): 50px x 50px.
- [Category logo](#): 110px x 36px.

Quick Use

This is a quick example to add custom icons and categories inside openshift.

To modify the icons inside openshift, we must to modify our master-config.yaml of our openshift cluster. This file is inside the openshift container and to obtain a copy of it, we must to know what's our openshift container name.

Obtain the master-config.yaml of our openshift cluster

Obtain the name of our openshift container

To obtain it, we can know it executing the next:

```
$ docker container ls
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS              PORTS
NAMES
83a4e3acda5b      openshift/origin:v3.7.0
"/usr/bin/openshift ..."   6 days ago       Up 6 days
origin
```

Here we can see that the name of the container is origin. Normally the container it's called as origin.

Copy the master-config.yaml of our openshift container to our directory

This file is inside the openshift container in the next directory: `/var/lib/origin/openshift.local.config/master/master-config.yaml` and we can copy it with the next command:

```
$ docker cp origin:/var/lib/origin/openshift.local.config/master/master-config.yaml ./
```

Now we have a file with the configuration of our openshift cluster.

Copy all customize files inside the openshift container

To use our customization of devonfw Openshift, we need to copy our files inside the openshift container.

To do this we need to copy the images, scripts and stylesheets from [here](#) inside openshift container, for example, we could put it all inside a folder called `openshift.local.devonfw`. On the step one we obtain the name of this container, for this example we assume that it's called `origin`. Then our images are located inside openshift container and we can see an access it in `/var/lib/origin/openshift.local.devonfw/images`.

```
$ docker cp ./openshift.local.devonfw origin:/var/lib/origin/
```

Edit and copy the master-config.yaml to use our customize files

The `master-config.yaml` have a sections to charge our custom files. All these sections are inside the `assetConfig` and their names are the next:

- The custom stylesheets are into `extensionStylesheets`.
- The custom scripts are into `extensionScripts`.
- The custom images are into `extensions`.

To use all our custom elements only need to add the directory routes of each element in their appropriate section of the `master-config.yaml`

```

...
assetConfig:
  ...
    extensionScripts:
      - /var/lib/origin/openshift.local.devonfw/scripts/catalog-categories.js
    extensionStylesheets:
      - /var/lib/origin/openshift.local.devonfw/stylesheet/icons.css
    extensions:
      - name: images
        sourceDirectory: /var/lib/origin/openshift.local.devonfw/images
  ...
...

```

Now we only need to copy that master-config.yaml inside openshift, and restart it to load the new configuration. To do that execute the next:

```
$ docker cp ./master-config.yaml
origin:/var/lib/origin/openshift.local.config/master/master-config.yaml
```

To re-start openshift do `oc cluster down` and start again your persistent openshift cluster.

More information

- [Customize icons](#) for Openshift.
- [Customize catalog](#) for Openshift.
- [Openshift docs](#) about customization.

58.7. ISTIO Guide

58.7.1. ISTIO Service Mesh Implementation Guide

Introduction

A service mesh separating applications from network functions like resilience, fault tolerance, etc.,

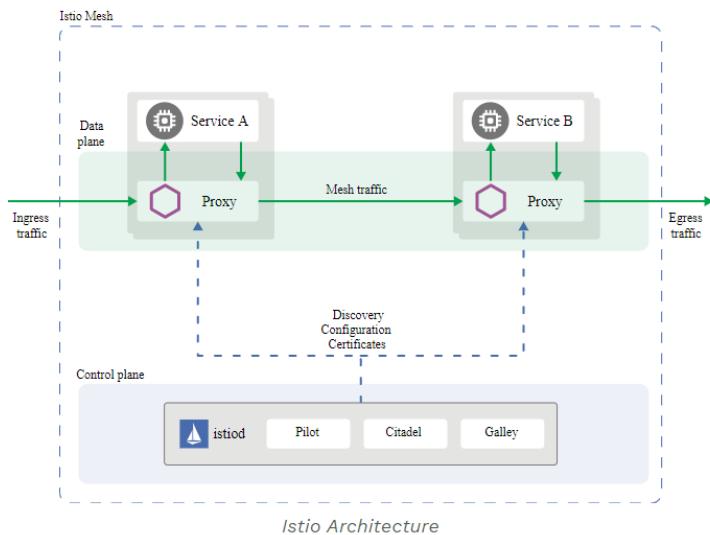
A service mesh addresses the below functions without changing the application code.

- Test the new versions of services without impacting the users.
- Scale the services.
- Find the services with the help of service registry.
- Test against failures.
- Secure service-to-service communication.
- Route traffic to a specific way.
- Circuit breaking and fault injection.

- Monitor the services and collect matrices.
- Tracing.

ISTIO service mesh is an open environment for Connecting, Securing, Monitoring services across the environments.

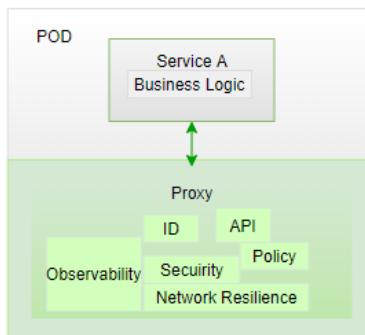
ISTIO Architecture



ISTIO is split into data plane and control plane. Refer [ISTIO Architecture](#)

Data Plane

The data plane is a set of intelligent proxies (Envoy) deployed as sidecars that mediate and control all network communication among microservices.



Control Plane

The control plane is managing and configuring proxies to route traffic and enforcing policies.

- Pilot manages all the proxies and responsible for routing
- Mixer collects telemetry and policy check
- Citadel does Certificate management (TLS certs to Envoy)

ISTIO installation

Download ISTIO from [releases](#)

```
istioctl install --set profile=demo
```

Here used demo profile, there are other profiles for production.

Verify installation:

```
kubectl get all -n istio-system
```

Inject sidecar container automatically by issuing the below command.

```
kubectl label namespace default istio-injection=enabled
```

Verify:

```
kubectl get namespace -L istio-injection
```

For more installation guides, refer [ISTIO Installation](#)

Traffic Management

ISITO's traffic management model relies on the Envoy proxies which deployed as sidecars to services.

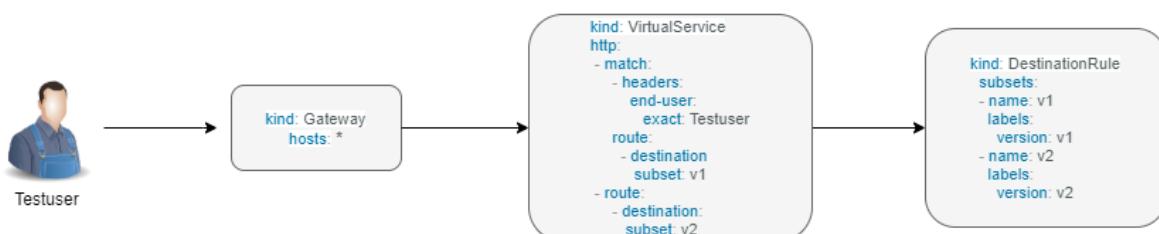
Below are the traffic management API resources

- Virtual Services
- Destination Rules
- Gateways
- Service Entries
- Sidecars

A virtual service, higher level abstraction of Kubernetes Service, lets you configure how requests are routed to a service within an Istio service mesh. Your mesh may have multiple virtual services or none. Virtual service consists of routing rules that are evaluated in order.

Dark Launch

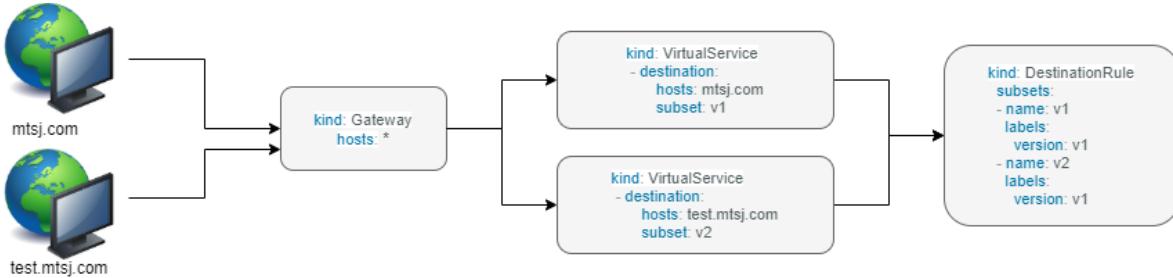
The following virtual service routes requests to different versions of a service depending on whether the request comes from a testuser. If the testuser calls then version v1 will be used, and for others version v2.



Blue/Green deployment

In blue/green deployment two versions of the application running. Both versions are live on different domain names, in this example it is mtsj.com and test.mtsj.com.

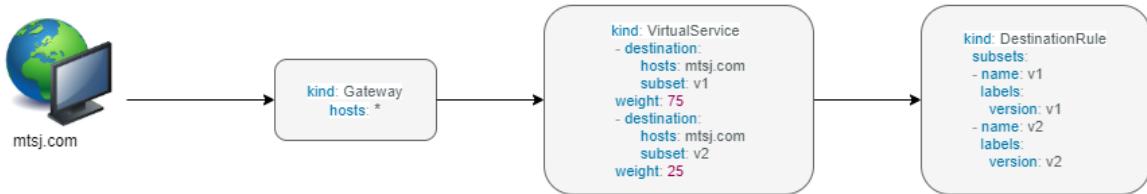
1. Define 2 virtual services for mtsj v1 and v2 versions.
2. Define DestinationRule and configure the subsets for v1 and v2.



When end user browses *mtsj.com*, the gateway call goes to subset v1 of the virtual service and redirects to destination version v1, and for *test.mtsj.com* to version v2.

Canary Deployment (Traffic Splitting)

In canary deployment old and new versions of the application alive. ISTIO can be configured, how much percentage of traffic can go to each version.



Here, the traffic is divided 75% to the version V1, and 25% to the version V2, as we gain confidence the percentage can be increased the latest version and gradually the traffic to the old version can be reduced and removed.

You may refer [ISTIO Traffic Management](#) for more details.

MyThaiStar Implementation

In this example dish will have two versions and the traffic will be routed alternately using the ISTIO configuration.

Find all configuration files in `istio/trafficmanagement/canary` directory under `mythaistar-microservices` example.

1. MyThaiStar defines below
 - a. Service
 - b. Service Account
 - c. Deployment

The above configurations are defined in a single yaml file for all the different services like angular, dish, image etc.

1. dish-v2: Dish Version 2 can be kept separately in different yaml file.
2. mts-gateway defines the ingress gateway which routes the outbound request to each service.
3. destination-rule-all defines the subsets here for later traffic routing
4. dish-50-50: traffic routing for different versions of dishmanagement.

Network Resilience

Timeout

Istio lets you adjust the timeouts using virtual services. The default timeout is 15 seconds.



Retry

A retry setting specifies the maximum number of times an Envoy proxy attempts to connect to a service if the initial call fails.



Retries can also be configured on Gateway Error, Connection failure, Connection Refused or any 5xx error from the application.

retryOn: gateway-error,connect-failure,refused-stream,5xx

Circuit Breakers

By defining the destination rule, set limits for calls to individual hosts within a service, such as the number of concurrent connections or how many times calls to this host have failed once the limit reached.

- Outlier Detection is an ISTIO Resiliency strategy to detect unusual host behaviour and evict the unhealthy hosts from the set of load balanced healthy hosts inside a cluster.

- If a request is sent to a service instance and it fails (returns a 50X error code), then ISTIO ejects the instance from the load balanced pool for a specified duration.

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: image-dr
spec:
  host: image
  trafficPolicy:
    connectionPool:
      tcp:
        maxConnections: 1
      http:
        http1MaxPendingRequests: 1
        maxRequestsPerConnection: 1
    outlierDetection:
      consecutiveErrors: 1
      interval: 1s
      baseEjectionTime: 3m
      maxEjectionPercent: 100
```

Fault Injection

Two types of faults can be generated using ISTIO. This is useful for the testing.

Delays: timing failures.

Aborts: crash failures.

Below example is a crash failure Virtual Service. The below example configured to receive http status 500 error for the testuser. The application works fine for all other users.

```
kind: VirtualService
...
spec:
  hosts:
    - ratings
  http:
    - fault:
        abort:
          httpStatus: 500
          percentage:
            value: 100
    match:
      - headers:
          end-user:
            exact: testuser
    route:
      - destination:
          host: ratings
          subset: v1
      - route:
          - destination:
              host: ratings
              subset: v1
```

The below virtual service configured to wait 10s for all requests.

```

kind: VirtualService
...
spec:
  hosts:
    - ratings
  http:
    - fault:
      delay:
        fixedDelay: 10s
      percentage:
        value: 100

```

Security

ISTIO provides security solution has the below functions.

- Traffic encryption
- Mutual TLS and fine-grained access policies.
- Auditing tools

Authentication

ISTIO provides two types of authentication.

- Peer authentication, secures service to service authentication
- Request authentication is end user authentication to verify credential attached to the request.

Mutual TLS Authentication

By default, the TLS protocol only proves the identity of the server to the client. Mutual TLS authentication ensures that traffic has been secure and trusted in both the directions between the client and server.

All traffic between services with proxies uses mutual TLS by default.

Peer Authentication

Peer authentication has Permissive, Strict and Disabled mode. With permissive mode, a service accepts both plain text and mutual TLS traffic. Permissive mode is good at the time of onboarding and should switch to Strict later.

The authentication policy can be applied to mesh-wide, namespace wide or workload specific using the selector field.

```
apiVersion: "security.istio.io/v1beta1"
kind: "PeerAuthentication"
spec:
  selector:
    matchLabels:
      app: bookings
  mTLS:
    mode: STRICT
```

Here the policy applied to the workload bookings.

Check the default mesh policy:

```
kubectl describe meshpolicy default
```

Request authentication

Request authentication policies specify the values needed to validate JWT tokens.

Authentication	Applies to	Uses	Identity
Peer authentication	Service to service	mTLS	source.principal
Request authentication	End User authentication	JWT	request.auth.principal

Authorization

Apply an authorization policy to the workload/namespace/mesh to enforce the access control. Supports ALLOW and DENY actions.

Deny All

Below example authorization policy without any rules denies access to all workloads in admin namespace.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
metadata:
  name: deny-all
  namespace: admin
spec:
  {}
```

Example below allowing the GET methods from order service.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
spec:
  selector:
    matchLabels:
      app: mtsj
  action: ALLOW
  rules:
    - from:
        - source:
            principals: ["cluster.local/ns/default/sa/order"]
        to:
          - operation:
              methods: ["GET"]
```

Example below denies the request to the /registered path for requests without request principals.

```
apiVersion: security.istio.io/v1beta1
kind: AuthorizationPolicy
spec:
  selector:
    matchLabels:
      app: mtsj
  action: DENY
  rules:
    - to:
        - operation:
            paths: ["/registered"]
      from
        - source:
            notRequestPrincipals: ["]
```

You may refer [ISTIO Security](#) for more details.

Observability

ISTIO generates

- Metrics - for monitor latency, traffic, errors and saturation.
- Distributed Traces to identify call flows and service dependencies
- Access Logs enables audit service behaviour to the individual service level.

Grafana dashboard

Grafana and Prometheus are preconfigured addons on ISTIO. To enable, choose the configuration profile which has Prometheus and Grafana enabled. Eg: Demo profile

Verify Prometheus and Grafana running in the cluster.

```
kubectl get pods -n istio-system
```

Kiali dashboard

The Kiali dashboard helps you understand the structure of your service mesh by displaying the topology. The demo profile enables Kiali dashboard also.

Access the Kiali dashboard. The default user name is admin and default password is admin.

```
istioctl dashboard kiali
```

You may refer [ISTIO Observability](#)

Minikube Troubleshooting Tips

This documentation provides the troubleshooting tips while working with minikube in a local machine.

1. Always start minikube with a minimum of 4GB of memory or more if available. Using command
`minikube start --memory=4096`
2. If minikube is not starting or throwing any error even after multiple attempts. Try the below tips:

- a. Delete the minikube in your local machine using `minikube delete` and do a fresh minikube start.
 - b. In any case, if minikube is not starting even after the above step, go to `.minikube` folder under the users directory and delete it manually. Now try starting minikube.
3. Set docker environment in minikube using `minikube docker-env`. Now all the docker commands that are run will be on the docker inside minikube. So building your application after executing the above command will have the application docker images available to minikube.
 - a. To exit minikube docker environment use `minikube docker-env -u`
 4. In any case, if you face any error related to docker image such as `Failed to pull image`, or `image not found` errors we will have to manually push the application docker image to minikube docker cache using the below commands.
 5. For better results - stop minikube using `minikube stop` command.
 6. Execute the command `minikube cache add imageName/tagName`.
 7. Now start the minikube. To verify if the docker image has been added to minikube docker execute `minikube ssh docker images`.
 8. To remove any docker image from minikube docker stop any containers running that docker image and then execute `minikube cache delete imageName/tagName`.
 9. To reload any docker image to minikube docker environment, execute `minikube cache reload`.
 10. In any case, if the docker images are not getting removed from minikube docker environment then navigate to `.minikube/cache/images` and then delete the particular image.

Execute the below command to make the Grafana available.

```
kubectl -n istio-system port-forward $(kubectl -n istio-system get pod -l app=grafana -o jsonpath='\{.items[0].metadata.name\}') 3000:3000 &
```

Use the below URLs to view the dashboard in local machine.

<http://localhost:3000/dashboard/db/istio-mesh-dashboard>

Part IX: cicdgen

59. CICDGEN

cicdgen is a devonfw tool for generate all code/files related to CICD. It will include/modify into your project all files that the project needs run a Jenkins cicd pipeline, to create a docker image based on your project, etc. It's based on angular schematics, so you can add it as a dependency into your project and generate the code using ng generate. In addition, it has its own CLI for those projects that are not angular based.

59.1. What is angular schematics?

Schematics are generators that transform an existing filesystem. They can create files, refactor existing files, or move files around.

What distinguishes Schematics from other generators, such as Yeoman or Yarn Create, is that schematics are purely descriptive; no changes are applied to the actual filesystem until everything is ready to be committed. There is no side effect, by design, in Schematics.

59.2. cicdgen CLI

In order to know more about how to use the cicdgen CLI, you can check the [CLI page](#)

59.3. cicdgen Schematics

In order to know more about how to use the cicdgen schematics, you can check the [schematics page](#)

59.4. Usage example

A [specific page](#) about how to use cicdgen is also available.

60. cicdgen CLI

60.1. CICDGEN CLI

cicdgen is a command line interface that helps you with some CICD in a devonfw project. At this moment we can only generate files related to CICD in a project but we plan to add more functionality in a future.

Installation

```
$ npm i -g @devonfw/cicdgen
```

Usage

Global arguments

- `--version`

Prints the cicdgen version number

- `--help`

Shows the usage of the command

Commands

Generate.

This command wraps the usage of angular schematics CLI. With this we generate files in a easy way and also print a better help about usage.

Available schematics that generate the code:

- `devon4j`
- `devon4ng`
- `devon4net`
- `devon4node`

Examples

- Generate all CICD files related to a devon4j project

```
$ cicdgen generate devon4j
```

- Generate all CICD files related to a devon4ng project with docker deployment.

```
$ cicdgen generate devon4ng --groupid com.devonfw --docker --registryurl docker-registry-devon.s2-eu.capgemini.com
```

- Generate all CICD files related to a devon4node project with OpenShift deployment.

```
$ cicdgen generate devon4ng --groupid com.devonfw --openshift --registryurl docker-registry-devon.s2-eu.capgemini.com --ocname default --ocn devonfw
```

60.2. cicdgen usage example

In this example we are going to show how to use cicdgen step by step in a devon4ng project.

1. Install cicdgen

cicdgen is already included in the devonfw distribution, but if you want to use it outside the devonfw console you can execute the following command:

```
$ npm i -g cicdgen
```

2. Generate a new devon4ng project using devonfw ide.

Inside a devonfw ide distribution execute the command (`devon ng create <app-name>`):

```
$ devon ng create devon4ng
```

3. Execute cicdgen generate command

As we want to send notifications to MS Teams, we need to create de connector first:

- Go to a channel in teams and click at the connectors button. Then click at the jenkins configure button.

Connectors for "devonfw shop floor" channel in "devonfw devs | Private" team X

Keep your group current with content and updates from other services.

Search
All
Sort by: Popularity ▾

MANAGE	Connectors for your team	Configure
Configured	Yammer <small>Updated</small> Receive updates from your Yammer network	Configure
My Accounts		
CATEGORY	All	Configure
Analytics	Trello Manage Trello cards and tasks all in one place.	Configure
CRM		
Customer Support	GitHub Manage and collaborate on code projects.	Configure
Developer Tools	Jenkins Continuous Integration and Continuous Delivery	Configure
HR		
Marketing	Forms Easily create surveys, quizzes, and polls.	Configure
News & Social		
Project Management	Azure DevOps Collaborate on and manage software projects online.	Add
Others	RSS Get RSS feeds for your group.	Add
	Incoming Webhook	Add
All connectors		

- Put a name for the connector

Connectors for "devonfw shop floor" channel in "devonfw devs | Private" team X

 Jenkins Send feedback

The Jenkins connector sends notifications about build-related activities. To use this connector, you'll need to install Office 365 Connector plugin from Jenkins update center and configure for your project by following a few easy steps. If you don't already have Jenkins installed, you can download it at [Jenkins website](#).

Fields marked with * are mandatory

Name *
Enter a name for your Jenkins connection.

devon4ng

Create Cancel

- Copy the name and the Webhook URL, we will use it later.

Connectors for "devonfw shop floor" channel in "devonfw devs | Private" team X

 Jenkins Send feedback

The Jenkins connector sends notifications about build-related activities. To use this connector, you'll need to install Office 365 Connector plugin from Jenkins update center and configure for your project by following a few easy steps. If you don't already have Jenkins installed, you can download it at [Jenkins website](#).

Fields marked with * are mandatory

Name *
Enter a name for your Jenkins connection.

Webhook URL
Copy the following URL to save it to the Clipboard. You'll need this URL when you go to the Jenkins website.
 

Notifications will be sent about the following events in Jenkins:

- Whenever activity occurs in Jenkins.

Instructions Hide details ↗

Follow these steps to create your Jenkins connector.

Step 1
Log into Jenkins and in the Jenkins dashboard (Home screen), click **Manage Jenkins** from the left-hand side menu.

With the values that we get in the previous steps, we will execute the cicdgen command inside the project folder. If you have any doubt you can use the help.

```

cmd
C:\Users\Capgemini\Documents\Devonfw\Projects\devonfw>\devon4ng (master)
λ cicdgen --help
Usage: cicdgen generate <technology> [Options]

The usage of the command
Commands:
  cicdgen generate <technology> Generate CICD files for a devonfw technology stack
Options:
  --version Show version number [boolean]
  --help     Show help [boolean]

C:\Users\Capgemini\Documents\Devonfw\Projects\devonfw>\devon4ng (master)
λ cicdgen generate --help
Usage: cicdgen generate <technology> [Options]

Commands:
  cicdgen generate devon4ng      Devon4ng CICD schematic. It will provide all changes related to CICD
  cicdgen generate devon4node    Devon4node CICD schematic. It will provide all changes related to CICD
  cicdgen generate devon4j       Devon4j CICD schematic. It will provide all changes related to CICD
  cicdgen generate devon4net     Devon4net CICD schematic. It will provide all changes related to CICD

Options:
  --help Show help [boolean]

Examples:
  cicdgen generate devon4ng  Generate all files for devon4ng

```

```

cmd
C:\Users\Capgemini\Documents\Devonfw\Projects\devonfw>\devon4ng (master)
λ cicdgen generate devon4ng --help
Usage: cicdgen generate devon4ng [Options]

Commands:
  cicdgen generate devon4ng      Devon4ng CICD schematic. It will provide all changes related to CICD
Options:
  --help          Show help [boolean]
  --docker        Should generate code for docker deployment? [boolean]
  --dockerurl     The external docker daemon URL [string]
  --dockercertid   The Jenkins secret id where the docker certificate is stored [string]
  --registryurl   The docker registry URL (without protocol) [string]
  --openshift      Should generate code for openshift deployment? [boolean]
  --ocname        Openshift cluster name defined in Jenkins [string]
  --ocn          Openshift project. If empty it will take the default project defined in the cluster. [string]
  --groupid       The project groupId. It will be used for store the project in Nexus3 [string] [required]
  --teams         Do you want MS Teams notifications? [boolean]
  --teamsname     The name of the MS Teams webhook. Used only if 'teams' is true. [string] [default: "jenkins"]
  --teamsurl      The url of the MS Teams webhook. Used only if 'teams' is true. [string] [default: "jenkins"]
  --mergecharts   The merge strategy. Check the documentation for more information [string] [default: "error"]
  --commit        If true, all changes will be committed at the end of the process (if possible). [boolean] [default: true]

Examples:
  cicdgen generate devon4ng  Generate all files for devon4ng

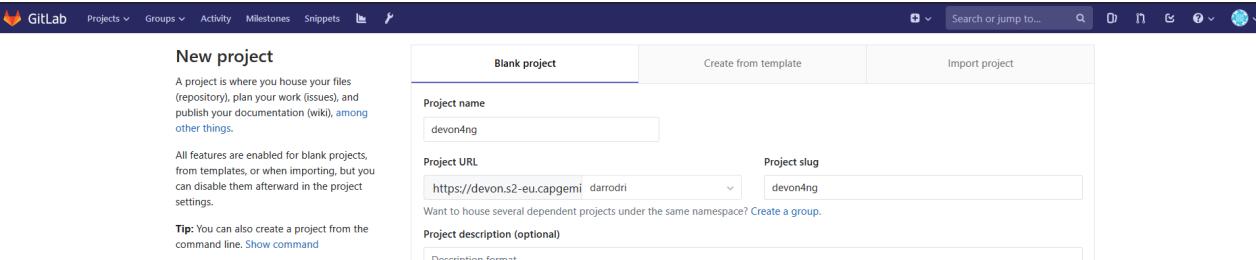
```

```
$ cicdgen generate devon4ng --groupid com.devonfw --docker --dockerurl
tpc://127.0.0.1:2376 --registryurl docker-registry-devon.s2-eu.capgemini.com
--teams --teamsname devon4ng --teamsurl https://outlook.office.com/webhook/...
```

```
C:\Users\plannedit\Documents\GitHub\devon4ng (master)
λ cicdgen generate devon4ng --groupid com.devonfw --docker --registryurl docker-registry-devon.s2-eu.capgemini.com --dockerurl tpc://127.0.0.1:2376
--teams --teamname devon4ng --teamsurl https://outlook.office.com/webhook/
CREATE /Jenkinsfile (12772 bytes)
CREATE /.dockernignore (25 bytes)
CREATE /Dockerfile (194 bytes)
CREATE /Dockerfile.ci (52 bytes)
UPDATE /karma.conf.js (1339 bytes)          merge-combine-
UPDATE /package.json (1366 bytes)           .es6code.png
UPDATE /angular.json (3663 bytes)
warning: CRLF will be replaced by LF in Jenkinsfile.
The file will have its original line endings in your working directory
[master 576edf1] Added CI/CD files to the project
 7 files changed, 372 insertions(+), 9 deletions(-)
create mode 100644 .dockernignore
create mode 100644 Dockerfile
create mode 100644 Dockerfile.ci
create mode 100644 Jenkinsfile

C:\Users\plannedit\Documents\GitHub\devon4ng (master)
λ
```

4. Create a git repository and upload the code



The screenshot shows the GitLab 'New project' creation interface. At the top, there are tabs for 'Blank project', 'Create from template', and 'Import project'. The 'Blank project' tab is selected. Below it, the 'Project name' field contains 'devon4ng'. The 'Project URL' field has the value 'https://devon.s2-eu.cagpemi.darrodri'. The 'Project slug' field also contains 'devon4ng'. A note below the URL says 'Want to house several dependent projects under the same namespace? [Create a group](#)'. The 'Project description (optional)' section has a placeholder 'Description format'. Under 'Visibility Level', the 'Private' option is selected, with a note that 'Project access must be granted explicitly to each user'. The 'Internal' and 'Public' options are also listed. At the bottom, there's a checkbox for 'Initialize repository with a README' which is unchecked, with a note that it allows immediate cloning. A large green 'Create project' button is at the bottom left, and a 'Cancel' button is at the bottom right.

New project

A project is where you house your files (repository), plan your work (issues), and publish your documentation (wiki), among other things.

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

Tip: You can also create a project from the command line. [Show command](#)

Blank project

Project name

devon4ng

Project URL

https://devon.s2-eu.cagpemi.darrodri

Project slug

devon4ng

Want to house several dependent projects under the same namespace? [Create a group](#).

Project description (optional)

Description format

Visibility Level ?

 Private
Project access must be granted explicitly to each user.

 Internal
The project can be accessed by any logged in user.

 Public
The project can be accessed without any authentication.

Initialize repository with a README
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

Create project

Cancel

Otherwise it is recommended you start with one of the options below.

[New file](#) [Add README](#) [Add CHANGELOG](#) [Add CONTRIBUTING](#) [Auto DevOps enabled](#)

Command line instructions

Git global setup

```
git config --global user.name "Dario Rodriguez Gonzalez"  
git config --global user.email "darrodrig@capgemini.com"
```

Create a new repository

```
git clone https://devon.s2-eu.capgemini.com/gitlab/darrodrig/devon4ng.git  
cd devon4ng  
touch README.md  
git add README.md  
git commit -m "add README"  
git push -u origin master
```

Existing folder

```
cd existing_folder  
git init  
git remote add origin https://devon.s2-eu.capgemini.com/gitlab/darrodrig/devon4ng.git  
git add .  
git commit -m "Initial commit"  
git push -u origin master
```

Existing Git repository

```
cd existing_repo  
git remote rename origin old-origin  
git remote add origin https://devon.s2-eu.capgemini.com/gitlab/darrodrig/devon4ng.git  
git push -u origin -all  
git push -u origin --tags
```

```
$ git remote add origin https://devon.s2-eu.capgemini.com/gitlab/darrodri/devon4ng.git
$ git push -u origin master
```

cmd

```
git clone https://gitlab-core:80/gitlab/darrodri/devon4ng.git
$ cd devon4ng
$ git remote add origin https://devon.s2-eu.capgemini.com/gitlab/darrodri/devon4ng.git
$ git push -u origin master
Enumerating objects: 47, done.
Counting objects: 100% (47/47), done.
Delta compression using up to 4 threads
Compressing objects: 100% (44/44), done.
Writing objects: 100% (47/47), 15.65 KiB | 640.00 KiB/s, done.
Total 47 (delta 5), reused 0 (delta 0)
To https://devon.s2-eu.capgemini.com/gitlab/darrodri/devon4ng.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

3. Create a git repository

As you can see, no git init or git commit is required, cicdgen do it for you.

5. Create a multibranch-pipeline in Jenkins

When you push the save button, it will download the repository and execute the pipeline defined in the Jenkinsfile. If you get any problem, check the environment variables defined in the Jenkinsfile. Here we show all variables related with Jenkins:

- chrome

Jenkins > Global Tool Configuration

Añadir un instalador ▾

Custom tool

Nombre: **Chrome-stable**

Custom Tool Configuration...

Instalar automáticamente

Ejecutar comando

Etiqueta:

Commando:

```
if l which google-chrome > /dev/null; then
    sudo su << EOF
    wget -q -O - https://dl.google.com/linux/linux_signing_key.pub | apt-key add -
    echo "deb http://dl.google.com/linux/chrome/deb/ stable main" > /etc/apt/sources.list.d/google-chrome.list
    apt-get update && apt-get -y install google-chrome-stable
EOF
fi
```

Directorio de la utilidad: /opt/chrome

Borrar Custom tool

Añadir un instalador ▾

Custom tool

Save **Apply**

- sonarTool

Jenkins > Global Tool Configuration

Sonar Scanner for MSBuild installations...

SonarQube Scanner

SonarQube Scanner installations **Add SonarQube Scanner**

SonarQube Scanner
Name: **SonarQube**

Install automatically

Install from Maven Central
Version: SonarQube Scanner 3.2.0.1227 ▾

Delete Installer

Add Installer ▾

Delete SonarQube Scanner

SonarQube Scanner
Name: **SonarQube-scanner**

Install automatically

Install from Maven Central
Version: SonarQube Scanner 3.2.0.1227 ▾

Delete Installer

Save **Apply**

- sonarEnv

SonarQube servers

Environment variables

Enable injection of SonarQube server configuration as build environment variables
If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

SonarQube installations

Name

Server URL
Default is http://localhost:9000

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

Add SonarQube

List of SonarQube installations

Selenium

You need to restart hub and than all configurations/nodes in order to apply settings for Selenium Hub.

Selenium Hub Port

Docker Slaves

Docker Provisioner

Save

- repositoryId

Jenkins

Manage Jenkins Config Files Add a new Config

Edit Configuration File

description

The configuration

ID	<input type="text" value="MavenSettings"/>
Name	<input type="text" value="MyGlobalSettings"/>
Comment	<input type="text" value="global settings"/>
<input checked="" type="checkbox"/> Replace All	
Server Credentials	<p>ServerId <input type="text" value="pl-nexus"/></p> <p>Credentials <input type="text" value="admin-devon/***** (admin for nexus)"/></p> <p><input type="button" value="Add"/> <input type="button" value="Delete"/></p>
Content	<pre> 1 <?xml version="1.0" encoding="UTF-8"?> 2 3 <!-- 4 Licensed to the Apache Software Foundation (ASF) under one 5 or more contributor license agreements. See the NOTICE file 6 distributed with this work for additional information 7 regarding copyright ownership. The ASF licenses this file </pre>

- globalSettingsId

The configuration

ID	MavenSettings
Name	MyGlobalSettings
Comment	global settings
<input checked="" type="checkbox"/> Replace All	
Server Credentials	ServerId: pl-nexus Credentials: admin-devon***** (admin for nexus)

Add Delete

Content

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <!--
4 Licensed to the Apache Software Foundation (ASF) under one
5 or more contributor license agreements. See the NOTICE file
6 distributed with this work for additional information
7 regarding copyright ownership. The ASF licenses this file

```

- mavenInstallation

Ant

Ant installations...

Maven

Maven installations

Add Maven
Maven
Name: Maven3
<input checked="" type="checkbox"/> Install automatically
Install from Apache
Version: 3.6.0

Delete Installer

Add Installer

Maven

Maven
Name: Maven2
<input checked="" type="checkbox"/> Install automatically
Install from Apache

Delete Maven

Save Apply

- dockerTool

The screenshot shows the Jenkins Global Tool Configuration page. A new custom tool named "docker-global" is being added. The "Run Shell Command" section contains a command to install Docker on a Debian-based system. The "Tool Home" field is set to "/usr/bin". Buttons for "Save", "Apply", "Delete Installer", and "Delete Custom tool" are visible at the bottom.

```

Custom tool
Name docker-global
Custom Tool Configuration...
Install automatically

Run Shell Command
Label
Command
if which docker > /dev/null; then
    sudo apt-get update
    sudo apt-get install -y apt-transport-https ca-certificates curl gnupg2 software-properties-common
    curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
    sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian ${lsb_release -cs} stable"
    sudo apt-get update
    sudo apt-get install -y docker-ce-cli
    docker -v
Tool Home /usr/bin
Delete Installer

Add Installer
Delete Custom tool

Save Apply

```

6. Add a webhook in GitLab

In order to run the pipeline every time that you push code to GitLab, you need to configure a webhook in your repository.

The screenshot shows the "Integrations Settings" page for the "devon4ng" project in GitLab. The "URL" field is set to "http://jenkins-core:8080/jenkins/project/devon4ng". The "Trigger" section has "Push events" checked, with a note that this URL will be triggered by a push to the repository. Other trigger options like "Tag push events", "Comments", etc., are listed but not selected.

Now your project is ready to work following a CICD strategy.

The last thing to take into account is the branch naming. We prepare the pipeline in order to work following the git-flow strategy. So all stages of the pipeline will be executed for the branches: develop, release/*, master. For the branches: feature/*, hotfix/*, bugfix/* only the steps related to unit testing will be executed.

61. cicdgen Schematics

61.1. CICDGEN SCHEMATICS

We use angular schematics to create and update an existing devonfw project in order to adapt it to a CICD environment. All schematics are prepared to work with Production Line, a Capgemini CICD platform, but it can also work in other environment which have the following tools:

- Jenkins
- GitLab
- Nexus 3
- SonarQube

The list of available schematics are:

- [devon4j](#)
- [devon4ng](#)
- [devon4net](#)
- [devon4node](#)

How to run the schematics

You can run the schematics using the schematics CLI provided by the angular team, but the easiest way to run it is using the [cicdgen CLI](#) which is a wrapper for the schematics CLI in order to use it in a easy way.

To generate files you only need to run the command

```
$ cicdgen generate <schematic-name> [arguments]
```

<schematic-name> is the name of the schematic that you want to execute.

You can find all information about arguments in the schematic section.

Merge Strategies

When you execute cicdgen in a project, is possible that you already have some files that cicdgen will generate. Until version 1.5 the behaviour in these cases was to throw an error and not create/modify any file. Since version 1.6 you can choose what to do in case of conflict. In this page we will explain who to choose one merge strategy and how it works.

Choose a merge strategy

To choose a merge strategy, you must pass to cicdgen the merge parameter followed by the name of the strategy. The strategies available are: **error**, **keep**, **override**, **combine**.

Example:

```
$ cicdgen generate devon4j --merge keep
```

Merge strategies

- **error**: The **error** strategy is the same as until version 1.5, throwing an error and do not create/modify any file. This is the default value, if you do not pass the merge parameter this value will be taken.
- **keep**: The **keep** strategy will keep the actual content of your files in case of conflict. If there is no conflict, the file will be created with the new content.
- **override**: The **override** strategy will override your current files, without throwing any error, and create a new ones with the new content. If there is no conflict, the file will be created with the new content.
- **combine**: The **combine** strategy will create a new file combining the current content with the new content. In order to combine both files, it will apply a diff algorithm and it will show the conflicts in the same way that git does. If there is no conflict, the file will be created with the new content.

By resolving the conflicts in the same way as git, you can use the same tools in order to solve them. For example, you can use VSCode:

The screenshot shows a code editor with a merge conflict. At the top, there are four options: 'Accept Current Change', 'Accept Incoming Change', 'Accept Both Changes', and 'Compare Changes'. Below these options, the code is displayed in three sections: 'HEAD (Current Change)' (green background), 'Line 1' (dark blue background), '=====' (dark blue background), 'Line 5' (dark blue background), and 'new_content (Incoming Change)' (blue background). The 'new_content' section contains the text 'Line 5'.

Examples:

keep Current file:

```
Line 1
Line 2
Line 3
Line 4
```

New file:

```
Line 5
Line 2
Line 3
Line 4
```

The result will be:

```
Line 1  
Line 2  
Line 3  
Line 4
```

override Current file:

```
Line 1  
Line 2  
Line 3  
Line 4
```

New file:

```
Line 5  
Line 2  
Line 3  
Line 4
```

The result will be:

```
Line 5  
Line 2  
Line 3  
Line 4
```

combine Current file:

```
Line 1  
Line 2  
Line 3  
Line 4
```

New file:

```
Line 5  
Line 2  
Line 3  
Line 4
```

The result will be:

```
<<<<< HEAD
Line 1
=====
Line 5
>>>>> new_content
Line 2
Line 3
Line 4
```

61.2. devon4j schematic

With the `cicdgen generate devon4j` command you can generate some files required for CICD. In this section we will explain the arguments of this command and also the files that will be generated.

devon4j schematic arguments

When you execute the `cicdgen generate devon4j` command you can also add some arguments in order to modify the behaviour of the command. Those arguments are:

- `--docker`

The type of this parameter if boolean. If it is present, docker related files and pipeline stage will be also generated. For more details see docker section of Jenkinsfile and [files generated for docker](#)

- `--dockerurl`

The URL of your external docker daemon. Example: `tcp://127.0.0.1:2376`

- `--dockercertid`

The Jenkins credential id for your docker daemon certificate. It is only required when your docker daemon is secure.

- `--registryurl`

Your docker registry URL. It is required when `--docker` is true, and it will be used to know where the docker image will be uploaded.

- `--openshift`

The type of this parameter if boolean. If it is present, OpenShift related files and pipeline stage will be also generated. For more details see OpenShift section of Jenkinsfile and [files generated for docker](#) (same as `--docker`)

- `--ocname`

The name used for register your Openshift cluster in Jenkins.

- `--ocn`

Openshift cluster namespace

- `--teams`

With this argument we can add the teams notification option in the [Jenkinsfile](#).

- `--teamsname`

The name of the Microsoft Teams webhook. It is defined at Microsoft Teams connectors.

- `--teamsurl`

The url of the Microsoft Teams webhook. It is returned by Microsoft Teams when you create a connector.

- `--merge`

If you have used cicdgen previously, you can choose what you want to do in case of file conflict. The default behavior is to throw an error and not modify any file. You can see the other strategies on their [specific page](#).

- `--commit`

If true, all changes will be committed at the end of the process (if possible). In order to send a false value, you need to write `--commit=false`

Devon4ng generated files

When you execute the generate devon4ng command, some files will be added/updated in your project.

Files

- `.gitignore`

Defines all files that git will ignore. e.g: compiled files, IDE configurations. It will download the content from: <https://gitignore.io/api/java,maven,eclipse,intellij,intellij+all,intellij+iml,visualstudiocode>

- `pom.xml`

The pom.xml is modified in order to add, if needed, the distributionManagement.

- `Jenkinsfile`

The Jenkinsfile is the file which define the Jenkins pipeline of our project. With this we can execute the test, build the application and deploy it automatically following a CICD methodology. This file is prepared to work with the Production Line default values, but it is also fully configurable to your needs.

- Prerequisites

- A Production Line instance. It can work also if you have a Jenkins, SonarQube and Nexus3, but in this case maybe you need to configure them properly.
- Java 11 installed in Jenkins as a global tool.
- SonarQube installed in Jenkins as a global tool.
- Maven3 installed in Jenkins as a global tool.
- A maven global settings properly configured in Jenkins.
- If you will use docker to deploy:
 - Docker installed in Jenkins as a global custom tool.
 - The Nexus3 with a docker repository.
 - A machine with docker installed where the build and deploy will happen.
 - A docker network called application.
- If you will use OpenShift to deploy:
 - An OpenShift instance
 - The OpenShift projects created
- The Jenkins syntax

In this section we will explain a little bit the syntax of the Jenkins, so if you need to change something you will be able to do it properly.

- agent: Here you can specify the Jenkins agent where the pipeline will be executed. The default value is any.
- options: Here you can set global options to the pipeline. By default, we add a build discarded to delete old artifacts/builds of the pipeline and also we disable the concurrent builds.

If the teams option is passed to cicdgen, we add a new option in order to send notifications to Microsoft Teams with the status of the pipeline executions.

- environment: Here all environment variables are defined. All values defined here matches with the Production Line defaults. If you Jenkins has other values, you need to update it manually.
- stages: Here are defined all stages that our pipeline will execute. Those stages are:
 - Loading Custom Tools: Load some custom tools that can not be loaded in the tools section. Also set some variables depending on the git branch which you are executing. Also, we set properly the version number in all pom files. It means that if your branch is develop, your version should end with the word **-SNAPSHOT**, in order case, if **-SNAPSHOT** is present it will be removed.
 - Fresh Dependency Installation: install all packages need to build/run your java project.
 - Unit Tests: execute the `mvn test` command.
 - SonarQube code analysis: send the project to SonarQube in order to get the static

code analysis of your project.

- Deliver application into Nexus: build the project and send all bundle files to Nexus3.
- If `--docker` is present:
 - Create the Docker image: build a new docker image that contains the new version of the project.
 - Deploy the new image: deploy a new version of the application using the image created in the previous stage. The previous version is removed.
- If `--openshift` is present:
 - Create the Docker image: build a new docker image that contains the new version of the project using a OpenShift build config.
 - Deploy the new image: deploy a new version of the application in OpenShift.
 - Check pod status: checks that the application deployed in the previous stage is running properly. If the application does not run the pipeline will fail.
- post: actions that will be executed after the stages. We use it to clean up all files.

devon4j Docker generated files

When you generate the files for a devon4ng you can also pass the option `--docker`. It will generate also some extra files related to docker.



If you pass the `--docker` option the option `--registryurl` is also required. It will be used to upload the images to a docker registry. Example: if your registry url is `docker-registry-test.s2-eu.capgemini.com` you should execute the command in this way: `cicdgen generate devon4node --groupid com.devonfw --docker --registryurl docker-registry-test.s2-eu.capgemini.com`.

Files

- Dockerfile

This file contains the instructions to build a docker image for your project. This Dockerfile is for local development purposes, you can use it in your machine executing:

```
$ cd <path-to-your-project>
$ docker build -t <project-name>/<tag> .
```

This build is using a multi-stage build. First, it uses a maven image in order to compile the source code, then it will use a java image to run the application. With the multi-stage build we keep the final image as clean as possible.

- Dockerfile.ci

This file contains the instructions to create a docker image for your project. The main difference with the Dockerfile is that this file will be only used in the Jenkins pipeline. Instead of compiling again the code, it takes the compiled war from Jenkins to the image.

61.3. devon4ng schematic

With the `cicdgen generate devon4ng` command you can generate some files required for CICD. In this section we will explain the arguments of this command and also the files that will be generated.

devon4ng schematic arguments

When you execute the `cicdgen generate devon4ng` command you can also add some arguments in order to modify the behaviour of the command. Those arguments are:

- `--docker`

The type of this parameter is boolean. If it is present, docker related files and pipeline stage will be also generated. For more details see docker section of Jenkinsfile and [files generated for docker](#)

- `--dockerurl`

The URL of your external docker daemon. Example: `tcp://127.0.0.1:2376`

- `--dockercertid`

The Jenkins credential id for your docker daemon certificate. It is only required when your docker daemon is secure.

- `--registryurl`

Your docker registry URL. It is required when `--docker` is true, and it will be used to know where the docker image will be uploaded.

- `--openshift`

The type of this parameter is boolean. If it is present, OpenShift related files and pipeline stage will be also generated. For more details see OpenShift section of Jenkinsfile and [files generated for OpenShift](#) (same as `--docker`)

- `--ocname`

The name used for register your Openshift cluster in Jenkins.

- `--ocn`

Openshift cluster namespace

- `--groupid`

The project groupId. This argument is required. It will be used for store the project in a maven repository at Nexus 3. Why maven? Because is the kind of repository where we can upload/download a zip file easily. Npm repository needs a package.json file but, as we compile the angular application to static javascript and html files, the package.json is no needed anymore.

- --teams

With this argument we can add the teams notification option in the [Jenkinsfile](#).

- --teamsname

The name of the Microsoft Teams webhook. It is defined at Microsoft Teams connectors.

- --teamsurl

The url of the Microsoft Teams webhook. It is returned by Microsoft Teams when you create a connector.

- --merge

If you have used cicdgen previously, you can choose what you want to do in case of file conflict. The default behavior is to throw an error and not modify any file. You can see the other strategies on their [specific page](#).

- --commit

If true, all changes will be committed at the end of the process (if possible). In order to send a false value, you need to write `--commit=false`

devon4ng generated files

When you execute the generate devon4ng command, some files will be added/updated in your project.

Files

- angular.json

The angular.json is modified in order to change the compiled files destination folder. Now, when you make a build of your project, the compiled files will be generated into dist folder instead of dist/<project-name> folder.

- package.json

The package.json is modified in order to add a script for test the application using Chrome Headless instead of a regular chrome. This script is called `test:ci`.

- karma.conf.js

The karma.conf.js is also modified in order to add the Chrome Headless as a browser to execute test. The coverage output folder is change to `./coverage` instead of `./coverage/<project-name>`

- Jenkinsfile

The Jenkinsfile is the file which define the Jenkins pipeline of our project. With this we can execute the test, build the application and deploy it automatically following a CICD methodology. This file is prepared to work with the Production Line default values, but it is also

fully configurable to your needs.

- Prerequisites
 - A Production Line instance. It can work also if you have a Jenkins, SonarQube and Nexus3, but in this case maybe you need to configure them properly.
 - NodeJS installed in Jenkins as a global tool.
 - Google Chrome installed in Jenkins as a global custom tool.
 - SonarQube installed in Jenkins as a global tool.
 - Maven3 installed in Jenkins as a global tool.
 - A maven global settings properly configured in Jenkins.
 - If you will use docker :
 - Docker installed in Jenkins as a global custom tool.
 - The Nexus3 with a docker repository.
 - A machine with docker installed where the build and deploy will happen.
 - A docker network called application.
 - If you will use OpenShift :
 - An OpenShift instance
 - The OpenShift projects created
- The Jenkins syntax

In this section we will explain a little bit the syntax of the Jenkins, so if you need to change something you will be able to do it properly.

- agent: Here you can specify the Jenkins agent where the pipeline will be executed. The default value is any.
- options: Here you can set global options for the pipeline. By default, we add a build discarded to delete old artifacts/builts of the pipeline and also we disable the concurrent builds.

If the teams option is passed to cicdgen, we add a new option in order to send notifications to Microsoft Teams with the status of the pipeline executions.

- tools: Here we define the global tools configurations. By default a version of nodejs is added here.
- environment: Here all environment variables are defined. All values defined here matches with the Production Line defaults. If you Jenkins has other values, you need to update it manually.
- stages: Here are defined all stages that our pipeline will execute. Those stages are:
 - Loading Custom Tools: in this stage some custom tools are loaded. Also we set some variables depending on the git branch which you are executing.
 - Fresh Dependency Installation: install all packages need to build/run your angular

project.

- Code Linting: execute the linter analysis.
- Execute Angular tests: execute the angular test in a Chrome Headless.
- SonarQube code analysis: send the project to SonarQube in order to get the static code analysis of your project.
- Build Application: compile the application to be ready to deploy in a web server.
- Deliver application into Nexus: store all compiled files in Nexus3 as a zip file.
- If `--docker` is present:
 - Create the Docker image: build a new docker image that contains the new version of the project.
 - Deploy the new image: deploy a new version of the application using the image created in the previous stage. The previous version is removed.
- If `--openshift` is present:
 - Create the Docker image: build a new docker image that contains the new version of the project using a OpenShift build config.
 - Deploy the new image: deploy a new version of the application in OpenShift.
 - Check pod status: checks that the application deployed in the previous stage is running properly. If the application does not run the pipeline will fail.
- post: actions that will be executed after the stages. We use it to clean up all files.

devon4ng Docker generated files

When you generate the files for a devon4ng you can also pass the option `--docker`. It will generate also some extra files related to docker.



If you pass the `--docker` option the option `--registryurl` is also required. It will be used to upload the images to a docker registry. Example: if your registry url is `docker-registry-test.s2-eu.capgemini.com` you should execute the command in this way: `cicdgen generate devon4node --groupid com.devonfw --docker --registryurl docker-registry-test.s2-eu.capgemini.com`.

Files

- `.dockerignore`

In this files are defined the folders that will not be copied to the docker image. Fore more information read the [official documentation](#).

- `Dockerfile`

This file contains the instructions to build a docker image for you project. This Dockerfile is for local development purposes, you can use it in your machine executing:

```
$ cd <path-to-your-project>
$ docker build -t <project-name>/<tag> .
```

This build is using a multi-stage build. First, it uses a node image in order to compile the source code, then it will use a nginx image as a web server for our devon4ng application. With the multi-stage build we avoid everything related to node.js in our final image, where we only have a nginx with our application compiled.

- Dockerfile.ci

This file contains the instructions to create a docker image for your project. The main difference with the Dockerfile is that this file will be only used in the Jenkins pipeline. Instead of compiling again the code, it takes all compiled files and the nginx.conf from Jenkins to the image.

61.4. devon4net schematic

With the `cicdgen generate devon4net` command you can generate some files required for CICD. In this section we will explain the arguments of this command and also the files that will be generated.

devon4net schematic arguments

When you execute the `cicdgen generate devon4net` command you can also add some arguments in order to modify the behaviour of the command. Those arguments are:

- `--appname`

The name of your devon4net application.

- `--appversion`

The initial version of your devon4net application

- `--docker`

The type of this parameter is boolean. If it is present, docker related files and pipeline stage will be also generated. For more details see docker section of Jenkinsfile and [files generated for docker](#)

- `--dockerurl`

The URL of your external docker daemon. Example: `tcp://127.0.0.1:2376`

- `--dockercertid`

The Jenkins credential id for your docker daemon certificate. It is only required when your docker daemon is secure.

- `--registryurl`

Your docker registry URL. It is required when `--docker` is true, and it will be used to know where the docker image will be uploaded.

- `--openshift`

The type of this parameter is boolean. If it is present, OpenShift related files and pipeline stage will be also generated. For more details see OpenShift section of Jenkinsfile and [files generated for OpenShift](#) (same as `--docker`)

- `--ocname`

The name used for register your Openshift cluster in Jenkins.

- `--ocn`

Openshift cluster namespace

- `--groupid`

The project groupId. This argument is required. It will be used for store the project in a maven repository at Nexus 3. Why maven? Because is the kind of repository where we can upload/download a zip file easily. Npm repository needs a package.json file but, as we compile the angular application to static javascript and html files, the package.json is no needed anymore.

- `--teams`

With this argument we can add the teams notification option in the [Jenkinsfile](#).

- `--teamsname`

The name of the Microsoft Teams webhook. It is defined at Microsoft Teams connectors.

- `--teamsurl`

The url of the Microsoft Teams webhook. It is returned by Microsoft Teams when you create a connector.

- `--merge`

If you have used cicdgen previously, you can choose what you want to do in case of file conflict. The default behavior is to throw an error and not modify any file. You can see the other strategies on their [specific page](#).

- `--commit`

If true, all changes will be committed at the end of the process (if possible). In order to send a false value, you need to write `--commit=false`

devon4net generated files

When you execute the generate devon4net command, some files will be added/updated in your

project.

Files

- Jenkinsfile

The Jenkinsfile is the file which define the Jenkins pipeline of our project. With this we can execute the test, build the application and deploy it automatically following a CICD methodology. This file is prepared to work with the Production Line default values, but it is also fully configurable to your needs.

- Prerequisites

- A Production Line instance. It can works also if you have a Jenkins, SonarQube and Nexus3, but in this case maybe you need to configure them properly.
- dotnet core installed in Jenkins as a global tool.
- SonarQube installed in Jenkins as a global tool.
- Maven3 installed in Jenkins as a global tool.
- A maven global settings properly configured in Jenkins.
- If you will use docker :
 - Docker installed in Jenkins as a global custom tool.
 - The Nexus3 with a docker repository.
 - A machine with docker installed where the build and deploy will happen.
- If you will use OpenShift :
 - An OpenShift instance
 - The OpenShift projects created

- The Jenkins syntax

In this section we will explain a little bit the syntax of the Jenkins, so if you need to change something you will be able to do it properly.

- agent: Here you can specify the Jenkins agent where the pipeline will be executed. The default value is any.
- options: Here you can set global options for the pipeline. By default, we add a build discarded to delete old artifacts/buils of the pipeline and also we disable the concurrent builds.

If the teams option is passed to cicdgen, we add a new option in order to send notifications to Microsoft Teams with the status of the pipeline executions.

- tools: Here we define the global tools configurations. By default a version of nodejs is added here.
- environment: Here all environment variables are defined. All values defined here matches with the Production Line defaults. If you Jenkins has other values, you need to update it manually.

- stages: Here are defined all stages that our pipeline will execute. Those stages are:
 - Loading Custom Tools: in this stage some custom tools are loaded. Also we set some variables depending on the git branch which you are executing.
 - Fresh Dependency Installation: install all dependencies need to build/run your dotnet project.
 - Execute dotnet tests: execute the tests.
 - SonarQube code analysis: send the project to SonarQube in order to get the static code analysis of your project.
 - Build Application: compile the application to be ready to deploy in a web server.
 - Deliver application into Nexus: store all compiled files in Nexus3 as a zip file.
 - If `--docker` is present:
 - Create the Docker image: build a new docker image that contains the new version of the project.
 - Deploy the new image: deploy a new version of the application using the image created in the previous stage. The previous version is removed.
 - If `--openshift` is present:
 - Create the Docker image: build a new docker image that contains the new version of the project using a OpenShift build config.
 - Deploy the new image: deploy a new version of the application in OpenShift.
 - Check pod status: checks that the application deployed in the previous stage is running properly. If the application does not run the pipeline will fail.
- post: actions that will be executed after the stages. We use it to clean up all files.

devon4net Docker generated files

When you generate the files for devon4net you can also pass the option `--docker`. It will generate also some extra files related to docker.



If you pass the `--docker` option the option `--registryurl` is also required. It will be used to upload the images to a docker registry. Example: if your registry url is `docker-registry-test.s2-eu.capgemini.com` you should execute the command in this way: `cicdgen generate devon4net --groupid com.devonfw --docker --registryurl docker-registry-test.s2-eu.capgemini.com`.

Files

- `.dockerignore`

In this files are defined the folders that will not be copied to the docker image. Fore more information read the [official documentation](#).

- `Dockerfile`

This file contains the instructions to build a docker image for your project. This Dockerfile is for

local development purposes, you can use it in your machine executing:

```
$ cd <path-to-your-project>
$ docker build -t <project-name>/<tag> .
```

- Dockerfile.ci

This file contains the instructions to create a docker image for your project. The main difference with the Dockerfile is that this file will be only used in the Jenkins pipeline. Instead of compiling again the code, it takes all compiled files from Jenkins to the image.

61.5. devon4node schematic

With the `cicdgen generate devon4node` command you can generate some files required for CICD. In this section we will explain the arguments of this command and also the files that will be generated.

devon4node schematic arguments

When you execute the `cicdgen generate devon4node` command you can also add some arguments in order to modify the behaviour of the command. Those arguments are:

- `--docker`

The type of this parameter is boolean. If it is present, docker related files and pipeline stage will be also generated. For more details see docker section of Jenkinsfile and [files generated for docker](#)

- `--dockerurl`

The URL of your external docker daemon. Example: `tcp://127.0.0.1:2376`

- `--dockercertid`

The Jenkins credential id for your docker daemon certificate. It is only required when your docker daemon is secure.

- `--registryurl`

Your docker registry URL. It is required when `--docker` is true, and it will be used to know where the docker image will be uploaded.

- `--openshift`

The type of this parameter is boolean. If it is present, OpenShift related files and pipeline stage will be also generated. For more details see OpenShift section of Jenkinsfile and [files generated for OpenShift](#) (same as `--docker`)

- `--ocname`

The name used for register your Openshift cluster in Jenkins.

- `--ocn`

Openshift cluster namespace

- `--groupid`

The project groupId. This argument is required. It will be used for store the project in a maven repository at Nexus 3. Why maven? Because is the kind of repository where we can upload/download a zip file easily. Npm repository needs a package.json file but, as we compile the angular application to static javascript and html files, the package.json is no needed anymore.

- `--teams`

With this argument we can add the teams notification option in the [Jenkinsfile](#).

- `--teamsname`

The name of the Microsoft Teams webhook. It is defined at Microsoft Teams connectors.

- `--teamsurl`

The url of the Microsoft Teams webhook. It is returned by Microsoft Teams when you create a connector.

- `--merge`

If you have used cicdgen previously, you can choose what you want to do in case of file conflict. The default behavior is to throw an error and not modify any file. You can see the other strategies on their [specific page](#).

- `--commit`

If true, all changes will be committed at the end of the process (if possible). In order to send a false value, you need to write `--commit=false`

devon4node generated files

When you execute the generate devon4node command, some files will be added/updated in your project.

Files

- `package.json`

The package.json is modified in order to add a script for run the linter and generate the json report. This script is called `lint:ci`.

- `Jenkinsfile`

The Jenkinsfile is the file which define the Jenkins pipeline of our project. With this we can execute the test, build the application and deploy it automatically following a CICD methodology. This file is prepared to work with the Production Line default values, but it is also fully configurable to your needs.

- Prerequisites
 - A Production Line instance. It can works also if you have a Jenkins, SonarQube and Nexus3, but in this case maybe you need to configure them properly.
 - NodeJS installed in Jenkins as a global tool.
 - SonarQube installed in Jenkins as a global tool.
 - Maven3 installed in Jenkins as a global tool.
 - A maven global settings properly configured in Jenkins.
 - If you will use docker :
 - Docker installed in Jenkins as a global custom tool.
 - The Nexus3 with a docker repository.
 - A machine with docker installed where the build and deploy will happen.
 - If you will use OpenShift :
 - An OpenShift instance
 - The OpenShift projects created
- The Jenkins syntax

In this section we will explain a little bit the syntax of the Jenkins, so if you need to change something you will be able to do it properly.

- agent: Here you can specify the Jenkins agente where the pipeline will be executed. The default value is any.
- options: Here you can set global options for the pipeline. By default, we add a build discarded to delete old artifacts/buils of the pipeline and also we disable the concurrent builds.

If the teams option is passed to cicdgen, we add a new option in order to send notifications to Microsoft Teams with the status of the pipeline executions.

- tools: Here we define the global tools configurations. By default a version of nodejs is added here.
- environment: Here all environment variables are defined. All values defined here matches with the Production Line defaults. If you Jenkins has other values, you need to update it manually.
- stages: Here are defined all stages that our pipeline will execute. Those stages are:
 - Loading Custom Tools: in this stage some custom tools are loaded. Also we set some variables depending on the git branch which you are executing.
 - Fresh Dependency Installation: install all packages need to build/run your node

project.

- Code Linting: execute the linter analysis.
- Execute tests: execute the tests.
- SonarQube code analysis: send the project to SonarQube in order to get the static code analysis of your project.
- Build Application: compile the application to be ready to deploy in a web server.
- Deliver application into Nexus: store all compiled files in Nexus3 as a zip file.
- If `--docker` is present:
 - Create the Docker image: build a new docker image that contains the new version of the project.
 - Deploy the new image: deploy a new version of the application using the image created in the previous stage. The previous version is removed.
- If `--openshift` is present:
 - Create the Docker image: build a new docker image that contains the new version of the project using a OpenShift build config.
 - Deploy the new image: deploy a new version of the application in OpenShift.
 - Check pod status: checks that the application deployed in the previous stage is running properly. If the application does not run the pipeline will fail.
- post: actions that will be executed after the stages. We use it to clean up all files.

devon4node Docker generated files

When you generate the files for a devon4node you can also pass the option `--docker`. It will generate also some extra files related to docker.



If you pass the `--docker` option the option `--registryurl` is also required. It will be used to upload the images to a docker registry. Example: if your registry url is `docker-registry-test.s2-eu.capgemini.com` you should execute the command in this way: `cicdgen generate devon4node --groupid com.devonfw --docker --registryurl docker-registry-test.s2-eu.capgemini.com`.

Files

- `.dockerignore`

In this files are defined the folders that will not be copied to the docker image. Fore more information read the [official documentation](#).

- `Dockerfile`

This file contains the instructions to build a docker image for you project. This Dockerfile is for local development purposes, you can use it in your machine executing:

```
$ cd <path-to-your-project>
$ docker build -t <project-name>/<tag> .
```

This build is installs all dependencies in ordre to build the project and then remove all devDependencies in order to keep only the production dependencies.

- `.dockerignore.ci`

Another `.dockerignore`. The purpose of this one is to define de file exclusions in your CI pipeline.

- `Dockerfile.ci`

This file contains the instructions to create a docker image for you project. The main difference with the Dockerfile is that this file will be only used in the Jenkins pipeline. Instead of compiling again the code, it takes all compiled files from Jenkins to the image.

Part X: Production Line Templates

This repository contains a collection of templates that can be used inside a Production Line Jenkins to setup/configure and execute certain tasks.

62. How to add a Template to your PL instance

- Go to Jenkins.
- On the upper left click on "New Element" to create a new Jenkins job.
- Choose a name for the job such as "MTS-template-seed-job". The job type has to be "Pipeline". Click on ok.

The screenshot shows the Jenkins interface for creating a new job. The title bar says "Jenkins" with a count of "2" notifications. The main form is titled "Geben Sie einen Element-Namen an" (Provide an element name) and contains a text input field with the value "MTS-template-seed-job". Below the input field is a note: "» Pflichtfeld" (Required field). There are three options listed: "Free Style"-Softwareprojekt bauen (Free Style Software Project), Maven-Projekt (Maven Project), and Pipeline. The "Pipeline" option is highlighted with a blue border, indicating it is selected. The Pipeline description states: "Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type."

- Scroll down to the bottom of the job creation page where you will find the "Pipeline" section.
 - Switch to "Pipeline script from SCM".
 - Set "SCM" to "Git".
 - Set "Repository URL" to: <https://github.com/devonfw/production-line.git>
 - Credentials can be left empty, because the repository is public.
 - Set "Script Path" to the template that you want to use e.g. "devon4j-mts/Jenkinsfile".

General Build Triggers Advanced Project Options **Pipeline**

Pipeline

Definition Pipeline script from SCM

SCM Git

Repositories

Repository URL <https://github.com/devonfw-forge/devon-production-line.git>

Credentials - leer -

Branches to build

Branch Specifier (blank for 'any') */master

Add Branch

Repository Browser (Auto)

Additional Behaviours

Script Path devon4j-mts/Jenkinsfile

Lightweight checkout

63. devonfw Technologies Templates

63.1. How to add a Template to your PL instance

- Go to Jenkins.
- On the upper left click on "New Element" to create a new Jenkins job.
- Chose a name for the job such as "MTS-template-seed-job". The job type has to be "Pipeline". Click on ok.

Geben Sie einen Element-Namen an
MTS-template-seed-job
» Pflichtfeld

"Free Style"-Softwareprojekt bauen
Dieses Profil ist das meistgenutzte in Jenkins. Jenkins baut Ihr Projekt, wobei Sie universell jedes SCM System mit jedem Build-Verfahren kombinieren können. Dieses Profil ist nicht nur auf das Bauen von Software beschränkt, sondern kann darüber hinaus auch für weitere Anwendungsgebiete verwendet werden.

Maven-Projekt
Dieses Profil baut ein Maven-Projekt. Jenkins wertet dabei Ihre POM Dateien aus und reduziert damit den Konfigurationsaufwand ganz erheblich.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

- Scroll down to the bottom of the job creation page where you will find the "Pipeline" section.
 - Switch to "Pipeline script from SCM".
 - Set "SCM" to "Git".
 - Set "Repository URL" to: <https://github.com/devonfw/production-line.git>
 - Credentials can be left empty, because the repository is public.
 - Set "Script Path" to the template that you want to use e.g. "devon4j-mts/Jenkinsfile".

General Build Triggers Advanced Project Options Pipeline

Pipeline

Definition Pipeline script from SCM

SCM Git

Repositories

Repository URL: <https://github.com/devonfw-forge/devon-production-line.git>

Credentials: - leer - [Add](#)

Branches to build

Branch Specifier (blank for 'any'): */master

Repository Browser: (Auto)

Additional Behaviours: Hinzufügen

Script Path: devon4j-mts/Jenkinsfile

Lightweight checkout:



63.2. devon4j Template for Production Line

63.2.1. Overview

This template will configure your PL instance to have a 'ready to use' devon4j template. It can be used as a starting point for your Java projects.

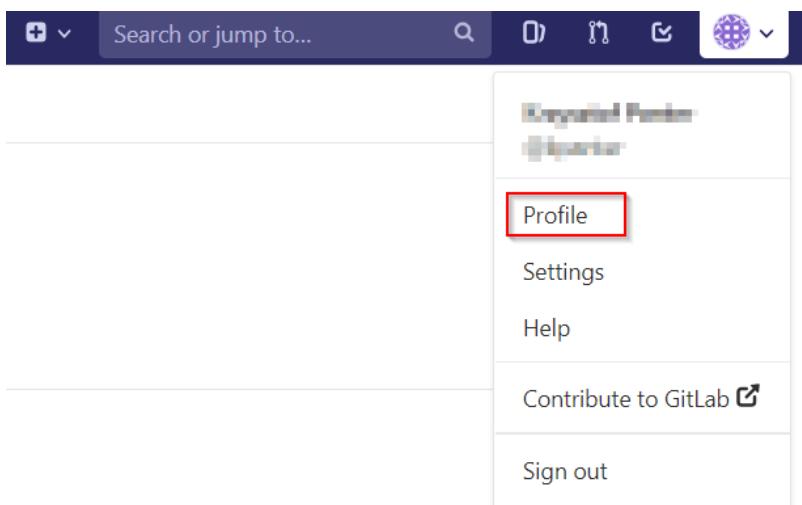
This includes CICD files for a devonfw technology stack with configuration for:

- docker or openshift deployment
- pushing artifacts to nexus3

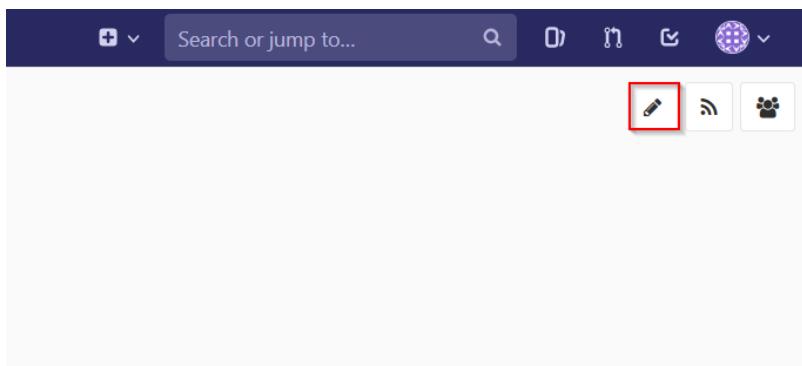
63.2.2. Prerequisites

To be able to run Jenkins devon4j job under ProductionLine you need to configure below settings in Jenkins and Gitlab

- Jenkins
 - Execute the [initialize instance template](#)
 - If you plan to deploy into OpenShift, you need to execute [openshift-configuration template](#) also.
- Gitlab
 - Generate User Private Token
Go to your Profile in Gitlab



Next click on the pen icon



On the left menu choose Access Tokens and put token name and check fields like below

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

test_token

Expires at

YYYY-MM-DD

**Scopes** **api**

Grants complete read/write access to the API, including all groups and projects.

 read_user

Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

 sudo

Grants permission to perform API actions as any user in the system, when authenticated as an admin user.

 read_repository

Grants read-only access to repositories on private projects using Git-over-HTTP (not using the API).

Create personal access token

Click "Create personal access token", you should receive notification about created token and token string. Copy the token string.

Your new personal access token has been created.

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Your New Personal Access Token

eHfQWvqJLjLwXk

Make sure you save it - you won't be able to access it again.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

The GitLab API user needs to have API access and the rights to create a new group. To set this permission follow the next steps:

- Enter the Admin control panel
- Select 'Users'
- Select the user(s) in question and click 'Edit'
- Scroll down to 'Access' and enable 'Can Create Group'

63.2.3. How to insert the Template

In order to add the template, you can follow the [guide](#).

63.2.4. How to run the Template

- Build the job with parameters:
 - PROJECT_NAME: The project name.
 - PROJECT_SUFFIX: The project name suffix. As your project can have multiple assets

(backend, frontend, middleware...), you can define a suffix in order to identify each one with a different name

- DB_TYPE: The type of the database. Possible values: h2 | postgresql | mysql | mariadb | oracle | hana | db2
- GROUP_ID: The group id of the project.
- GITLAB_USER_PRIVATE_TOKEN: Private Token of a Production Line Gitlab User that can be used to create repositories. Created as prerequisite, you only need to add it as credential with GitLab API token **Kind**.
- GITLAB_CREATE_GROUP_NAME: Name of the GitLab group. The repository will be created inside this group.
- GITLAB_CREATE_PROJECT_DESCRIPTION: Description of the repository.
- DEPLOY: Choose the environment where you want to deploy. The deployment could be **none**, **docker** or **openshift**. If **docker** or **openshift** were selected, extra parameters will be required in their dedicated steps:
 - Configuring DOCKER:
 - DOCKER_URL: The remote docker daemon URL
 - DOCKER_CERT: Credentials to access docker daemon. If the daemon is not secure, you can leave this empty.
 - Configuring Openshift:
 - OC_NAME: Openshift cluster name. It was defined in the Openshift Configuration template
 - DOCKER_REGISTRY_CREDENTIALS: Nexus docker registry user credentials. It was created in the initialize instance pipeline. The default username is nexus-api, the default password is the same as your service account.

After executing this template, you will have:

- A new GitLab repository.
 - The repository group is the value passed in the GITLAB_CREATE_GROUP_NAME parameter.
 - The repository name is *PROJECT_NAME-PROJECT_SUFFIX*
 - The repository contains a clean devon4j project.
 - The repository contains a Jenkinsfile.
 - The repository has already setted the jenkins webhook.
 - The repository protects the branches master and release/* to only maintainers to push. Develop is the default branch.
- A new multibranch pipeline in jenkins inside the folder *PROJECT_NAME* with the name *PROJECT_NAME-PROJECT_SUFFIX*. As the webhook is already configured, it should be executed on every push to GitLab repository.
- If you choose docker for deployment, your Jenkinsfile should contain two extra stages in order to build and deploy the docker image. Also, the repository should contain the Dockerfiles to

create the docker images.

- If you choose OpenShift for deployment, three new applications should be created in your OpenShift. Those applications represent three environments of your application: develop, uat and stage. Also, your Jenkinsfile should contain three extra stages in order to build and deploy the docker image and check that the pod is running without errors. Also, the repository should contain the Dockerfiles to create the docker images.



63.3. devon4ng Template for Production Line

63.3.1. Overview

This template will configure your PL instance to have a 'ready to use' devon4ng template. It can be used as a starting point for your Angular projects.

This includes CICD files for a devonfw technology stack with configuration for:

- ProductionLine instance
- docker or openshift deployment
- pushing artifacts to nexus3

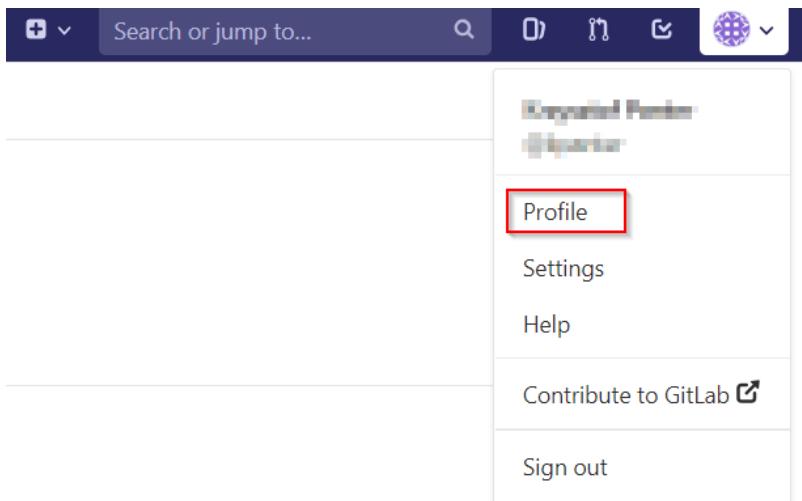
63.3.2. Prerequisites

To be able to run Jenkins Angular job under ProductionLine you need to configure below settings in Jenkins and Gitlab

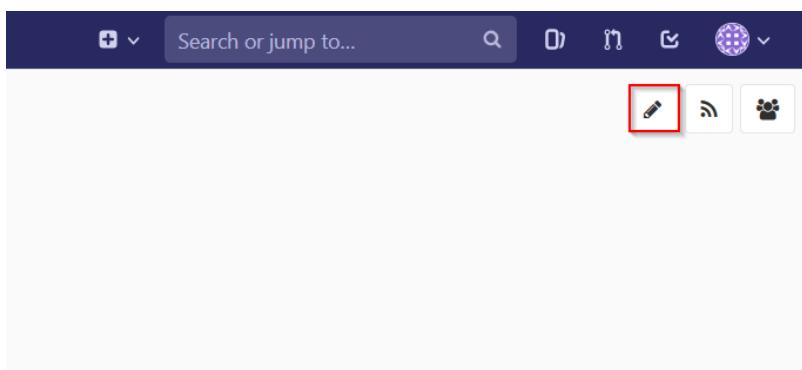
- Jenkins
 - Execute the [initialize instance template](#)
 - If you plan to deploy into OpenShift, you need to execute [openshift-configuration template](#) also.
- Gitlab

- Generate User Private Token

Go to your Profile in Gitlab



Next click on the pen icon



On the left menu choose Access Tokens and put token name and check fields like below

User Settings > Access Tokens

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name
test_token

Expires at
YYYY-MM-DD

Scopes

api
Grants complete read/write access to the API, including all groups and projects.

read_user
Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

sudo
Grants permission to perform API actions as any user in the system, when authenticated as an admin user.

read_repository
Grants read-only access to repositories on private projects using Git-over-HTTP (not using the API).

Create personal access token

Click "Create personal access token", you should receive notification about created token and token string. Copy the token string.

Your new personal access token has been created.

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Your New Personal Access Token

`gHJ1234567890abcd1234567890`

Make sure you save it - you won't be able to access it again.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

The GitLab API user needs to have API access and the rights to create a new group. To set this permission follow the next steps:

- Enter the Admin control panel
- Select 'Users'
- Select the user(s) in question and click 'Edit'
- Scroll down to 'Access' and un-tick 'Can Create Group'

63.3.3. How to insert the Template

In order to add the template, you can follow the [guide](#).

63.3.4. How to run the Template

- Build the job with parameters:
 - PROJECT_NAME: The project name.
 - PROJECT_SUFFIX: The project name suffix. As your project can have multiple assets (backend, frontend, middleware...), you can define a suffix in order to identify each one with a different name
 - GROUP_ID: The group id of the project.
 - GITLAB_USER_PRIVATE_TOKEN: Private Token of a Production Line Gitlab User that can be used to create repositories. Created as prerequisite, you only need to add it as credential with GitLab API token **Kind**.
 - GITLAB_CREATE_GROUP_NAME: Name of the GitLab group. The repository will be created inside this group.
 - GITLAB_CREATE_PROJECT_DESCRIPTION: Description of the repository.
 - DEPLOY: Choose the environment where you want to deploy. The deployment could be **none**, **docker** or **openshift**. If **docker** or **openshift** were selected, extra parameters will be required in their dedicated steps:
 - Configuring DOCKER:
 - DOCKER_URL: The remote docker daemon URL
 - DOCKER_CERT: Credentials to access docker daemon. If the daemon is not secure, you can leave this empty.
 - Configuring Openshift:

- OC_NAME: Openshift cluster name. It was defined in the Openshift Configuration template
- DOCKER_REGISTRY_CREDENTIALS: Nexus docker registry user credentials. It was created in the initialize instance pipeline. The default username is nexus-api, the default password is the same as your service account.

After executing this template, you will have:

- A new GitLab repository.
 - The repository group is the value passed in the GITLAB_CREATE_GROUP_NAME parameter.
 - The repository name is *PROJECT_NAME-PROJECT_SUFFIX*
 - The repository contains a clean devon4ng project.
 - The repository contains a Jenkinsfile.
 - The repository has already setted the jenkins webhook.
 - The repository protects the branches master and release/* to only maintainers to push. Develop is the default branch.
- A new multibranch pipeline in jenkins inside the folder *PROJECT_NAME* with the name *PROJECT_NAME-PROJECT_SUFFIX*. As the webhook is already configured, it should be executed on every push to GitLab repository.
- If you choose docker for deployment, your Jenkinsfile should contain two extra stages in order to build and deploy the docker image. Also, the repository should contain the Dockerfiles to create the docker images.
- If you choose OpenShift for deployment, three new applications should be created in your OpenShift. Those applications represent three environments of your application: develop, uat and stage. Also, your Jenkinsfile should contain three extra stages in order to build and deploy the docker image and check that the pod is running without errors. Also, the repository should contain the Dockerfiles to create the docker images.

Unresolved directive in production-line.wiki/master-production-line.asciidoc - include::devon4net-pl.asciidoc[leveloffset=2]





63.4. devon4node Template for Production Line

63.4.1. Overview

This template will configure your PL instance to have a 'ready to use' devon4node template. It can be used as a starting point for your Node projects.

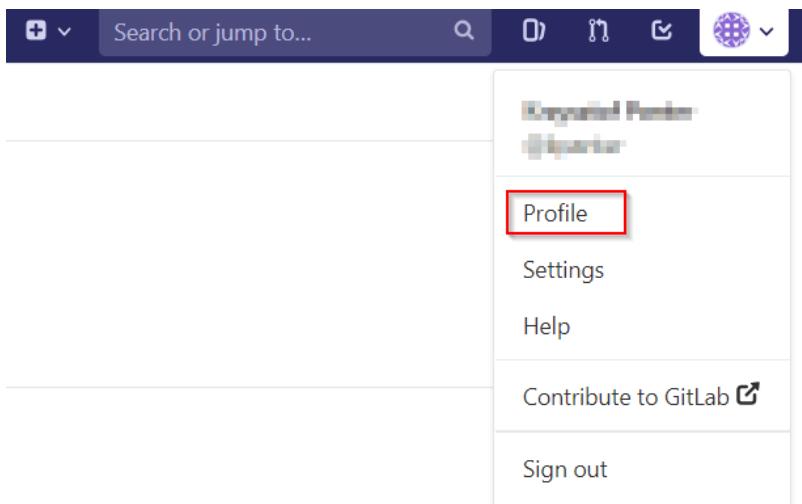
This includes CICD files for a devonfw technology stack with configuration for:

- ProductionLine instance
- docker or openshift deployment
- pushing artifacts to nexus3

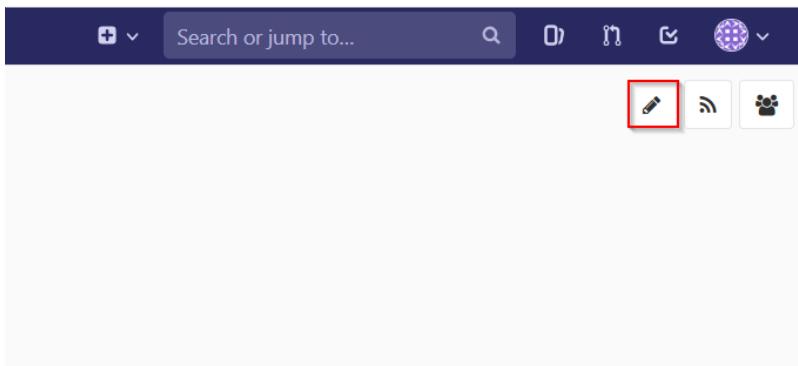
63.4.2. Prerequisites

To be able to run Jenkins Node job under ProductionLine you need to configure below settings in Jenkins and Gitlab

- Jenkins
 - Execute the [initialize instance template](#)
 - If you plan to deploy into OpenShift, you need to execute [openshift-configuration template](#) also.
- Gitlab
 - Generate User Private Token
Go to your Profile in Gitlab



Next click on the pen icon



On the left menu choose Access Tokens and put token name and check fields like below

User Settings > Access Tokens

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

Expires at

Scopes

api

Grants complete read/write access to the API, including all groups and projects.

read_user

Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

sudo

Grants permission to perform API actions as any user in the system, when authenticated as an admin user.

read_repository

Grants read-only access to repositories on private projects using Git-over-HTTP (not using the API).

Create personal access token

Click "Create personal access token", you should receive notification about created token and token string. Copy the token string.

User Settings > Access Tokens

Your new personal access token has been created.

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Your New Personal Access Token

Make sure you save it - you won't be able to access it again.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

The GitLab API user needs to have API access and the rights to create a new group. To set this permission follow the next steps:

- Enter the Admin control panel
- Select 'Users'
- Select the user(s) in question and click 'Edit'
- Scroll down to 'Access' and un-tick 'Can Create Group'

63.4.3. How to insert the Template

In order to add the template, you can follow the [guide](#).

63.4.4. How to run the Template

- Build the job with parameters:
 - PROJECT_NAME: The project name.
 - PROJECT_SUFFIX: The project name suffix. As your project can have multiple assets (backend, frontend, middleware...), you can define a suffix in order to identify each one with a different name
 - GROUP_ID: The group id of the project.
 - GITLAB_USER_PRIVATE_TOKEN: Private Token of a Production Line Gitlab User that can be used to create repositories. Created as prerequisite, you only need to add it as credential with GitLab API token **Kind**.
 - GITLAB_CREATE_GROUP_NAME: Name of the GitLab group. The repository will be created inside this group.
 - GITLAB_CREATE_PROJECT_DESCRIPTION: Description of the repository.
 - DEPLOY: Choose the environment where you want to deploy. The deployment could be **none**, **docker** or **openshift**. If **docker** or **openshift** were selected, extra parameters will be required in their dedicated steps:
 - Configuring DOCKER:
 - DOCKER_URL: The remote docker daemon URL
 - DOCKER_CERT: Credentials to access docker daemon. If the daemon is not secure, you can leave this empty.
 - Configuring Openshift:
 - OC_NAME: Openshift cluster name. It was defined in the Openshift Configuration template
 - DOCKER_REGISTRY_CREDENTIALS: Nexus docker registry user credentials. It was created in the initialize instance pipeline. The default username is nexus-api, the default password is the same as your service account.

After executing this template, you will have:

- A new GitLab repository.
 - The repository group is the value passed in the GITLAB_CREATE_GROUP_NAME parameter.
 - The repository name is *PROJECT_NAME-PROJECT_SUFFIX*
 - The repository contains a clean devon4node project.
 - The repository contains a Jenkinsfile.
 - The repository has already setted the jenkins webhook.
 - The repository protects the branches master and release/* to only maintainers to push.

Develop is the default branch.

- A new multibranch pipeline in jenkins inside the folder *PROJECT_NAME* with the name *PROJECT_NAME-PROJECT_SUFFIX*. As the webhook is already configured, it should be executed on every push to GitLab repository.
- If you choose docker for deployment, your Jenkinsfile should contain two extra stages in order to build and deploy the docker image. Also, the repository should contain the Dockerfiles to create the docker images.
- If you choose OpenShift for deployment, three new applications should be created in your OpenShift. Those applications represent three environments of your application: develop, uat and stage. Also, your Jenkinsfile should contain three extra stages in order to build and deploy the docker image and check that the pod is running without errors. Also, the repository should contain the Dockerfiles to create the docker images.



63.5. From existing devonfw Template for Production Line

63.5.1. Overview

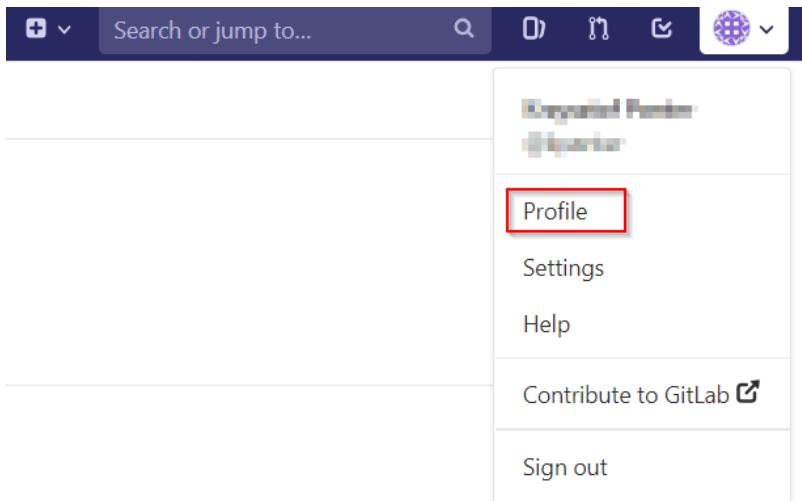
From existing devonfw template is very similar to devon4j, devon4ng, devon4net and devon4node templates. The main difference is from existing devonfw template will no create a new devonfw project, it takes an existing project from GitLab and then add/create everything in order to apply a CICD strategy to your project.

63.5.2. Prerequisites

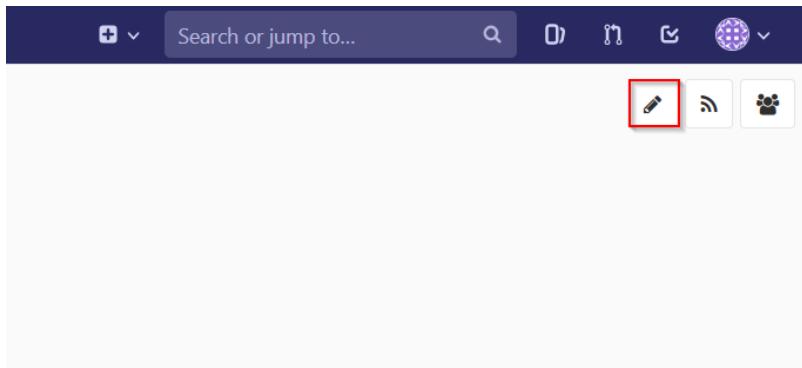
To be able to run Jenkins Node job under ProductionLine you need to configure below settings in Jenkins and Gitlab

- Jenkins

- Execute the [initialize instance template](#)
- If you plan to deploy into OpenShift, you need to execute [openshift-configuration template](#) also.
- Gitlab
 - Create a project and upload your current code. In order to start a new project in your local machine, you can use the [devonfw-ide](#). The project must be a devon4j, devon4ng, devon4net or devon4node project.
 - Generate User Private Token
Go to your Profile in Gitlab



Next click on the pen icon



On the left menu choose Access Tokens and put token name and check fields like below

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

Expires at

Scopes **api**

Grants complete read/write access to the API, including all groups and projects.

 read_user

Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

 sudo

Grants permission to perform API actions as any user in the system, when authenticated as an admin user.

 read_repository

Grants read-only access to repositories on private projects using Git-over-HTTP (not using the API).

Click "Create personal access token", you should receive notification about created token and token string. Copy the token string.

Your new personal access token has been created.

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Your New Personal Access Token

Make sure you save it - you won't be able to access it again.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

The GitLab API user needs to have API access and the rights to create a new group. To set this permission follow the next steps:

- Enter the Admin control panel
- Select 'Users'
- Select the user(s) in question and click 'Edit'
- Scroll down to 'Access' and un-tick 'Can Create Group'

63.5.3. How to insert the Template

In order to add the template, you can follow the [guide](#).

63.5.4. How to run the Template

- Build the job with parameters:
 - REPOSITORY_URL: The internal repository URL. Without protocol. Example: gitlab-core:80/gitlab/mygroup/myproject-frontend.
 - GIT_BRANCH: The branch where you want to apply the CICD changes.
 - MERGE_STRATEGY: Choose the merge strategy for cicdgen. For more information see the

CICDGEN merge documentation page

- GITLAB_USER_PRIVATE_TOKEN: Private Token of a Production Line Gitlab User that can be used to create/update repositories. The token proprietary user must have admin rights in the repository. Created as prerequisite, you only need to add it as credential with GitLab API token **Kind**.
- DEPLOY: Choose the environment where you want to deploy. The deployment could be **none**, **docker** or **openshift**. If **docker** or **openshift** were selected, extra parameters will be required in their dedicated steps:
 - Configuring DOCKER:
 - DOCKER_URL: The remote docker daemon URL
 - DOCKER_CERT: Credentials to access docker daemon. If the daemon is not secure, you can leave this empty.
 - Configuring Openshift:
 - OC_NAME: Openshift cluster name. It was defined in the Openshift Configuration template
 - DOCKER_REGISTRY_CREDENTIALS: Nexus docker registry user credentials. It was created in the initialize instance pipeline. The default username is nexus-api, the default password is the same as your service account.

After executing this template, you will have:

- Your GitLab project updated.
 - Added a Jenkinsfile with all CICD stages.
 - The repository is updated in order to have the jenkins webhook.
- A new multibranch pipeline in jenkins inside the folder *PROJECT_NAME* with the name *PROJECT_NAME-PROJECT_SUFFIX*. As the webhook is already configured, it should be executed on every push to GitLab repository.
- If you choose docker for deployment, your Jenkinsfile should contain two extra stages in order to build and deploy the docker image. Also, the repository should contain the Dockerfiles to create the docker images.
- If you choose OpenShift for deployment, three new applications should be created in your OpenShift. Those applications represent three environments of your application: develop, uat and stage. Also, your Jenkinsfile should contain three extra stages in order to build and deploy the docker image and check that the pod is running without errors. Also, the repository should contain the Dockerfiles to create the docker images.

64. Utility Templates

64.1. Initialize Instance Template for Production Line

64.1.1. Introduction

Production Line Templates allows you to create/configure certain task. In order to work properly, Production Line Templates needs some previous configurations. You can do it [manually](#) or executing the Initialize Instance Template.

64.1.2. Prerequisites

In order to be able to start this template, you need:

- The [Production Line Shared Lib](#) added in Jenkins
- The following plugins:
 - [Custom Tools Plugin](#)
 - [HTTP Request Plugin](#)
 - [Job DSL Plugin](#)
 - [SonarQube plugin](#)
 - [Pipeline Maven Plugin](#)
 - [NodeJS Plugin](#)
- A service account added in the LAM

Production Line provides by default the Shared Lib and the plugins, so no actions are required. The only thing that you need to do manually is the creation of the service account.

In order to create the service account you need:

1. Open the LAM
2. Press the **New User** button

LDAP Account Manager - 6.6 (Logged in as: admin)

Users Groups

+ New user X Delete selected users File upload

User count: 100

3. Enter the required parameters

Save Set password

New user

Personal Unix

First name * Jenkins

Last name * Jenkins

Address

State + ?

Office name + ?

Add photo

Contact data

Email address * jenkins@example.com + ?

4. Change to **Unix** tab and enter the required parameters

Jenkins Jenkins
jenkins@example.com

Personal

Unix

User name * jenkins

Common name Jenkins Jenkins

UID number 10140

Primary group admins

Additional groups Edit groups

Home directory * /home/jenkins

Login shell /bin/bash

The user name will be used later in order to login. As this user will do some configuration changes, its primary group must be admins.

- Set a password for the user.

nkins
mple.com

Personal

User name * jenkins

Common name Jenkins Jenkins

UID number 10140

Primary group admins

Additional groups Edit groups

Home directory * /home/jenkins

Set password

Password [REDACTED] ?

Repeat password [REDACTED]

Unix

Ok Set random password Cancel

- Press the **Save** button

The screenshot shows the LDAP Account Manager interface. At the top, there's a header with a logo and the text "LDAP Account Manager - 6.6 (Logged in as: admin > pl > s2-eu > capgemini > local)". Below the header, there are two tabs: "Users" and "Groups". The "Users" tab is active. In the center, there's a form for creating a new user. The user name is "Jenkins Jenkins" and the email is "jenkins@example.com". Below the form are two buttons: "Save" and "Set password". The "Save" button is highlighted with a red box and has a cursor icon pointing to it, indicating it's the next step to be taken.

64.1.3. Template

In order to execute this template, you need to add it into Jenkins manually. In order to do that, you can follow [this guide](#)

Parameters

The required parameters are:

- **svcaccount**: The service account created as prerequisite. It must be added as a Jenkins credential.
- **installDeploymentPlugins**: With this parameter you can install extra plugins into Jenkins. Also, you can add extra template utils.

Execution

1. Press the Build with Parameters button
2. Insert the parameters.
3. If the service account is not added as credential, please add a new entry.
4. Press the **Build** button.
5. Wait until the pipeline ends.



if any plugin is installed, Jenkins will be restarted and the pipeline will fail. You need to execute it again with the same parameters.

Jenkins > initialize_instance >

Pipeline initialize_instance

This build requires parameters:

svcacount	<input type="checkbox"/> List user credentials - current -	Add
installDeploymentPlugins	None	Install extra plugins related with the deployment method.

Build

Stage View

#11	Average stage times: (Average full run time: ~33s)
Nov 28 16:03	No Changes

Average stage times:
(Average full run time: ~33s)

Declarative: Checkout SCM Install plugins Configure SonarQube Create UTIL pipelines Configure Nexus 3 Configure Maven File

Declarative: Checkout SCM	Install plugins	Configure SonarQube	Create UTIL pipelines	Configure Nexus 3	Configure Maven File
3s	4s	36s	4s	1s	590ms
1s	1s	7s	4s	1s	467ms

The result

- Install plugins stage

In this stage the following plugins will be installed:

- SSH Credentials Plugin
- Custom Tools Plugin
- HTTP Request Plugin
- Job DSL Plugin
- SonarQube plugin
- Ansible Plugin
- Pipeline Maven Plugin
- NodeJS Plugin
- GitLab Plugin
- OWASP Dependency-Check Plugin
- If `installDeploymentPlugins` is Docker or Docker+Openshift, extra plugins will be installed:
 - Docker Plugin
 - Docker build step plugin
 - Docker Pipeline Plugin
 - JClouds Plugin
- If `installDeploymentPlugins` is Openshift or Docker+Openshift, extra plugins will be installed:

- [OpenShift Client Plugin](#)

- Configure SonarQube stage

This stage is the responsible of configure the Jenkins-SonarQube integration. It will:

- Generate a SonarQube API token for the user **Admin**
- Register the token in Jenkins as credential with the id **sonar-token**
- Add the SonarQube server in Jenkins → Manage Jenkins → Configure System → SonarQube servers. The values used are:
 - Name: **SonarQube**
 - Server URL: <http://sonarqube-core:9000/sonarqube> (default Production Line SonarQube URL)
 - Server authentication token: **sonar-token** (generated in the previous step)
- Add a webhook in SonarQube:
 - Name: **jenkins**
 - URL: <http://jenkins-core:8080/jenkins/sonarqube-webhook/>
- Install the following SonarQube plugins:
 - java
 - javascript
 - typescript
 - csharp
 - web
 - cssfamily
 - jacoco
 - checkstyle
 - cobertura
 - smells
 - findbugs
 - scmgit
 - ansible
 - sonar-dependency-check-plugin
- Restart the SonarQube server in order to enable the plugins installed.

- Create UTIL templates stage

Some templates needs that Jenkins has installed some plugins. If the plugins are not installed, the template will fail. In order to prevent this behaviour, we use the **initialize-instance** to install all plugins required in order templates. Then, we create another templates that will use the plugins installed by **initialize-instance**. In this stage we create some template utils to

configure Jenkins after all required plugins are installed. Those templates are:

- [Install_SonarQube_Plugin](#)
- If `installDeploymentPlugins` is `Docker` or `Docker+Openshift`: [Docker_Configuration](#)
- If `installDeploymentPlugins` is `Openshift` or `Docker+Openshift`: [Openshift_Configuration](#)
- Configure Nexus 3 stage

This stage will configure the Production Line Nexus3

- Enable anonymous access
- Add a internal user to download/upload docker images
 - username: `nexus-api`
 - password: The same as the service account created in LAM
- Create the maven repositories: maven-central, maven-snapshots, maven-release, maven-plugin
- Create the docker repository
- Create the npmjs repositories: npmjs, npm-registry, npm
- Create in Jenkins a new credential with the id `nexus-api` with the username and password created in nexus3
- Configure Maven File stage

This stage adds the nexus3 credentials creadted in the previous stage to the maven global configuration file with the id `pl-nexus`



Edit Configuration File

description

The configuration

ID	MavenSettings				
Name	MyGlobalSettings				
Comment	global settings				
<input checked="" type="checkbox"/> Replace All					
Server Credentials	<table border="1"> <tbody> <tr> <td>ServerId</td> <td>pl-nexus</td> </tr> <tr> <td>Credentials</td> <td>nexus-api/***** (Credentials for connect to docker registry)</td> </tr> </tbody> </table>	ServerId	pl-nexus	Credentials	nexus-api/***** (Credentials for connect to docker registry)
ServerId	pl-nexus				
Credentials	nexus-api/***** (Credentials for connect to docker registry)				

Now, you are able to execute other templates adding them manually or using the Production Line Market Place.

64.2. Install SonarQube Plugin

64.2.1. Introduction

SonarQube can extends its behaviour by adding plugins. Some on them can be installed by using the SonarQube Marketplace, others can be installed by copying the .jar into the SonarQube plugins folder.

Overview

This template will help you to install SonarQube plugins by copying the .jar into the SonarQube plugins folder. As you do not have access to the Production Line volumes, it will help you when you want to install a plugin that is not installed in the SonarQube Marketplace.

It will:

- Download the .jar file from a provided URL.
- Copy the .jar file to the plugins folder.
- Restart the SonarQube server in order to enable the plugin.



this template only works in a Production Line instance.

64.2.2. Template

This template will be automatically created in your jenkins after executing the [Initialize_Instance](#) template inside the [UTILS](#) folder with the name [Install_SonarQube_Plugin](#).

For manual creation see: [How to add a Template](#)



This template needs the [devonfw Production Line Shared Lib](#)

Parameters

The only parameter required is the plugin download URL.

Execution

1. Press the Build with Parameters button
2. Insert plugin the download url. Example: <https://github.com/dependency-check/dependency-check-sonar-plugin/releases/download/1.2.6/sonar-dependency-check-plugin-1.2.6.jar>
3. Press the **Build** button.
4. Wait until the pipeline ends.

Jenkins > UTILS > Install_SonarQube_Plugin

Pipeline Install_SonarQube_Plugin

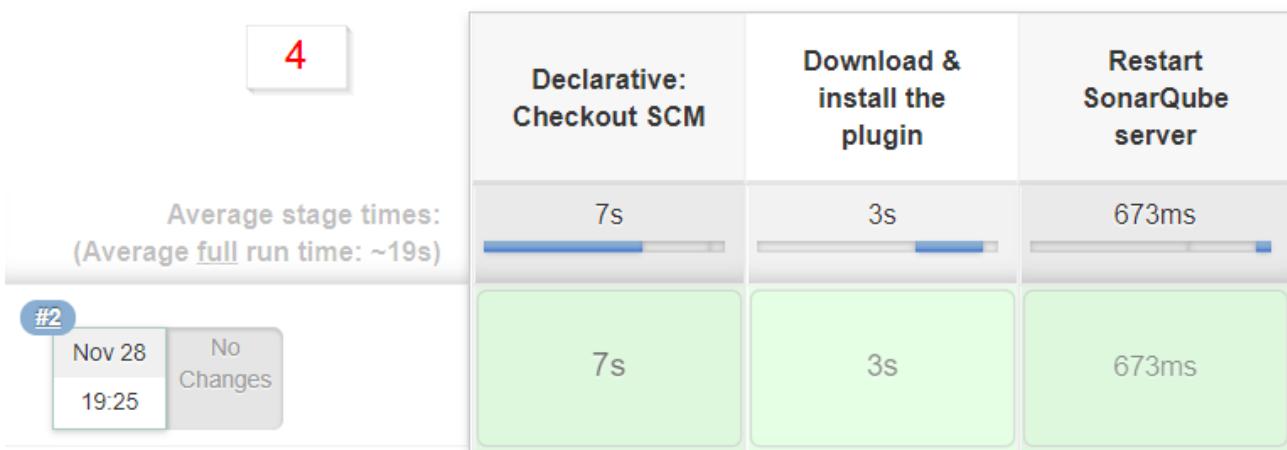
This build requires parameters:

1 **Build with Parameters**

2 **plugin_download_url** <https://github.com/dependency-check/dependency-check-sonar-plugin/releases/download/1.2.6/sonar-dependency-check-plugin-1.2.6.jar>

3 **Build**

Stage View



After the execution, when the SonarQube is restarted, you can check that your plugin is installed visiting the Marketplace.

sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

Administration Configuration Security Projects System Marketplace

Dependency-Check Integrates Dependency-Check reports into SonarQube	1.2.6 installed	Homepage Issue Tracker Licensed under GNU LGPL 3 Developed by OWASP	Uninstall
Findbugs External Analyzers Analyze Java, Scala, Closure and JSP code with SpotBugs 3.1.2	3.11.1 installed	Homepage Issue Tracker Licensed under GNU LGPL 3 Developed by SpotBugs Team	Uninstall
Git Integration Git SCM Provider for SonarQube	1.9.1 (build 1834) installed	Homepage Issue Tracker Licensed under GNU LGPL 3 Developed by SonarSource	Uninstall

64.3. Docker Configuration

64.3.1. Introduction

Docker is the most popular container technology. It allows you to build your application in an [image](#) and then deploy it into a [container](#).

Overview

This template allow you to configure Jenkins in order to work with docker.

It will:

- Add docker client as custom tool.
- Configure docker to work with an external docker dameon.

64.3.2. Prerequisites

In order to execute this template, you need the following plugins installed in your Jenkins:

- [Docker Plugin](#)
- [Docker build step plugin](#)
- [Docker Pipeline](#)
- [JClouds Plugin](#)



The initialize instance template will install all plugins if you select 'Docker' or 'Docker+Openshift' in the `installDeploymentPlugins` parameter

64.3.3. Template

This template will be automatically created in your jenkins after executing the `Initialize_Instance` template inside the `UTILS` folder with the name `Docker_Configuration`.

For manual creation see: [How to add a Template](#)



This template needs the [devonfw Production Line Shared Lib](#)

Parameters

The only parameter required is remote docker daemon URL. Example: `tcp://127.0.0.1:2367`



You need to expose the docker daemon manually in your machine. Here you can find [how to do it](#)



This configuration requires that the docker daemon has no security. It's prepared for development environments, for production environments please add [security](#) to your docker daemon.

Execution

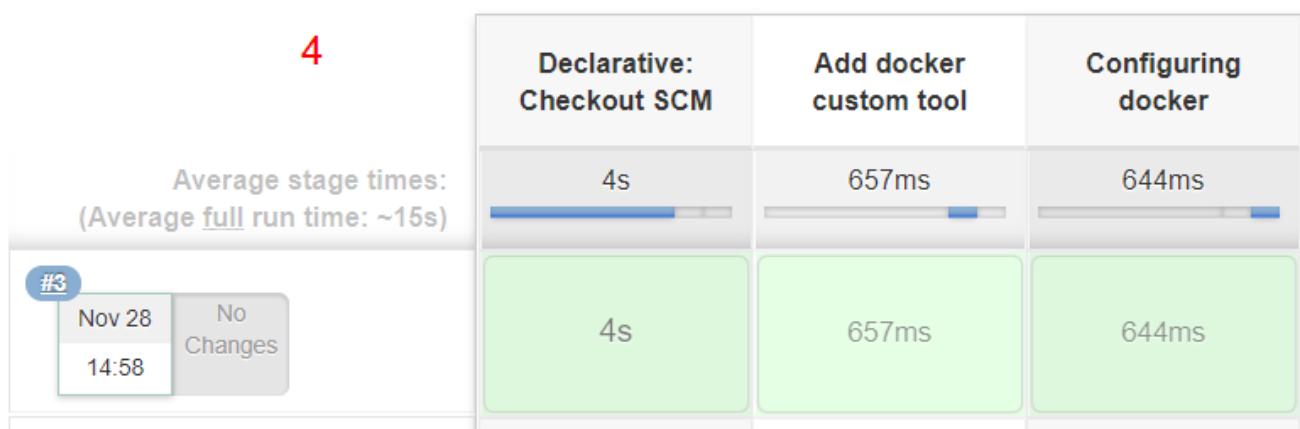
1. Press the Build with Parameters button
2. Insert remote docker daemon URL.
3. Press the `Build` button.
4. Wait until the pipeline ends.

1

2

3

Stage View



Then, you can see that the docker is configured and the remote docker daemon environment variable is set:

Global properties

- Disable deferred wipeout on this node
- Environment variables

List of variables

Name	DOCKER_HOST
Value	tcp://10.36.39.36:2376

Add Delete

Custom tool

Name docker-global

Install automatically

Run Shell Command

Label

Command

```
if ! which docker > /dev/null; then
    sudo apt-get update
    sudo apt-get install -y apt-transport-https ca-certificates curl gnupg2 software-properties-common
    curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
    sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian ${lsb_release -cs} stable"
    sudo apt-get update
    sudo apt-get install -y docker-ce-cli
    docker -v
fi
```

Tool Home /usr/bin

Delete Installer

The environment variable is setted globally, if you want to use another remote docker daemon for a specific build, you can override the `DOCKER_HOST` environment variable in your job.

If the `DOCKER_HOST` is already setted globally, when you execute again this template the value will not be changed. You need to change the value manually at: Jenkins → Manage Jenkins → Configure System → Global properties

64.4. Docker Configuration

64.4.1. Introduction

OpenShift is a docker container orchestrator built on top Kubernetes.

Overview

This template allow you to configure Jenkins in order to work with OpenShift.

It will:

- Add OpenShift client as custom tool.
- Configure an OpenShift cluster to work with.

64.4.2. Prerequisites

In order to execute this template, you need the following plugins installed in your Jenkins:

- [OpenShift Client Plugin](#)



The initialize instance template will install all plugins if you select **Openshift** or **Docker+Openshift** in the `installDeploymentPlugins` parameter

64.4.3. Template

This template will be automatically created in your jenkins after executing the `Initialize_Instance` template inside the `UTILS` folder with the name `Openshift_Configuration`.

For manual creation see: [How to add a Template](#)



This template needs the `devonfw Production Line Shared Lib`

Parameters

The required parameters are:

- `ocName`: The name of the OpenShift connection. You can define multiple OpenShift connections by changing the name.
- `ocUrl`: The OpenShift URL.
- `ocProject`: The OpenShift Project.
- `ocToken`: The OpenShift token. In order to have a long-term token, this token should be a service account token.

Execution

1. Press the Build with Parameters button
2. Insert the parameters.
3. If the OpenShift token is not added as credential, please add a new entry.
4. Press the `Build` button.
5. Wait until the pipeline ends.



If a cluster already exists with the provided name, it will not modify anything.

Jenkins

Jenkins > UTILS > Openshift_Configuration

Pipeline Openshift_Configuration

This build requires parameters:

- ocName: default (2)
- ocUrl: https://ocp.itaas.s2-eu.capgemini.com (3)
- ocProject: s2portaldev
- ocToken: - none - (3)

Add (4)

Jenkins Credentials Provider: Jenkins (3)

Add Credentials

Domain: Global credentials (unrestricted)

Kind: OpenShift Token for OpenShift Client Plugin (2)

Scope: Global (Jenkins, nodes, items, all child items, etc)

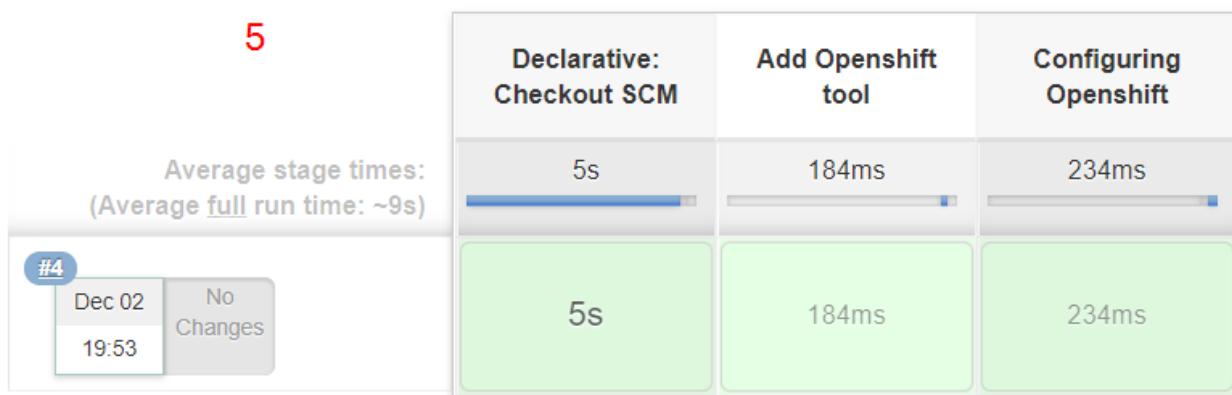
Token: (redacted)

ID: openshift-default-token

Description: Openshift token for default connection (redacted)

Add (1) **Cancel**

Stage View



You can add more clusters by executing the template again or in Jenkins → Manage Jenkins → Configure System

OpenShift Client Plugin

Cluster Configurations

OpenShift Cluster	
Cluster Name	default
API Server URL	https://ocp.itaas.s2-eu.capgemini.com
Credentials	openshift-token (openshift-token) ▾ Add
Disable TLS Verify	<input checked="" type="checkbox"/>
Server Certificate Authority	
Default Project	s2portaldev
Add OpenShift Cluster ▾	

Xunc

65. MrChecker

65.1. MrChecker under ProductionLine



65.1.1. Introduction

MrChecker is end to end automation test framework written in Java. It has been released by devonfw but it is not supported by the devonfw core team.

This framework consist of eight test modules:

- Core test module
- Selenium test module
- WebAPI test module
- Security test module
- DataBase test module
- Standalone test module
- DevOps module

65.1.2. Prerequisites

To be able to run Jenkins MrChecker job under ProductionLine you need to configure below settings in Jenkins and Gitlab

- Jenkins
 - Add Jenkins Shared Library using documentation <https://github.com/devonfw/production-line-shared-lib>
 - Install required plugins:
 - HTTP Request Plugin
 - Allure Jenkins Plugin

- In Jenkins Global Tool Configuration configure Allure Commandline and Maven like

Allure Commandline

Allure Commandline installations

Add Allure Commandline

Allure Commandline

Name

Install automatically

From Maven Central

Version

Maven

Maven installations

Add Maven

Maven

Name

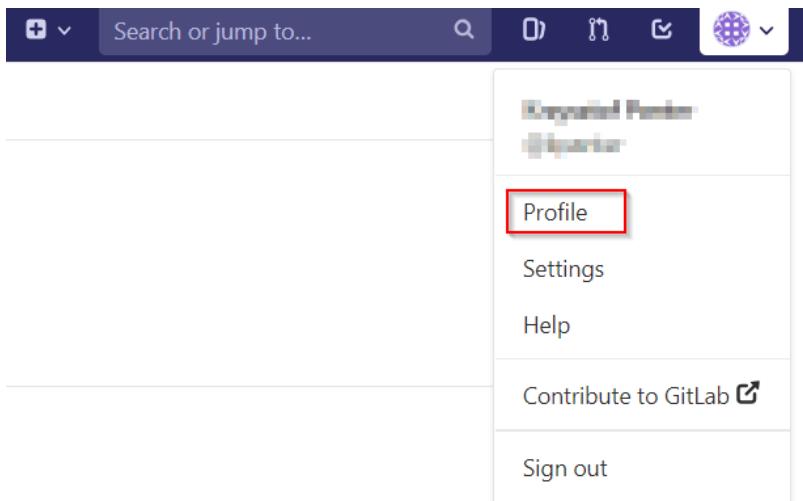
Install automatically

Install from Apache

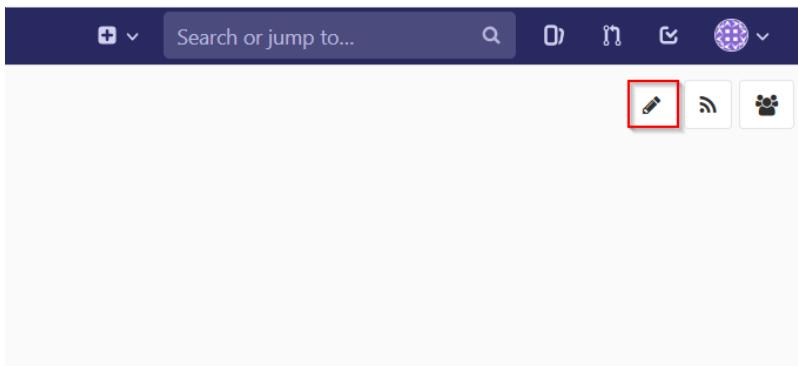
Version

- Gitlab

- Generate User Private Token
Go to your Profile in Gitlab



Next click on the pen icon



On the left menu choose Access Tokens and put token name and check fields like below

User Settings > Access Tokens

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

Expires at

Scopes

api

Grants complete read/write access to the API, including all groups and projects.

read_user

Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

sudo

Grants permission to perform API actions as any user in the system, when authenticated as an admin user.

read_repository

Grants read-only access to repositories on private projects using Git-over-HTTP (not using the API).

Create personal access token

Click "Create personal access token", you should receive notification about created token and token string. Copy the token string.

User Settings > Access Tokens

Your new personal access token has been created.

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Your New Personal Access Token

Make sure you save it - you won't be able to access it again.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

The GitLab API user needs to have API access and the rights to create a new group. To set this permission follow the next steps:

- Enter the Admin control panel
- Select 'Users'
- Select the user(s) in question and click 'Edit'
- Scroll down to 'Access' and un-tick 'Can Create Group'

65.1.3. How to insert the Template

- Create new Jenkins Pipeline Job
- In job configuration check "This project is parametrized", choose "String parameter and provide Name: GITLAB_USER_PRIVATE_TOKEN
Default Value: <GITLAB_TOKEN_STRING_YOU JUST_CREATED>
- Add the template
The guide on how to add a template to your Jenkins can be found in the root directory of the template repository: <https://github.com/devonfw/production-line.git>
- Save job configuration

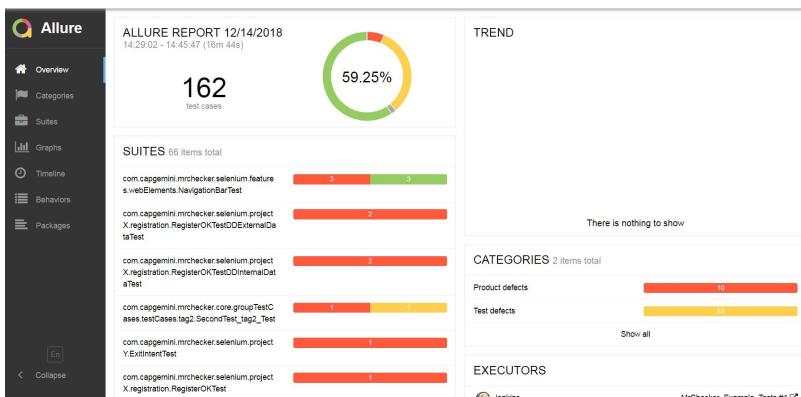
65.1.4. How to run the Template

- Build the job
- After job ends with success wait few seconds for repository import to Gitlab
- As output of the build new Jenkins Pipeline job is created with name "MrChecker_Example_Tests" also new repository "Mrchecker" will be created in Gitlab
- Build "MrChecker_Example_Tests" job

Stage	Average Time
Prepare environment	1s
Git Pull	3s
Build Compile	29s
Integration test	17min 41s

65.1.5. Expected Result

- As output of this job Allure Report will be generated



65.1.6. Summary

Using this documentation you should be able to run MrChercker test framework on ProductionLine.

MrChecker offers two projects to your disposal:

- First project "mrchecker-app-under-test/pipelines/CI/Jenkinsfile_ProductionLine.groovy" has all tests included in the project and is the default project used in "MrChecker_Example_Tests" job.
- Second project "mrchecker-app-under-testboilerplate/pipelines/CI/Jenkinsfile_ProductionLine.groovy" here tests are not included, therefore if you choose to run "MrChecker_Example_Tests" job Allure report will be not generated.

To change the project change script path at the bottom of the "MrChecker_Example_Tests" job.



66. Samples

66.1. devon4j My-Thai-Star Sample Application Template for Production Line

66.1.1. Introduction

Please read all of the following sections carefully.

66.1.2. Overview

This template will configure your PL instance to have a 'ready to use' My-Thai-Star devonfw application. It is only an example. In order to start a new project, please use the other templates. This includes:

- Cloning the official [My-Thai-Star](https://github.com/devonfw/my-thai-star) (<https://github.com/devonfw/my-thai-star>) repository into your GitLab, which allows you to do customizations on your own.
- Adding a build job for the Angular front-end, including a SonarQube analysis and a delivery to Nexus as zip and docker image.
- Adding a build job for the Java back-end, including a SonarQube analysis and a deployment to Nexus as zip and docker image.
- Adding a deployment job for the Angular front-end
- Adding a deployment job for the Java back-end
- Adding a deployment job for the reverse proxy. Please see [My Thai Star deployment documentation](#)

Especially the build and deployment jobs require several additional Jenkins plugins, which are not part of the PL by default. The Template will also take care of those installations.

All build and deployment jobs are taken from the official [My-Thai-Star](https://github.com/devonfw/my-thai-star) (<https://github.com/devonfw/my-thai-star>) repository. The created build and deployment jobs inside Jenkins will use the Jenkinsfiles from the cloned repo in Gitlab. These are currently the following Jenkinsfiles:

Jenkins Jobs

Table 44. Jenkins Jobs

Jenkins job name	Path to Jenkinsfile in repo	Description
MyThaiStar_FRONTEND_BUILD	jenkins/angular/cicd/Jenkinsfile	Builds and tests the Angular frontend. Pushes artifacts to Nexus.
MyThaiStar_SERVER_BUILD	jenkins/java/cicd/Jenkinsfile	Builds and tests the Java backend. Pushes artifacts to Nexus.

Jenkins job name	Path to Jenkinsfile in repo	Description
MyThaiStar_FRONTEND_DEPLOY	jenkins/angular/deployment/Jenkinsfile	Frontend deployment job. Downloads the docker images from Nexus3 and starts a new container usign that image.
MyThaiStar_SERVER_DEPLOY	jenkins/java/deployment/Jenkinsfile	Backend deployment job. Downloads the docker images from Nexus3 and starts a new container usign that image.
MyThaiStar_REVERSE-PROXY_DEPLOY	jenkins/deployment/Jenkinsfile	Reverse proxy deployment job. Downloads the docker images from Nexus3 and starts a new container usign that image. With this job you can also build the reverse proxy image.

66.1.3. How to report Issues

This template is independent from PL and devonfw releases and is also not really connected to one of the projects. Therefore issues that occur during the template setup or execution should be tracked in the issue section of this GitHub project.

66.1.4. How to contribute

In case you see improvements we would love to see a Pull Request.

66.1.5. Prerequisites before running the template

Production Line Components

To use the template you need to make sure that your PL has the following components installed:

- Jenkins (required to run the template and to execute the build/deployment Jobs)
- SonarQube (required for a static code analysis)
- GitLab (required as a reposiitory)
- Nexus3 (required to store the build artifacts)



Additional components can be ordered from the ProductionLine service team.

Technical User Setup

In order to configure the services, we need technical users for the following components:

- Gitlab

- Nexus3
- SonarQube

The following sections describe how to configure the components to enable technical users and tokens.

Manual configuration

In order to configure the Production Line components manually you can follow [this guide](#)

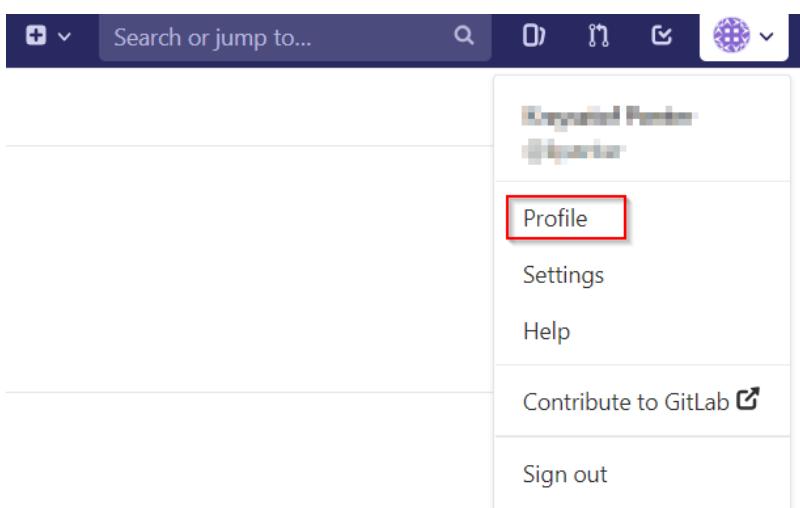
Automatic configuration

In order to configure the Production Line components automatically you can follow [this guide](#)

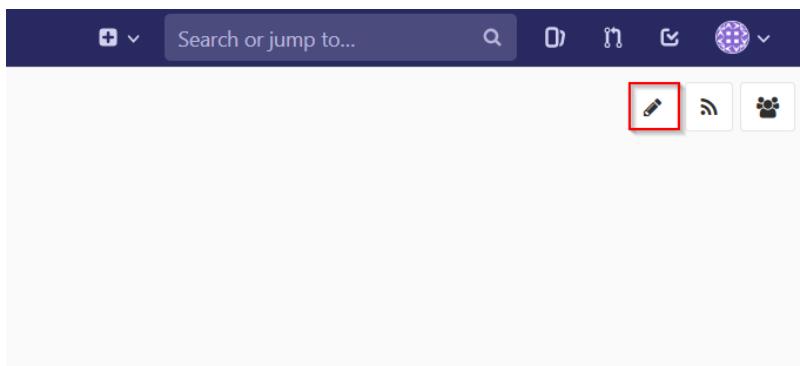
There is one thing that **initialize-template** can not do automatically: the gitlab token creation.

The creation of the GitLab Group and Project will require a private GitLab token which has to be created manually. The token can be obtained like this:

1. Go to your Profile in Gitlab



1. Next click on the pen icon



1. On the left menu choose Access Tokens and put token name and check fields like below

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

test_token

Expires at

YYYY-MM-DD

**Scopes** **api**

Grants complete read/write access to the API, including all groups and projects.

 read_user

Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

 sudo

Grants permission to perform API actions as any user in the system, when authenticated as an admin user.

 read_repository

Grants read-only access to repositories on private projects using Git-over-HTTP (not using the API).

Create personal access token

1. Click "Create personal access token", you should receive notification about created token and token string. Copy the token string.

Your new personal access token has been created.

Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Your New Personal Access Token

Make sure you save it - you won't be able to access it again.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

! The GitLab API user needs to have API access and the rights to create a new group. To set this permission follow the next steps:

1. Enter the Admin control panel
2. Select 'Users'
3. Select the user(s) in question and click 'Edit'
4. Scroll down to 'Access' and un-tick 'Can Create Group'

Build/Deployment Requirements

The My Thai Star CICD pipelines will create a docker image and then the deployment pipelines will use it in order to deploy the application. As Production Line do not include a docker daemon, you need an additional server to do it. Those server needs:

- Docker-CE has to be installed
- **Docker daemon exposed**

66.1.6. How to run it



If Jenkins needs to install plugins, a restart will be performed. So please make sure, that nothing important is running.



We have job-parameters inside the template Jenkinsfile that will only be active if Jenkins has run the job at least once!

Setup template job in Jenkins

The guide on how to add a template to your Jenkins can be found in the root directory of the template repository: <https://github.com/devonfw/production-line.git>

Execute the Jenkins job in your Jenkins

- Go to the Jenkins job.
- Execute job.
- It will try to configure and setup the PL components such as Jenkins/Gitlab and Nexus.



If a restart was needed, you need to trigger the job again!

- The job should now show the required parameters, you only need to change the GITLAB PRIVATE TOKEN that you should have generated in the prerequisite section

Pipeline MTS_Template

This build requires parameters:

GITLAB_USER_PRIVATE_TOKEN	<input type="text" value="GitLab API token (gitlab api token)"/> <input type="button" value="Add"/>
NEEDS TO BE SET! Private Token of a Production Line Gitlab User that can be used to create repos	
GITLAB_CREATE_GROUP_NAME	<input type="text" value="devon"/>
GitLab group name where the repository will be created.	
GITLAB_CREATE_PROJECT_NAME	<input type="text" value="MyThaiStar"/>
GitLab repository name where My Thai Star will be cloned to.	
GITLAB_CREATE_BRANCH	<input type="text" value="master"/>
GitLab repository default branch name.	
GITLAB_CREATE_PROJECT_DESCRIPTION	<input type="text" value="MyThaiStar Sample Devon4J application"/>
GitLab repository description.	
GITLAB_CLONE_URL	<input type="text" value="https://github.com/devonfw/my-thai-star.git"/>
Private Token of a Production Line Gitlab User that can be used to create repositories.	
JENKINS_FOLDER	<input type="text" value="MTS"/>
Folder where all pipelines will be created.	
NEW_DOCKER_HOST	<input type="text" value="tcp://127.0.0.1:2375"/>
Remote docker url	

Build

When everything is "green" the template is done and you can have a look in the created "MTS" folder in Jenkins.



It will take a few minutes to clone the official MTS repository to the internal Gitlab. So you need to wait before executing the build jobs at the first time.

Build Jobs

You can now execute the build for the frontend and also the backend. They do not require any parameters to run. The expected result is, that both jobs can run without any errors. They will build, test and deploy the artifacts to Nexus3.

Deployment Jobs

All deployment jobs have several parameters configured in their Jenkinsfile. Unfortunately, Jenkins does not pick them up immediatly, **so you need to execute the job once, by pressing the "Build now" button**. The run should fail quite fast and once you refresh the page, the "Build now" button should have changed to "Build with Parameters". If you now click on the button you should see the parameters below:

Pipeline MyThaiStar_FRONTEND_DEPLOY

This build requires parameters:

registryUrl	<input type="text" value="https://docker-registry-devon.s2-eu.capgemini.com"/>
	docker registry url
registryCredentialsId	<input checked="" type="checkbox"/> List user credentials <input type="text" value="nexus-api***** (Nexus internal admin user)"/> ▼ Add ▼ registry credentials
VERSION	<input type="text" value="3.2.0"/>
	Version number
dockerNetwork	<input type="text" value="my-thai-star"/>
	The docker network for the deployed container
Build	

You need to set the following parameters in order to get it running:

Table 45. Required Parameters

Parameter	Description
registryUrl	The docker registry URL where image is stored.
registryCredentialsId	The nexus credentials to access to the docker registry.
VERSION	The version of the image that was built in the build jobs. For example "1.12.3-SNAPSHOT".
dockerNetwork	The docker network where the container will be deployed.

Also, the reverse proxy deployment has two more parameters:

Table 46. Reverse Proxy extra parameters

Parameter	Description
buildReverseProxy	If true, it will build a new reverse proxy docker image and then deploy that image.
port	The port where the application will be listening. It's a host port, not a container port.



You can deploy multiple versions of My Thai Star in the same machine by changing the docker network in all deployments and the port in the reverse proxy deployment.



You must choose the same docker network for all deployments



You need to deploy the angular and java applications before the reverse proxy. Also, the first you need to check the `buildReverseProxy` parameter in order to create the reverse proxy image and then deploy the container.

67. Troubleshooting

67.1. Troubleshooting

67.1.1. Introduction

In this section you can find the solution of the most common errors using the templates.

67.1.2. Template startup failed

Sometimes, when you execute any template you will see this an error like:

```
org.codehaus.groovy.control.MultipleCompilationErrorsException: startup failed:  
/home/pl/jobs/devon4j-  
mts_PL_Template/builds/8/libs/ProductionLineTemplateLib/src/com/capgemini/productionline/  
configuration/JenkinsConfiguration.groovy: 38: unable to resolve class  
ru.yandex.qatools.allure.jenkins.tools.AllureCommandlineInstaller @ line 38, column 1. import  
ru.yandex.qatools.allure.jenkins.tools.AllureCommandlineInstaller
```

In most of our templates we use the [Production Line Shared Lib](#). In order to work, the Shared Lib needs some plugins installed in your Jenkins, so to solve this error you need to install those plugins manually using the [Manage Plugins](#).

In this specific case the problem is the [Allure](#) plugin is not installed. Just install it, restart Jenkins and execute again the template.

67.1.3. Build Now instead Build with Parameters

Sometimes, when you go to execute a template, mostly the first time, the [Build Now](#) button is available instead [Build with Parameters](#) button. The root cause of this problem is the parameters are defined in the [Jenkinsfile](#) and, as you never execute it before, Jenkins do not have those [Jenkinsfile](#) yet. For this reason it does not know the parameters required.

To solve this problem, you only need to press the [Build Now](#) button. Then, the execution will start and fail. It's not a problem as you do not enter any parameter. Now you only need to reload the page and the [Build with Parameters](#) button will be available.

67.1.4. Error at Install plugins stage

In some templates you can see the [Install plugins](#) stage. In this stage some plugins required for the template will be installed. In order to properly load the plugins, Jenkins needs to be restarted, for that reason the pipeline fails on that stage. It is not a bug or problem, so do not worry about that. You only need to wait until Jenkins is restarted and execute the template again.

Part XI: CobiGen — Code-based incremental Generator

68. Document Description

This document contains the documentation of the CobiGen core module as well as all CobiGen plugins and the CobiGen eclipse integration.



DISCLAIMER: All Cobigen plugins are compatible with the latest release of Devonfw unless otherwise denoted.

Current versions:

- CobiGen - Eclipse Plug-in v7.0.0
 - CobiGen - Maven Build Plug-in v7.0.0
 - CobiGen CLI v1.2.0
-
- CobiGen v7.0.0
 - CobiGen - Java Plug-in v7.0.0
 - CobiGen - XML Plug-in v7.0.0
 - CobiGen - TypeScript Plug-in v7.0.0
 - CobiGen - Property Plug-in v7.0.0
 - CobiGen - Text Merger v7.0.0
 - CobiGen - JSON Plug-in v7.0.0
 - CobiGen - HTML Plug-in v7.0.0
 - CobiGen - Open API Plug-in v7.0.0
 - CobiGen - FreeMarker Template Engine v7.0.0
 - CobiGen - Velocity Template Engine v7.0.0

Authors:

- Malte Brunnlieb
- Jaime Diaz Gonzalez
- Steffen Holzer
- Ruben Diaz Martinez
- Joerg Hohwiller
- Fabian Kreis
- Lukas Goerlach
- Krati Shah
- Christian Richter
- Erik Grüner

-
- Mike Schumacher
 - Marco Rose
 - Mohamed Ghanmi

68.1. Guide to the Reader

Dependent on the intention you are reading this document, you might be most interested in the following chapters:

- If this is **your first contact with CobiGen**, you will be interested in the [general purpose](#) of CobiGen, in the [licensing of CobiGen](#), as well as in the [Shared Service](#) provided for CobiGen. Additionally, there are some [general use cases](#), which are currently implemented and maintained to be used out of the box.
- As a **user of the CobiGen Eclipse integration**, you should focus on the [Installation](#) and [Usage](#) chapters to get a good introduction how to use CobiGen in eclipse.
- As a **user of the Maven integration**, you should focus on the [Maven configuration](#) chapter, which guides you through the integration of CobiGen into your build configuration.
- If you like to **adapt the configuration of CobiGen**, you have to step deeper into the [configuration guide](#) as well as into the plug-in configuration extensions for the [Java Plug-in](#), [XML-Plugin](#), [Java Property Plug-in](#), as well as for the [Text-Merger Plug-in](#).
- Finally, if want to **develop your own templates**, you will be thankful for [helpful links](#) in addition to the plug-ins documentation as referenced in the previous point.

68.2. CobiGen - Code-based incremental Generator

68.2.1. Overview

CobiGen is a **generic incremental generator** for end to end code generation tasks, mostly used in Java projects. Due to a template-based approach, CobiGen **generates any set of text-based documents and document fragments**.

Input (currently):

- Java classes
- XML-based files
- OpenAPI documents
- Possibly more inputs like WSDL, which is currently not implemented.

Output:

- any text-based document or document fragments specified by templates

68.2.2. Architecture

CobiGen is build as an extensible framework for incremental code generation. It provides extension

points for new input readers which allow reading new input types and converting them to an internally processed model. The model is used to process templates of different kinds to generate patches. The template processing will be done by different template engines. There is an extension point for template engines to support multiple ones as well. Finally, the patch will be structurally merged into potentially already existing code. To allow structural merge on different programming languages, the extension point for structural mergers has been introduced. Here you will see an overview of the currently available extension points and plug-ins:

68.2.3. Features and Characteristics

- Generate fresh files across all the layers of a application - ready to run.
- Add on to existing files merging code into it. E.g. generate new methods into existing java classes or adding nodes to an XML file. Merging of contents into existing files will be done using structural merge mechanisms.
- Structural merge mechanisms are currently implemented for Java, XML, Java Property Syntax, JSON, Basic HTML, Text Append, TypeScript.
- Conflicts can be resolved individually but automatically by former configuration for each template.
- CobiGen provides an [Eclipse integration](#) as well as a [Maven Integration](#).
- CobiGen comes with an extensive documentation for [users](#) and [developers](#).
- Templates can be fully tailored to project needs - this is considered as a simple task.

68.2.4. Selection of current and past CobiGen applications

General applications:

- Generation of a **Java CRUD application based on devonfw architecture** including all software-layers on the server plus code for js-clients (Angular). You can find details [here](#).
- Generation of a **Java CRUD application according to the Register Factory architecture**. Persistence entities are the input for generation.
- Generation of **builder classes for generating test data** for JUnit-Tests. Input are the persistence entities.
- Generation of a **EXT JS 6** client with full CRUD operations connected a devon4j server.
- Generation of a **Angular 6** client with full CRUD operations connected a devon4j server.

Project-specific applications in the past:

- Generation of an **additional Java type hierarchy on top of existing Java classes** in combination with additional methods to be integrated in the modified classes. Hibernate entities were considered as input as well as output of the generation. The rational in this case, was to generate an additional business object hierarchy on top of an existing data model for efficient business processing.
- Generation of **hash- and equals-methods** as well as copy constructors depending on the field types of the input Java class. Furthermore, CobiGen is able to re-generate these

methods/constructors triggered by the user, i.e., when fields have been changed.

- **Extraction of JavaDoc** of test classes and their methods for generating a csv test documentation. This test documentation has been further processed manually in Excel to provide a good overview about the currently available tests in the software system, which enables further human analysis.

68.3. General use cases

In addition to the [selection of CobiGen applications](#) introduced before, this chapter provides a more detailed overview about the currently implemented and maintained general use cases. These can be used by any project following a supported reference architecture as e.g. the [devonfw](#) or [Register Factory](#).

68.3.1. devon4j

With our templates for [devon4j](#), you can generate a whole CRUD application from a single Entity class. You save the effort for creating, DAOs, Transfer Objects, simple CRUD use cases with REST services and even the client application can be generated.

CRUD server application for devon4j

For the server, the required files for all architectural layers (Data access, logic, and service layer) can be created based on your Entity class. After the generation, you have CRUD functionality for the entity from bottom to top which can be accessed via a RESTful web service. Details are provided in the [Devon wiki](#).

CRUD client application for devon4ng

Based on the REST services on the server, you can also generate an [Angular](#) client based on [devon4ng](#). With the help of [Node.js](#), you have a working client application for displaying your entities within minutes!

Testdata Builder for devon4j

Generating a builder pattern for POJOs to easily create test data in your tests. CobiGen is not only able to generate a plain builder pattern but rather builder, which follow a specific concept to minimize test data generation efforts in your unit tests. The following [Person](#) class as an example:

Listing 94. Person class

```
public class Person {

    private String firstname;
    private String lastname;
    private int birthyear;
    @NotNull
    private Address address;

    @NotNull
    public String getFirstname() {
        return this.firstname;
    }

    // additional default setter and getter
}
```

It is a simple POJO with a validation annotation, to indicate, that `firstname` should never be `null`. Creating this object in a test would imply to call every setter, which is kind of nasty. Therefore, the Builder Pattern has been introduced for quite a long time in software engineering, allowing to easily create POJOs with a fluent API. See below.

Listing 95. Builder pattern example

```
Person person = new PersonBuilder()
    .firstname("Heinz")
    .lastname("Erhardt")
    .birthyear(1909)
    .address(
        new AddressBuilder().postcode("22222")
            .city("Hamburg").street("Luebecker Str. 123")
            .createNew())
    .addChild(
        new PersonBuilder()[...].createNew()).createNew();
```

The Builder API generated by CobiGen allows you to set any setter accessible field of a POJO in a fluent way. But in addition lets assume a test, which should check the birth year as precondition for any business operation. So specifying all other fields of `Person`, especially `firstname` as it is mandatory to enter business code, would not make sense. The test behavior should just depend on the specification of the birth year and on no other data. So we would like to just provide this data to the test.

The Builder classes generated by CobiGen try to tackle this inconvenience by providing the ability to declare default values for any mandatory field due to validation or database constraints.

Listing 96. Builder Outline

```

public class PersonBuilder {

    private void fillMandatoryFields() {
        firstname("lasdjfaöskdlfja");
        address(new AddressBuilder().createNew());
    };
    private void fillMandatoryFields_custom() {...};

    public PersonBuilder firstname(String value);
    public PersonBuilder lastname(String value);
    ...

    public Person createNew();
    public Person persist(EntityManager em);
    public List<Person> persistAndDuplicate(EntityManager em, int count);
}

```

Looking at the plotted builder API generated by CobiGen, you will find two `private` methods. The method `fillMandatoryFields` will be generated by CobiGen and regenerated every time CobiGen generation will be triggered for the `Person` class. This method will set every automatically detected field with not `null` constraints to a default value. However, by implementing `fillMandatoryFields_custom` on your own, you can reset these values or even specify more default values for any other field of the object. Thus, running `new PersonBuilder().birthyear(1909).createNew();` will create a valid object of `Person`, which is already pre-filled such that it does not influence the test execution besides the fact that it circumvents database and validation issues.

This even holds for complex data structures as indicated by `address(new AddressBuilder().createNew());`. Due to the use of the `AddressBuilder` for setting the default value for the field `address`, also the default values for `Address` will be set automatically.

Finally, the builder API provides different methods to create new objects.

- `createNew()` just creates a new object from the builder specification and returns it.
- `persist(EntityManager)` will create a new object from the builder specification and persists it to the database.
- `persistAndDuplicate(EntityManager, int)` will create the given amount of objects form the builder specification and persists all of these. After the initial generation of each builder, you might want to adapt the method body as you will most probably not be able to persist more than one object with the same field assignments to the database due to `unique` constraints. Thus, please see the generated comment in the method to adapt `unique` fields accordingly before persisting to the database.

Custom Builder for Business Needs

CobiGen just generates basic builder for any POJO. However, for project needs you probably would like to have even more complex builders, which enable the easy generation of more complex test

data which are encoded in a large object hierarchy. Therefore, the generated builders can just be seen as a tool to achieve this. You can define your own business driven builders in the same way as the generated builders, but explicitly focusing on your business needs. Just take this example as a demonstration of that idea:

```
University uni = new ComplexUniversityBuilder()  
    .withStudents(200)  
    .withProfessors(4)  
    .withExternalStudent()  
    .createNew();
```

E.g. the method `withExternalStudent()` might create a person, which is a student and is flagged to be an external student. Basing this implementation on the generated builders will even assure that you would benefit from any default values you have set before. In addition, you can even imagine any more complex builder methods setting values driven by your reusable testing needs based on the specific business knowledge.

68.3.2. Register Factory

CRUD server application

Generates a CRUD application with persistence entities as inputs. This includes DAOs, TOs, use cases, as well as a CRUD JSF user interface if needed.

Testdata Builder

Analogous to [Testdata Builder for devon4J](#)

Test documentation

Generate test documentation from test classes. The input are the doclet tags of several test classes, which e.g. can specify a description, a cross-reference, or a test target description. The result currently is a csv file, which lists all tests with the corresponding meta-information. Afterwards, this file might be styled and passed to the customer if needed and it will be up-to-date every time!

69. CobiGen

69.1. Configuration

CobiGen will be configured using a configuration folder containing a context configuration, multiple template folders with a templates configuration per template folder, and a number of templates in each template folder. Find some examples [here](#). Thus, a simple folder structure might look like this:

```
CobiGen_Templates
|- templateFolder1
  |- templates.xml
|- templateFolder2
  |- templates.xml
|- context.xml
```

69.1.1. Context Configuration

The context configuration (`context.xml`) always has the following root structure:

Listing 97. Context Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<contextConfiguration xmlns="http://capgemini.com"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      version="1.0">
  <triggers>
    ...
  </triggers>
</contextConfiguration>
```

The context configuration has a `version` attribute, which should match the XSD version the context configuration is an instance of. It should not state the version of the currently released version of CobiGen. This attribute should be maintained by the context configuration developers. If configured correctly, it will provide a better feedback for the user and thus higher user experience. Currently there is only the version v1.0. For further version there will be a changelog later on.

Trigger Node

As children of the `<triggers>` node you can define different triggers. By defining a `<trigger>` you declare a mapping between special inputs and a `templateFolder`, which contains all templates, which are worth to be generated with the given input.

Listing 98. trigger configuration

```
<trigger id="..." type="..." templateFolder="..." inputCharset="UTF-8" >
  ...
</trigger>
```

- The attribute `id` should be unique within an context configuration. It is necessary for efficient internal processing.
- The attribute `type` declares a specific *trigger interpreter*, which might be provided by additional plug-ins. A *trigger interpreter* has to provide an *input reader*, which reads specific inputs and creates a template object model out of it to be processed by the FreeMarker template engine later on. Have a look at the plug-in's documentation of your interest and see, which trigger types and thus inputs are currently supported.
- The attribute `templateFolder` declares the relative path to the template folder, which will be used if the trigger gets activated.
- The attribute `inputCharset` (*optional*) determines the charset to be used for reading any input file.

Matcher Node

A trigger will be activated if its matchers hold the following formula:

$$!(\text{NOT} \mid\mid \cdots \mid\mid \text{NOT}) \&& \text{AND} \&& \cdots \&& \text{AND} \&& \text{OR} \mid\mid \cdots \mid\mid \text{OR}$$

Whereas NOT/AND/OR describes the accumulationType of a *matcher* (see below) and e.g. `NOT` means 'a *matcher* with accumulationType NOT matches a given input'. Thus additionally to an *input reader*, a *trigger interpreter* has to define at least one set of *matchers*, which are satisfiable, to be fully functional. A `<matcher>` node declares a specific characteristics a valid input should have.

Listing 99. Matcher Configuration

```
<matcher type="..." value="..." accumulationType="...">
  ...
</matcher>
```

- The attribute `type` declares a specific type of *matcher*, which has to be provided by the surrounding *trigger interpreter*. Have a look at the plug-in's documentation, which also provides the used trigger type for more information about valid matcher and their functionalities.
- The attribute `value` might contain any information necessary for processing the *matcher's* functionality. Have a look at the relevant plug-in's documentation for more detail.
- The attribute `accumulationType` (*optional*) specifies how the matcher will influence the trigger activation. Valid values are:
 - OR (default): if any matcher of accumulation type OR *matches*, the trigger will be activated as long as there are no further matchers with different accumulation types
 - AND: if any matcher with AND accumulation type does *not match*, the trigger will *not* be activated

- NOT: if any matcher with NOT accumulation type *matches*, the trigger will *not* be activated

VariableAssignment Node

Finally, a `<matcher>` node can have multiple `<variableAssignment>` nodes as children. *Variable assignments* allow to parametrize the generation by additional values, which will be added to the object model for template processing. The variables declared using *variable assignments*, will be made accessible in the templates.xml as well in the object model for template processing via the namespace `variables.*`.

Listing 100. Complete Configuration Pattern

```
<?xml version="1.0" encoding="UTF-8"?>
<contextConfiguration xmlns="http://capgemini.com"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      version="1.0">
    <triggers>
        <trigger id="..." type="..." templateFolder="...">
            <matcher type="..." value="...">
                <variableAssignment type="..." key="..." value="..." />
            </matcher>
        </trigger>
    </triggers>
</contextConfiguration>
```

- The attribute `type` declares the type of *variable assignment* to be processed by the *trigger interpreter* providing plug-in. This attribute enables *variable assignments* with different dynamic value resolutions.
- The attribute `key` declares the namespace under which the resolved value will be accessible later on.
- The attribute `value` might declare a constant value to be assigned or any hint for value resolution done by the *trigger interpreter* providing plug-in. For instance, if `type` is `regex`, then on `value` you will assign the matched group number by the regex (1, 2, 3...)

ContainerMatcher Node

The `<containerMatcher>` node is an additional matcher for matching containers of multiple input objects. Such a container might be a package, which encloses multiple types or--more generic--a model, which encloses multiple elements. A container matcher can be declared side by side with other matchers:

Listing 101. ContainerMatcher Declaration

```

<?xml version="1.0" encoding="UTF-8"?>
<contextConfiguration xmlns="http://capgemini.com"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      version="1.0">
    <triggers>
        <trigger id="..." type="..." templateFolder="..." >
            <containerMatcher type="..." value="..." retrieveObjectsRecursively="..." />
            <matcher type="..." value="..." >
                <variableAssignment type="..." variable="..." value="..." />
            </matcher>
        </trigger>
    </triggers>
</contextConfiguration>

```

- The attribute `type` declares a specific type of *matcher*, which has to be provided by the surrounding *trigger interpreter*. Have a look at the plug-in's documentation, which also provides the used trigger type for more information about valid matcher and their functionalities.
- The attribute `value` might contain any information necessary for processing the *matcher's* functionality. Have a look at the relevant plug-in's documentation for more detail.
- The attribute `retrieveObjectsRecursively` (*optional boolean*) states, whether the children of the input should be retrieved recursively to find matching inputs for generation.

The semantics of a container matchers are the following:

- A `<containerMatcher>` does not declare any `<variableAssignment>` nodes
- A `<containerMatcher>` matches an input if and only if one of its enclosed elements satisfies a set of `<matcher>` nodes of the same `<trigger>`
- Inputs, which match a `<containerMatcher>` will cause a generation for each enclosed element

69.1.2. Templates Configuration

The template configuration (`templates.xml`) specifies, which templates exist and under which circumstances it will be generated. There are two possible configuration styles:

- Configure the template meta-data for each template file by `template nodes`
- (since `cobigen-core-v1.2.0`): Configure `templateScan nodes` to automatically retrieve a default configuration for all files within a configured folder and possibly modify the automatically configured templates using `templateExtension nodes`

To get an intuition of the idea, the following will initially describe the first (more extensive) configuration style. Such an configuration root structure looks as follows:

Listing 102. Extensive Templates Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<templatesConfiguration xmlns="http://capgemini.com"
                           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                           version="1.0" templateEngine="FreeMarker">
    <templates>
        ...
    </templates>
    <increments>
        ...
    </increments>
</templatesConfiguration>
```

The root node `<templatesConfiguration>` specifies two attributes. The attribute `version` provides further usability support and will be handled analogous to the `version` attribute of the `context configuration`. The optional attribute `templateEngine` specifies the template engine to be used for processing the templates (*since cobigen-core-4.0.0*). By default it is set to `FreeMarker`. The node `<templatesConfiguration>` allows two different grouping nodes as children. First, there is the `<templates>` node, which groups all declarations of templates. Second, there is the `<increments>` node, which groups all declarations about increments.

Template Node

The `<templates>` node groups multiple `<template>` declarations, which enables further generation. Each template file should be registered at least once as a template to be considered.

Listing 103. Example Template Configuration

```
<templates>
    <template name="..." destinationPath="..." templateFile="..." mergeStrategy="..."
              targetCharset="..." />
    ...
</templates>
```

A template declaration consist of multiple information:

- The attribute `name` specifies an unique ID within the templates configuration, which will later be reused in the `increment definitions`.
- The attribute `destinationPath` specifies the destination path the template will be generated to. It is possible to use all variables defined by `variable assignments` within the path declaration using the FreeMarker syntax `${variables.*}`. While resolving the variable expressions, each dot within the value will be automatically replaced by a slash. This behavior is accounted for by the transformations of Java packages to paths as CobiGen has first been developed in the context of the Java world. Furthermore, the destination path variable resolution provides the following additional built-in operators analogue to the FreeMarker syntax:
 - `?cap_first` analogue to `FreeMarker`
 - `?uncap_first` analogue to `FreeMarker`

- ?lower_case analogue to FreeMarker
 - ?upper_case analogue to FreeMarker
 - ?replace(regex, replacement) - Replaces all occurrences of the regular expression `regex` in the variable's value with the given `replacement` string. (since cobigen-core v1.1.0)
 - ?removeSuffix(suffix) - Removes the given `suffix` in the variable's value iff the variable's value ends with the given `suffix`. Otherwise nothing will happen. (since cobigen-core v1.1.0)
 - ?removePrefix(prefix) - Analogue to `?removeSuffix` but removes the prefix of the variable's value. (since cobigen-core v1.1.0)
- The attribute `templateFile` describes the relative path dependent on the template folder specified in the `trigger` to the template file to be generated.
 - The attribute `mergeStrategy` (*optional*) can be *optionally* specified and declares the type of merge mechanism to be used, when the `destinationPath` points to an already existing file. CobiGen by itself just comes with a `mergeStrategy override`, which enforces file regeneration in total. Additional available merge strategies have to be obtained from the different plug-in's documentations (see here for `java`, `XML`, `properties`, and `text`). Default: *not set* (means not mergeable)
 - The attribute `targetCharset` (*optional*) can be *optionally* specified and declares the encoding with which the contents will be written into the destination file. This also includes reading an existing file at the destination path for merging its contents with the newly generated ones. Default: `UTF-8`

(Since version 4.1.0) It is possible to reference external `template` (templates defined on another trigger), thanks to using `<incrementRef ...>` that are explained [here](#).

TemplateScan Node

(since cobigen-core-v1.2.0)

The second configuration style for template meta-data is driven by initially scanning all available templates and automatically configure them with a default set of meta-data. A scanning configuration might look like this:

Listing 104. Example of Template-scan configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<templatesConfiguration xmlns="http://capgemini.com"
                         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                         version="1.2">
    <templateScans>
        <templateScan templatePath="templates" templateNamePrefix="prefix_"
                      destinationPath="src/main/java"/>
    </templateScans>
</templatesConfiguration>
```

You can specify multiple `<templateScan ...>` nodes for different `templatePaths` and different `templateNamePrefixes`.

- The `name` can be specified to later on reference the templates found by a template-scan within an `increment`. (since cobigen-core-v2.1.)
- The `templatePath` specifies the relative path from the `templates.xml` to the root folder from which the template scan should be performed.
- The `templateNamePrefix` (*optional*) defines a common id prefix, which will be added to all found and automatically configured templates.
- The `destinationPath` defines the root folder all found templates should be generated to, whereas the root folder will be a prefix for all found and automatically configured templates.

A `templateScan` will result in the following **default configuration of templates**. For each file found, new `template` will be created virtually with the following default values:

- `id`: file name without `.ftl` extension prefixed by `templateNamePrefix` from `template-scan`
- `destinationPath`: relative file path of the file found with the prefix defined by `destinationPath` from `template-scan`. Furthermore,
 - it is possible to use the syntax for accessing and modifying variables as described for the attribute `destinationPath` of the `template node`, besides the only difference, that due to file system restrictions you have to replace all `?-signs` (for built-ins) with `#-signs`.
 - the files to be scanned, should provide their final file extension by the following file naming convention: `<filename>.<extension>.ftl` Thus the file extension `.ftl` will be removed after generation.
- `templateFile`: relative path to the file found
- `mergeStrategy`: (*optional*) not set means not mergeable
- `targetCharset`: (*optional*) defaults to UTF-8

(Since version 4.1.0) It is possible to reference external `templateScan` (`templateScans` defined on another trigger), thanks to using `<incrementRef ...>` that are explained [here](#).

TemplateExtension Node

(since cobigen-core-v1.2.0)

Additionally to the `templateScan declaration` it is easily possible to rewrite specific attributes for any scanned and automatically configured template.

Listing 105. Example Configuration of a TemplateExtension

```

<templates>
    <templateExtension ref="prefix_FooClass.java" mergeStrategy="javamerge" />
</templates>

<templateScans>
    <templateScan templatePath="foo" templateNamePrefix="prefix_" destinationPath=
    "src/main/java/foo"/>
</templateScans>

```

Lets assume, that the above example declares a `template-scan` for the folder `foo`, which contains a file `FooClass.java.ftl` in any folder depth. Thus the template scan will automatically create a virtual `template` declaration with `id=prefix_FooClass.java` and further `default configuration`.

Using the `templateExtension` declaration above will reference the scanned template by the attribute `ref` and overrides the `mergeStrategy` of the automatically configured template by the value `javamerge`. Thus we are able to minimize the needed templates configuration.

(Since version 4.1.0) It is possible to reference external `templateExtension` (`templateExtensions` defined on another trigger), thanks to using `<incrementRef ...>` that are explained [here](#).

Increment Node

The `<increments>` node groups multiple `<increment>` nodes, which can be seen as a collection of templates to be generated. An increment will be defined by a unique `id` and a human readable `description`.

```
<increments>
  <increment id="..." description="...">
    <incrementRef ref="..." />
    <templateRef ref="..." />
    <templateScanRef ref="..." />
  </increment>
</increments>
```

An increment might contain multiple increments and/or templates, which will be referenced using `<incrementRef ...>`, `<templateRef ...>`, resp. `<templateScanRef ...>` nodes. These nodes only declare the attribute `ref`, which will reference an increment, a template, or a template-scan by its `id` or `name`.

(Since version 4.1.0) A special case of `<incrementRef ...>` is the external incrementsRef. By default, `<incrementRef ...>` are used to reference increments defined in the same `templates.xml` file. So for example, we could have:

```
<increments>
  <increment id="incA" description="...">
    <incrementRef ref="incB" />
  </increment>
  <increment id="incB" description="...">
    <templateRef .... />
    <templateScan .... />
  </increment>
</increments>
```

However, if we want to reference an increment that it is not defined inside our `templates.xml` (an increment defined for another trigger), then we can use external incrementRef as shown below:

```
<increment name="..." description="...>
  <incrementRef ref="trigger_id::increment_id"/>
</increment>
```

The ref string is split using as delimiter `::`. The first part of the string, is the `trigger_id` to reference. That trigger contains an `increment_id`. Currently, this functionality only works when both templates use the same kind of input file.

69.1.3. Java Template Logic

since cobigen-core-3.0.0 which is included in the Eclipse and Maven Plugin since version 2.0.0 In addition, it is possible to implement more complex template logic by custom Java code. To enable this feature, you can simply import the the [CobiGen_Templates](#) by clicking on *Adapt Templates*, turn it into a simple maven project (if it is not already) and implement any Java logic in the common maven layout (e.g. in the source folder `src/main/java`). Each Java class will be instantiated by CobiGen for each generation process. Thus, you can even store any state within a Java class instance during generation. However, there is currently no guarantee according to the template processing order.

As a consequence, you have to implement your Java classes with a public default (non-parameter) constructor to be used by any template. Methods of the implemented Java classes can be called within templates by the simple standard FreeMarker expression for calling Bean methods: `SimpleType.methodName(param1)`. Until now, CobiGen will shadow multiple types with the same simple name non-deterministically. So please prevent yourself from that situation.

Finally, if you would like to do some reflection within your Java code accessing any type of the template project or any type referenced by the input, you should load classes by making use of the classloader of the util classes. CobiGen will take care of the correct classloader building including the classpath of the input source as well as of the classpath of the template project. If you use any other classloader or build it by your own, there will be no guarantee, that generation succeeds.

69.1.4. Template Properties

since cobigen-core-4.0.0 Using a configuration with [template scan](#), you can make use of properties in templates specified in property files named `cobigen.properties` next to the templates. The property files are specified as [Java property files](#). Property files can be nested in subfolders. Properties will be resolved including property shading. Properties defined nearest to the template to be generated will take precedence. In addition, a `cobigen.properties` file can be specified in the target folder root (in eclipse plugin, this is equal to the source project root). These properties take precedence over template properties specified in the template folder.



It is not allowed to override context variables in `cobigen.properties` specifications as we have not found any interesting use case. This is most probably an error of the template designer, CobiGen will raise an error in this case.

Multi module support or template target path redirects

since cobigen-core-4.0.0 One special property you can specify in the template properties is the property `relocate`. It will cause the current folder and its subfolders to be relocated at destination path resolution time. Take the following example:

```
folder
- sub1
  Template.java.ftl
  cobigen.properties
```

Let the `cobigen.properties` file contain the line `relocate=../sub2/${cwd}`. Given that, the relative destination path of `Template.java.ftl` will be resolved to `folder/sub2/Template.java`. Compare `template scan` configuration for more information about basic path resolution. The `${cwd}` placeholder will contain the remaining relative path from the `cobigen.properties` location to the template file. In this basic example it just contains `Template.java.ftl`, but it may even be any relative path including subfolders of `sub1` and its templates. Given the `relocate` feature, you can even step out of the root path, which in general is the project/maven module the input is located in. This enables template designers to even address, e.g., maven modules located next to the module the input is coming from.

69.1.5. Basic Template Model

In addition to what is served by the different model builders of the different plug-ins, CobiGen provides a minimal model based on context variables as well as CobiGen properties. The following model is independent of the input format and will be served as a template model all the time:

- variables
 - all triggered `context variables` mapped to its assigned/mapped value
 - all `template properties`
- all simple names of `Java template logic` implementation classes
- all full qualified names of `Java template logic` implementation classes
- further input related model, e.g. `model from Java inputs`

69.1.6. Plugin Mechanism

Since cobigen-core 4.1.0, we changed the plug-in discovery mechanism. So far it was necessary to register new plugins programmatically, which introduces the need to let every tool integration, i.e. for eclipse or maven, be dependent on every plug-in, which should be released. This made release cycles take long time as all plug-ins have to be integrated into a final release of maven or eclipse integration.

Now, plug-ins are automatically discovered by the Java `Service Loader` mechanism from the classpath. This also effects the setup of `eclipse` and `maven` integrations to allow modular releases of CobiGen in future. We are now able to provide faster rollouts of bug-fixes in any of the plug-ins as

they can be released completely independently.

69.2. Plug-ins

69.2.1. Java Plug-in

The CobiGen Java Plug-in comes with a new input reader for java artifacts, new java related trigger and matchers, as well as a merging mechanism for Java sources.

Trigger extension

The Java Plug-in provides a new trigger for Java related inputs. It accepts different representations as inputs (see [Java input reader](#)) and provides additional matching and variable assignment mechanisms. The configuration in the `context.xml` for this trigger looks like this:

- type 'java'

Listing 106. Example of a java trigger definition

```
<trigger id="..." type="java" templateFolder="..."><br/>    ...<br/></trigger>
```

This trigger type enables Java elements as inputs.

Matcher types

With the trigger you might define matchers, which restrict the input upon specific aspects:

- type 'fqn' → full qualified name matching

Listing 107. Example of a java trigger definition with a full qualified name matcher

```
<trigger id="..." type="java" templateFolder="...">
    <matcher type="fqn" value="(.)\..persistence\.([^\.]+)\..entity\.([^\.]+)">
        ...
    </matcher>
</trigger>
```

This trigger will be enabled if the full qualified name (`fqn`) of the declaring input class matches the given regular expression (`value`).

- type 'package' → package name of the input

Listing 108. Example of a java trigger definition with a package name matcher

```
<trigger id="..." type="java" templateFolder="...">
  <matcher type="package" value="(.)\persistence\.(^\.)+\.\entity">
  ...
  </matcher>
</trigger>
```

This trigger will be enabled if the package name (**package**) of the declaring input class matches the given regular expression (**value**).

- type 'expression'

Listing 109. Example of a java trigger definition with a package name matcher

```
<trigger id="..." type="java" templateFolder="...">
  <matcher type="expression" value="instanceof java.lang.String">
  ...
  </matcher>
</trigger>
```

This trigger will be enabled if the expression evaluates to true. Valid expressions are

- **instanceof fqn**: checks an 'is a' relation of the input type
- **isAbstract**: checks, whether the input type is declared abstract

ContainerMatcher types

Additionally, the java plugin provides the ability to match packages (containers) as follows:

- type 'package'

Listing 110. Example of a java trigger definition with a container matcher for packages

```
<trigger id="..." type="java" templateFolder="...">
  <containerMatcher type="package" value=
    "com\example\app\component1\persistence.entity" />
</trigger>
```

The container matcher matches packages provided by the type **com.cagemini.cobigen.javaplugin.inputreader.to.PackageFolder** with a regular expression stated in the **value** attribute. (See [ContainerMatcher semantics](#) to get more information about containerMatchers itself.)

VariableAssignment types

Furthermore, it provides the ability to extract information from each input for further processing in the templates. The values assigned by variable assignments will be made available in template and the **destinationPath** of context.xml through the namespace **variables.<key>**. The Java Plug-in

currently provides two different mechanisms:

- type 'regex' → regular expression group

```
<trigger id="..." type="java" templateFolder="...">
    <matcher type="fqn" value="(.)\persistence\.(^\.)+\.\entity\.(^\.)+">
        <variableAssignment type="regex" key="rootPackage" value="1" />
        <variableAssignment type="regex" key="component" value="2" />
        <variableAssignment type="regex" key="pojoName" value="3" />
    </matcher>
</trigger>
```

This variable assignment assigns the value of the given regular expression group number to the given **key**.

- type 'constant' → constant parameter

```
<trigger id="..." type="java" templateFolder="...">
    <matcher type="fqn" value="(.)\persistence\.(^\.)+\.\entity\.(^\.)+">
        <variableAssignment type="constant" key="domain" value="restaurant" />
    </matcher>
</trigger>
```

This variable assignment assigns the **value** to the **key** as a constant.

Java input reader

The Cobigen Java Plug-in implements an input reader for parsed java sources as well as for java **Class<?>** objects (loaded by reflection). So API user can pass **Class<?>** objects as well as **JavaClass** objects for generation. The latter depends on **QDox**, which will be used for parsing and merging java sources. For getting the right parsed java inputs you can easily use the **JavaParserUtil**, which provides static functionality to parse java files and get the appropriate **JavaClass** object.

Furthermore, due to restrictions on both inputs according to model building (see below), it is also possible to provide an array of length two as an input, which contains the **Class<?>** as well as the **JavaClass** object of the same class.

Template object model

No matter whether you use reflection objects or parsed java classes as input, you will get the following object model for template creation:

- **classObject** ('Class' :: Class object of the Java input)
- **pojo**
 - **name** ('String' :: Simple name of the input class)
 - **package** ('String' :: Package name of the input class)

- **canonicalName** ('String' :: Full qualified name of the input class)
- **annotations** ('Map<String, Object>' :: Annotations, which will be represented by a mapping of the full qualified type of an annotation to its value. To gain template compatibility, the key will be stored with '_' instead of '.' in the full qualified annotation type. Furthermore, the annotation might be recursively defined and thus be accessed using the same type of mapping. Example `#{pojo.annotations.java_persistence_Id}`)
- **javaDoc** ('Map<String, Object>') :: A generic way of addressing all available javaDoc doclets and comments. The only fixed variable is **comment** (see below). All other provided variables depend on the doclets found while parsing. The value of a doclet can be accessed by the doclets name (e.g. `#{...javaDoc.author}`). In case of doclet tags that can be declared multiple times (currently **@param** and **@throws**), you will get a map, which you access in a specific way (see below).
 - **comment** ('String' :: javaDoc comment, which does not include any doclets)
 - **params** ('Map<String, String>' :: javaDoc parameter info. If the comment follows proper conventions, the key will be the name of the parameter and the value being its description. You can also access the parameters by their number, as in **arg0**, **arg1** etc, following the order of declaration in the signature, not in order of javadoc)
 - **throws** ('Map<String, String>' :: javaDoc exception info. If the comment follows proper conventions, the key will be the name of the thrown exception and the value being its description)
- **extendedType** ('Map<String, Object>' :: The supertype, represented by a set of mappings (*since cobigen-javaplugin v1.1.0*)
 - **name** ('String' :: Simple name of the supertype)
 - **canonicalName** ('String' :: Full qualified name of the supertype)
 - **package** ('String' :: Package name of the supertype)
- **implementedTypes** ('List<Map<String, Object>>' :: A list of all implementedTypes (interfaces) represented by a set of mappings (*since cobigen-javaplugin v1.1.0*)
 - **interface** ('Map<String, Object>' :: List element)
 - **name** ('String' :: Simple name of the interface)
 - **canonicalName** ('String' :: Full qualified name of the interface)
 - **package** ('String' :: Package name of the interface)
- **fields** ('List<Map<String, Object>>' :: List of fields of the input class) (*renamed since cobigen-javaplugin v1.2.0; previously **attributes***)
 - **field** ('Map<String, Object>' :: List element)
 - **name** ('String' :: Name of the Java field)
 - **type** ('String' :: Type of the Java field)
 - **canonicalType** ('String' :: Full qualified type declaration of the Java field's type)
 - **'isId'** ('Deprecated' :: 'boolean' :: true if the Java field or its setter or its getter is annotated with the javax.persistence.Id annotation, false otherwise. Equivalent to `#{pojo.attributes[i].annotations.java_persistence_Id?has_content}`)

- **javaDoc** (see pojo.javaDoc)
- **annotations** (see pojo.annotations with the remark, that for fields all annotations of its setter and getter will also be collected)
- **methodAccessibleFields** ('List<Map<String, Object>>' :: List of fields of the input class or its inherited classes, which are accessible using setter and getter methods)
 - same as for **field** (but without javaDoc!)
- **methods** ('List<Map<String, Object>>' :: The list of all methods, whereas one method will be represented by a set of property mappings)
 - **method** ('Map<String, Object>' :: List element)
 - **name** ('String' :: Name of the method)
 - **javaDoc** (see pojo.javaDoc)
 - **annotations** (see pojo.annotations)

Furthermore, when providing a `Class<?>` object as input, the Java Plug-in will provide additional functionalities as template methods (*deprecated*):

1. **isAbstract(String fqn)** (Checks whether the type with the given full qualified name is an abstract class. Returns a boolean value.) (*since cobigen-javaplugin v1.1.1*) (*deprecated*)
2. **isSubtypeOf(String subType, String superType)** (Checks whether the `subType` declared by its full qualified name is a sub type of the `superType` declared by its full qualified name. Equals the Java expression `subType instanceof superType` and so also returns a boolean value.) (*since cobigen-javaplugin v1.1.1*) (*deprecated*)

Model Restrictions

As stated before both inputs (`Class<?>` objects and `JavaClass` objects) have their restrictions according to model building. In the following these restrictions are listed for both models, the ParsedJava Model which results from an `JavaClass` input and the ReflectedJava Model, which results from a `Class<?>`` input.

It is important to understand, that these restrictions are only present if you work with either Parsed Model **OR** the Reflected Model. If you use the *Maven Build Plug-in* or *Eclipse Plug-in* these two models are merged together so that they can mutually compensate their weaknesses.

Parsed Model

- annotations of the input's supertype are not accessible due to restrictions in the `QDox` library. So `pojo.methodAccessibleFields[i].annotations` will always be empty for super type fields.
- annotations' parameter values are available as Strings only (e.g. the Boolean value `true` is transformed into "`true`"). This also holds for the Reflected Model.
- fields of "supersupertypes" of the input `JavaClass` are not available at all. So `pojo.methodAccessibleFields` will only contain the input type's and the direct superclass's fields.
- [resolved, since cobigen-javaplugin 1.3.1] field types of supertypes are always canonical. So `pojo.methodAccessibleFields[i].type` will always provide the same value as

`pojo.methodAccessibleFields[i].canonicalType` (e.g. `java.lang.String` instead of the expected `String`) for super type fields.

Reflected Model

- annotations' parameter values are available as Strings only (e.g. the Boolean value `true` is transformed into "`true`"). This also holds for the Parsed Model.
- annotations are only available if the respective annotation has `@Retention(value=RUNTIME)`, otherwise the annotations are to be discarded by the compiler or by the VM at run time. For more information see [RetentionPolicy](#).
- information about generic types is lost. E.g. a field's/ `methodAccessibleField`'s type for `List<String>` can only be provided as `List<?>`.

Merger extensions

The Java Plug-in provides two additional merging strategies for Java sources, which can be configured in the `templates.xml`:

- Merge strategy `javamerge` (merges two Java resources and keeps the existing Java elements on conflicts)
- Merge strategy `javamerge_override` (merges two Java resources and overrides the existing Java elements on conflicts)

In general merging of two Java sources will be processed as follows:

Precondition of processing a merge of generated contents and existing ones is a common Java root class resp. surrounding class. If this is the case this class and all further inner classes will be merged recursively. Therefore, the following Java elements will be merged and conflicts will be resolved according to the configured merge strategy:

- `extends` and `implements` relations of a class: Conflicts can only occur for the `extends` relation.
- Annotations of a class: Conflicted if an annotation declaration already exists.
- Fields of a class: Conflicted if there is already a field with the same name in the existing sources. (Will be replaced / ignored in total, also including annotations)
- Methods of a class: Conflicted if there is already a method with the same signature in the existing sources. (Will be replaced / ignored in total, also including annotations)

69.2.2. Property Plug-in

The CobiGen Property Plug-in currently only provides different merge mechanisms for documents written in [Java property syntax](#).

Merger extensions

There are two merge strategies for Java properties, which can be configured in the `templates.xml`:

- Merge strategy `propertymerge` (merges two properties documents and keeps the existing properties on conflicts)

- Merge strategy `propertymerge_override` (merges two properties documents and overrides the existing properties on conflicts)

Both documents (base and patch) will be parsed using the [Java 7 API](#) and will be compared according their keys. Conflicts will occur if a key in the patch already exists in the base document.

69.2.3. XML Plug-in

The CobiGen XML Plug-in comes with an input reader for xml artifacts, xml related trigger and matchers and provides different merge mechanisms for XML result documents.

Trigger extension

(since `cobigen-xmlplugin v2.0.0`)

The XML Plug-in provides a trigger for xml related inputs. It accepts xml documents as input (see [XML input reader](#)) and provides additional matching and variable assignment mechanisms. The configuration in the `context.xml` for this trigger looks like this:

- type 'xml'

Listing 111. Example of a xml trigger definition.

```
<trigger id="..." type="xml" templateFolder="...>
  ...
</trigger>
```

This trigger type enables xml documents as inputs.

- type 'xpath'

Listing 112. Example of a xpath trigger definition.

```
<trigger id="..." type="xpath" templateFolder="...>
  ...
</trigger>
```

This trigger type enables xml documents as container inputs, which consists of several subdocuments.

ContainerMatcher type

A ContainerMatcher check if the input is a valid container.

- xpath: type: 'xpath'

Listing 113. Example of a xml trigger definition with a nodename matcher.

```
<trigger id="..." type="xml" templateFolder="...">
  <containerMatcher type="xpath" value=
    "./uml:Model//packagedElement[@xmi:type='uml:Class']">
    ...
  </matcher>
</trigger>
```

Before applying any Matcher, this containerMatcher checks if the XML file contains a node "uml:Model" with a childnode "packagedElement" which contains an attribute "xmi:type" with the value "uml:Class".

Matcher types

With the trigger you might define matchers, which restrict the input upon specific aspects:

- xml: type 'nodename' → document's root name matching

Listing 114. Example of a xml trigger definition with a nodename matcher

```
<trigger id="..." type="xml" templateFolder="...">
  <matcher type="nodename" value="\D\w*">
    ...
  </matcher>
</trigger>
```

This trigger will be enabled if the root name of the declaring input document matches the given regular expression (**value**).

- xpath: type: 'xpath' → matching a node with a xpath value

Listing 115. Example of a xpath trigger definition with a xpath matcher.

```
<trigger id="..." type="xml" templateFolder="...">
  <matcher type="xpath" value="/packagedElement[@xmi:type='uml:Class']">
    ...
  </matcher>
</trigger>
```

This trigger will be enabled if the XML file contains a node "/packagedElement" where the "xmi:type" property equals "uml:Class".

VariableAssignment types

Furthermore, it provides the ability to extract information from each input for further processing in the templates. The values assigned by variable assignments will be made available in template and the **destinationPath** of context.xml through the namespace **variables.<key>**. The XML Plug-in currently provides only one mechanism:

- type 'constant' → constant parameter

```
<trigger id="..." type="xml" templateFolder="...">
    <matcher type="nodename" value="\D\w*">
        <variableAssignment type="constant" key="domain" value="restaurant" />
    </matcher>
</trigger>
```

This variable assignment assigns the **value** to the **key** as a constant.

XML input reader

The Cobigen XML Plug-in implements an input reader for parsed xml documents. So API user can pass `org.w3c.dom.Document` objects for generation. For getting the right parsed xml inputs you can easily use the `xmlplugin.util.XmlUtil`, which provides static functionality to parse xml files or input streams and get the appropriate `Document` object.

Template object

Due to the heterogeneous structure an xml document can have, the xml input reader does not always create exactly the same model structure (in contrast to the java input reader). For example the model's depth differs strongly, according to it's input document. To allow navigational access to the nodes, the model also depends on the document's element's node names. All child elements with unique names, are directly accessible via their names. In addition it is possible to iterate over all child elements with help of the child list `Children`. So it is also possible to access child elements with non unique names.

The XML input reader will create the following object model for template creation (`EXAMPLEROOT`, `EXAMPLENODE1`, `EXAMPLENODE2`, `EXAMPLEATTR1`, ...) are just used here as examples. Of course they will be replaced later by the actual node or attribute names):

- ~**EXAMPLEROOT**~ ('Map<String, Object>' :: common element structure)
 - **_nodeName_** ('String' :: Simple name of the root node)
 - **_text_** ('String' :: Concatenated text content (PCDATA) of the root node)
 - **TextNodes** ('List<String>' :: List of all the root's text node contents)
 - **_at_~EXAMPLEATTR1~** ('String' :: String representation of the attribute's value)
 - **_at_~EXAMPLEATTR2~** ('String' :: String representation of the attribute's value)
 - **_at_...**
 - **Attributes** ('List<Map<String, Object>>' :: List of the root's attributes)
 - at ('Map<String, Object>' :: List element)
 - **_attName_** ('String' :: Name of the attribute)
 - **_attValue_** ('String' :: String representation of the attribute's value)
 - **Children** ('List<Map<String, Object>>' :: List of the root's child elements)

- child ('Map<String, Object>' :: List element)
 - ...common element sub structure...
- ~EXAMPLENODE1~ ('Map<String, Object>' :: One of the root's child nodes)
 - ...common element structure...
- ~EXAMPLENODE2~ ('Map<String, Object>' :: One of the root's child nodes)
 - ...common element sub structure...
- ~EXAMPLENODE21~ ('Map<String, Object>' :: One of the nodes' child nodes)
 - ...common element structure...
- ~EXAMPLENODE...~
- ~EXAMPLENODE...~

In contrast to the java input reader, this xml input reader does currently not provide any additional template methods.

Merger extensions

The XML plugin uses the [LeXeMe](#) merger library to produce semantically correct merge products. The merge strategies can be found in the [MergeType enum](#) and can be configured in the [templates.xml](#) as a mergeStrategy attribute:

- mergeStrategy 'xmlmerge'

Listing 116. Example of a template using the mergeStrategy `xmlmerge`

```
<templates>
  <template name="..." destinationPath="..." templateFile="..." mergeStrategy=
    "xmlmerge"/>
</templates>
```

Currently only the document types included in LeXeMe are supported. On how the merger works consult the [LeXeMe Wiki](#).

69.2.4. Text Merger Plug-in

The Text Merger Plug-in enables merging result free text documents to existing free text documents. Therefore, the algorithms are also very rudimentary.

Merger extensions

There are currently three main merge strategies that apply for the whole document:

- merge strategy `textmerge_append` (appends the text directly to the end of the existing document) _Remark_: If no anchors are defined, this will simply append the patch.
- merge strategy `textmerge_appendWithNewLine` (appends the text after adding a new line break to the existing document) _Remark_: empty patches will not result in appending a new line any

more since v1.0.1 *Remark:* Only suitable if no anchors are defined, otherwise it will simply act as `textmerge_append`

- merge strategy `textmerge_override` (replaces the contents of the existing file with the patch)
Remark: If anchors are defined, `override` is set as the default mergestrategy for every text block if not redefined in an anchor specification.

Anchor functionality

If a template contains text that fits the definition of `anchor:${documentpart}: ${mergestrategy}:anchorend` or more specifically the regular expression `(.*)anchor:([:]+)(newline)?([:]+)(newline)?::anchorend\\s*(\\r\\n|\\r|\\n)`, some additional functionality becomes available about specific parts of the incoming text and the way it will be merged with the existing text. These anchors always change things about the text to come up until the next anchor, text before it is ignored.

If no anchors are defined, the complete patch will be appended depending on your choice for the template in the file `templates.xml`.

Anchor Definition

Anchors should always be defined as a comment of the language the template results in, as you do not want them to appear in your readable version, but cannot define them as freemarker comments in the template, or the merger will not know about them. Anchors will also be read when they are not comments due to the merger being able to merge multiple types of text-based languages, thus making it practically impossible to filter for the correct comment declaration. **That is why anchors have to always be followed by line breaks.** That way there is a universal way to filter anchors that should have anchor functionality and ones that should appear in the text. *Remark:* If the resulting language has closing tags for comments, they have to appear in the next line. *Remark:* If you do not put the anchor into a new line, all the text that appears before it will be added to the anchor.

Documentparts

In general, `${documentpart}` is an id to mark a part of the document, that way the merger knows what parts of the text to merge with which parts of the patch (e.g. if the existing text contains `anchor:table:${}:anchorend` that part will be merged with the part tagged `anchor:table:${}:anchorend` of the patch).

If the same documentpart is defined multiple times, it can lead to errors, so instead of defining `table` multiple times, use `table1, table2, table3` etc.

If a `${documentpart}` is defined in the document but not in the patch and they are in the same position, it is processed in the following way: If only the documentparts `header`, `test` and `footer` are defined in the document in that order, and the patch contains `header`, `order` and `footer`, the resulting order will be `header, test, order` then `footer`.

The following documentparts have default functionality:

1. `anchor:header:${mergestrategy}:anchorend` marks the beginning of a header, that will be added once when the document is created, but not again. *Remark:* This is only done once, if you have

`header` in another anchor, it will be ignored

2. `anchor:footer:${mergestrategy}:anchorend` marks the beginning of a footer, that will be added once when the document is created, but not again. Once this is invoked, all following text will be included in the footer, including other anchors.

Mergestrategies

Mergestrategies are only relevant in the patch, as the merger is only interested in how text in the patch should be managed, not how it was managed in the past.

1. `anchor:${documentpart}::anchorend` will use the merge strategy from templates.xml, see [Merger-Extensions](#).
2. `anchor:${}:${mergestrategy}_newline:anchorend` or
`anchor:${}:newline_${mergestrategy}:anchorend` states that a new line should be appended before or after this anchors text, depending on where the newline is (before or after the mergestrategy). `anchor:${documentpart}:newline:anchorend` puts a new line after the anchors text. *Remark:* Only works with appending strategies, not merging/replacing ones. These strategies currently include: `appendbefore`, `append/appendafter`
3. `anchor:${documentpart}:override:anchorend` means that the new text of this documentpart will replace the existing one completely
4. `anchor:${documentpart}:appendbefore:anchorend` or
`anchor:${documentpart}:appendafter:anchorend/anchor:${documentpart}:append:anchorend` specifies whether the text of the patch should come before the existing text or after.

Usage Examples

General

Below you can see how a file with anchors might look like (using Asciidoc comment tags), with examples of what you might want to use the different functions for.

```
// anchor:header:append:anchorend

Table of contents
Introduction/Header

// anchor:part1:appendafter:anchorend

Lists
Table entries

// anchor:part2:nomerge:anchorend

Document Separators
Asciidoc table definitions

// anchor:part3:override:anchorend

Anything that you only want once but changes from time to time

// anchor:footer:append:anchorend

Copyright Info
Imprint
```

Merging

In this section you will see a comparison on what files look like before and after merging

override

Listing 117. Before

```
// anchor:part:override:anchorend
Lorem Ipsum
```

Listing 118. Patch

```
// anchor:part:override:anchorend
Dolor Sit
```

Listing 119. After

```
// anchor:part:override:anchorend
Dolor Sit
```

Appending

Listing 120. Before

```
// anchor:part:append:anchorend
Lorem Ipsum
// anchor:part2:appendafter:anchorend
Lorem Ipsum
// anchor:part3:appendbefore:anchorend
Lorem Ipsum
```

Listing 121. Patch

```
// anchor:part:append:anchorend
Dolor Sit
// anchor:part2:appendafter:anchorend
Dolor Sit
// anchor:part3:appendbefore:anchorend
Dolor Sit
```

Listing 122. After

```
// anchor:part:append:anchorend
Lorem Ipsum
Dolor Sit
// anchor:part2:appendafter:anchorend
Lorem Ipsum
Dolor Sit
// anchor:part3:appendbefore:anchorend
Dolor Sit
Lorem Ipsum
```

Newline*Listing 123. Before*

```
// anchor:part:newline_append:anchorend
Lorem Ipsum
// anchor:part:append_newline:anchorend
Lorem Ipsum
(end of file)
```

Listing 124. Patch

```
// anchor:part:newline_append:anchorend
Dolor Sit
// anchor:part:append_newline:anchorend
Dolor Sit
(end of file)
```

Listing 125. After

```
// anchor:part:newline_append:anchorend
Lorem Ipsum

Dolor Sit
// anchor:part:append_newline:anchorend
Lorem Ipsum
Dolor Sit

(end of file)
```

Error List

- If there are anchors in the text, but either base or patch do not start with one, the merging process will be aborted, as text might go missing this way.
- Using `_newline` or `newline_` with mergestrategies that don't support it , like `override`, will abort the merging process. See [Merge Strategies](#) → 2 for details.
- Using undefined mergestrategies will abort the merging process.
- Wrong anchor definitions, for example `anchor:${{}:anchorend` will abort the merging process, see [Anchor Definition](#) for details.

69.2.5. JSON Plug-in

At the moment the plug-in can be used for merge generic JSON files depending on the merge strategy defined at the templates.

Merger extensions

There are currently these merge strategies:

Generic JSON Merge

- merge strategy `jsonmerge`(add the new code respecting the existent in case of conflict)
- merge strategy `jsonmerge_override` (add the new code overwriting the existent in case of conflict)
 1. JSONArray's will be ignored / replaced in total
 2. JSONObject's in conflict will be processed recursively ignoring adding non existent elements.

Merge Process**Generic JSON Merging**

The merge process will be:

1. Add non existent JSON Objects from patch file to base file.
2. For existent object in both files, will add non existent keys from patch to base object. This

process will be done recursively for all existent objects.

3. For Json Arrays existent in both files, the arrays will be just concatenated.

69.2.6. TypeScript Plug-in

The TypeScript Plug-in enables merging result TS files to existing ones. This plug-in is used at the moment for generate an Angular2 client with all CRUD functionalities enabled. The plug-in also generates de i18n functionality just appending at the end of the word the ES or EN suffixes, to put into the developer knowledge that this words must been translated to the correspondent language. Currently, the generation of Angular2 client requires an ETO java object as input so, there is no need to implement an input reader for ts artifacts for the moment.

Trigger Extensions

As for the Angular2 generation the input is a java object, the trigger expressions (including matchers and variable assignments) are implemented as [Java](#).

Merger extensions

This plugin uses the [OASP TypeScript Merger](#) to merge files. There are currently two merge strategies:

- merge strategy `tsmerge` (add the new code respecting the existing in case of conflict)
- merge strategy `tsmerge_override` (add the new code overwriting the existent in case of conflict)

The merge algorithm mainly handles the following AST nodes:

- **ImportDeclaration**

- Will add non existent imports whatever the merge strategy is.
- For different imports from same module, the import clauses will be merged.

```
import { a } from 'b';
import { c } from 'b';
//Result
import { a, c } from 'b';
```

- **ClassDeclaration**

- Adds non existent base properties from patch based on the name property.
- Adds non existent base methods from patch based on the name signature.
- Adds non existent annotations to class, properties and methods.

- **PropertyDeclaration**

- Adds non existent decorators.
- Merge existent decorators.
- With override strategy, the value of the property will be replaced by the patch value.

- **MethodDeclaration**

- With override strategy, the body will be replaced.
- The parameters will be merged.

- **ParameterDeclaration**

- Replace type and modifiers with override merge strategy, adding non existent from patch into base.

- **ConstructorDeclaration**

- Merged in the same way as Method is.

- **FunctionDeclaration**

- Merged in the same way as Method is.

Input reader

The TypeScript input reader is based on the one that the [TypeScript merger](#) uses. The current extensions are additional module fields giving from which library any entity originates. `module: null` specifies a standard entity or type as `string` or `number`.

Object model

To get a first impression of the created object after parsing, let us start with analyzing a small example, namely the parsing of a simple [type-orm](#) model written in TypeScript.

```
import {Entity, PrimaryGeneratedColumn, Column} from "typeorm";

@Entity()
export class User {

    @PrimaryGeneratedColumn()
    id: number;

    @Column()
    firstName: string;

    @Column()
    lastName: string;

    @Column()
    age: number;

}
```

The returned object has the following structure

```
{
  "importDeclarations": [
```

```
{  
    "module": "typeorm",  
    "named": [  
        "Entity",  
        "PrimaryGeneratedColumn",  
        "Column"  
    ],  
    "spaceBinding": true  
}  
],  
"classes": [  
    {  
        "identifier": "User",  
        "modifiers": [  
            "export"  
        ],  
        "decorators": [  
            {  
                "identifier": {  
                    "name": "Entity",  
                    "module": "typeorm"  
                },  
                "isCallExpression": true  
            }  
        ],  
        "properties": [  
            {  
                "identifier": "id",  
                "type": {  
                    "name": "number",  
                    "module": null  
                },  
                "decorators": [  
                    {  
                        "identifier": {  
                            "name": "PrimaryGeneratedColumn",  
                            "module": "typeorm"  
                        },  
                        "isCallExpression": true  
                    }  
                ]  
            },  
            {  
                "identifier": "firstName",  
                "type": {  
                    "name": "string",  
                    "module": null  
                },  
                "decorators": [  
                    {  
                        "identifier": {  
                            "name": "Column",  
                            "module": "typeorm"  
                        },  
                        "isCallExpression": true  
                    }  
                ]  
            }  
        ]  
    }  
]
```

```

        "name": "Column",
        "module": "typeorm"
    },
    "isCallExpression": true
}
]
},
{
    "identifier": "lastName",
    "type": {
        "name": "string",
        "module": null
    },
    "decorators": [
        {
            "identifier": {
                "name": "Column",
                "module": "typeorm"
            },
            "isCallExpression": true
        }
    ]
},
{
    "identifier": "age",
    "type": {
        "name": "number",
        "module": null
    },
    "decorators": [
        {
            "identifier": {
                "name": "Column",
                "module": "typeorm"
            },
            "isCallExpression": true
        }
    ]
}
]
}
]
```

If we only consider the first level of the JSON response, we spot two lists of **imports** and **classes**, providing information about the only import statement and the only **User** class, respectively. Moving one level deeper we observe that:

- Every import statement is translated to an import declaration entry in the declarations list, containing the module name, as well as a list of entities imported from the given module.

- Every class entry provides besides the class identifier, its decoration(s), modifier(s), as well as a list of properties that the original class contains.

Note that, for each given type, the module from which it is imported is also given as in

```
"identifier": {
  "name": "Column",
  "module": "typeorm"
}
```

Returning to the general case, independently from the given TypeScript file, an object having the following Structure will be created.

- **importDeclarations**: A list of import statement as described above
- **exportDeclarations**: A list of export declarations
- **classes**: A list of classes extracted from the given file, where each entry is full of class specific fields, describing its properties and decorator for example.
- **interfaces**: A list of interfaces.
- **variables**: A list of variables.
- **functions**: A list of functions.
- **enums**: A list of enumerations.

69.2.7. HTML Plug-in

The HTML Plug-in enables merging result HTML files to existing ones. This plug-in is used at the moment for generate an Angular2 client. Currently, the generation of Angular2 client requires an ETO java object as input so, there is no need to implement an input reader for ts artifacts for the moment.

Trigger Extensions

As for the Angular2 generation the input is a java object, the trigger expressions (including matchers and variable assignments) are implemented as [Java](#).

Merger extensions

There are currently two merge strategies:

- merge strategy `html-ng*` (add the new code respecting the existing in case of conflict)
- merge strategy `html-ng*_override` (add the new code overwriting the existent in case of conflict)

The merging of two Angular2 files will be processed as follows:

The merge algorithm handles the following AST nodes:

- md-nav-list

- a
- form
- md-input-container
- input
- name (for name attribute)
- ngIf



Be aware, that the HTML merger is not generic and only handles the described tags needed for merging code of a basic Angular client implementation. For future versions, it is planned to implement a more generic solution.

70. CobiGen CLI

70.1. Cobigen Command line Interface generation

Our new command line interface (CLI) for CobiGen enables the generation of code using few commands. This feature allows us to decouple CobiGen from Eclipse.

You can check out our interactive katacoda tutorial where you can setup and use the cobigen cli step by step.

[CobiGen CLI Katacoda Scenario](#)

70.1.1. Install CobiGen CLI

In order to install the CobiGen CLI you will need to use the [devonfw/ide](#). In a console run `devon cobigen`.

70.1.2. Commands and options

Using the following command and option you will be able to customize your generation as follows:

- `cobigen, cg`: Main entry point of the CLI. If no arguments are passed, man page will be printed.
- `[generate, g]`: Command used for code generation.
 - `< --increment, -i >` : Specifies an increment ID to be generated. You can also search increments by name and CobiGen will output the resultant list. If an exact match found, code generation will happen.
 - `< --template, -t >` : specifies a template ID to be generated. You can also search templates by name and CobiGen will output the resultant list.
 - `< --outputRootPath, -out >`: The project file path in which you want to generate your code. If no output path is given, CobiGen will use the project of your input file.
- `[adapt-templates, a]`: Generates a new templates folder next to the cobigen cli and stores its location inside a configuration file. After executing this command, the CLI will attempt to use the specified Templates folder.
 - `< --custom-location, -cl >` : Allows the user to choose an absolute file path to a custom location where the CobiGen Templates should be stored and read from.
- `< --verbose, -v >`: Prints debug information, verbose log.
- `< --help, -h >`: Prints man page.
- `< update, u>` : This command compare the artificial pom plug-ins version with central latest version available and user can update any outdated plug-ins version .

70.1.3. CLI Execution steps:

CobiGen CLI is installed inside your devonfw distribution. In order to execute it follow the next steps:

1. Run `console.bat`, this will open a console.
2. Execute `cobigen` or `cg` and the man page should be printed.
3. Use a valid CobiGen input file and run `cobigen generate <pathToInputFile>`. **Note:** On the first execution of the CLI, CobiGen will download all the needed dependencies, please be patient.
4. A list of increments will be printed so that you can start the generation.

Preview of the man page for `generate` command:

```
C:\MyData\IDE4\workspaces\cobigen-master -> origin\tools-cobigen\cobigen-cli\src\main\java (master -> origin)
λ cobigen generate
Missing required parameter: <inputFiles>
Usage: [-hv] [-o[=<outputRootPath>]] [-i=<increments>[,<increments>...]]... [-t=<templates>[,<templates>...]]... <inputFiles>[,<inputFiles>...]
Using an input file (Java entity or ETO, OpenAPI definition, XML...) can
generate code to a location on your computer
<inputFiles>[,<inputFiles>...]...
    Input files (Java entity or ETO, OpenAPI definition, XML...) elements
    that will be parsed by Cobigen and generate code from them.
    You can use glob patterns on the path, for using multiple
    input files. Also you can specify input files one by one
    separated by comma.
-h, --help      Show this help message and exit.
-i, --increments=<increments>[,<increments>...]
    List of increments that will be generated. They need to be
    specified with numbers separated by comma.
-o, --out[=<outputRootPath>]
    Location where the generated code will be stored.
-t, --templates=<templates>[,<templates>...]
    List of templates that will be generated. They need to be
    specified with numbers separated by comma.
-v, --[no-]verbose  If this options is enabled, we will print also debug messages
-V, --version     Print version information and exit.

C:\MyData\IDE4\workspaces\cobigen-master -> origin\tools-cobigen\ccbigen-cli\src\main\java (master -> origin)
```

70.1.4. Examples

A selection of commands that you can use with the CLI:

- `cobigen generate foo\bar\EmployeeEntity.java`: As no output path has been defined, CobiGen will try to find the `pom.xml` of the current project in order to set the generation root path.
- `cobigen generate foo\bar*.java --out other\project`: Will retrieve all the Java files on that input folder and generate the code on the path specified by `--out`.
- `cg g foo\bar\webServices.yml --increment T0`: Performs a string search using `T0` and will print the closest increments like in the following image:

```
[INFO ] Here are the options you have for your choice. Which increments do you want to generate? Please list the increments number you want separated by comma:
2,4,5
[INFO ] (2) CRUD UC logic (CTOs)
[INFO ] (4) CRUD REST services (CTOs)
[INFO ] (5) TO's
[INFO ] Generating templates for input 'AuthorEntity.java', this can take a while...
[INFO ] Successfull generation.

[DEBUG] Commands were executed correctly

C:\MyData\IDE4\workspaces\cobigen-master -> origin\tools-cobigen\ccbigen-cli\src\main\java (master -> origin) updated 2019-06-26 13:56:33 +0550
λ
```

After choose some increment by user like below

- `cg g foo\bar\webServices.yml -i 1,4,6`: Directly generates increments with IDs `1`, `4` and `6`. CobiGen will not request you any other input.
- `cg a`: Downloads the latest CobiGen_Templates and unpacks them next to the CLI. CobiGen will from now on use these unpacked Templates for generation.

- `cg a -cl C:\my\custom\location`: Downloads the latest CobiGen_Templates and unpacks them in `C:\my\custom\location`. CobiGen will from now on use these unpacked Templates for generation.

70.1.5. CLI update command

Example of Update Command :

```
C:\MyData\IDE4\workspaces\cobigen-development\master\cobigen-cli\src\main\java (updatecommand -> origin)
λ cobigen update
[INFO] (0) All
[INFO] (1)core , 5.3.4
[INFO] (2)core-api , 5.3.4
[INFO] Here are the components that can be updated, which ones do you want to update? Please list the number of artifact(s) to update separated by comma:
```

Select the plug-ins which you want to update like below :

```
C:\MyData\IDE4\workspaces\cobigen-development\master\cobigen-cli\src\main\java (updatecommand -> origin)
λ cobigen update
[INFO] In order to do that, the CLI needs to find the compiled source code of your project.
[INFO] (0) All
[INFO] (1)core , 5.3.4
[INFO] (2)core-api , 5.3.4
[INFO] Here are the components that can be updated, which ones do you want to update? Please list the number of artifact(s) to update separated by comma:
1,2
[INFO] Updating the following components:
[INFO] (1)core
[INFO] (2)core-api
[INFO] As you need to run mvn clean install on the input project so that a new target folder gets created with the needed compiled sources.
[INFO] Updated successfully
```

Troubleshooting

70.1.6. Troubleshooting

When generating code from a Java file, CobiGen makes use of Java reflection for generating templates. In order to do that, the CLI needs to find the compiled source code of your project.

If you find an error like `Compiled class foo\bar\EmployeeEntity.java has not been found`, it means you need to run `mvn clean install` on the input project so that a new `target` folder gets created with the needed compiled sources.

71. Maven Build Integration

71.1. Maven Build Integration

For maven integration of CobiGen you can include the following build plugin into your build:

Listing 126. Build integration of CobiGen

```
<build>
  <plugins>
    <plugin>
      <groupId>com.devonfw.cobigen</groupId>
      <artifactId>cobigen-maven-plugin</artifactId>
      <version>VERSION-YOU-LIKE</version>
      <executions>
        <execution>
          <id>cobigen-generate</id>
          <phase>generate-resources</phase>
          <goals>
            <goal>generate</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Available goals

- **generate**: Generates contents configured by the standard non-compiled configuration folder. Thus generation can be controlled/configured due to an location URI of the configuration and template or increment ids to be generated for a set of inputs.

Available phases are all phases, which already provide compiled sources such that CobiGen can perform reflection on it. Thus possible phases are for example package, site.

71.1.1. Provide Template Set

For generation using the CobiGen maven plug-in, the CobiGen configuration can be provided in two different styles:

1. By a **configurationFolder**, which should be available on the file system whenever you are running the generation. The value of **configurationFolder** should correspond to the maven file path syntax.

Listing 127. Provide CobiGen configuration by configuration folder (file)

```
<build>
  <plugins>
    <plugin>
      ...
      <configuration>
        <configurationFolder>cobigen-templates</configurationFolder>
      </configuration>
      ...
    </plugin>
  </plugins>
</build>
```

2. By maven dependency, whereas the maven dependency should stick on the same conventions as the configuration folder. This explicitly means that it should contain non-compiled resources as well as the `context.xml` on top-level.

Listing 128. Provide CobiGen configuration by maven dependency (jar)

```
<build>
  <plugins>
    <plugin>
      ...
      <dependencies>
        <dependency>
          <groupId>com.devonfw.cobigen</groupId>
          <artifactId>templates-XYZ</artifactId>
          <version>VERSION-YOU-LIKE</version>
        </dependency>
      </dependencies>
      ...
    </plugin>
  </plugins>
</build>
```

We currently provide a generic deployed version of the templates on the devonfw-nexus for Register Factory (`<artifactId>cobigen-templates-rf</artifactId>`) and for the devonfw itself (`<artifactId>cobigen-templates-devonfw</artifactId>`).

71.1.2. Build Configuration

Using the following configuration you will be able to customize your generation as follows:

- `<destinationRoot>` specifies the root directory the relative `destinationPath` of `CobiGen templates configuration` should depend on. *Default \$./*
- `<inputPackage>` declares a package name to be used as input for batch generation. This refers directly to the CobiGen Java Plug-in `container matchers of type package` configuration.

- <`inputFile`> declares a file to be used as input. The CobiGen maven plug-in will try to parse this file to get an appropriate input to be interpreted by any CobiGen plug-in.
- <`increment`> specifies an `increment` ID to be generated. You can specify one single increment with content `ALL` to generate all increments matching the input(s).
- <`template`> specifies a `template` ID to be generated. You can specify one single template with content `ALL` to generate all templates matching the input(s).
- <`forceOverride`> specifies an overriding behavior, which enables non-mergeable resources to be completely rewritten by generated contents. For mergeable resources this flag indicates, that conflicting fragments during merge will be replaced by generated content. *Default: false*
- <`failOnNothingGenerated`> specifies whether the build should fail if the execution does not generate anything.

Listing 129. Example for a simple build configuration

```

<build>
  <plugins>
    <plugin>
      ...
      <configuration>
        <destinationRoot>${basedir}</destinationRoot>
        <inputPackages>
          <inputPackage>package.to.be.used.as.input</inputPackage>
        </inputPackages>
        <inputFiles>
          <inputFile>path/to/file/to/be/used/as/input</inputFile>
        </inputFiles>
        <increments>
          <increment>IncrementID</increment>
        </increments>
        <templates>
          <template>TemplateID</template>
        </templates>
        <forceOverride>false</forceOverride>
      </configuration>
      ...
    </plugin>
  </plugins>
</build>

```

71.1.3. Plugin Injection Since v3

Since version 3.0.0, the `plug-in mechanism` has changed to support modular releases of the CobiGen plug-ins. Therefore, you need to add all plug-ins to be used for generation. Take the following example to get the idea:

Listing 130. Example of a full configuration including plugins

```
<build>
  <plugins>
    <plugin>
      <groupId>com.devonfw.cobigen</groupId>
      <artifactId>cobigen-maven-plugin</artifactId>
      <version>VERSION-YOU-LIKE</version>
      <executions>
        ...
      </executions>
      <configuration>
        ...
      </configuration>
    <dependencies>
      <dependency>
        <groupId>com.devonfw.cobigen</groupId>
        <artifactId>templates-devon4j</artifactId>
        <version>2.0.0</version>
      </dependency>
      <dependency>
        <groupId>com.devonfw.cobigen</groupId>
        <artifactId>tempeng-freemarker</artifactId>
        <version>1.0.0</version>
      </dependency>
      <dependency>
        <groupId>com.devonfw.cobigen</groupId>
        <artifactId>javaplugin</artifactId>
        <version>1.6.0</version>
      </dependency>
    </dependencies>
  </plugin>
  </plugins>
</build>
```

71.1.4. A full example

1. A complete maven configuration example

```
<build>
  <plugins>
    <plugin>
      <groupId>com.devonfw.cobigen</groupId>
      <artifactId>cobigen-maven-plugin</artifactId>
      <version>6.0.0</version>
      <executions>
        <execution>
          <id>generate</id>
          <phase>package</phase>
          <goals>
            <goal>generate</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <inputFiles>

<inputFile>src/main/java/io/github/devonfw/cobigen/generator/dataaccess/api/InputEntity.java</inputFile>
        </inputFiles>
        <increments>
          <increment>dataaccess_infrastructure</increment>
          <increment>daos</increment>
        </increments>
        <failOnNothingGenerated>false</failOnNothingGenerated>
      </configuration>
      <dependencies>
        <dependency>
          <groupId>com.devonfw.cobigen</groupId>
          <artifactId>templates-devon4j</artifactId>
          <version>2.0.0</version>
        </dependency>
        <dependency>
          <groupId>com.devonfw.cobigen</groupId>
          <artifactId>tempeng-freemarker</artifactId>
          <version>2.0.0</version>
        </dependency>
        <dependency>
          <groupId>com.devonfw.cobigen</groupId>
          <artifactId>javaplugin</artifactId>
          <version>1.6.0</version>
        </dependency>
      </dependencies>
    </plugin>
  </plugins>
</build>
```

72. Eclipse Integration

72.1. Installation

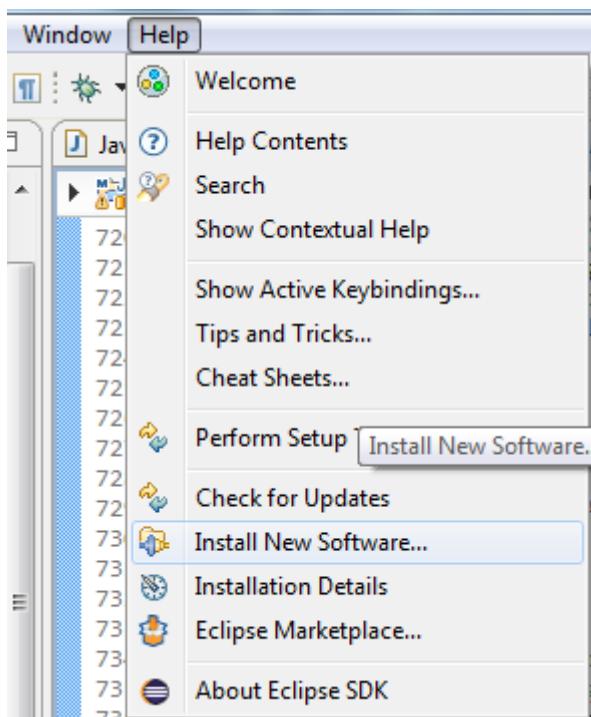
Remark: CobiGen is preinstalled in the [devonfw/devon-ide](#).

72.1.1. Preconditions

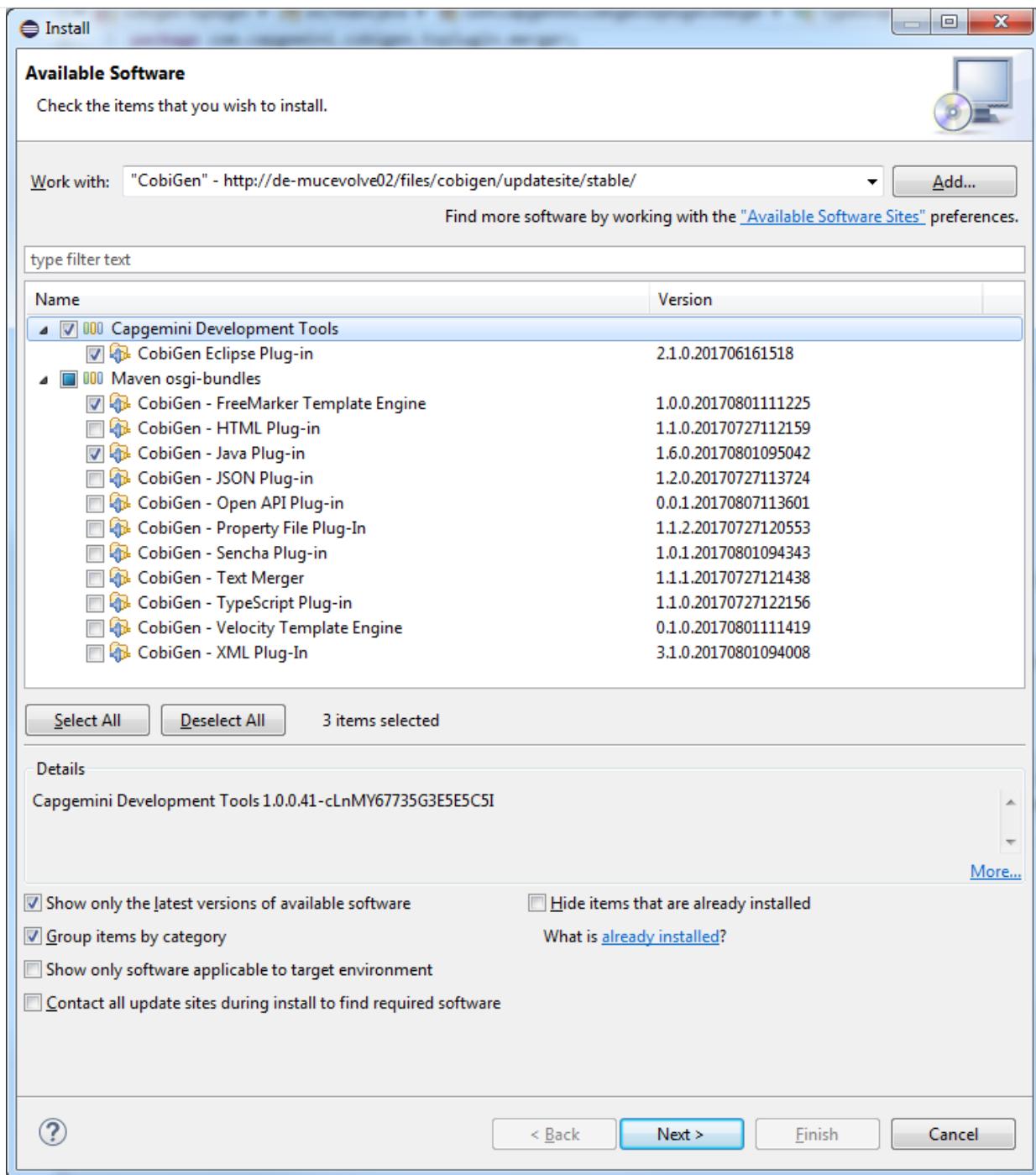
- Eclipse 4.x
- Java 7 Runtime (for starting eclipse with CobiGen). This is independent from the target version of your developed code.

72.1.2. Installation steps

1. Open the eclipse installation dialog
menu bar → *Help* → *Install new Software...*



2. Open CobiGen's update site
Insert the update site of your interest into the field *Work with* and press *Add ...*
 - Stable releases: <https://dl.bintray.com/devonfw/cobigen.p2/>



3. Follow the installation wizard

Select *CobiGen Eclipse Plug-in* → *Next* → *Next* → accept the license → *Finish* → *OK* → *Yes*

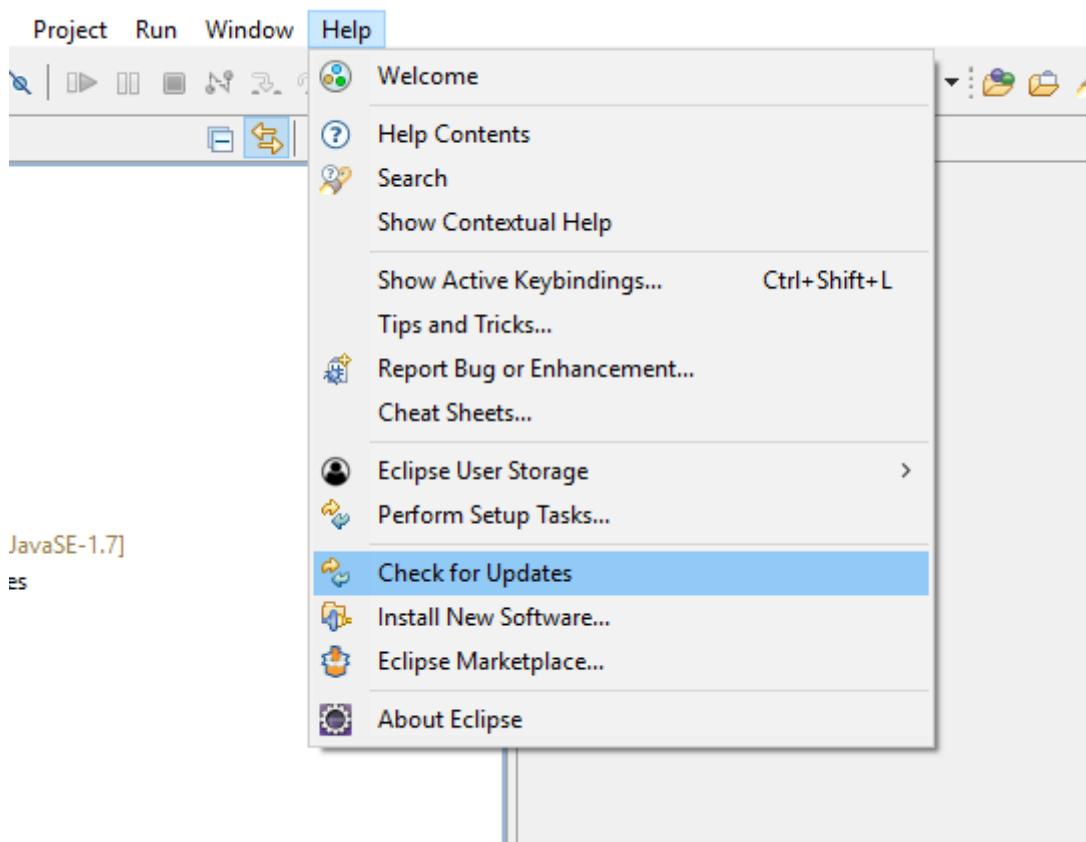
4. Once installed, a new menu entry named "CobiGen" will show up in the *Package Explorer*'s context menu. In the sub menu there will be the *Generate...* command, which may ask you to update the templates, and then you can start the generation wizard of CobiGen. You can adapt the templates by clicking on *Adapt Templates* which will give you the possibility to import the *CobiGen_Templates* automatically so that you can modify them.
5. Checkout (clone) your project's templates folder or use the current templates released with CobiGen (<https://github.com/devonfw/cobigen/tree/master/cobigen-templates>) and then choose Import -> General -> Existing Projects into Workspace to import the templates into your workspace.
6. Now you can start generating. To get an introduction of CobiGen try the devon4j templates and work on the devon4j sample application. There you might want to start with Entity objects as a

selection to run CobiGen with, which will give you a good overview of what CobiGen can be used for right out of the box in devon4j based development. If you need some more introduction in how to come up with your templates and increments, please be referred to the documentation of the [context configuration](#) and the [templates configuration](#)

Dependent on your context configuration menu entry *Generate...* may be greyed out or not. See for more information about valid selections for generation.

72.1.3. Updating

In general updating CobiGen for eclipse is done via the update mechanism of eclipse directly, as shown on image below:



Upgrading eclipse CobiGen plug-in to v3.0.0 needs some more attention of the user due to a changed plug-in architecture of CobiGen's [core module](#) and the eclipse integration. Eventually, we were able to provide any plug-in of CobiGen separately as its own eclipse bundle (fragment), which is automatically discovered by the main CobiGen Eclipse plug-in after installation.

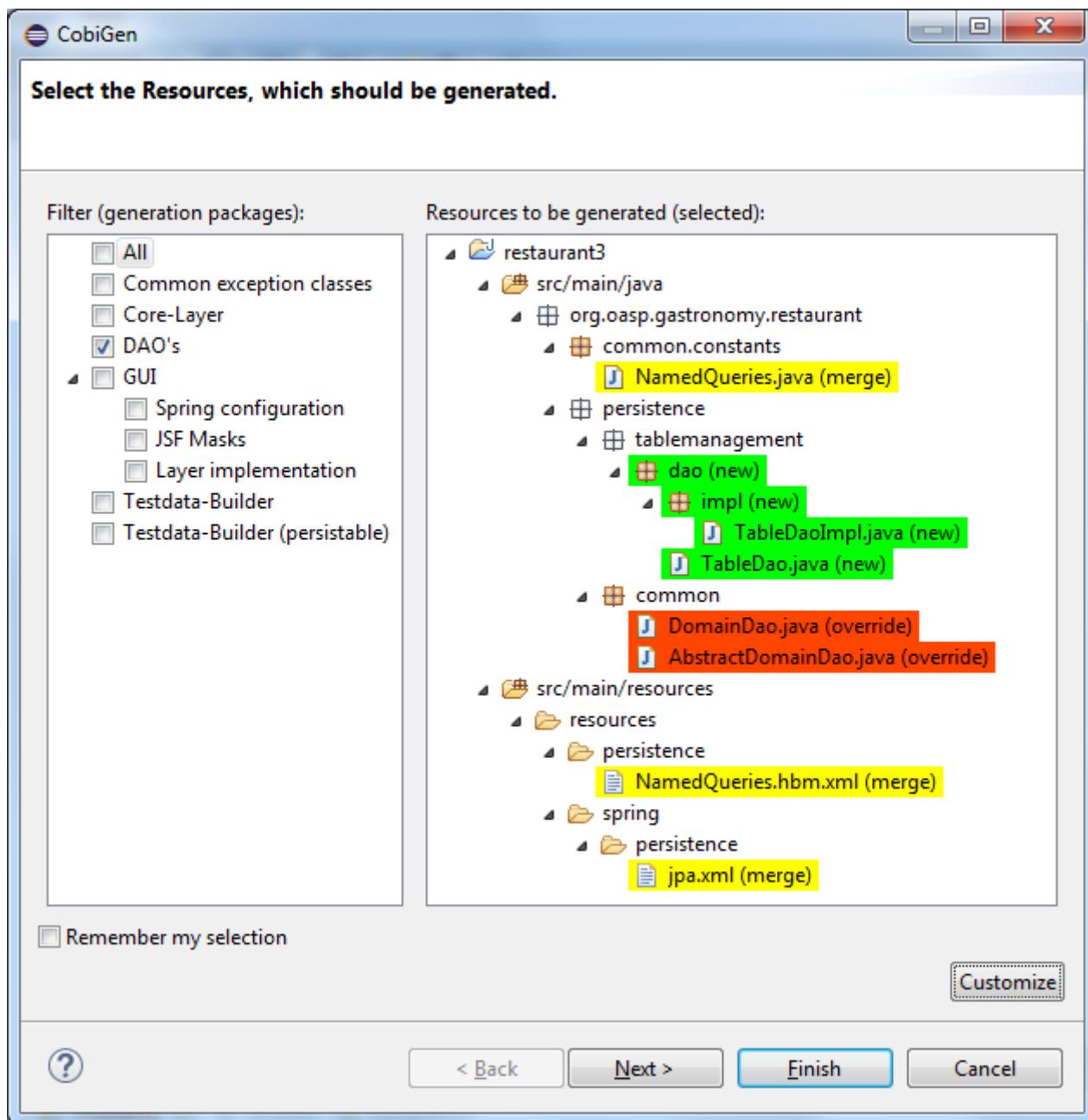
72.2. Usage

CobiGen has two different generation modes depending on the input selected for generation. The first one is the *simple mode*, which will be started if the input contains only one input artifact, e.g. for Java an input artifact currently is a Java file. The second one is the *batch mode*, which will be started if the input contains multiple input artifacts, e.g. for Java this means a list of files. In general this means also that the batch mode might be started when selecting complex models as inputs, which contain multiple input artifacts. The latter scenario has only been covered in the research

group,yet.

72.2.1. Simple Mode

Selecting the menu entry *Generate...* the generation wizard will be opened:



The left side of the wizard shows all available increments, which can be selected to be generated. Increments are a container like concept encompassing multiple files to be generated, which should result in a semantically closed generation output. On the right side of the wizard all files are shown, which might be effected by the generation - dependent on the increment selection of files on the left side. The type of modification of each file will be encoded into following color scheme if the files are selected for generation:

- **green:** files, which are currently non-existent in the file system. These files will be created during generation
- **yellow:** files, which are currently existent in the file system and which are configured to be

merged with generated contents.

- **red:** files, which are currently existent in the file system. These files will be overwritten if manually selected.
- **no color:** files, which are currently existent in the file system. Additionally files, which were unselected and thus will be ignored during generation.

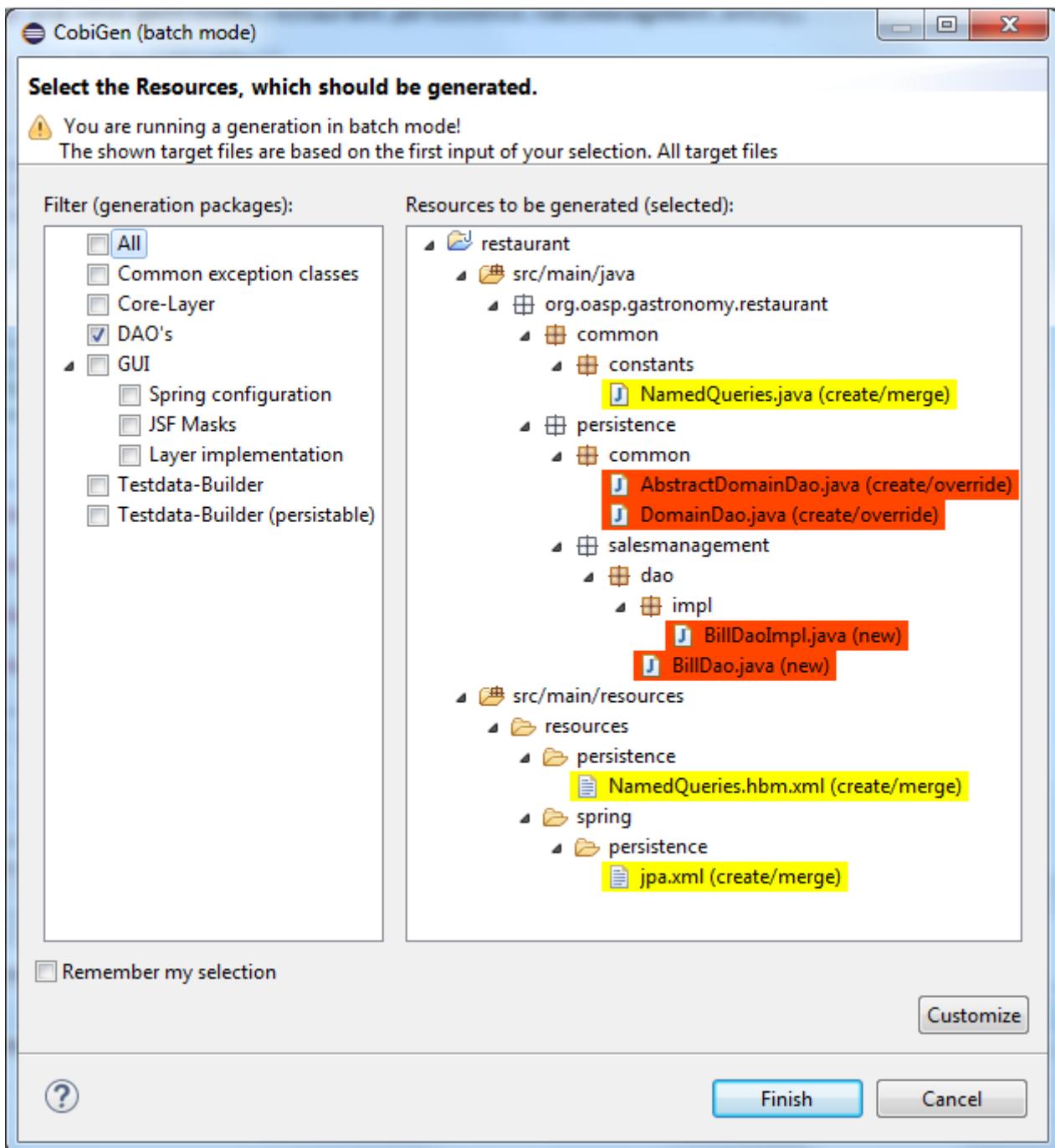
Selecting an increment on the left side will initialize the selection of all shown files to be generated on the right side, whereas green and yellow categorized files will be selected initially. A manual modification of the pre-selection can be performed by switching to the customization tree using the *Customize* button on the right lower corner.

Optional: If you want to customize the generation object model of a Java input class, you might continue with the *Next >* button instead of finishing the generation wizard. The next generation wizard page is currently available for Java file inputs and lists all non-static fields of the input. Unselecting entries will lead to an adapted object model for generation, such that unselected fields will be removed in the object model for generation. By default all fields will be included in the object model.

Using the *Finish* button, the generation will be performed. Finally, CobiGen runs the eclipse internal *organize imports* and *format source code* for all generated sources and modified sources. Thus it is possible, that--especially *organize imports* opens a dialog if some types could not be determined automatically. This dialog can be easily closed by pressing on *Continue*. If the generation is finished, the *Success!* dialog will pop up.

72.2.2. Batch mode

If there are multiple input elements selected, e.g., Java files, CobiGen will be started in batch mode. For the generation wizard dialog this means, that the generation preview will be constrained to the first selected input element. It does *not* preview the generation for each element of the selection or of a complex input. The selection of the files to be generated will be generated for each input element analogously afterwards.



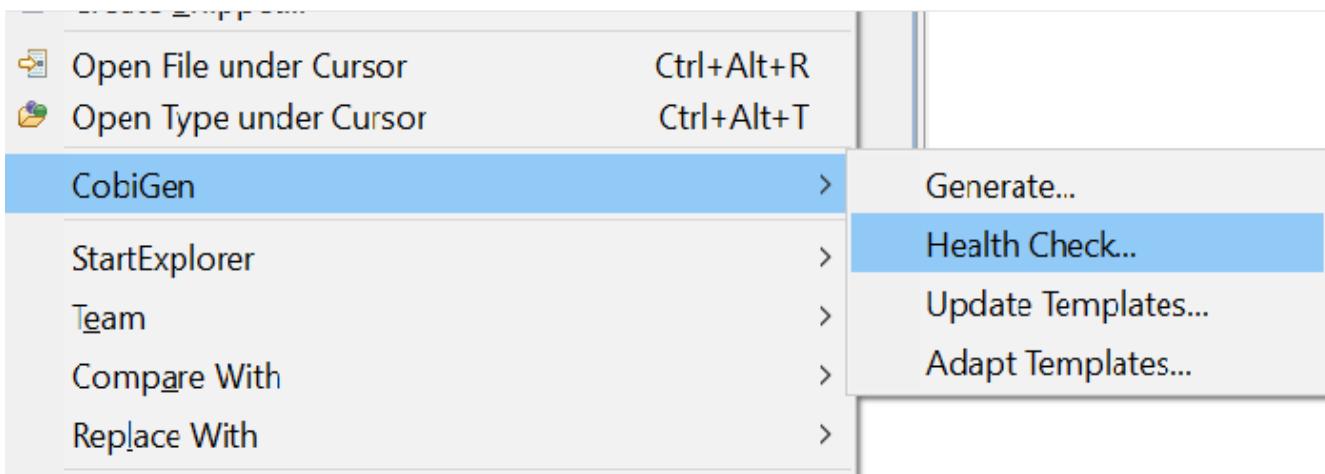
Thus the color encoding differs also a little bit:

- **yellow:** files, which are configured to be merged.
- **red:** files, which are not configured with any merge strategy and thus will be created if the file does not exist or overwritten if the file already exists
- **no color:** files, which will be ignored during generation

Initially all possible files to be generated will be selected.

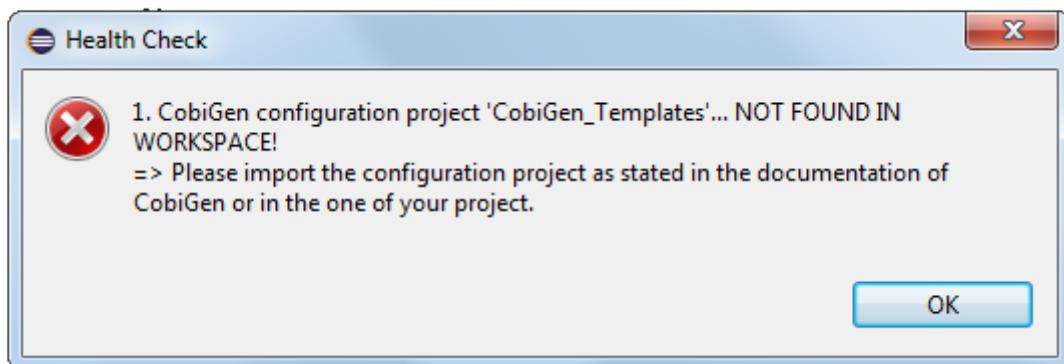
72.2.3. Health Check

To check whether CobiGen runs appropriately for the selected element(s) the user can perform a *Health Check* by activating the respective menu entry as shown below.



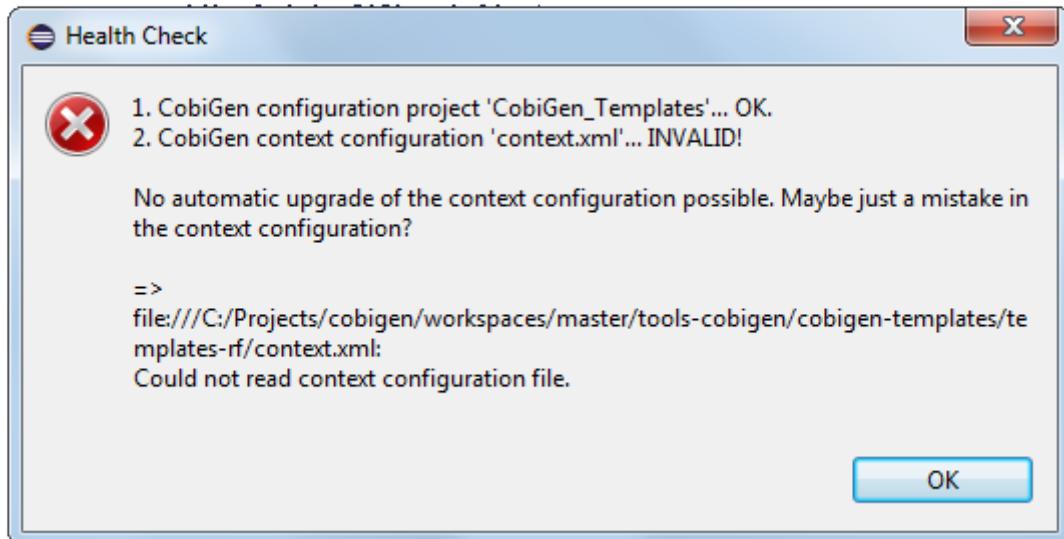
The simple *Health Check* includes 3 checks. As long as any of these steps fails, the *Generate* menu entry is grayed out.

The first step is to check whether the generation configuration is available at all. If this check fails you will see the following message:



This indicates, that there is no Project named *CobiGen_Templates* available in the current workspace. To run CobiGen appropriately, it is necessary to have a configuration project named *CobiGen_Templates* imported into your workspace. For more information see chapter [Eclipse Installation](#).

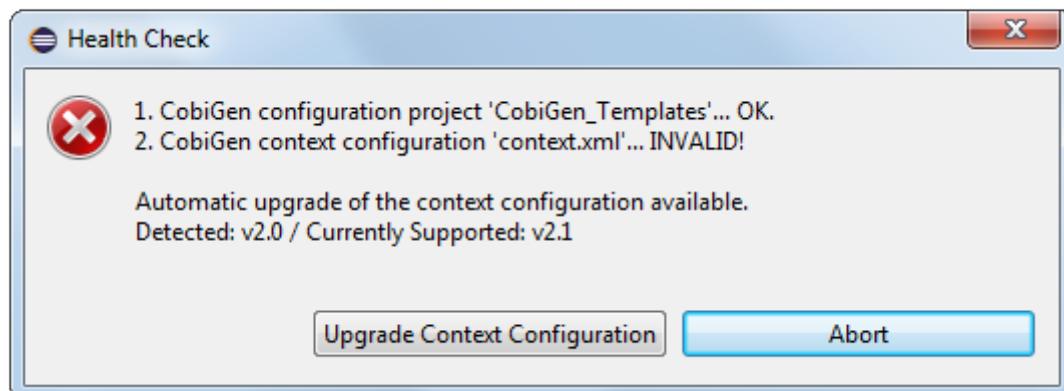
The second step is to check whether the template project includes a valid *context.xml*. If this check fails, you will see the following message:



This means that either your *context.xml*

- does not exist (or has another name)
- or it is not valid one in any released version of CobiGen
- or there is simply no automatic routine of upgrading your context configuration to a valid state.

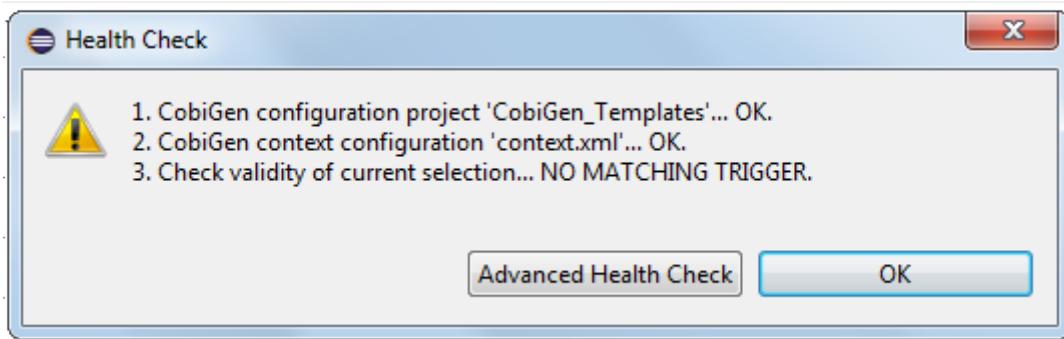
If all this is not the case, such as, there is a *context.xml*, which can be successfully read by CobiGen, you might get the following information:



This means that your *context.xml* is available with the correct name but it is outdated (belongs to an older CobiGen version). In this case just click on *Upgrade Context Configuration* to get the latest version.

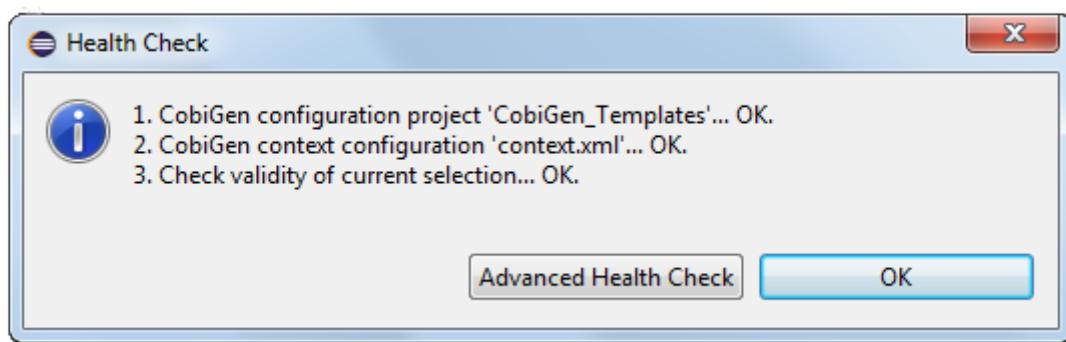
Remark: This will create a backup of your current context configuration and converts your old configuration to the new format. The upgrade will remove all comments from the file, which could be retrieved later on again from the backup. If the creation of the backup fails, you will be asked to continue or to abort.

The third step checks whether there are templates for the selected element(s). If this check fails, you will see the following message:



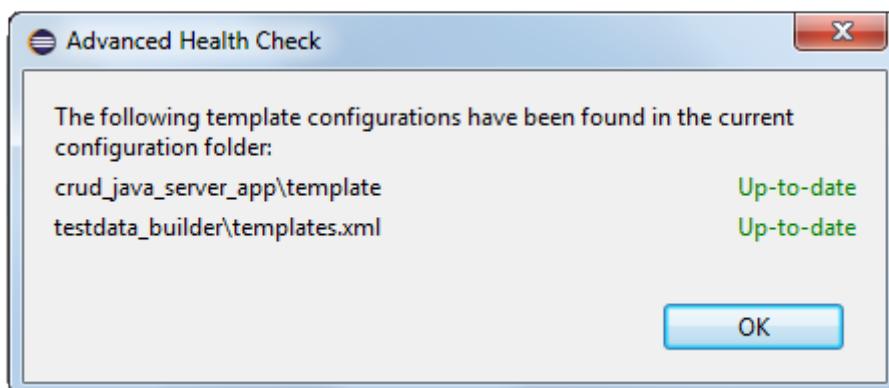
This indicates, that there no trigger has been activated, which matches the current selection. The reason might be that your selection is faulty or that you imported the wrong template project (e.g. you are working on a devon4j project, but imported the Templates for the Register Factory). If you are a template developer, have a look at the [trigger configuration](#) and at the corresponding available plug-in implementations of triggers, like e.g., [Java Plug-in](#) or [XML Plug-in](#).

If all the checks are passed you see the following message:



In this case everything is OK and the *Generate* button is not grayed out anymore so that you are able to trigger it and see the [simple-mode](#).

In addition to the basic check of the context configuration, you also have the opportunity to perform an *Advanced Health Check*, which will check all available templates configurations (*templates.xml*) of path-depth=1 from the configuration project root according to their compatibility.



Analogous to the upgrade of the *context configuration*, the *Advanced Health Check* will also provide upgrade functionality for *templates configurations* if available.

72.2.4. Update Templates

Update Template: Select Entity file and right click then select cobigen Update Templates after that click on download then download successfully message will be come .

72.2.5. Adapt Templates

Adapt Template: Select any file and right click, then select cobigen → *Adapt Templates* .If cobigen templates jar is not available then it downloads them automatically. If Cobigen templates is already present then it will override existing template in workspace and click on OK then imported template successfully message will be come.

Finally, please change the Java version of the project to 1.8 so that you don't have any compilation errors.

72.3. Logging

If you have any problem with the CobiGen eclipse plug-in, you might want to enable logging to provide more information for further problem analysis. This can be done easily by adding the `logback.xml` to the root of the CobiGen_templates configuration folder. The file should contain at least the following contents, whereas you should specify an absolute path to the target log file (at the `TODO`). If you are using the ([cobigen-templates](#) project, you might have the contents already specified but partially commented.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This file is for logback classic. The file contains the configuration for sl4j
logging -->
<configuration>
    <appender name="FILE" class="ch.qos.logback.core.FileAppender">
        <file><!-- TODO choose your log file location --></file>
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <Pattern>%n%date %d{HH:mm:ss.SSS} [%thread] %-5level %logger - %msg%n
            </Pattern>
        </encoder>
    </appender>
    <root level="DEBUG">
        <appender-ref ref="FILE" />
    </root>
</configuration>
```

73. How to

73.1. Angular 8 Client Generation

The generation can create a full Angular 8 client using the `devon4ng-application-template` package located at `workspaces/examples` folder of the distribution. For more details about this package, please refer [here](#).

You can also try out the katacoda tutorial for angular client generation.

CobiGen Angular Katacoda Scenario

Take into account that the TypeScript merging for CobiGen needs Node 6 or higher to be installed at your machine.



This is a short introduction to the Angular generation. For a deeper tutorial including the generation of the backend, we strongly recommend you to follow [this document](#).

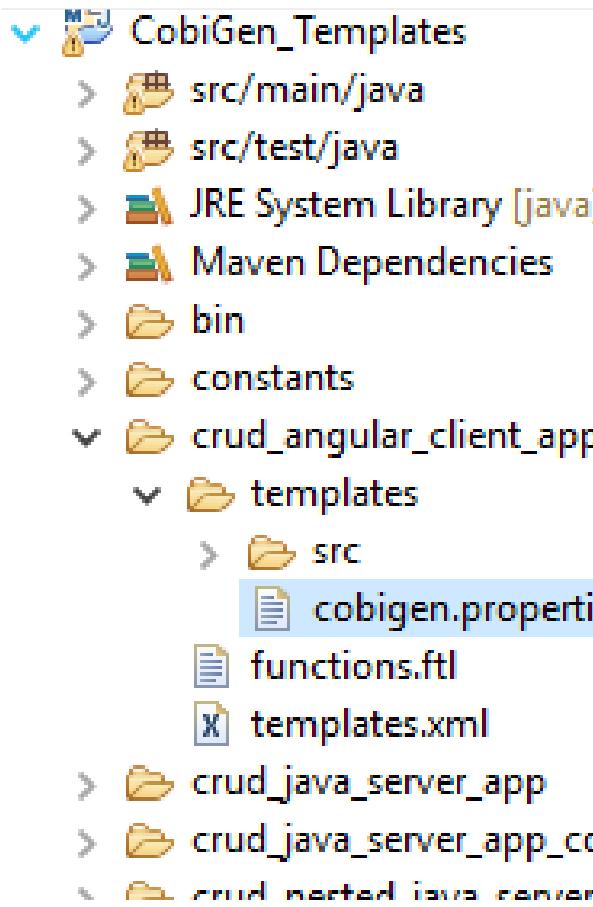
73.1.1. Requisites

Install yarn globally:

```
npm install -g yarn
```

73.1.2. Angular 8 workspace

The output location of the generation can be defined editing the `cobigen.properties` file located at `crud_angular_client_app/templates` folder of the `CobiGen_Templates` project.



By default, the output path would be into the *devon4ng-application-template* folder at the root of the devon4j project parent folder:

```
root/
|- devon4ng-application-template/
|- devon4j-project-parent/
  |- core/
  |- server/
```

However, this path can be changed, for example to *src/main/client* folder of the devon4j project:

```
relocate: ./src/main/client/${cwd}
```

```
root/
|- devon4j-project-parent/
  |- core/
    |- src
      |- main
        |- client
  |- server/
```

Once the output path is chosen, copy the files of **DEVON4NG-APPLICATION-TEMPLATE** repository into this output path.

73.1.3. Install Node dependencies

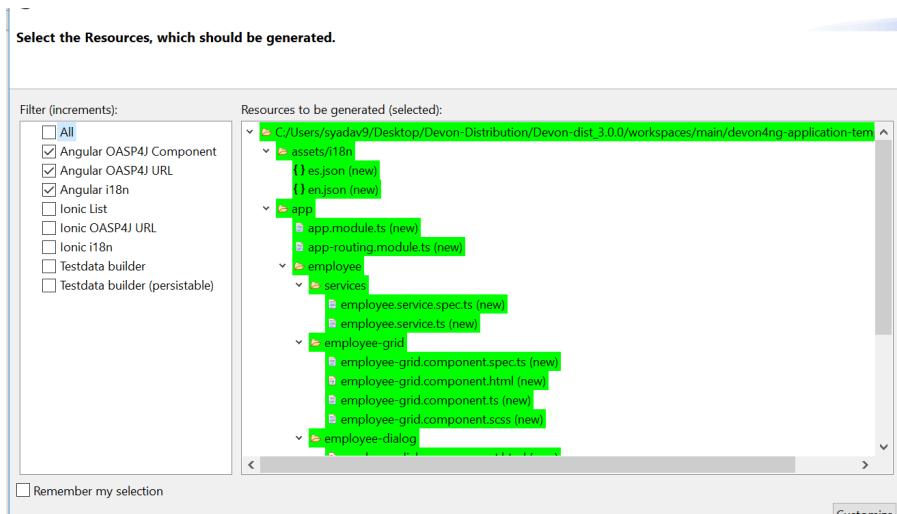
Open a terminal into devon4ng-application-template copied and just run the command:

```
yarn
```

This will start the installation of all node packages needed by the project into the node_modules folder.

73.1.4. Generating

From an Eto object, right click, CobiGen → Generate will show the CobiGen wizard relative to client generation:

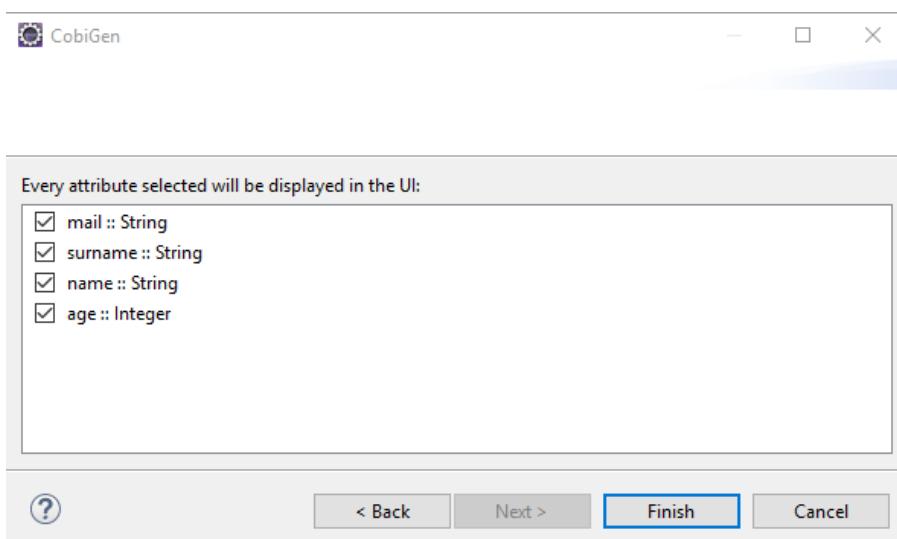


Check all the increments relative to Angular:



The Angular devon4j URL increment is only needed for the first generations however, checking it again on next generation will not cause any problem.

As we done on other generations, we click Next to choose which fields to include at the generation or simply clicking Finish will start the generation.



73.1.5. Routing

Due to the nature of the TypeScript merger, currently is not possible to merge properly the array of paths objects of the routings at app.routing.ts file so, this modification should be done by hand on this file. However, the import related to the new component generated is added.

This would be the generated `app-routing.module` file:

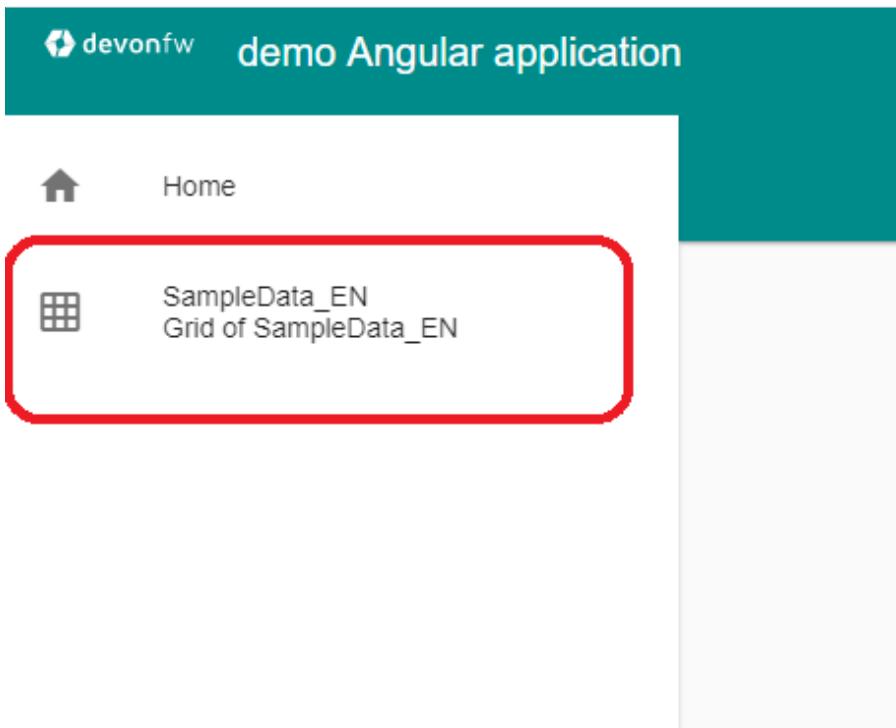
```
import { Routes, RouterModule } from '@angular/router';
import { LoginComponent } from './login/login.component';
import { AuthGuard } from './shared/security/auth-guard.service';
import { InitialPageComponent } from './initial-page/initial-page.component';
import { HomeComponent } from './home/home.component';
import { SampleDataGridComponent } from './sampledata/sampledata-grid/sampledata-grid.component';
//Routing array
const appRoutes: Routes = [
  {
    path: 'login',
    component: LoginComponent
  },
  {
    path: 'home',
    component: HomeComponent,
    canActivate: [AuthGuard],
    children: [
      {
        path: '',
        redirectTo: '/home/initialPage',
        pathMatch: 'full',
        canActivate: [AuthGuard]
      },
      {
        path: 'initialPage',
        component: InitialPageComponent,
        canActivate: [AuthGuard]
      }
    ]
  },
  {
    path: '**',
    redirectTo: '/login',
    pathMatch: 'full'
  }];
export const routing = RouterModule.forRoot(appRoutes);
```

Adding the following into the children object of `home`, will add into the side menu the entry for the component generated:

```
{
  path: 'sampleData',
  component: SampleDataGridComponent,
  canActivate: [AuthGuard],
}
```

```
import { Routes, RouterModule } from '@angular/router';
import { LoginComponent } from './login/login.component';
import { AuthGuard } from './shared/security/auth-guard.service';
import { InitialPageComponent } from './initial-page/initial-page.component';
import { HomeComponent } from './home/home.component';
import { SampleDataGridComponent } from './sampledata/sampledata-grid/sampledata-
grid.component';
//Routing array
const appRoutes: Routes = [{
  path: 'login',
  component: LoginComponent
}, {
  path: 'home',
  component: HomeComponent,
  canActivate: [AuthGuard],
  children: [{

    path: '',
    redirectTo: '/home/initialPage',
    pathMatch: 'full',
    canActivate: [AuthGuard]
  }, {
    path: 'initialPage',
    component: InitialPageComponent,
    canActivate: [AuthGuard]
  }, {
    path: 'sampleData',
    component: SampleDataGridComponent,
    canActivate: [AuthGuard],
  }]
}, {
  path: '**',
  redirectTo: '/login',
  pathMatch: 'full'
}];
export const routing = RouterModule.forRoot(appRoutes);
```



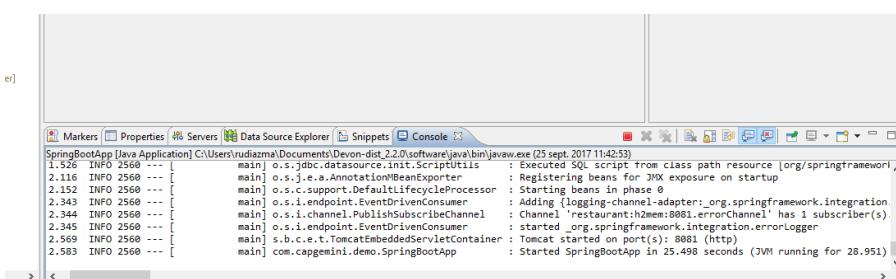
73.1.6. JWT Authentication

If you are using a back end server with JWT Authentication (there is a sample in workspaces/folder called **sampleJwt**) you have to specify the Angular application to use this kind of authentication.

By default the variable is set to 'JWT' but you can change it to CSRF by going to the [Enviroment.ts](#) and setting **security: 'csrf'**.

73.1.7. Running

First of all, run your devon4j java server by right clicking over `SpringBootApp.java` Run As → Java Application. This will start to run the SpringBoot server. Once you see the Started SpringBoot in XX seconds, the backend is running.



Once the the server is running, open a Devon console at the output directory defined previously and run:

```
ng serve --open
```

This will run the Angular 8 application at:

<http://localhost:4200>

```
λ ng serve -o
Your global Angular CLI version (1.2.0) is greater than your local
version (1.0.6). The local Angular CLI version is used.

To disable this warning use "ng set --global warnings.versionMismatch=false".
** NG Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200 ***
13% building modules 31/44 modules 13 active ...@angular\forms@angular\forms.es5.jswebpack: wait until bundle
e finished:
Hash: 655086d5df7d30892486
Time: 56749ms
chunk {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 397 kB {4} [initial] [rendered]
chunk {1} main.bundle.js, main.bundle.js.map (main) 54.8 kB {3} [initial] [rendered]
chunk {2} styles.bundle.js, styles.bundle.js.map (styles) 489 kB {4} [initial] [rendered]
chunk {3} vendor.bundle.js, vendor.bundle.js.map (vendor) 5.49 MB [initial] [rendered]
chunk {4} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry] [rendered]
webpack: Compiled successfully.

  Date: 2019-01-15T11:15:29.000Z  Hash: 655086d5df7d30892486  Time: 56749ms
  chunk {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 397 kB {4} [initial] [rendered]
  chunk {1} main.bundle.js, main.bundle.js.map (main) 54.8 kB {3} [initial] [rendered]
  chunk {2} styles.bundle.js, styles.bundle.js.map (styles) 489 kB {4} [initial] [rendered]
  chunk {3} vendor.bundle.js, vendor.bundle.js.map (vendor) 5.49 MB [initial] [rendered]
  chunk {4} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry] [rendered]
  webpack: Compiled successfully.

  Date: 2019-01-15T11:15:29.000Z  Hash: 655086d5df7d30892486  Time: 56749ms
  chunk {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 397 kB {4} [initial] [rendered]
  chunk {1} main.bundle.js, main.bundle.js.map (main) 54.8 kB {3} [initial] [rendered]
  chunk {2} styles.bundle.js, styles.bundle.js.map (styles) 489 kB {4} [initial] [rendered]
  chunk {3} vendor.bundle.js, vendor.bundle.js.map (vendor) 5.49 MB [initial] [rendered]
  chunk {4} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry] [rendered]
  webpack: Compiled successfully.
```

Once finished, the browser will open automatically at the previous localhost URL showing the Angular 8 application, using the credentials set at the devon4j java server you will be able to access.

73.2. Ionic client generation

We are going to show you **how to** generate a CRUD Ionic application from an **ETO** using CobiGen.



This is a short introduction to the Ionic generation. For a deeper tutorial including the generation of the backend, we strongly recommend you to follow [this document](#).

73.2.1. Prerequisites

Before starting, make sure you already have in your computer:

- **Ionic:** by following the steps defined on that page. It includes installing:
 - **NodeJS:** We have to use "NPM" for downloading packages.
 - Ionic CLI.
- **Capacitor:** Necessary to access to native device features.

If *CobiGen_Templates* are not already downloaded, follow the next steps:

- Right click on any file of your workspace *CobiGen > Update Templates* and now you are able to start the generation.
- If you want to adapt them, click *Adapt Templates* and you should have the *CobiGen_Templates* as a new project in Eclipse's workspace.

After following those steps correctly, you should have the latest version of the templates ready to use.

73.2.2. Generation

We are going to generate the CRUD into a **sample application** that we have developed for testing this functionality. It is present on your **workspaces/examples** folder (`devon4ng-ionic-application-template`). If you do not see it, you can clone or download it from [here](#).

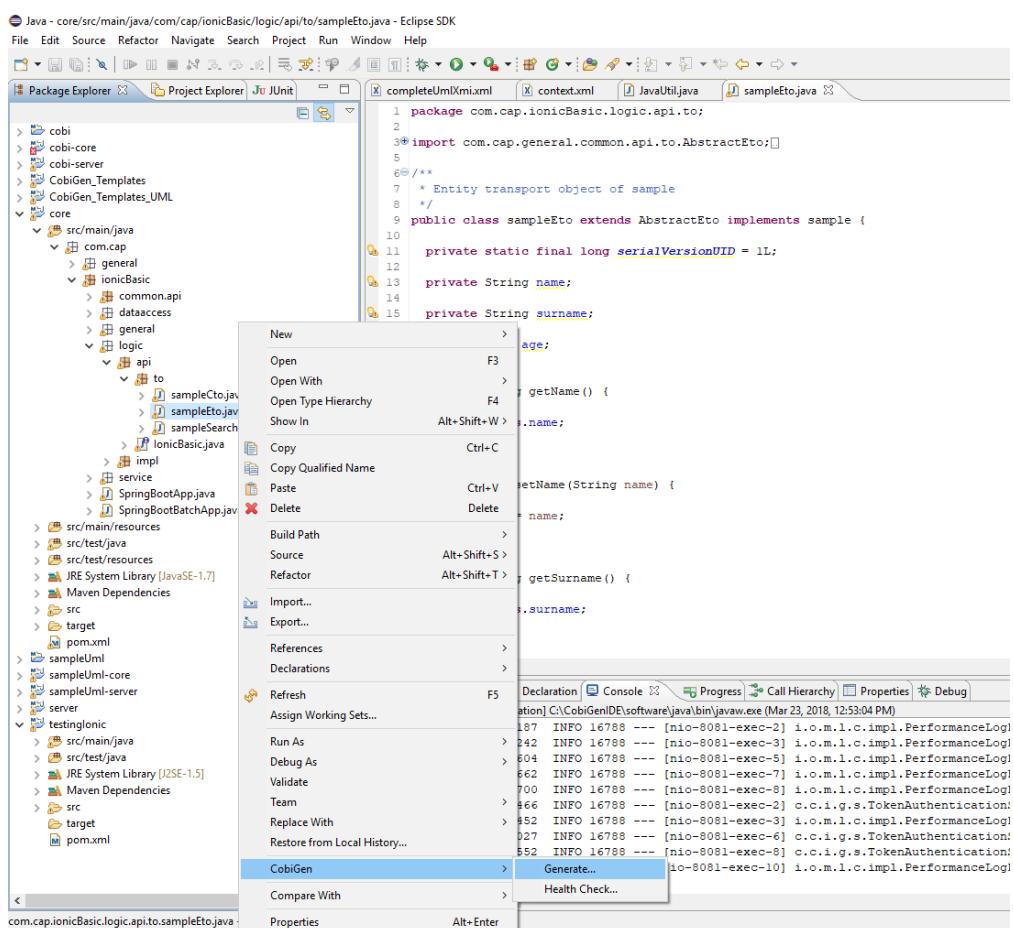
After having that sample app, please create an **devon4j** project and then start implementing the ETO: You will find an example [here](#).

As you can see, **TableEto** contains 3 attributes: 2 of them are **Long** and the third one **TableState** is an enum that you will find [here](#). The Ionic generation works fine for any Java primitive attribute (Strings, floats, chars, boolean...) and enums. However, if you want to use your own objects, you should override the **toString()** method, as explained [here](#).

The attributes explained above will be used for generating a page that shows a list. Each item of that list will show the values of those attributes.

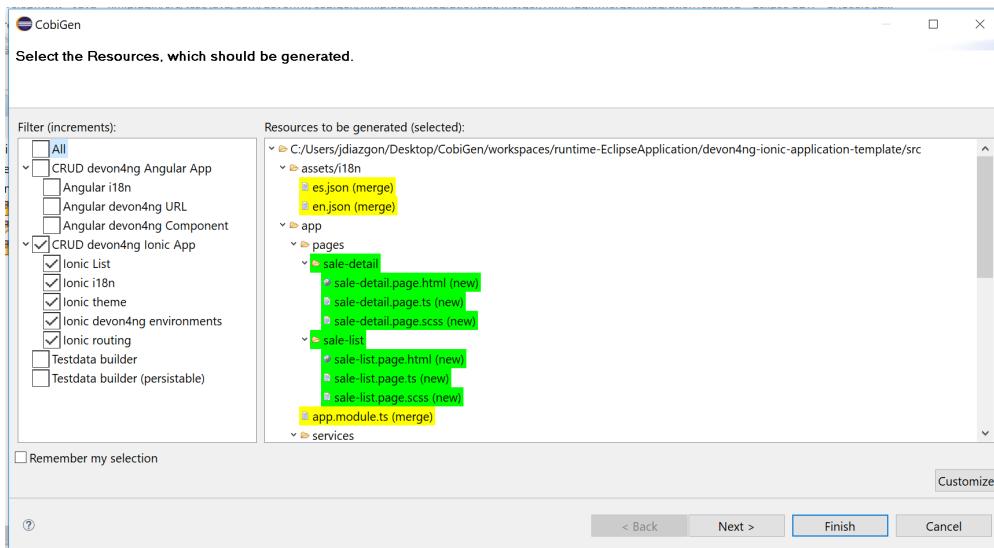
For generating the files:

- Right click your ETO file and click on *CobiGen > Generate* as shown on the figure below.

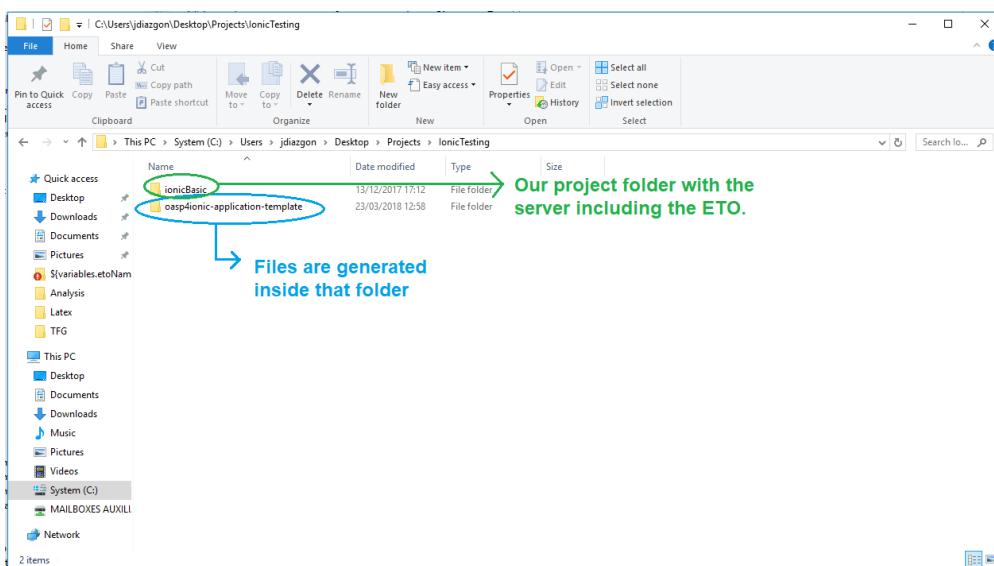


- Select the Ionic increments for generating as shown below. *Increments group a set of templates for generating different projects.*
 - Ionic List** used for generating the page containing the list.
 - Ionic devon4ng environments** is for stating the server path.
 - Ionic i18n** used for generating the different language translations for the *translationService* (currently English and Spanish).
 - Ionic routing** adds an *app-routing.module.ts* file to allow navigation similar to the one available in Angular.
 - Ionic theme** generates the *variables.scss* file which contains variables to style the

application.



By default, the generated files will be placed inside "devon4ng-ionic-application-template", next to the root of your project's folder. See the image below to know where they are generated. For **changing the generation path** and the name of the application go to *CobiGen_Templates/crud_ionic_client_app/cobigen.properties*.



Now that we have generated the files, lets start testing them:

- First change the **SERVER_URL** of your application. For doing that, modify *src/environments/environments.ts*, also modify *src/environments/environments.android.ts* (android) and *src/environments/environments.prod.ts* (production) if you want to test in different environments.
- Check that there are no duplicated imports. Sometimes there are duplicated imports in *src/app/app.module.ts*. This happens because the merger of CobiGen prefers to duplicate rather than to delete.
- Run **npm install** to install all the required dependencies.
- Run **'ionic serve** on your console.

After following all these steps your application should start. However, remember that you will need your **server** to be running for access to the list page.

73.2.3. Running it on Android

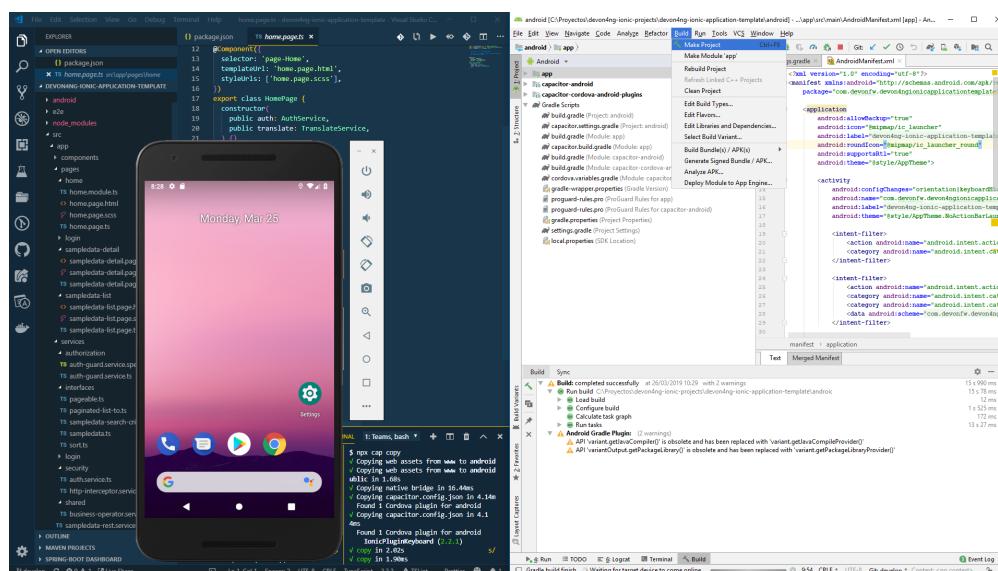
To run the application in an android emulated device, it is necessary to have Android Studio and Android SDK. After its installation, the following commands have to be run on your console:

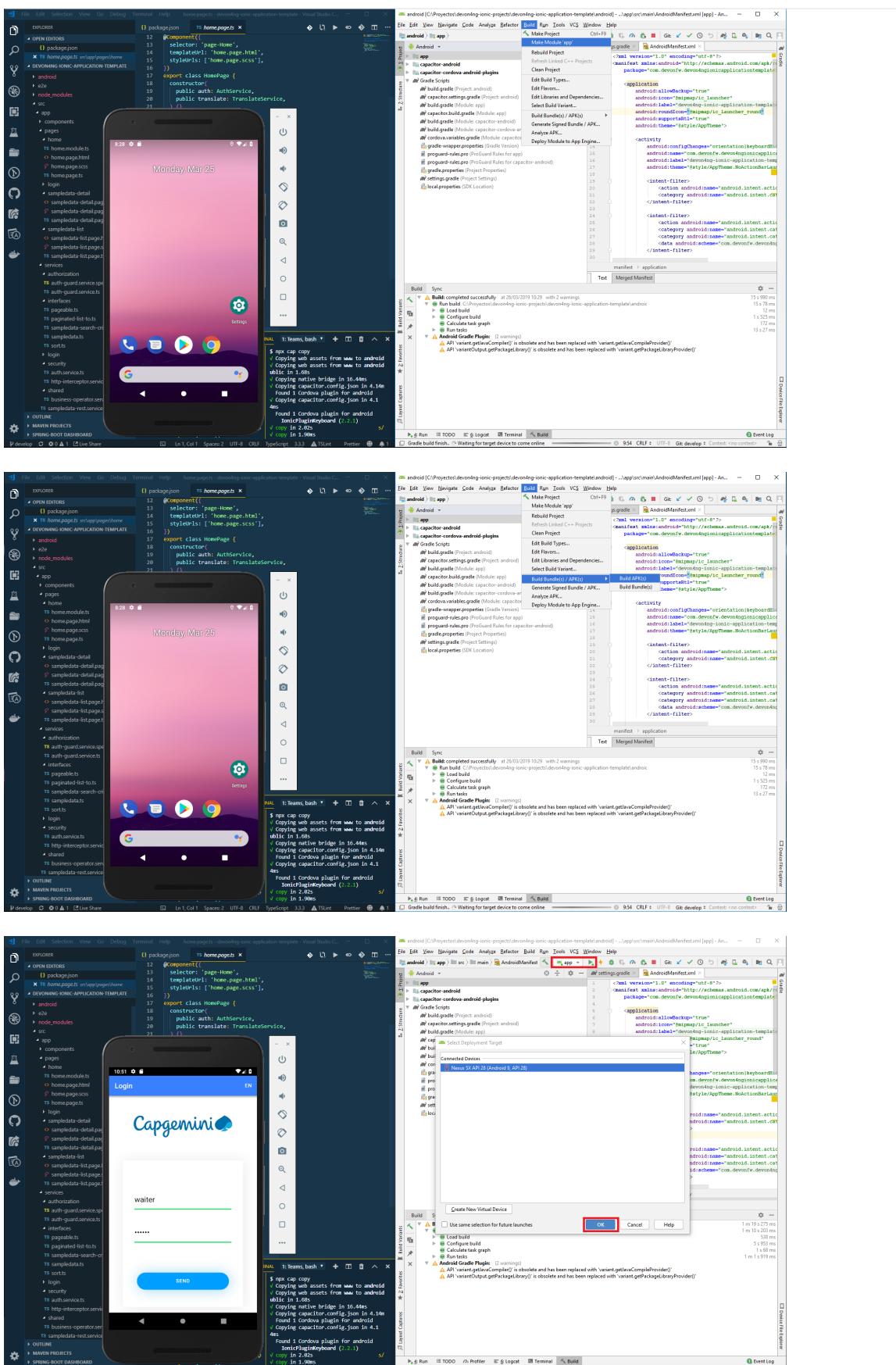
- `npx cap init "name-for-the-app (between quotes)" "id-for-the-app (between quotes)"`
- `ionic build --configuration=android`. To use this command, you must add an android build configuration at `angular.json`

```
"build": {
  ...
  "configurations": {
    ...
    "android": {
      "fileReplacements": [
        {
          "replace": "src/environments/environment.ts",
          "with": "src/environments/environment.android.ts"
        }
      ],
    },
  }
}
```

- `npx cap add android`
- `npx cap copy`
- `npx cap open android`

The last steps are done in Android studio: make the project, make the app, build and APK and run in a device.





73.3. Implementing a new Plug-in

New plug-ins can implement an input reader, a merger, a matcher, a trigger interpreter, and/or a template engine as explained [here](#).



It is discouraged to have `cobigen-core` dependencies at runtime, except for `cobigen-core-api` which definitely must be present.

73.3.1. Plugin Activator

Each plug-in has to have an plug-in activator class implementing the interface `GeneratorPluginActivator` from the core-api. This class will be used to load the plug-in using the `PluginRegistry` as explained [here](#). This class implements two methods:

1. `bindMerger()` → returns a mapping of merge strategies and its implementation to be registered.
2. `bindTriggerInterpreter()` → returns the trigger interpreters to be provided by this plug-in.

Both methods create and register instances of mergers and trigger interpreters to be provided by the new plug-in.

73.3.2. Adding TriggerInterpreter

The trigger interpreter has to implement the `TriggerInterpreter` interface from the core. The trigger interpreter defines the type for the new plugin and creates new `InputReader` and new `Matcher` objects.

73.3.3. Adding InputReader

The input reader is responsible of read the input object and parse it into FreeMarker models. The input reader must be implemented for the type of the input file. If there is any existent plugin that has the same file type as input, there will be no need to add a new input reader to the new plug-in.

InputReader Interface

The interface needed to add a new input reader is defined at the core. Each new sub plug-in must implements this interface if is needed an input reader for it.

The interface implements the basic methods that an input reader must have, but if additional methods are required, the developer must add a new interface that extends the original interface `InputReader.java` from the core-api and implement that on the sub plug-in.

The methods to be implemented by the input reader of the new sub plugin are:

Method	Return Type	Description
<code>isValidInput(Object input)</code>	<code>boolean</code>	This function will be called if matching triggers or matching templates should be retrieved for a given input object.
<code>createModel(Object input)</code>	<code>Map<String, Object></code>	This function should create the FreeMarker object model from the given input.

Method	Return Type	Description
<code>combinesMultipleInputObjects(Object input)</code>	<code>boolean</code>	States whether the given input object combines multiple input objects to be used for generation.
<code>getInputObjects(Object input, Charset inputCharset)</code>	<code>List<Object></code>	Will return the set of combined input objects if the given input combines multiple input objects.
<code>getTemplateMethods(Object input)</code>	<code>Map<String, Object></code>	This method returns available template methods from the plugins as Map. If the plugin which corresponds to the input does not provide any template methods an empty Map will be returned.
<code>getInputObjectsRecursively(Object input, Charset inputCharset)</code>	<code>List<Object></code>	Will return the set of combined input objects if the given input combines multiple input objects.

Model Constants

The Input reader will create a model for FreeMarker. A Freemarker model must have variables to use them at the `.ftl` template file. Refer to [Java Model](#) to see the FreeMarker model example for java input files.

Registering the Input Reader

The input reader is an object that can be retrieved using the correspondent get method of the trigger interpreter object. The trigger interpreter object is loaded at the eclipse plug-in using the load plug-in method explained [here](#). That way, when the core needs the input reader, only needs to call that `getInputReader` method.

73.3.4. Adding Matcher

The matcher implements the `MatcherInterpreter` interface from the core-api. Should be implemented for providing a new input matcher. Input matcher are defined as part of a trigger and provide the ability to restrict specific inputs to a set of templates. This restriction is implemented with a `MatcherType` [enum](#).

E.g JavaPlugin

```
private enum MatcherType {
    /** Full Qualified Name Matching */
    FQN,
    /** Package Name Matching */
    PACKAGE,
    /** Expression interpretation */
    EXPRESSION
}
```

Furthermore, matchers may provide several variable assignments, which might be dependent on any information of the matched input and thus should be resolvable by the defined matcher.

E.g JavaPlugin

```
private enum VariableType {
    /** Constant variable assignment */
    CONSTANT,
    /** Regular expression group assignment */
    REGEX
}
```

73.3.5. Adding Merger

The merger is responsible to perform merge action between new output with the existent data at the file if it already exists. Must implement the Merger interface from the core-api. The implementation of the Merge interface must override the following methods:

Method	Return Type	Description
<code>getType()</code>	<code>String</code>	Returns the type, this merger should handle.
<code>merge(File base, String patch, String targetCharset)</code>	<code>String</code>	Merges the patch into the base file.

Is important to know that any exception caused by the merger must throw a MergeException from the core-api to the eclipse-plugin handle it.

73.3.6. Changes since Eclipse / Maven 3.x

Since version 3.x the Eclipse and Maven plugins of CobiGen utilize the Java [ServiceLoader](#) mechanic to find and register plugins at runtime. To enable a new plugin to be discovered by this mechanic the following steps are needed:

- create the file `META-INF/services/com.capgemini.cobigen.api.extension.GeneratorPluginActivator` containing just the full qualified name of the class implementing the `GeneratorPluginActivator` interface, if the plugin provides a `Merger` and/or a `TriggerInterpreter`
- create the file `META-INF/services/com.capgemini.cobigen.api.extension.TextTemplateEngine`

containing just the full qualified name of the class implementing the [TextTemplateEngine](#) interface, if provided by the plugin

- include **META-INF** into the target bundle (i.e. the folder **META-INF** has to be present in the target jar file)

Example: Java Plugin

The java plugin provides both a [Merger](#) and a [TriggerInterpreter](#). It contains therefore a `com.capgemini.cobigen.api.extension.GeneratorPluginActivator` file with the following content:

```
com.capgemini.cobigen.javaplugin.JavaPluginActivator
```

This makes the `JavaPluginActivator` class discoverable by the [ServiceLoader](#) at runtime.

- to properly include the plugin into the current system and use existing infrastructure, you need to add the plugin as a module in `/cobigen/pom.xml` (in case of a [Merger/TriggerInterpreter](#) providing plugin) and declare that as the plugin's parent in it's own `pom.xml` via

```
<parent>
  <groupId>com.capgemini</groupId>
  <artifactId>cobigen-parent</artifactId>
  <version>dev-SNAPSHOT</version>
</parent>
```

or `/cobigen/cobigen-templateengines/pom.xml` (in case of a [Merger/TriggerInterpreter](#) providing plugin) and declare that as the plugin's parent in it's own `pom.xml` via

```
<parent>
  <groupId>com.capgemini</groupId>
  <artifactId>cobigen-tempeng-parent</artifactId>
  <version>dev-SNAPSHOT</version>
</parent>
```

If the plugin provides both just use the `/cobigen/pom.xml`.

- The dependencies of the plugin are included in the bundle
- To make the plugin available to the Eclipse plugin it must be included into the current `compositeContent.xml` and `compositeArtifacts.xml` files. Both files are located in `http://de-mucevolve02/files/cobigen/updatesite/{experimental|nightly|stable}`. To do so, add an `<child>` entry to the `<children>` tag in both files and adapt the `size` attribute to match the new number of references. The `location` attribute of the new `<child>` tag needs to be the artifact id of the plugins `pom.xml`.

Example: Java Plugin

In case of the Java plugin, the entry is

```
<child location="cobiGen-javaplugin"/>
```

Deployment

For the Maven Plugin

Execute `mvn clean deploy` from the plugins project folder. You need to configure write access to the [devon nexus](#) (e.g in the CobiGen IDE via the [variables-customized](#) script)

For the Eclipse Plugin

Depending on the kind of release you want to publish you can chose from the following maven profiles:

- [experimental](#) is for, as the name suggests, experimental snapshot builds. In case of new plugins this is a good place to upload first drafts.
- [nightly](#) is for periodically CI deployment.
- [stable](#) is solely for releases.

E.g. you want an experimental release you need to follow these steps:

```
# Builds the Manifest and bundles the dependencies
mvn clean package bundle:bundle -Pp2-bundle
# Uses the created bundle and builds a p2 update site for it. Do NOT use clean
mvn install bundle:bundle -Pp2-bundle,p2-build-mars,p2-build-experimental p2:site
# Uploads the p2 update site to the experimental repository. Do NOT use clean
mvn deploy -Pp2-build-mars,p2-build-experimental -Dp2.upload=experimental
```

You need write access to the [iCSD file server](#) configured (e.g in the CobiGen IDE via the [variables-customized](#) script).

73.4. Introduction to CobiGen external plug-ins

Since September of 2019, a major change on CobiGen has taken place. CobiGen is written in Java code and previously, it was very hard for developers to create new plug-ins in other languages.

Creating a new plug-in means:

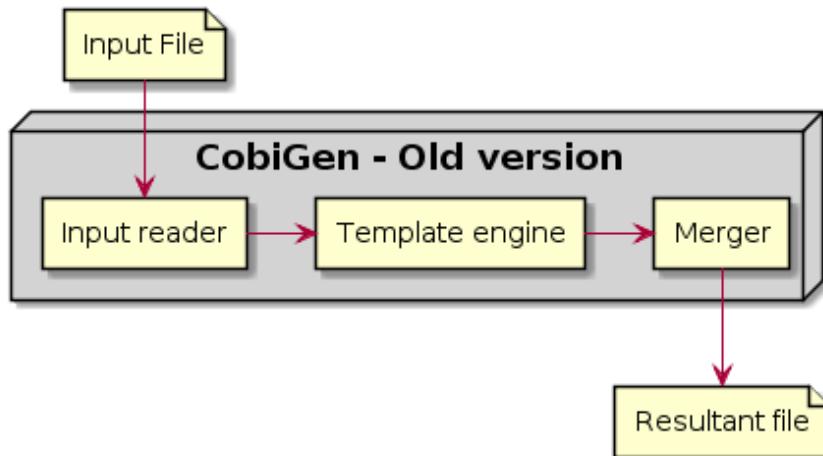
- Being able to parse a file in that language.
- Create a human readable model that can be used to generate templates (by retrieving properties from the model).

- Enable merging files, so that user's code does not get removed.

For the Java plug-in it was relatively easy. As you are inside the Java world, you can use multiple utilities or libraries in order to get the [AST](#) or to merge Java code. With this new feature, we wanted that behaviour to be possible for any programming language.

73.4.1. General intuition

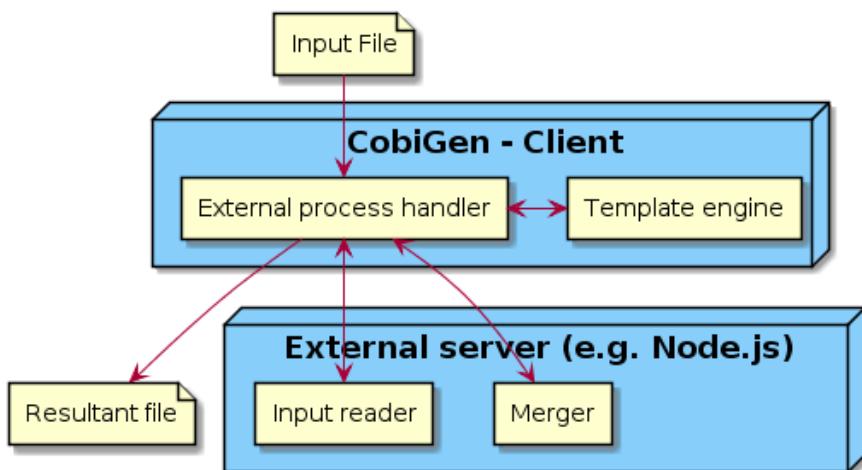
Below you will find a very high level description of how CobiGen worked in previous versions:



Basically, when a new input file was sent to CobiGen, it called the input reader to create a model of it (see [here](#) an example of a model). That model was sent to the template engine.

Afterwards, the template engine generated a new file which had to be merged with the original one. All this code was implemented in Java.

On the new version, we have implemented a handler ([ExternalProcessHandler](#)) which connects through TCP/IP connection to a server (normally on localhost:5000). This server can be implemented in any language (.Net, Node.js, Python...) it just needs to implement a REST API defined [here](#). The most important services are the input reading and merging:



CobiGen acts as a client that sends requests to the server in order to read the input file and create a model. The model is returned to the template engine so that it generates a new file. Finally, it is sent back to get merged with the original file.

73.4.2. How to create new external plug-in

The creation of a new plug-in consists mainly in three steps:

- Creation of the server (external process).
- Creation of a CobiGen plug-in.
- Creation of templates.

Server (external process)

The server can be programmed in any language that is able to implement REST services endpoints. The API that needs to implement is defined with [this contract](#). You can paste the content to <https://editor.swagger.io/> for a better look.

We have already created a NestJS server that implements the API defined above. You can find the code [here](#) which you can use as an example.

As you can see, the endpoints have the following naming convention: `processmanagement/todoplugin/nameOfService` where you will have to change `todo` to your plug-in name (e.g. rustplugin, pyplugin, goplugin...)

When implementing service `getInputModel` which returns a model from the input file there are only two restrictions:

- A `path` key must be added. Its value can be the full path of the input file or just the file name. It is needed because in CobiGen there is a `batch mode`, in which you can have multiple input objects inside the same input file. You do not need to worry about batch mode for now.
- On the root of your model, for each found key that is an object (defined with brackets `[{}]`), CobiGen will try to use it as an input object. For example, this could be a valid model:

```
{
  "path": "example/path/employee.entity.ts"
  "classes": [
    {
      "identifier": "Employee",
      "modifiers": [
        "export"
      ],
      "decorators": [
        {
          "identifier": {
            "name": "Entity",
            "module": "typeorm"
          },
          "isCallExpression": true
        }
      ],
      "properties": [
        {
          "identifier": "id",
          ...
          ...
          ...
        }]
    "interfaces": [
      ...
    ]
  }
}
```

For this model, CobiGen would use as input objects all the `classes` and `interfaces` defined. On the templates we would be able to do `model.classes[0].identifier` to get the class name. These input objects depend on the language, therefore you can use any key.

In order to test the server, you will have to deploy it on your local machine (localhost), default port is 5000. If that port is already in use, you can deploy it on higher port values (5001, 5002...). Nevertheless, we explain [later](#) the testing process as you need to complete the next step before.

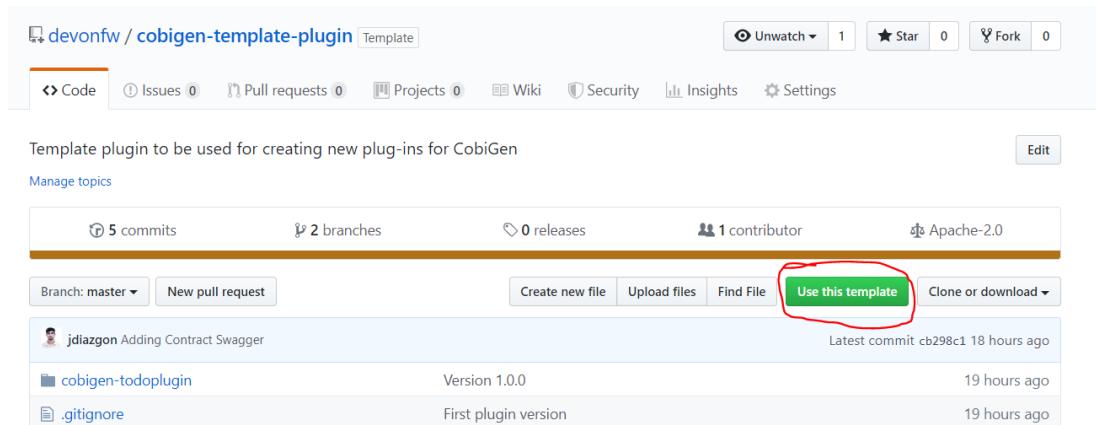


Your server must accept one argument when running it. The argument will be the port number (as an integer). This will be used for CobiGen in order to handle blocked ports when deploying your server. Check this [code](#) to see how we implemented that argument on our NestJS server.

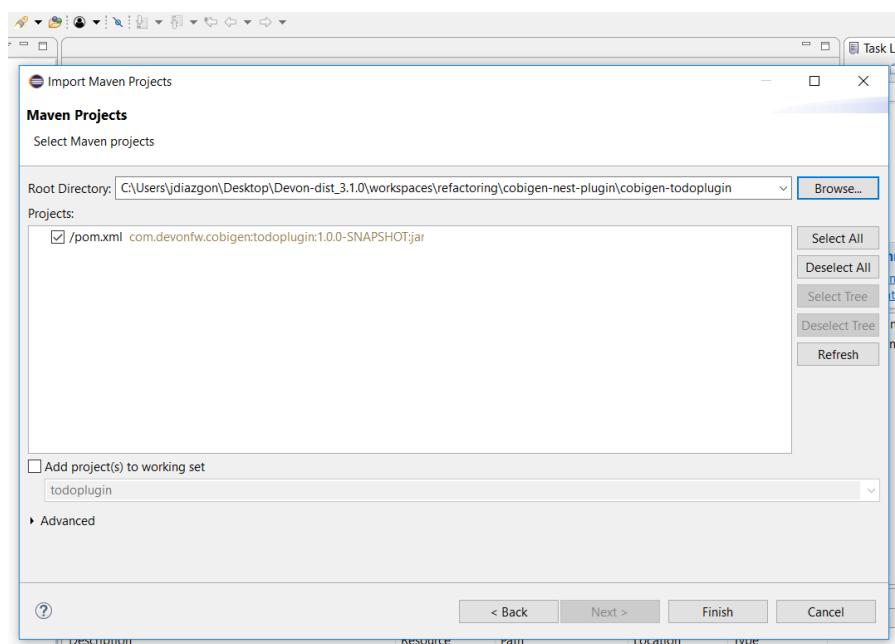
CobiGen plug-in

You will have to create a new CobiGen plug-in that connects to the server. But **do not worry**, you will not have to implement anything new. We have a CobiGen plug-in template available, the only changes needed are renaming files and setting some properties on the pom.xml. Please follow these steps:

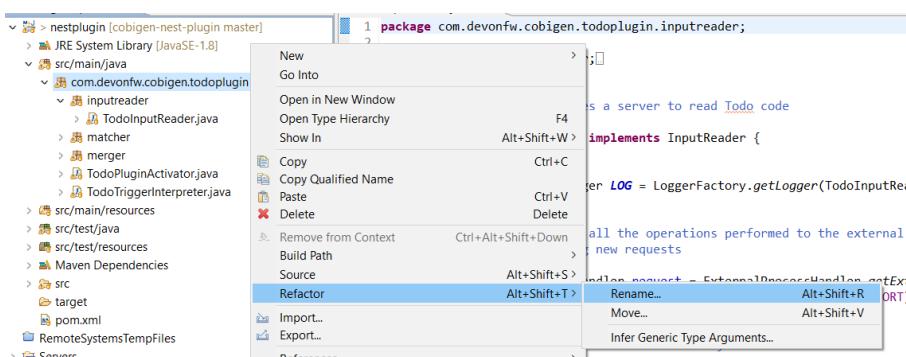
- Get the CobiGen plug-in template from [here](#). It is a template repository (new GitHub feature), so you can click on "Use this template" as shown below:



- Name your repo as **cobigen-name-plugin** where **name** can be python, rust, go... In our case we will create a **nest** plug-in. It will create a repo with only one commit which contains all the needed files.
- Clone your just created repo and import folder **cobigen-todoplugin** as a Maven project on any Java IDE, though we recommend you devonfw ;)

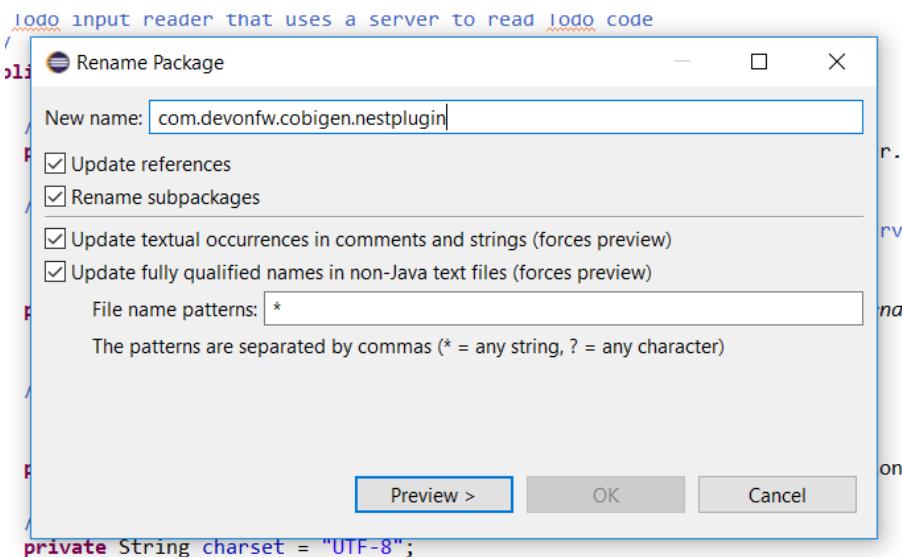


- Rename all the **todoPlugin** folders, files and class names to **nameplugin**. In our case **nestplugin**. In Eclipse you can easily rename by right clicking and then refactor → rename:



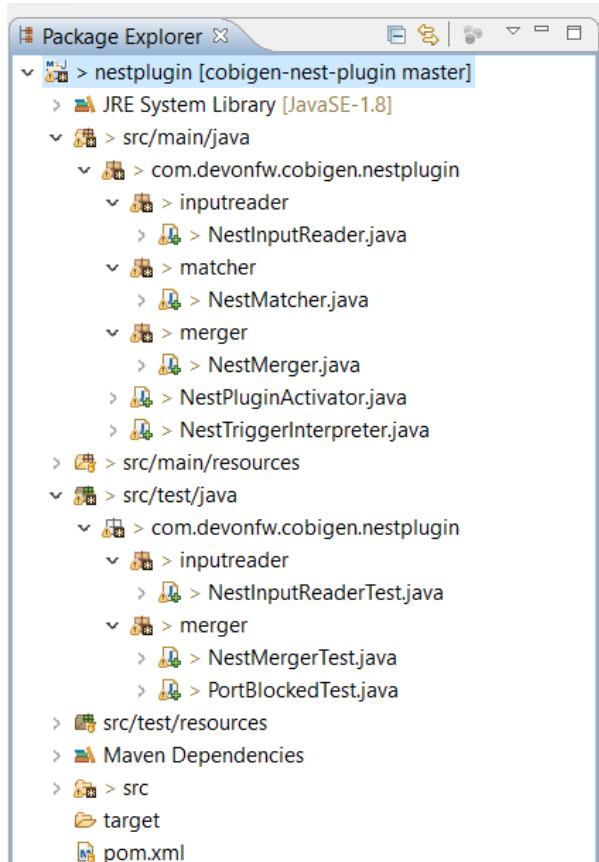


We recommend you to select all the checkboxes



```
private String charset = "UTF-8";
```

- Remember to change in `src/main/java` and `src/test/java` all the package, files and class names to use your plug-in name. The final result would be:



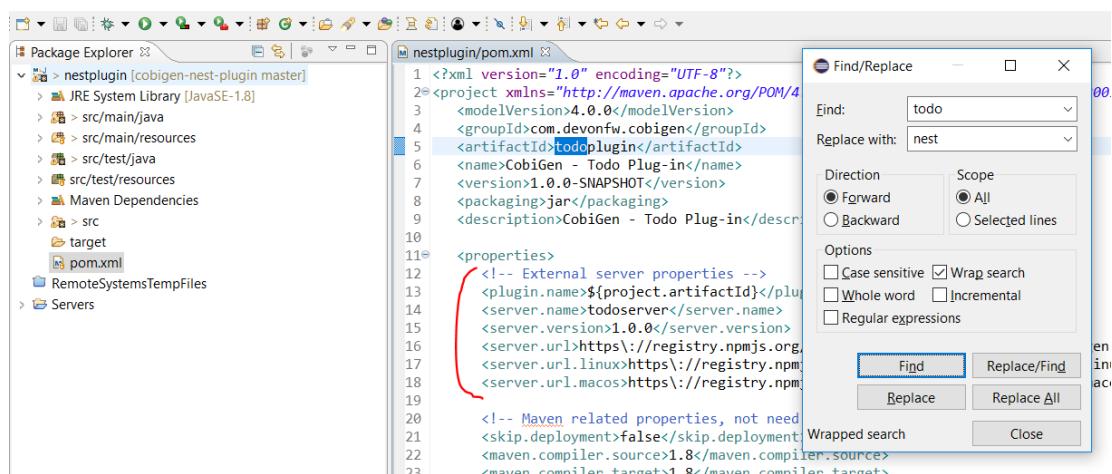
- Now we just need to change some strings, this is needed for CobiGen to register all the different plugins (they need unique names). In class `TodoPluginActivator` (in our case `NestPluginActivator`), change all the `todo` to your plug-in name. See below the 3 strings that need to be changed:

```

1 package com.devonfw.cobigen.nestplugin;
2
3* import java.util.List;[]
10
11 /**
12 * Todo Plug-in Activator to be registered in the PluginRegistry of CobiGen.
13 */
14 public class NestPluginActivator implements GeneratorPluginActivator {
15
16 /**
17 * Defines the trigger type
18 */
19 private static final String TRIGGER_TYPE = "nest";
20
21 @Override
22 public List<Merger> bindMerger() {
23
24     List<Merger> merger = Lists.newArrayList();
25     merger.add(new NestMerger("nestmerge", false));
26     merger.add(new NestMerger("nestmerge_override", true));
27
28     return merger;
}

```

- Finally, we will change some properties from the `pom.xml` of the project. These properties define the server (external process) that is going to be used:
 - Inside `pom.xml`, press `Ctrl + F` to perform a find and replace operation. Replace all `todo` with your plugin name:



- We are going to explain the server properties:
 - artifactId:** This is the name of your plug-in, that will be used for a future release on Maven Central.
 - plugin.name:** does not need to be changed as it uses the property from the `artifactId`. When connecting to the server, it will send a request to `localhost:5000/{plugin.name}plugin/isConnectionReady`, that is why it is important to use an unique name for the plug-in.
 - server.name:** This defines how the server executable (.exe) file will be named. This .exe file contains all the needed resources for deploying the server. You can use any name you want.
 - server.version:** You will specify here the server version that needs to be used. The .exe file will be named as `{server.name}-{server.version}.exe`.
 - server.url:** This will define from where to download the server. We **really** recommend you using NPM which is a package manager we know it works well. We explain [here](#)

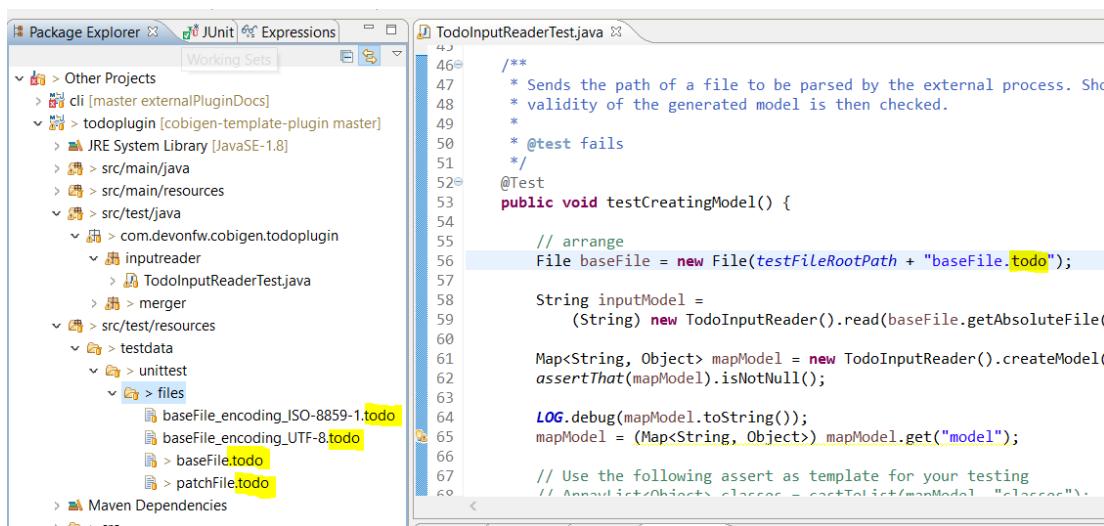
how to release the server on NPM. This will download the .exe file for Windows.

- vi. server.url.linux: Same as before, but this should download the .exe file for Linux systems. If you do not want to implement a Linux version of the plug-in, just use the same URL from Windows or MacOS.
- vii. server.url.macos: Same as before, but this should download the .exe file for MacOS systems. If you do not want to implement a MacOS version of the plug-in, just use the same URL from Linux or Windows.

73.4.3. Testing phase

Now that you have finished with the implementation of the server and the creation of a new CobiGen plug-in, we are going to explain how you can test that everything works fine:

1. Deploy the server on port 5000.
2. Run `mvn clean test` on the CobiGen-plugin or run the JUnit tests directly on Eclipse.
 - a. If the server and the plug-in are working properly, some tests will pass and other will fail (we need to tweak them).
 - b. If every test fails, something is wrong in your code.
3. In order to fix the failing tests, go to `src/test/java`. The failing tests make use of sample input files that we added in sake of example:



Replace those files (on `src/test/resources/testadata/unittest/files/…`) with the correct input files for your server.

73.4.4. Releasing

Now that you have already tested that everything works fine, we are going to explain how to release the server and the plug-in.

Release the server

We are going to use [NPM](#) to store the executable of our server. Even though NPM is a package manager for JavaScript, it can be used for our purpose.

- Get the CobiGen server template from [here](#). It is a template repository (new GitHub feature), so you can click on "Use this template" as shown below:

Template repository for CobiGen servers (external process capable of parsing and merging an input file)

Manage topics

2 commits 1 branch 0 releases 1 contributor Apache-2.0

Branch: master New pull request Create new file Upload files Find File Use this template Clone or download ▾

jdiazgon NPM ready to publish Latest commit 1d67133 2 minutes ago

cobigen-todo-server NPM ready to publish 2 minutes ago

.gitignore NPM ready to publish 2 minutes ago

- Name your repo as `cobigen-name-server` where `name` can be `python`, `rust`, `go...` In our case we will create a `nest` plug-in. It will create a repo with only one commit which contains all the needed files.
- Clone your just created repo and go to folder `cobigen-todo-server`. It will just contain two files: `ExternalProcessContract.yml` is the OpenAPI definition which you can modify with your own server definition (this step is optional), and `package.json` is a file needed for NPM in order to define where to publish this package:

```
{
  "name": "@devonfw/cobigen-todo-server",
  "version": "1.0.0",
  "description": "Todo server to implement the input reader and merger for CobiGen",
  "author": "CobiGen Team",
  "license": "Apache"
}
```

Those are the default properties. This would push a new package `cobigen-todo-server` on the `devonfw` organization, with version 1.0.0. We have no restrictions here, you can use any organization, though we always recommend `devonfw`.



Remember to change all the `todo` to your server name.

- Add your executable file into the `cobigen-todo-server` folder, just like below. As we said previously, this `.exe` is the server ready to be deployed.

```
cobigen-template-server/
 |- cobigen-todo-server/
   |- ExternalProcessContract.yml
   |- package.json
   |- todoserver-1.0.0.exe
```

- Finally, we have to publish to NPM. If you have never done it, you can follow this [tutorial](#). Basically you need to login into NPM and run:

```
cd cobigen-todo-server/
npm publish --access=public
```



To release Linux and MacOS versions of your plug-in, just add the suffix into the package name (e.g. [@devonfw/cobigen-todo-server-linux](#))

That's it! You have published the first version of your server. Now you just need to modify the properties defined on the pom of your CobiGen plug-in. Please see next section for more information.

Releasing CobiGen plug-in

- Change the pom.xml to define all the properties. You can see below a final example for **nest**:

```
...
<groupId>com.devonfw.cobigen</groupId>
<artifactId>nestplugin</artifactId>
<name>CobiGen - Nest Plug-in</name>
<version>1.0.0</version>
<packaging>jar</packaging>
<description>CobiGen - nest Plug-in</description>

<properties>
    <!-- External server properties -->
    <plugin.name>devonfw-guide-tutorial-sources-wiki</plugin.name>
    <server.name>nestserver</server.name>
    <server.version>1.0.0</server.version>
    <server.url>https://registry.npmjs.org/@devonfw/cobigen-nest-server-/
    /cobigen-nest-server-${server.version}.tgz</server.url>
    <server.url.linux>https://registry.npmjs.org/@devonfw/cobigen-nest-server-
    linux-/cobigen-nest-server-linux-${server.version}.tgz</server.url.linux>
    <server.url.macos>https://registry.npmjs.org/@devonfw/cobigen-nest-server-
    macos-/cobigen-nest-server-macos-${server.version}.tgz</server.url.macos>
    ...

```

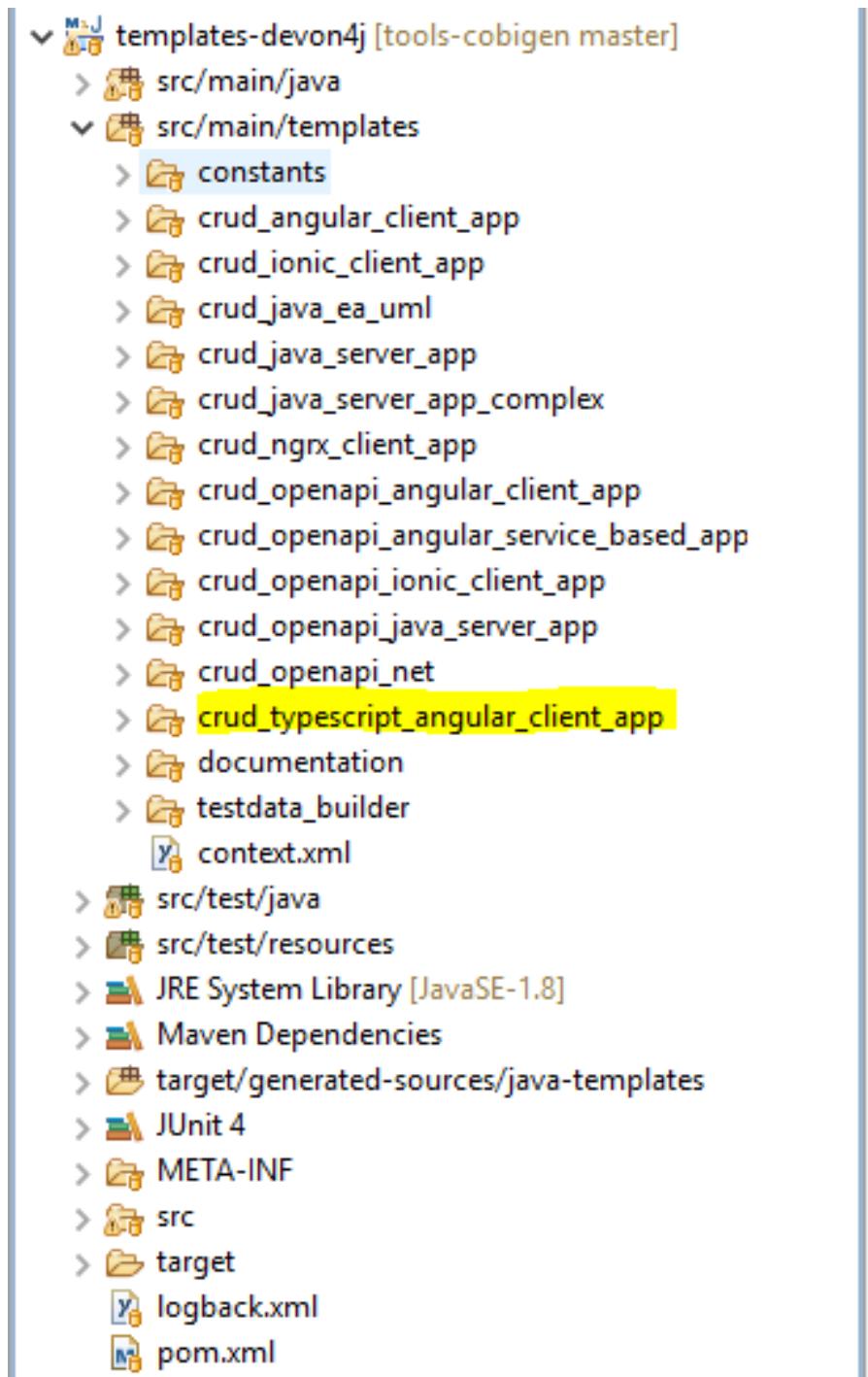
- Deploy to Maven Central.

73.4.5. Templates creation

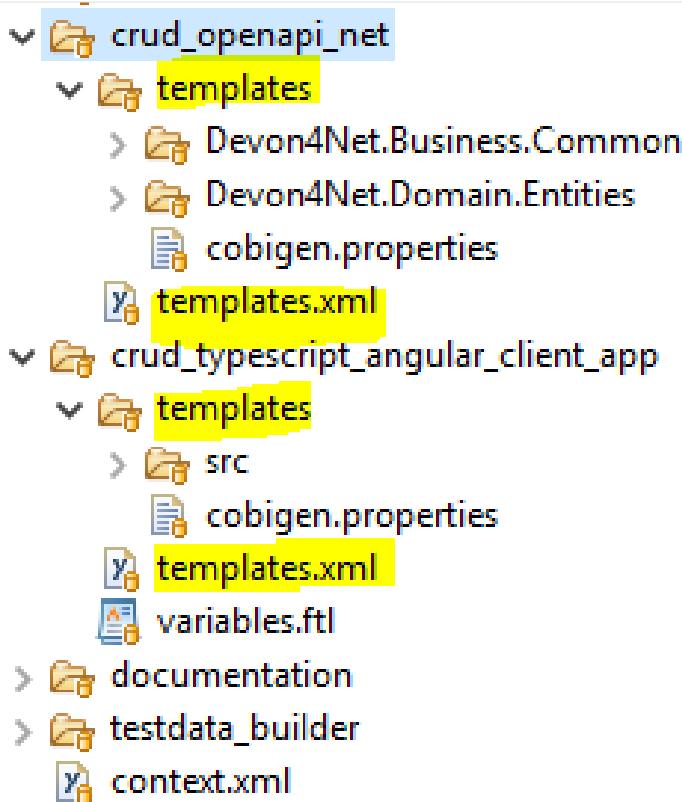
After following above steps, we now have a CobiGen plug-in that connects to a server (external process) which reads your input files, returns a model and is able to merge files.

However, we need a key component for our plug-in to be useful. We need to define templates:

- Fork our CobiGen main repository, from [here](#) and clone it into your PC. Stay in the **master** branch and import into your IDE `cobigen-templates\templates-devon4j`. Set the Java version of the project to 1.8 if needed.
- Create a new folder on `src/main/templates`, this will contain all your templates. You can use any name, but please use underscores as separators. In our case, we created a folder `crud_typescript_angular_client_app` to generate an Angular client from a TypeORM entity (NodeJS entity).



- Inside your folder, create a `templates` folder. As you can see below, the folder structure of the generated files starts here (the sources). Also we need a configuration file `templates.xml` that should be on the same level as `templates/` folder. For now, copy and paste a `templates.xml` file from any of the templates folder.



- Start creating your own templates. Our default templates language is Freemarker, but you can also use Velocity. Add the extension to the file (**.ftl**) and start developing templates! You can find useful documentation [here](#).
- After creating all the templates, you need to modify **context.xml** which is located on the root of **src/main/templates**. There you need to define a trigger, which is used for CobiGen to know when to trigger a plug-in. I recommend you to copy and paste the following trigger:

```
<trigger id="crud_typescript-angular-client-app" type="nest" templateFolder="crud_typescript-angular-client-app">
    <matcher type="fqn" value="([^\.]+).entity.ts">
        <variableAssignment type="regex" key="entityName" value="1"/>
        <variableAssignment type="regex" key="component" value="1"/>
        <variableAssignment type="constant" key="domain" value="demo"/>
    </matcher>
</trigger>
```

- Change **templateFolder** to your templates folder name. **id** you can use any, but it is recommendable to use the same as the template folder name. **type** is the **TRIGGER_TYPE** we defined above on the **NestPluginActivator** class. On **matcher** just change the **value**: **([^\.]+).entity.ts** means that we will only accept input files that contain "anyString.entity.ts". This improves usability, so that users only generate using the correct input files. You will find more info about **variableAssignment** [here](#).
- Finally, is time to configure **templates.xml**. It is needed for organizing templates into increments, please take a look into this [documentation](#).

Testing templates

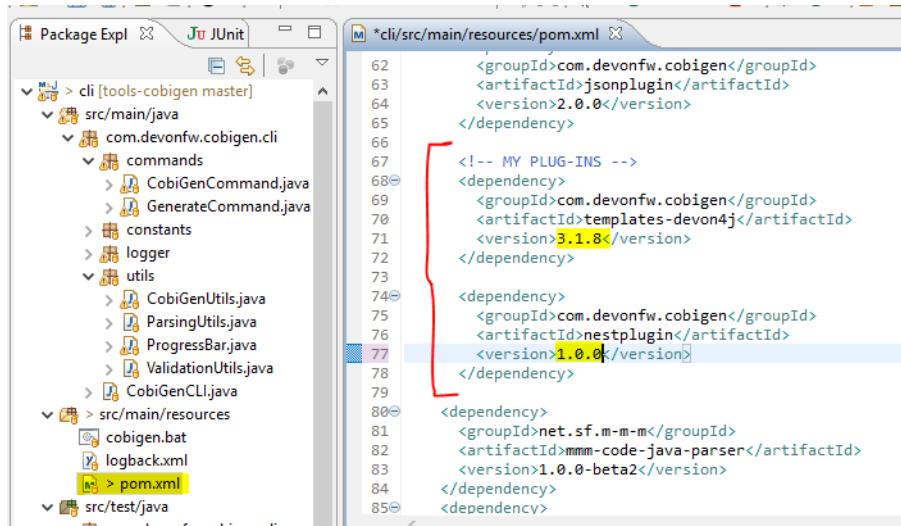
- When you have finished your templates you will like to test them. On the templates-devon4j `pom.xml` remove the SNAPSHOT from the version (in our case the version will be 3.1.8). Run `mvn clean install -DskipTests` on the project. We skip tests because you need special permissions to download artifacts from our Nexus. Remember the version that has just been installed:

```
[INFO] --- maven-install-plugin:2.4:install (default-install) @ templates-devon4j ---
[INFO] Installing C:\Users\jdiazgon\Desktop\HDD\cobigenDist\tools-cobigen\cobigen-templates\templates-devon4j\target\templates-devon4j-3.1.8.jar to C:\Users\jdiazgon\Desktop\Devon-dist-3.1.0\conf.m2\repository\com\devonfw\cobigen\templates-devon4j\3.1.8\templates-devon4j-3.1.8.jar
[INFO] Installing C:\Users\jdiazgon\Desktop\HDD\cobigenDist\tools-cobigen\cobigen-templates\templates-devon4j\flattened-pom.xml to C:\Users\jdiazgon\Desktop\Devon-dist-3.1.0\conf.m2\repository\com\devonfw\cobigen\templates-devon4j\3.1.8\templates-devon4j-3.1.8-pom.xml
[INFO] Installing C:\Users\jdiazgon\Desktop\HDD\cobigenDist\tools-cobigen\cobigen-templates\templates-devon4j\target\templates-devon4j-3.1.8-sources.jar to C:\Users\jdiazgon\Desktop\Devon-dist-3.1.0\conf.m2\repository\com\devonfw\cobigen\templates-devon4j\3.1.8\templates-devon4j-3.1.8-sources.jar
[INFO] Installing C:\Users\jdiazgon\Desktop\HDD\cobigenDist\tools-cobigen\cobigen-templates\templates-devon4j\target\templates-devon4j-3.1.8-javadoc.jar to C:\Users\jdiazgon\Desktop\Devon-dist-3.1.0\conf.m2\repository\com\devonfw\cobigen\templates-devon4j\3.1.8\templates-devon4j-3.1.8-javadoc.jar
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 01:13 min
[INFO] Finished at: 2019-09-20T13:39:15+02:00
[INFO]
```



We always recommend using the devonfw console, which already contains a working Maven version.

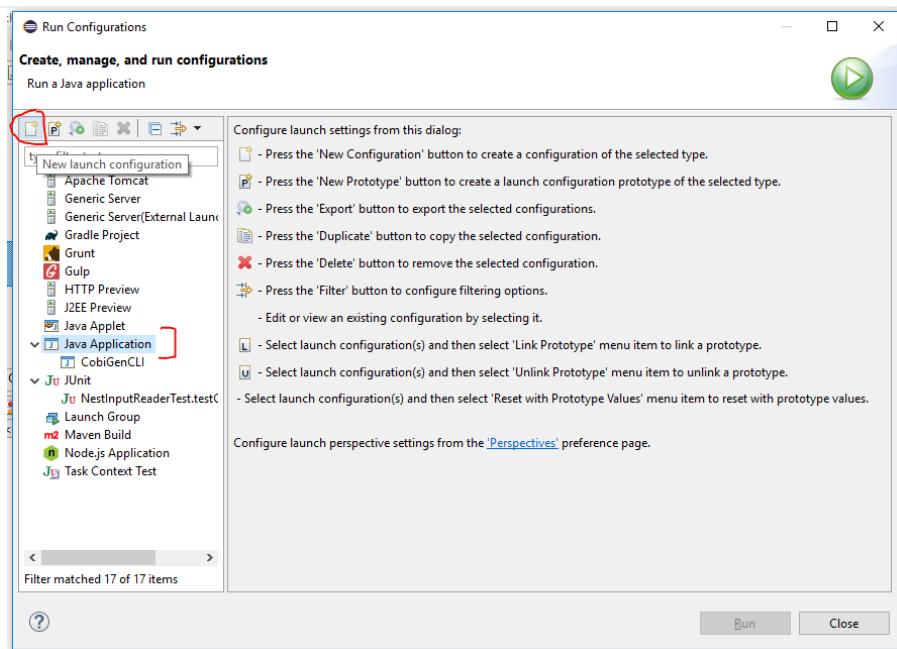
- Now we have your last version of the templates ready to be used. We need to use that latest version in CobiGen. We will use the CobiGen CLI that you will find in your cloned repo, at `cobigen-cli/cli`. Import the project into your IDE.
- Inside the project, go to `src/main/resources/pom.xml`. This pom.xml is used on runtime in order to install all the CobiGen plug-ins and templates. Add there your latest templates version and the previously created plug-in:



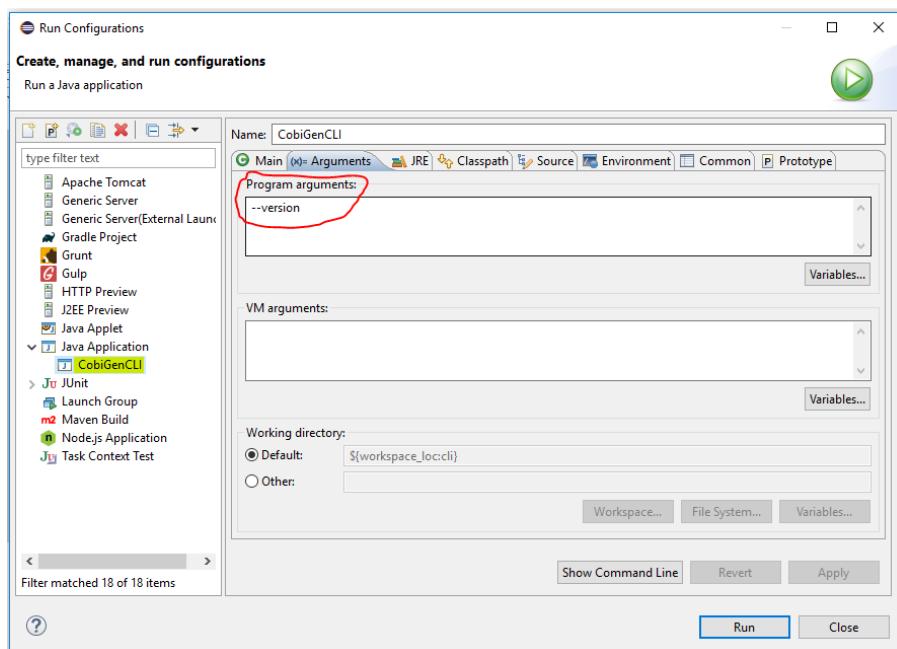
- Afterwards, run `mvn clean install -DskipTests` and CobiGen will get your plug-ins. Now you have three options to test templates:

- Using Eclipse run as:

- Inside Eclipse, you can run the CobiGen-CLI as a Java application. Right click class `CobiGenCLI.java` → run as → run configurations... and create a new Java application as shown below:



- That will create a **CobiGenCLI** configuration where we can set arguments to the CLI. Let's first begin with showing the CLI version, which should print a list of all plug-ins, including ours.



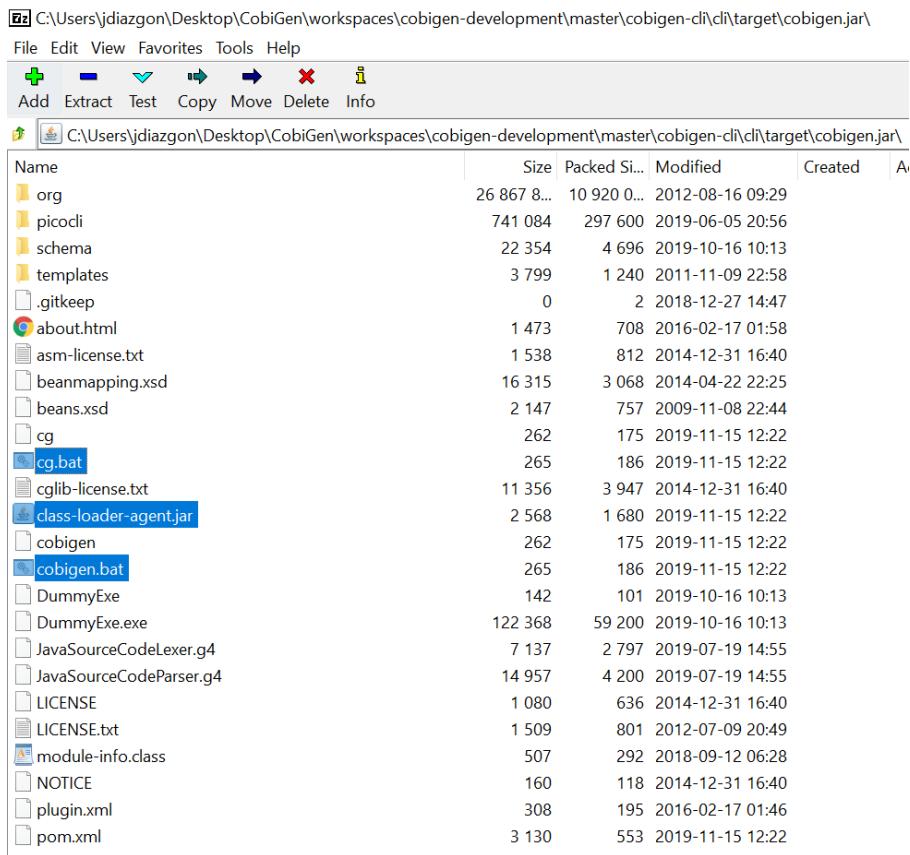
```
...
name:= propertyplugin version = 2.0.0
name:= jsonplugin version = 2.0.0
name:= templates-devon4j version = 3.1.8
name:= nestplugin version = 1.0.0
...
```

- If that worked, now you can send any arguments to the CLI in order to generate with your templates. Please follow [this guide](#) that explains all the CLI commands.
- Modify the already present JUnit tests on the CLI project: They test the generation of

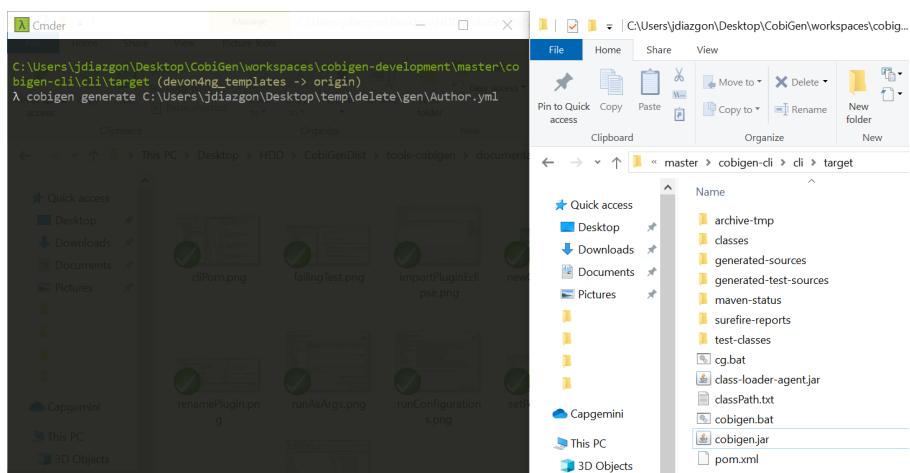
templates from multiple plug-ins, you can add your own tests and input files.

3. Use the CLI jar to execute commands:

1. The `mvn clean install -DskipTests` command will have created a Cobigen.jar inside your target folder (`cobigen-cli/cli/target`). Open the jar with any unzipped and extract to the current location `class-loader-agent.jar`, `cobigen.bat` and `cg.bat`:



2. Now you can run any CobiGen CLI commands using a console. [This guide](#) explains all the CLI commands.



73.4.6. devon4net Cobigen Guide

Overview

In this guide we will explain how to generate a new WebApi project from an OpenAPI 3.0.0

specification. This means that we are going to use a “contract first” strategy. This is going to be possible due to these type of files that contain all the information about entities, operations, etc...

In order to make it work we are using [CobiGen](#), a powerful tool for generating source code. CobiGen allows users to generate all the structure and code of the components, helping to save a lot of time otherwise wasted on repetitive tasks.

Getting things ready

devonfw-ide

First, we will install the devonfw-ide. It is a tool that will setup your IDE within minutes. Please follow the install guide [here](#).

devon4net Templates

We are going to use the template of devon4net as a base to generate all the code, so what we have to do now is to download said template using the following steps.

First of all you have to set up all the environment for .NET, you can do this using [the following tutorial](#). Next we are going to create a new folder where we want to have the WebAPI project, lastly we are going to open the terminal there.

Type the following:

```
dotnet new -i Devon4Net.WebAPI.Template
```

and then:

```
dotnet new Devon4NetAPI
```

OpenAPI File

In order to let CobiGen generate all the files, we first have to make some modifications to our OpenAPI file.

It is obligatory to put the “*x-rootpackage*” tag to indicate where CobiGen will place the generated files as well as the “*x-component*” tags for each component, keep in mind that due to CobiGen’s limitations each component **must** have its own entity.

You can read more information about how to configure your OpenAPI file and a working example [here](#).

Generating files

Cobigen allow us to generate the files in two different ways. One of them is using Eclipse which it can be done by using the its grafical interface. The other way to generate the code is using the Cobigen CLI tool.

Generating files through Eclipse

In order to generate the files using Eclipse we need to follow some simple steps.

First we are going to import our basic devon4net WebAPI Project into Eclipse. to do so open Eclipse with the “eclipse-main.bat” file that can be found in the devon distribution root folder. Once we are inside of Eclipse we go to **File > Open projects from file system...** and, under "Directory", search for your project.

Next we copy our OpenAPI file into the root folder of the project.

And then we right click on OpenAPI file and then select **CobiGen > Generate...** It will display a window like this:

To select all .NET features choose **CRUD devon4net Server** otherwise you can select only those that interest you.

Once you select all the files that you want to generate, click on the “*Finish*” button to generate all the source code.

Generating files through Cobigen CLI

In order to generate the files using the Cobigen CLI it is needed to do the following steps:

1. Go to devonfw distribution folder
2. Run **console.bat**, this will open a console.
3. Go to the folder you downloaded the **devon4net template** and your **yml** file.
4. Run the command:

```
cobigen generate {yourOpenAPIfile}.yml
```

5. A list of increments will be printed so that you can start the generation. It has to be selected **CRUD devon4net Server** increment.

Configuration

Data base

Cobigen is generating an empty context that has to be filled with manually in order to be able to work with the database. The context can be found in **[Project_Name]/Devon4Net.WebAPI.Implementation/Domain/Database/CobigenContext.cs**.

Run the application

After the configuration of the database, open a terminal in path:

[Project_Name]/Devon4Net.Application.WebAPI and then type:

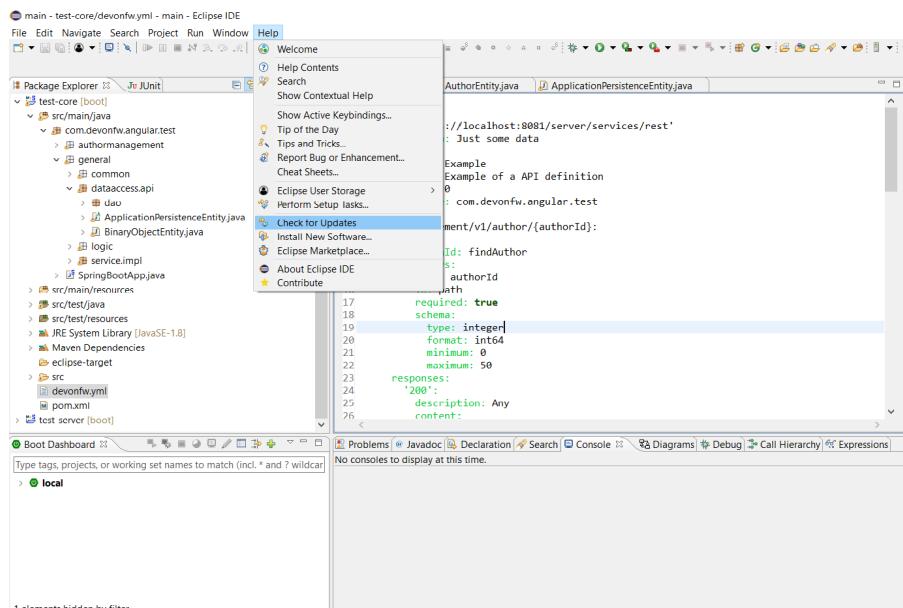
```
dotnet run
```

This will deploy our application in our localhost with the port 8082, so when you click [here](https://localhost:8082/swagger) (<https://localhost:8082/swagger>) you can see, in swagger, all the services and the data model.

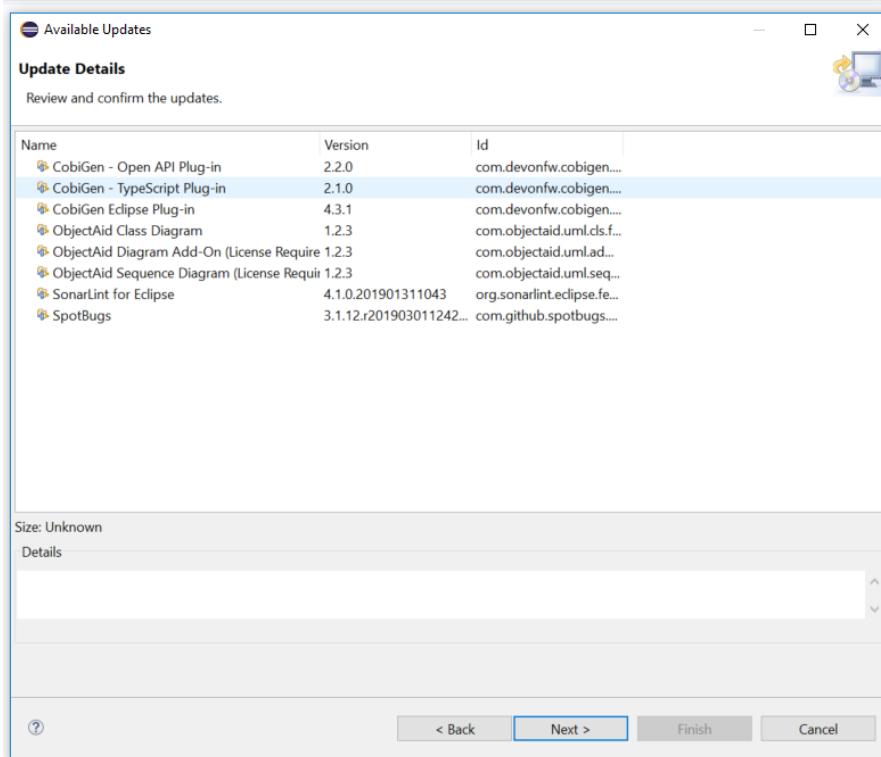
73.5. How to update CobiGen

In order to update CobiGen from our devonfw distribution, we have two options:

- Open Eclipse, click on *Help* → *Check for updates*

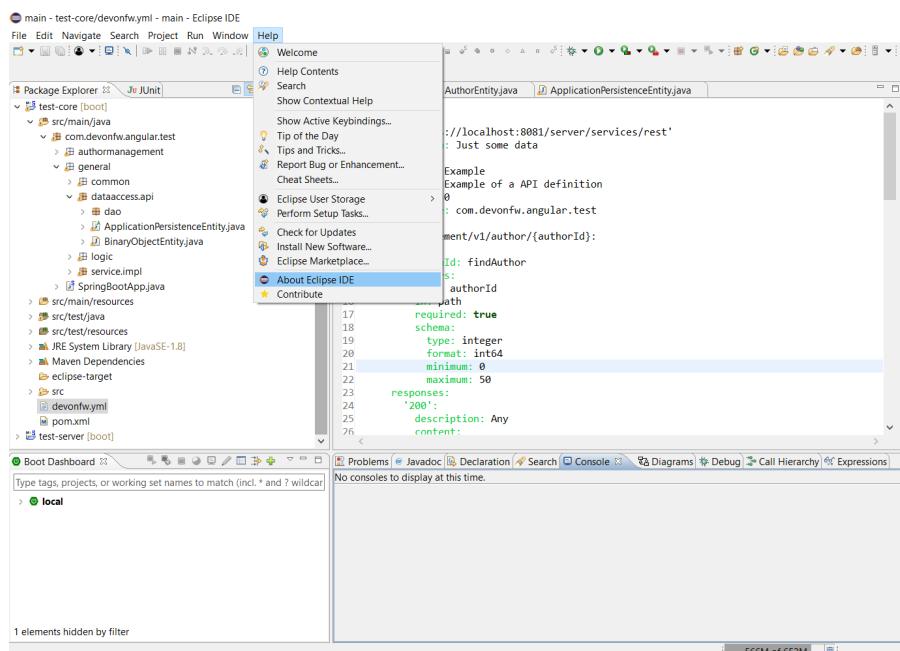


- Select all the CobiGen plugins listed and click on *Next*.

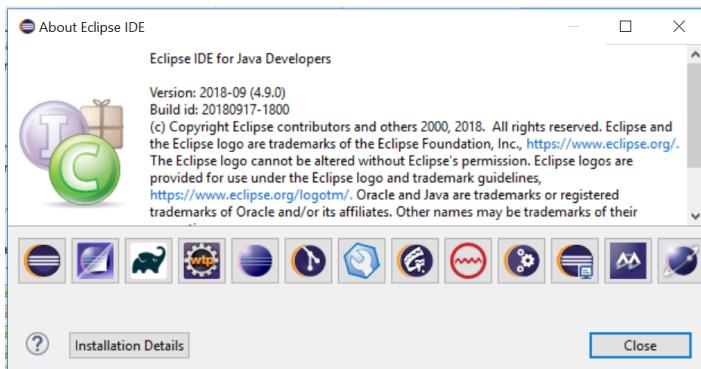


If this option is not working properly, then you can try the second option:

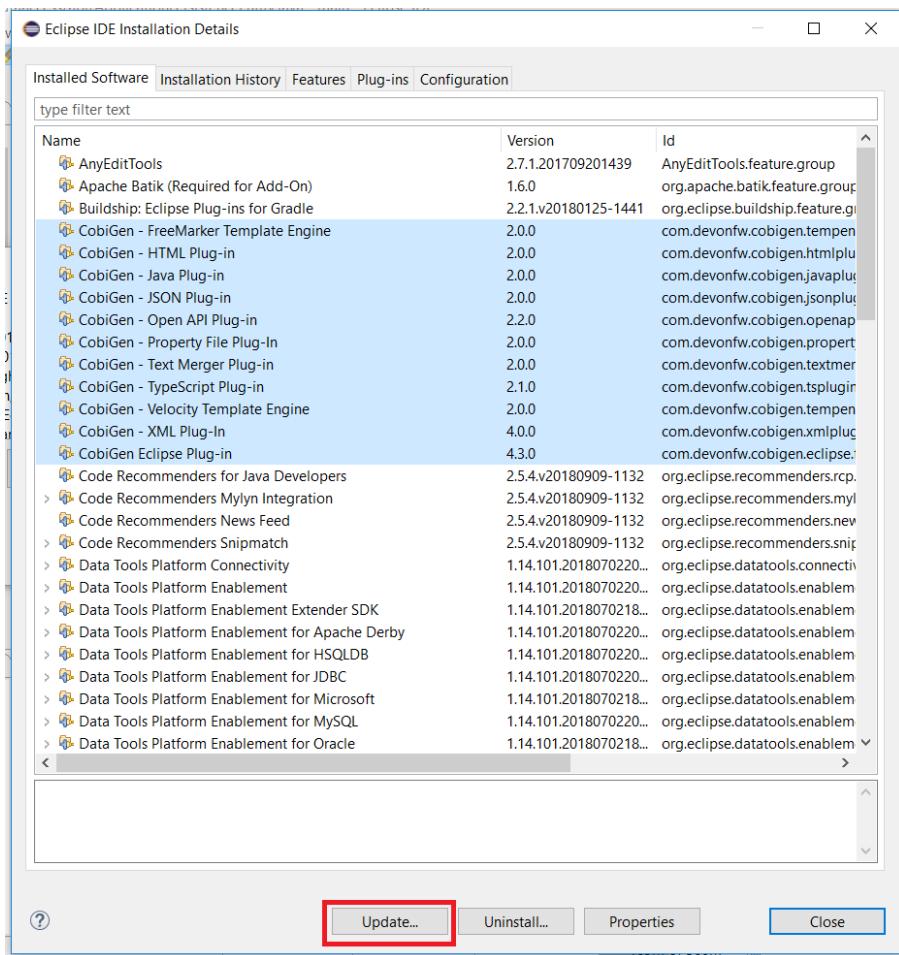
- Open Eclipse, click on *Help* → *About Eclipse IDE*:



- Click on *Installation details*:



- Select all the CobiGen plugins and click on *Update*:

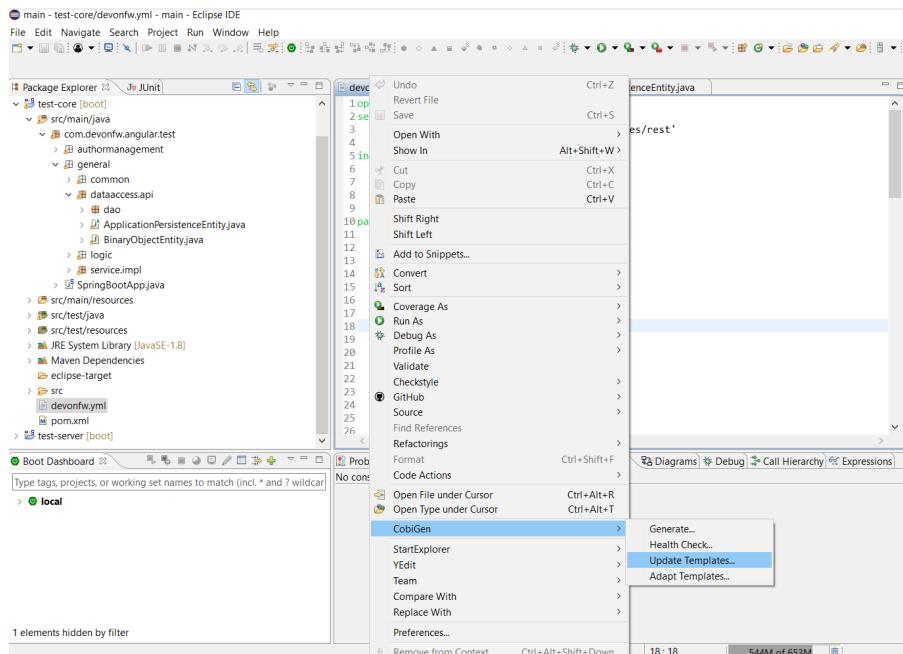


After the update process finishes, remember to restart Eclipse.

73.5.1. Updating templates:

To update your CobiGen templates to the latest version, you just need to do one step:

- Right click any file on your package explorer, click on *CobiGen* → *Update templates*, then click on *download*:



Now you will have the latest templates ready!

73.6. CobiGen Release creation

In this guide we explain how to create CobiGen related releases, i.e. release of a new core version using our useful release automation [script](#).

73.6.1. Usage

Fire up a command prompt from the CobiGen IDE environment (using [console.bat](#) for example). Then, you will need to execute the following command:

```
python "<path_to_release_script_parent_folder>/create_release.py" -d -g devonfw/cobigen -r "<path_of_your_just_cloned_fork>" -k "yourcapgemini@mail.com" -c
```



The CobiGen development environment comes with all required python packages needed for the release script. However, if you encounter errors like [no module named xyz found](#) you might want to consider running the following command:

```
python -m pip install -r "<path_to_release_script_parent_folder>/requirements.txt"
```

73.7. End to End POC Code generation using OpenAPI

This article helps to create a sample application using cobigen.

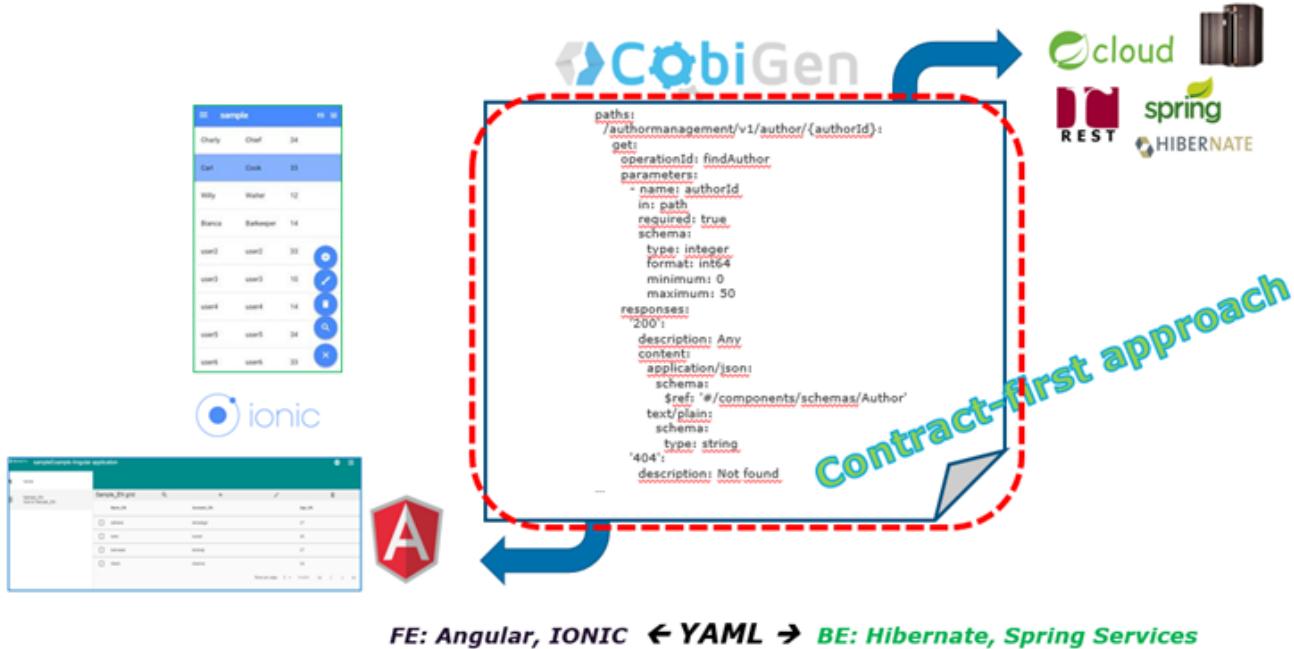
73.7.1. Prerequisites

Download and install devonfw IDE [here](#),

73.7.2. Steps to create a Sample Project using Cobigen

The HOW_TO is divided in 2 parts:

1. BE-Back End generator (DB + DAO + services) – CONTRACT FIRST APPROACH
2. FE-Front End generator (Web App Angular + Ionic App) – CONTRACT FIRST APPROACH



So, ready to go! We're going to star

t from the BE part ...

Back End

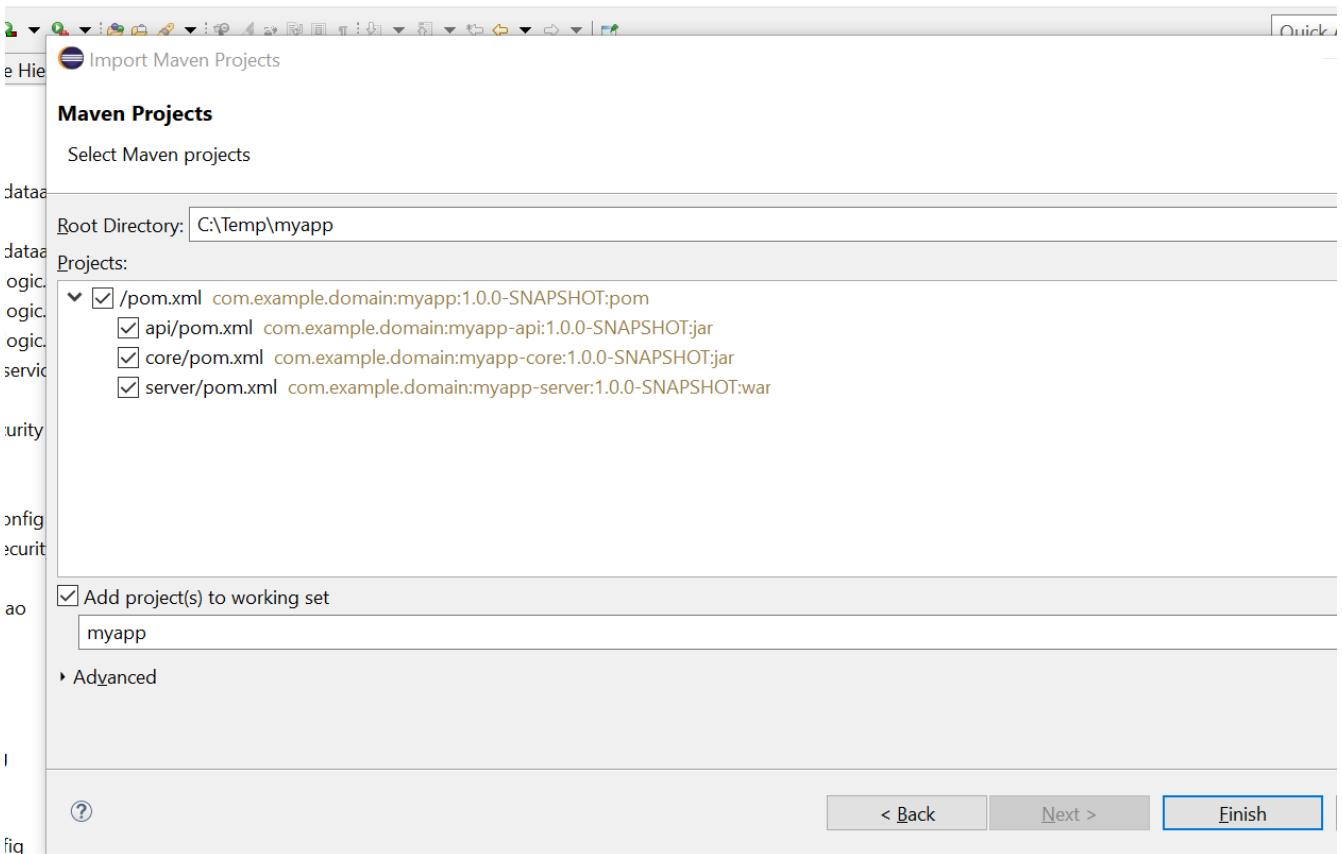
run \devonfw-ide-scripts-3.2.4\eclipse-main.bat

It will open eclipse

create a project using below command from the command prompt

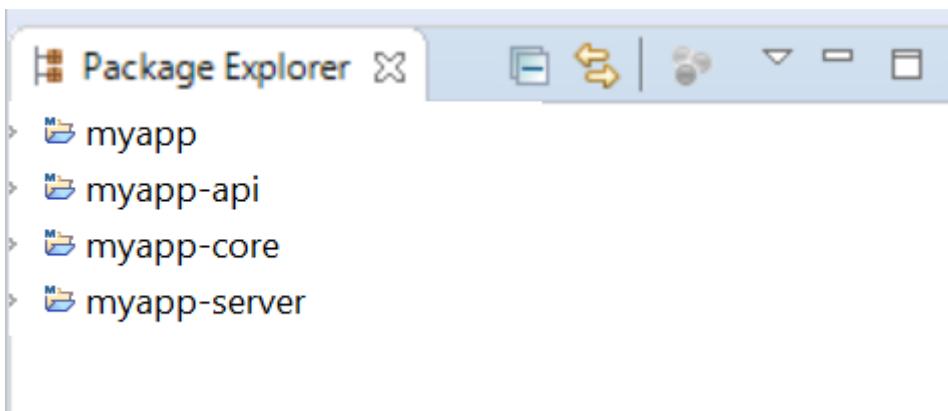
```
devon java create com.example.domain.myapp
```

Import the project to eclipse as maven project



Click **FINISH**

Now We have the following 4 projects.



BEFORE to start to create an Entity class, remember to create the tables !

7. Create a new **SQL file** (i.e: `V0005CreateTables_ItaPoc.sql`) inside `jwtsample-__core` and insert the following script:

```
CREATE TABLE EMPLOYEE (
    id BIGINT auto_increment, modificationCounter *INTEGER* *NOT* *NULL*, 
    employeeid BIGINT auto_increment,
    name VARCHAR(255),
    surname VARCHAR(255),
    email VARCHAR(255),
    PRIMARY KEY (employeeid)
);
```

WARNING: please note that there are 2 underscore in the name !

The left side shows a file tree for a project named 'myapp-core'. It includes 'src/main/java' and 'src/main/resources' directories. Under 'src/main/resources/db/migration/type/h2', there are several SQL files: V0001_Create_Sequence.sql, V0002_Create_RevInfo.sql, V0003_Create_BinaryObject.sql, and V0005_CreateTables_ItaPoc.sql. The right side shows a SQL editor window with the following content:

```
Type: [dropdown] Name: [dropdown] Database: [dropdown] Status: [dropdown]
1 CREATE TABLE EMPLOYEE (
2     id BIGINT auto_increment, modificationCounter INTEGER NOT NULL,
3     employeeid BIGINT auto_increment,
4     name VARCHAR(255),
5     surname VARCHAR(255),
6     email VARCHAR(255),
7     PRIMARY KEY (employeeid)
8 );
9 |
```

- Now create another SQL file (i.e: V0006_PopulateTables-ItaPoc.sql) and add following script about the INSERT in order to populate the table created before

WARNING: please note that there are 2 underscore in the name !

```
INSERT INTO EMPLOYEE (id, modificationCounter, employeeid, name, surname, email) VALUES
(1, 1, 1, 'Stefano', 'Rossini', 'stefano.rossini@capgemini.com');
INSERT INTO EMPLOYEE (id, modificationCounter, employeeid, name, surname, email) VALUES
(2, 2, 2, 'Angelo', 'Muresu', 'angelo.muresu@capgemini.com');
INSERT INTO EMPLOYEE (id, modificationCounter, employeeid, name, surname, email) VALUES
(3, 3, 3, 'Jaime', 'Gonzalez', 'jaime.diaz-gonzalez@capgemini.com');
```

The left side shows the same file tree as the previous screenshot. The right side shows a SQL editor window with the following content:

```
Type: [dropdown] Name: [dropdown] Database: [dropdown] Status: Disconnected, Auto Commit
1 INSERT INTO EMPLOYEE (id, modificationCounter, employeeid, name, surname, email) VALUES (1, :^
2 INSERT INTO EMPLOYEE (id, modificationCounter, employeeid, name, surname, email) VALUES (2, :^
3 INSERT INTO EMPLOYEE (id, modificationCounter, employeeid, name, surname, email) VALUES (3, :^
4 |
```

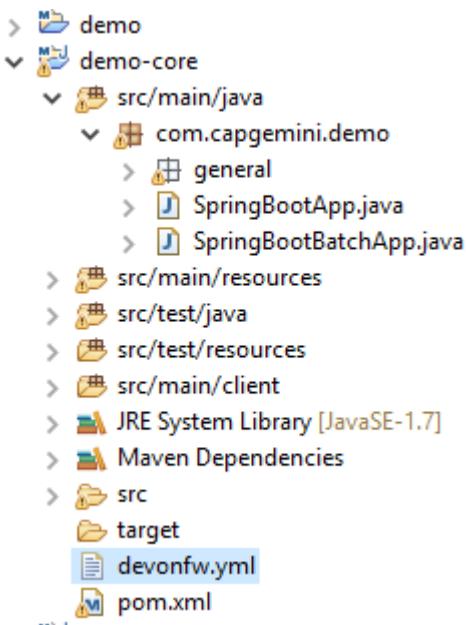
A new file 'V0006_PopulateTables-ItaPoc.sql' has been added to the 'h2' directory.

Let's create the yml file for the code generation

- Now create a new file *devonfw.yml* in the root of your core folder. This will be our OpenAPI contract, like shown below. Then, copy the contents of [this file](#) into your OpenAPI. It defines some REST service endpoints and a *EmployeeEntity* with its properties defined.

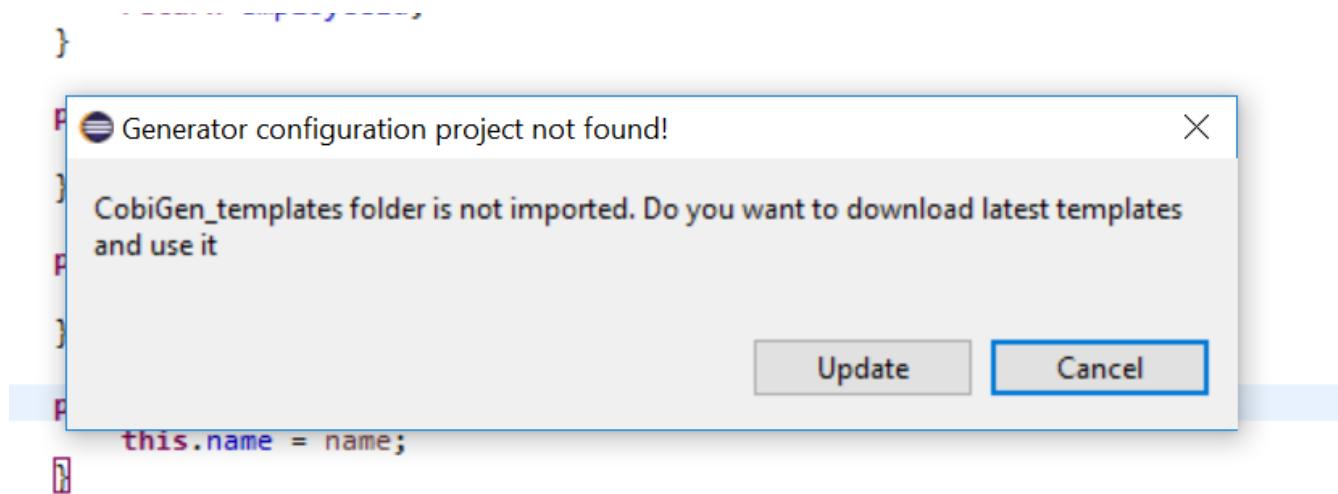
Important: if you want to know how to write an OpenAPI contract compatible with CobiGen,

please read [this tutorial](#).



10. Right click *devonfw.yml*. CobiGen → Generate

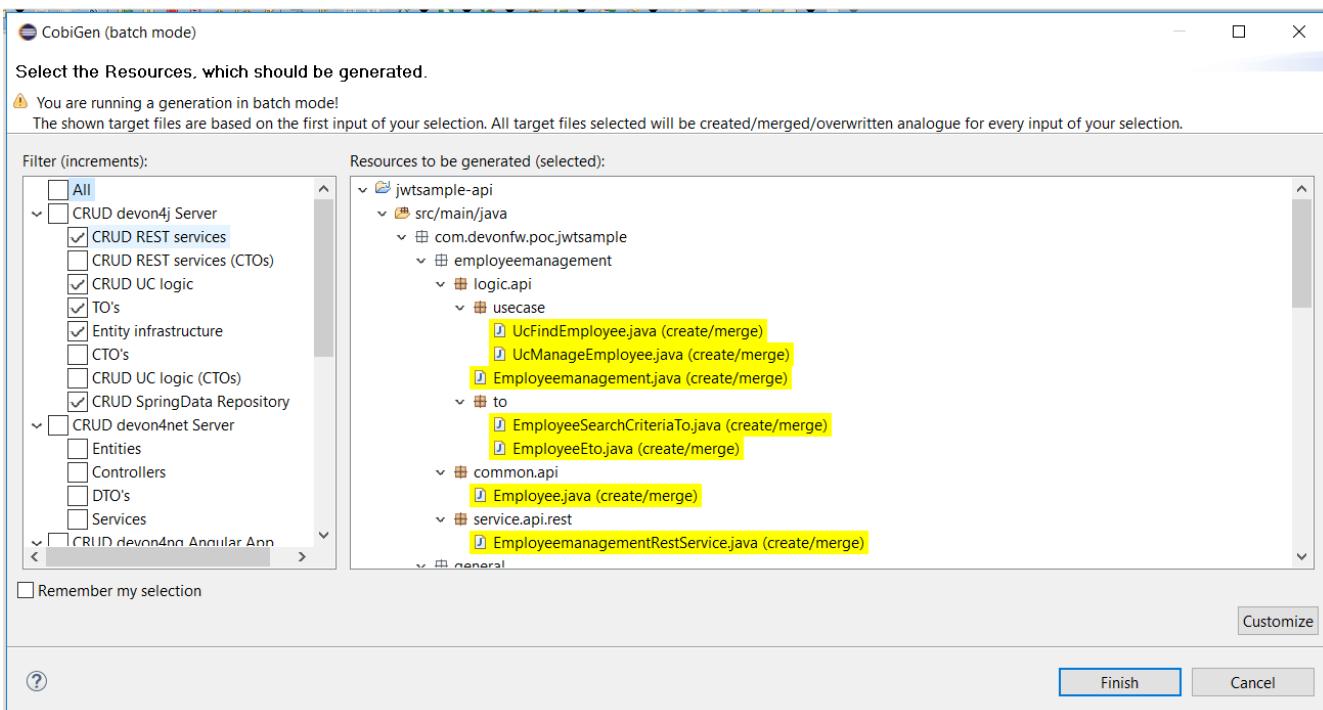
It will ask you to download the templates, click on *update*:



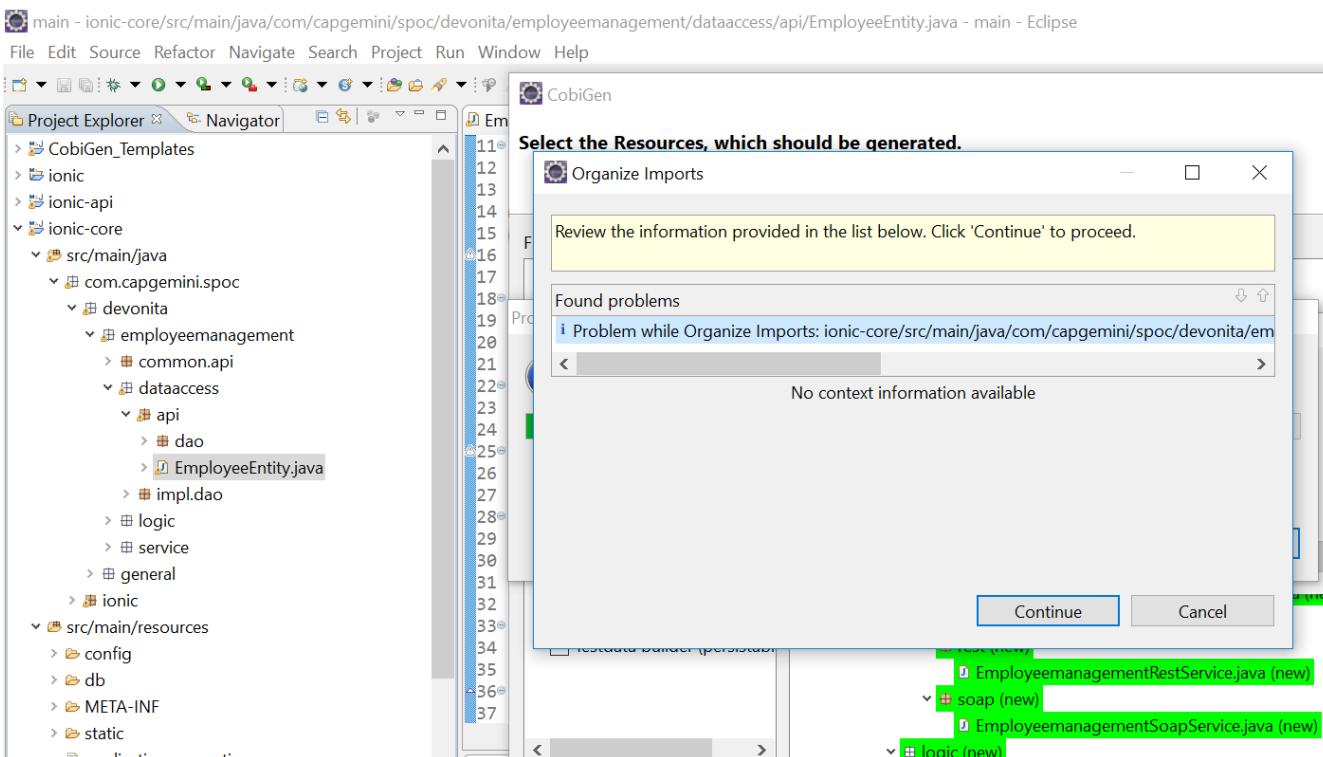
It will automatically download the latest version of *CobiGen_Templates*.

Attention: If you want to adapt the *CobiGen_Templates*, (normally this is not necessary), you will find at the end of this document a tutorial on how to import them and adapt them!

11. Click on all the option selected as below:

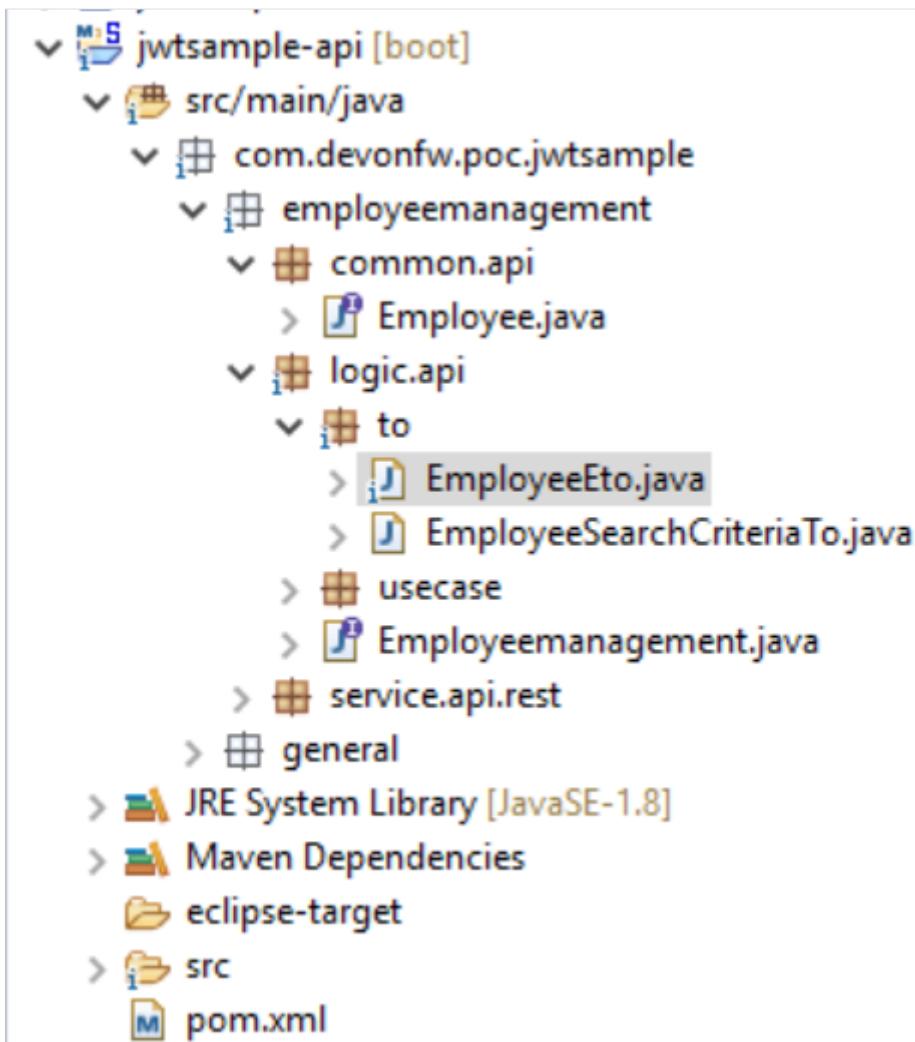


12. Click on finish. Below Screen would be seen. Click on continue

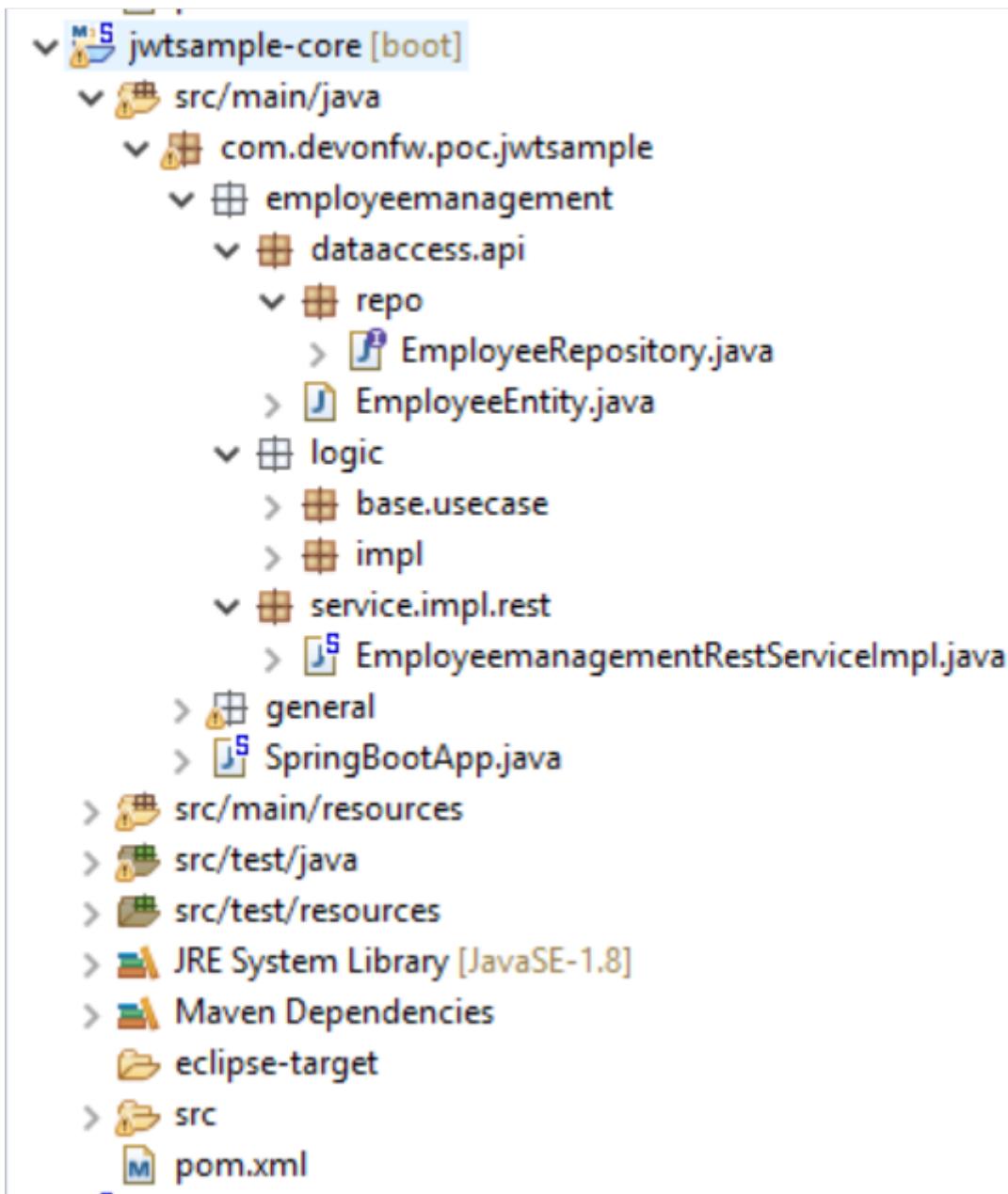


The entire BE layer structure having CRUD operation methods will be auto generated.

Some classes will be generated on the api part (*jwtsample-api*), normally it will be interfaces, as shown below:

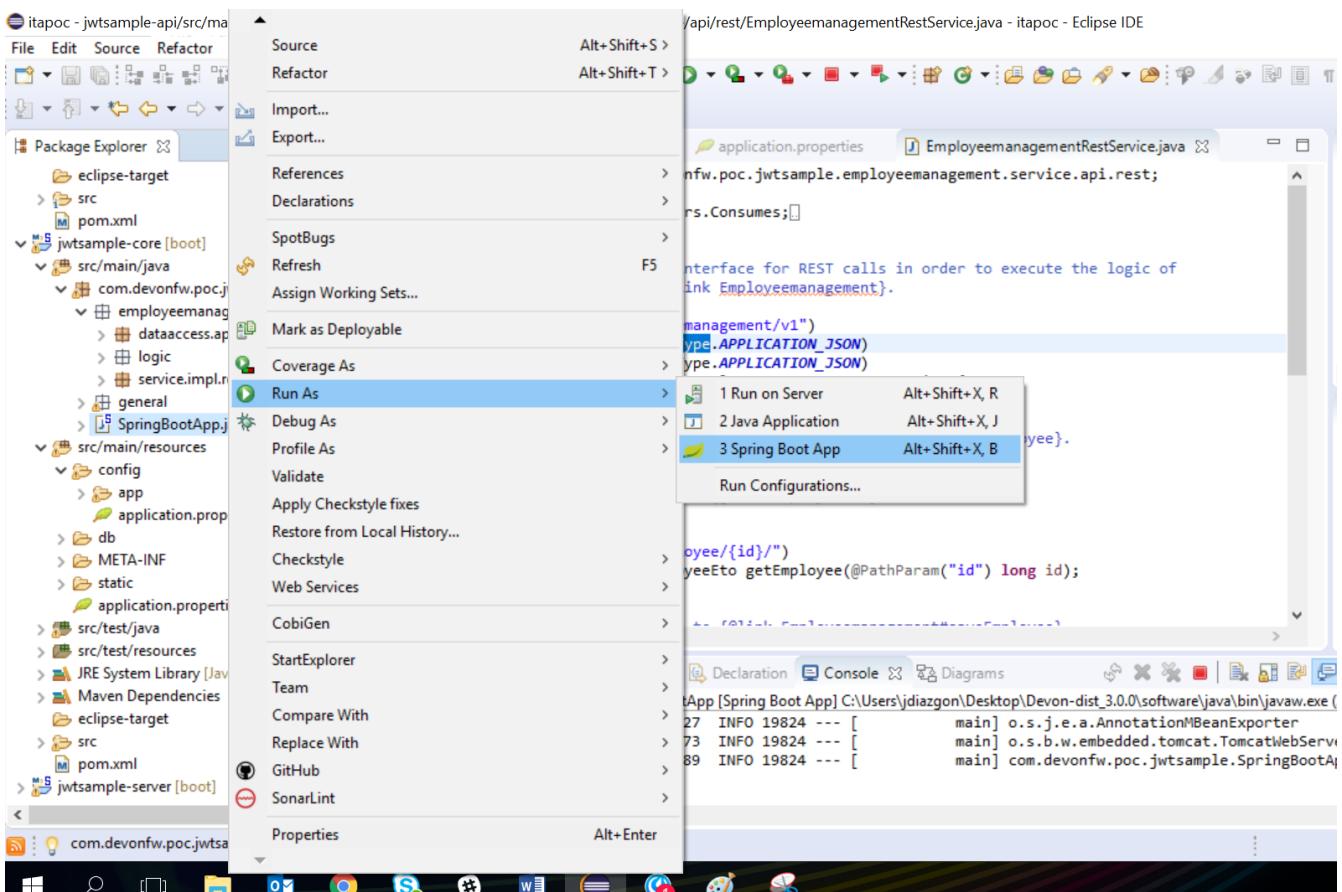
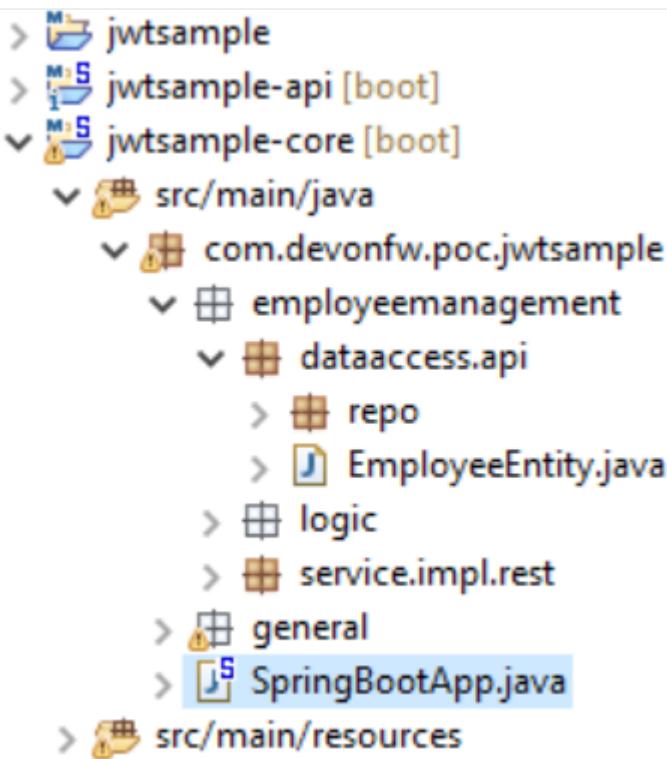


Some other classes will be generated on the core part (*jwtsample-core*), normally it will be implementations as shown below:



BEFORE to generate the FE, please start the Tomcat server to check that BE Layer has been generated properly.

To start a server you just have to right click on `SpringBootApp.java` → *run as* → *Spring Boot app*



The screenshot shows the Eclipse IDE interface. The Project Explorer view on the left lists the project structure, including packages like ionic-api, ionic-core, and src/main/java, which contains com.capgemini.spoc.ionic.employeemanagement, general, and SpringBootApp.java. The central editor window displays the code for EmployeeEntity.java:

```

1 package com.capgemini.spoc.ionic.employeemanagement.dataaccess.api;
2
3 import javax.persistence.Column;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.GenerationType;
7
8 import com.capgemini.spoc.ionic.employeemanagement.common.api.Employee;
9 import com.capgemini.spoc.ionic.general.dataaccess.api.ApplicationPersistenceEntity;
10
11 /**
12  * @author SROSSINI
13 */
14 @Entity
15 @javax.persistence.Table(name = "FMPI_NYFF")
<

```

The code defines an Entity named Employee with a table name of FMPI_NYFF. Below the code, the Console tab shows the application's startup logs:

```

<terminated> SpringBootApp [Java Application] C:\Devon-dist-2.4.0\software\java\bin\javaw.exe (03 ott 2018, 15:28:06)

\ \ / \
( ) \ / \
\ \ / \ / \
=====: / = / /
:: Spring Boot ::          (v1.5.3.RELEASE)
2018-10-03 15:28:08.788 INFO 11784 --- [           main] com.capgemini.spoc.ionic.SpringBootApp : Starting SpringBootApp on LIT
2018-10-03 15:28:08.794 INFO 11784 --- [           main] com.capgemini.spoc.ionic.SpringBootApp : No active profile set, falling back to default profiles: default
2018-10-03 15:28:09.138 INFO 11784 --- [           main] o.s.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext : Refreshing org.springframework.web.context.support.AnnotationConfigEmbeddedWebApplicationContext@63333333: startup date [2018-10-03 15:28:09.138]

```

BE DONE

Last but not least: We make a quick REST services test !

See in the application.properties the TCP Port and the PATH

The screenshot shows the Eclipse IDE interface with the application.properties file open in the center. The left side shows the Package Explorer with the project structure, including src/main/java and src/main/resources. The application.properties file contains the following configuration:

```

# This is the spring boot configuration file for development. It will not be included
# In order to set specific configurations in a regular installed environment create an
# config/application.properties in the server. If you are deploying the application to
# a WAR file you can locate this config folder in ${symbol_dollar}{CATALINA_BASE}/lib. In
# the same container (not recommended by default) you need to ensure the WARS are extracted
# the config folder inside the WEB-INF/classes folder of the webapplication.
#
server.port=8081
server.servlet.context-path=
#
# Datasource for accessing the database
# See https://github.com/devonfw-wiki/devon4j/wiki/guide-configuration#security-config
# jasypt.encryptor.password=None
# spring.datasource.password=ENC(7CnHiadYc0Wh2FnWADNjJg==)
spring.datasource.password=
spring.datasource.url=jdbc:h2:./.jwtsample;
#
# Enable JSON pretty printing
spring.jackson.serialization.INDENT_OUTPUT=true
#
# Flyway for Database Setup and Migrations
spring.flyway.enabled=true
spring.flyway.clean-on-validation-error=true

```

Now compose the Rest service URL:

service class path>/<service method path>

- <server> refers to server with port no. (ie: localhost:8081)
- <app> is in the application.properties (empty in our case, see above)
- <rest service class path> refers to EmployeemanagementRestService: /employeemanagement/v1) (i.e.: for getEmployee method)
- <service method path>/employee/{id} (i.e: for getEmployee method)

The screenshot shows the Eclipse IDE interface with the following details:

- File Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Java development toolbar with icons for file operations, search, and project management.
- Package Explorer:** Shows the project structure:
 - jtapsample
 - jtapsample-api [boot]
 - src/main/java
 - com.devonfw.poc.jtapsample
 - employeemanagement
 - common.api
 - logic.api
 - to
 - EmployeeEto.java
 - EmployeeSearchCriteriaTo.java
 - usecase
 - EmployeeManagement.java
 - service.api.rest
 - EmployeeManagementRestService.java
 - JRE System Library [JavaSE-1.8]
 - Maven Dependencies
 - eclipse-target
 - src
 - pom.xml
 - EmployeeEntity.java** tab: Shows the Java code for the Employee entity.
 - application.properties** tab: Shows configuration properties.
 - EmployeeManagementRestService.java** tab: Shows the REST service interface code.

URL of getEmployee for this example is:

For all employees

<http://localhost:8081/services/rest/employeemanagement/v1/employee/search>

For the specific employee

<http://localhost:8081/services/rest/employeemanagement/v1/employee/1>

Now download [Postman](#) to test the rest services.

Once done, you have to create a POST Request for the LOGIN and insert in the body the JSON containing the username and password *waiter*

The screenshot shows the Postman application interface with the following details:

- Header Bar:** Postman, File, Edit, View, Help.
- Toolbar:** New, Import, Runner, Filter, My Workspace, Invite, Examples (0).
- History:** Nothing in your history yet. Requests that you send through Postman are automatically saved here.
- Collection:** ItaPocRequest.
- Request Details:**
 - Method:** POST, URL: <http://localhost:8081/services/rest/login>.
 - Headers:** (1)
 - Body:** (Green dot) indicates JSON format.
 - Params:**
 - Send:** Send button.
 - Save:** Save button.
 - Authorization:** (radio buttons) form-data, x-www-form-urlencoded, raw, binary.
 - Body Content:** JSON (application/json) field containing the following JSON:


```
1 + [
2 "j_username": "waiter",
3 "j_password": "waiter"
4 }
```
 - Tests:**

Once done with success (**Status: 200 OK**) ...

The screenshot shows the Postman interface with an 'Untitled Request'. The method is set to 'GET' and the URL is 'http://localhost:8081/services/rest/employeemanagement/v1/employee/1'. The 'Body' tab is selected, showing a JSON object:

```

1 {
2   "id": 1,
3   "modificationCounter": 1,
4   "employeeId": 1,
5   "name": "Stefano",
6   "surname": "Rossini",
7   "email": "stefano.rossini@capgemini.com"
8 }

```

... We create a NEW POST Request and We copy the Authorization Bearer field (see above) and We paste it in the Token field (see below)

The screenshot shows a new POST request in Postman. The URL is 'http://localhost:8081/services/rest/employeemanagement/v1/employee/search'. The 'Body' tab is selected, showing a JSON object:

```

1 {
2   "pageable": {
3     "pageSize": 8,
4     "pageNumber": 0
5   }
6 }

```

and specific the JSON parameters for the pagination of the Request that We're going to send:

The screenshot shows the same POST request in Postman. The URL is 'http://localhost:8081/services/rest/employeemanagement/v1/employee/search'. The 'Body' tab is selected, showing a JSON object:

```

1 {
2   "pageable": {
3     "pageSize": 8,
4     "pageNumber": 0
5   }
6 }

```

Below the body, there is a large blue 'Send' button.

Now you can click

Now you 've to check that response has got **Status: 200 OK** and to see the below list of Employee

```

1▼ {
2  "content": [
3    {
4      "id": 1,
5      "modificationCounter": 1,
6      "employeeId": 1,
7      "name": "Stefano",
8      "surname": "Rossini",
9      "email": "stefano.rossini@capgemini.com"
10     },
11    {
12      "id": 2,
13      "modificationCounter": 2,
14      "employeeId": 2,
15      "name": "Angelo",
16      "surname": "Muresu",
17      "email": "angelo.muresu@capgemini.com"
18    },
19    {
20      "id": 3,
21      "modificationCounter": 3,
22      "employeeId": 3,
23      "name": "Jaime",
24      "surname": "Gonzalez",
25      "email": "jaime.diaz-gonzalez@capgemini.com"
26    }
27  ],
28  "pageable": {
29    "pageNumber": 0,
30    "pageSize": 8
31  },
32  "last": true,
33  "totalElements": 3,
34  "totalPages": 1,
35  "size": 8,
36  "number": 0,
37  "sort": null,
38  "numberOfElements": 3,
39  "first": true
40 }

```

Now that We have successfully tested the BE is time to go to create the FE !

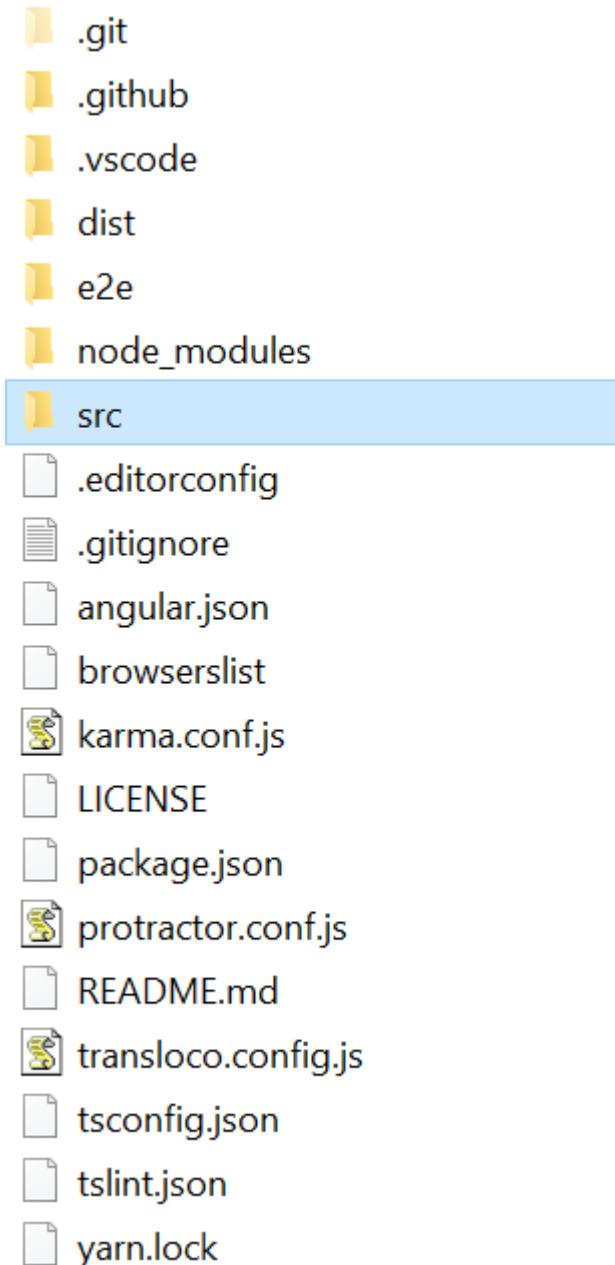
Front End

Let's start now with angular Web and then Ionic app.

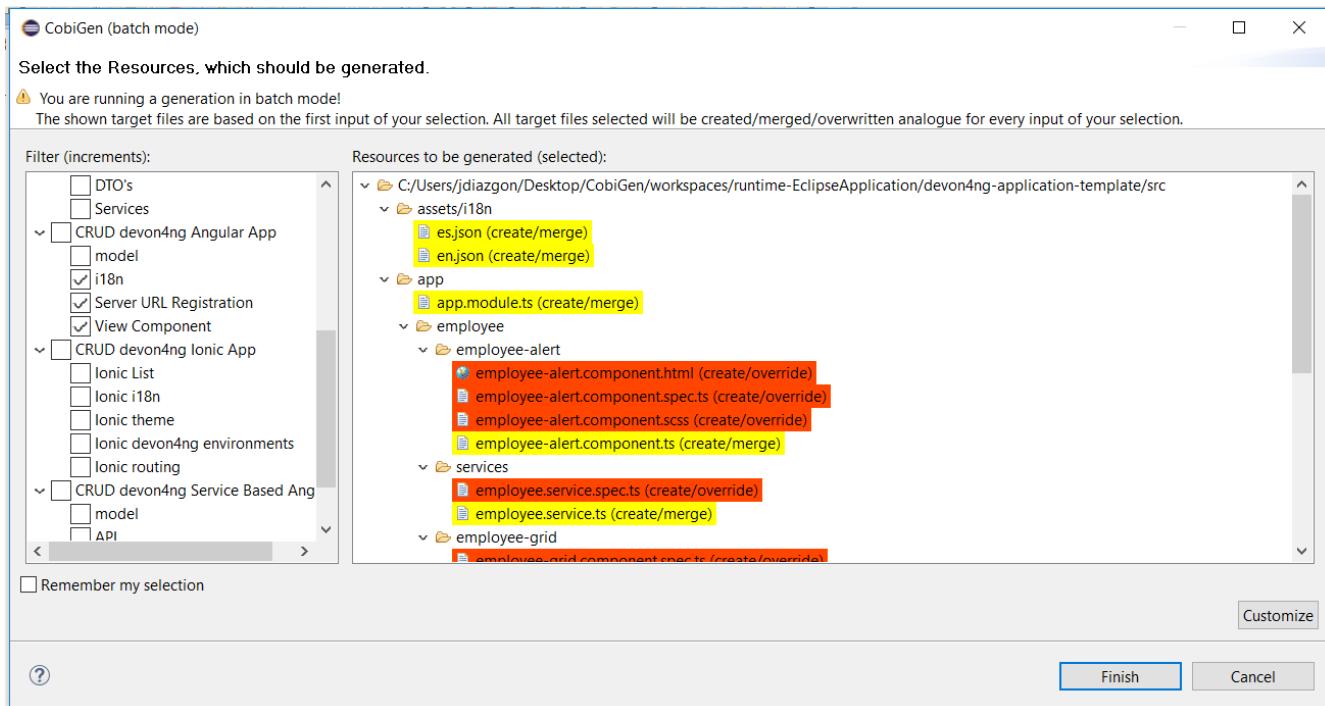
Angular Web App

1. To generate angular structure, download or clone **devon4ng-application-template** from

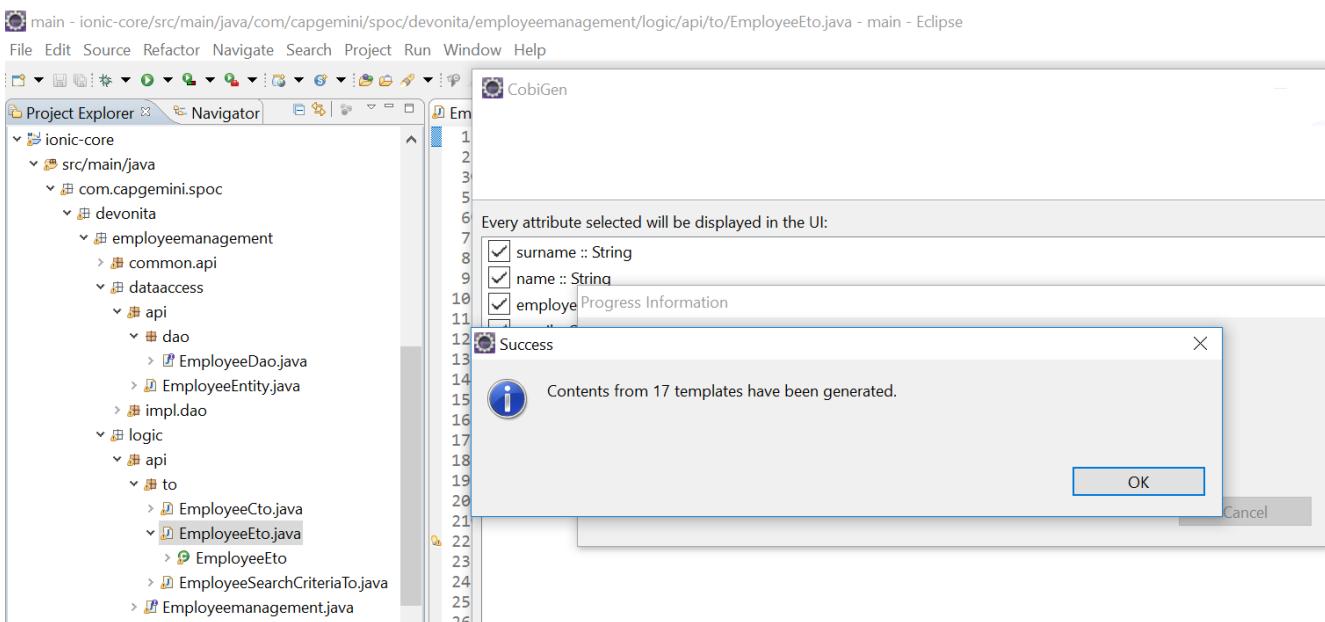
<https://github.com/devonfw/devon4ng-application-template>



2. Once done, right click on *devonfw.yml* again (the OpenAPI contract). CobiGen → Generate
3. Click on the selected options as seen in the screenshot:



4. Click on Finish



5. The entire ANGULAR structure has been auto generated. The generated code will be merged to the existing.

```

1 // Karma configuration file, see link for more information
2 // https://karma-runner.github.io/1.0/config/configuration-file.html
3
4 module.exports = function (config) {
5   config.set({
6     basePath: '',
7     frameworks: ['jasmine', '@angular/cli'],
8     plugins: [
9       require('karma-jasmine'),
10      require('karma-chrome-launcher'),
11      require('karma-jasmine-html-reporter'),
12      require('karma-coverage-istanbul-reporter'),
13      require('@angular/cli/plugins/karma')
14    ],
15    client:{},
16    clearContext: false // leave Jasmine Spec Runner output visible in browser
17  },
18  coverageIstanbulReporter: {
19    reports: [ 'html', 'lcovonly' ],
20    fixWebpackSourcePaths: true
21  },
22  angularCli: {
23    environment: 'dev'
24  },
25  reporters: ['progress', 'kjhtml'],
26  port: 9876,
27  colors: true,
28  logLevel: config.LOG_INFO,
29  autoWatch: true,
30  browsers: ['Chrome'],
31  singleRun: false

```

6. IMPORTANT now you have to add in the **app-routing.module.ts** file the next content, as a child of HomeComponent, in order to enable the route of the new generated component

```

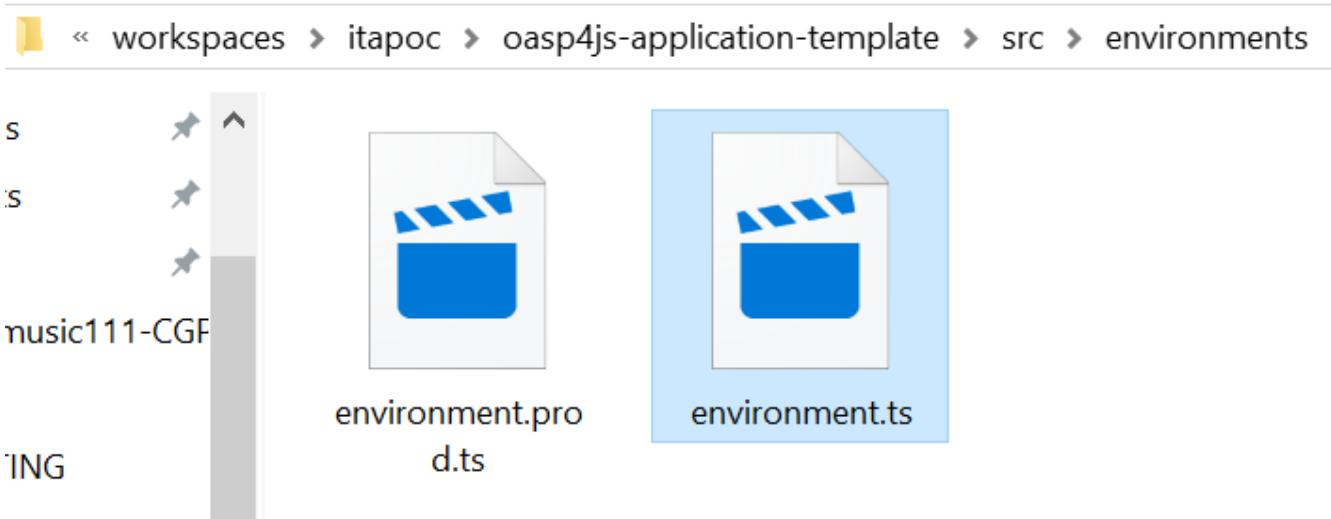
,{\n  path: 'employee',\n  component: EmployeeGridComponent,\n  canActivate: [AuthGuard],\n},

```

Following picture explain where to place the above content:

```
const routes: Routes = [{  
    path: 'login',  
    component: LoginComponent  
}, {  
    path: 'home',  
    component: HomeComponent,  
    canActivate: [AuthGuard],  
    children: []{  
        path: '',  
        redirectTo: '/home/initialPage',  
        pathMatch: 'full',  
        canActivate: [AuthGuard]  
    }, {  
        path: 'initialPage',  
        component: InitialPageComponent,  
        canActivate: [AuthGuard]  
    }, {  
        path: 'sampleData',  
        component: SampleDataGridComponent,  
        canActivate: [AuthGuard]  
    }, {  
        path: 'employee',  
        component: EmployeeGridComponent,  
        canActivate: [AuthGuard]  
    }]  
}, {  
    path: '',  
    redirectTo: '/login',  
    pathMatch: 'full'
```

7. Open the command prompt and execute `devon yarn install` from the base folder, which would download all the required libraries..
8. Check the file **environment.ts** if the server path is correct. (for production you will have to change also the environment.prod.ts file)



In order to do that it's important to look at the application.properties to see the values as PATH, TCP port etc ...

The screenshot shows the devonfw IDE interface. On the left, the Project Explorer shows a Java-based Spring Boot project structure with packages like 'CobiGen_Templates', 'ionic', 'ionic-api', 'ionic-core', 'src/main/java' (containing 'com.capgemini.spoc.ionic' and 'EmployeeManagement' sub-packages), 'src/main/resources' (containing 'config', 'app', and 'application.properties'), and 'db'. The 'application.properties' file is open in the main editor area, displaying configuration properties. On the right, the status bar shows the application name 'SpringBootApp [Java Application]', the path 'C:\Devon-dist_2.4.0\software\java\bin\javaw.exe', the date and time '03 ott 2018, 15:32:16', and the log message '2018-10-03 15:32:30.919 INFO 6112 --- [main] f.a.AutowiredAnnotationBeanPostProce'.

```

# This is the spring boot configuration file for development. It will not be included into
# In order to set specific configurations in a regular installed environment create an acc
# config/application.properties in the server. If you are deploying the application to a s
# WAR file you can locate this config folder in ${symbol_dollar}{CATALINA_BASE}/lib. If yo
# the same container (not recommended by default) you need to ensure the WARS are extracte
# the config folder inside the WEB-INF/classes folder of the webapplication.
#
# server.port=8081
# server.context-path=/
#
# Datasource for accessing the database
# See https://github.com/oasp/oasp4j/wiki/guide-configuration#security
# jasypt.encryptor.password=null
#spring.datasource.password=ENC(7CnHiadYc0Wh2FnWADNjJg==)
spring.datasource.password=
spring.datasource.url=jdbc:h2:./ionic;
#
# Enable JSON pretty printing
spring.jackson.serialization.INDENT_OUTPUT=true
#
# Flyway for Database Setup and Migrations
flyway.enabled=true
flyway.clean-on-validation-error=true

```

For example in this case the URL should be since the context path is empty the server URLs should be like:

```

export const environment = {
  production: false,
  restPathRoot: 'http://localhost:8081/',
  restServiceRoot: 'http://localhost:8081/services/rest/',
  security: 'jwt'
};

```

Warning: REMEMBER to set security filed to **jwt**, if it is not configured already.

- Now run the *ng serve -o* command to run the Angular Application.

```
C:\> C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\sadevadi>D:
D:\>cd D:\Devon\30Jan_2018\Devon-dist_2.2.0\Devon-dist_2.2.0\workspaces\main\oasp4js-application-template

D:\Devon\30Jan_2018\Devon-dist_2.2.0\Devon-dist_2.2.0\workspaces\main\oasp4js-application-template>ng serve -o
```

```
7601]
corporation. All rights reserved.

n-dist_2.2.0\Devon-dist_2.2.0\workspaces\main\oasp4js-ap
2.2.0\Devon-dist_2.2.0\workspaces\main\oasp4js-ap
```

10. If the command execution is **successful**, the below screen will **appear** and it would be automatically redirected to the url:

`http://localhost:4200/login`

employeeId	Name_EN	Surname_EN	Email_EN
1	Stefano	Rossini	stefano.rossini@cap...
2	Angelo	Muresu	angelo.muresu@cap...
3	Jaime	Diaz	jaime.diaz-gonzalez@capgemini.com

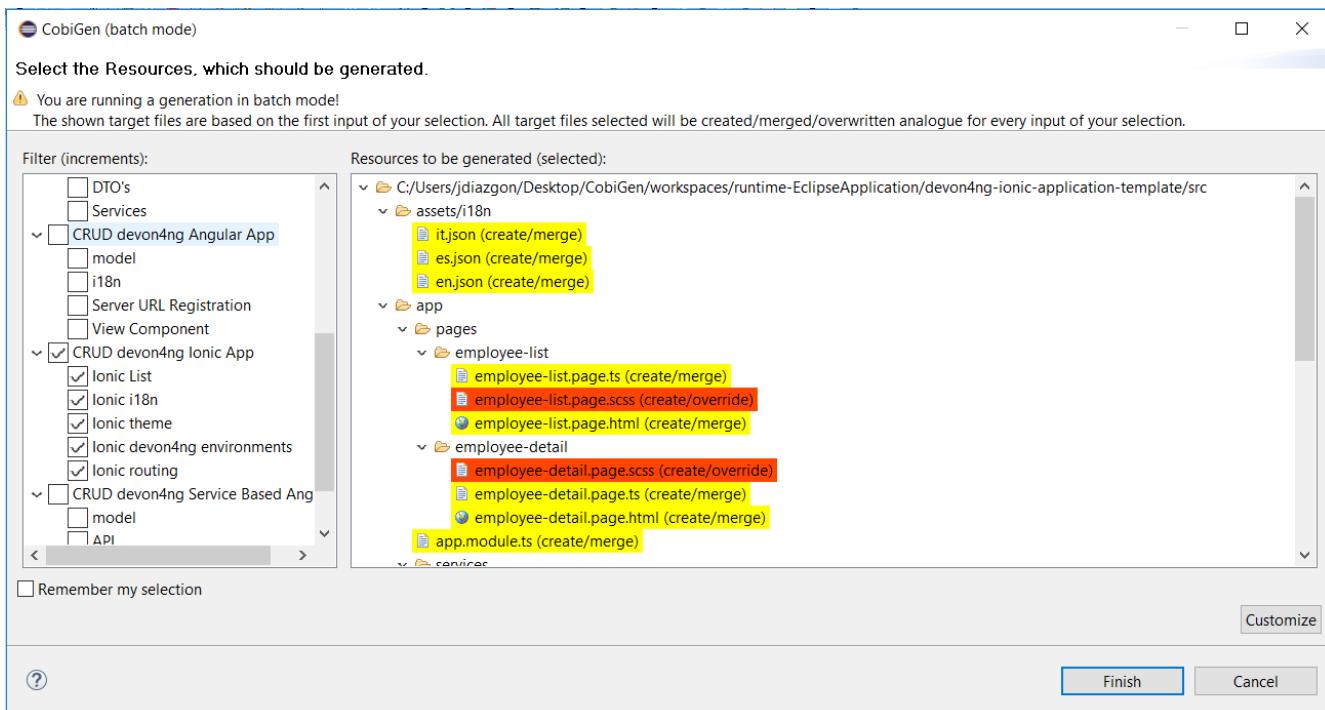
WebApp DONE

Ionic Mobile App

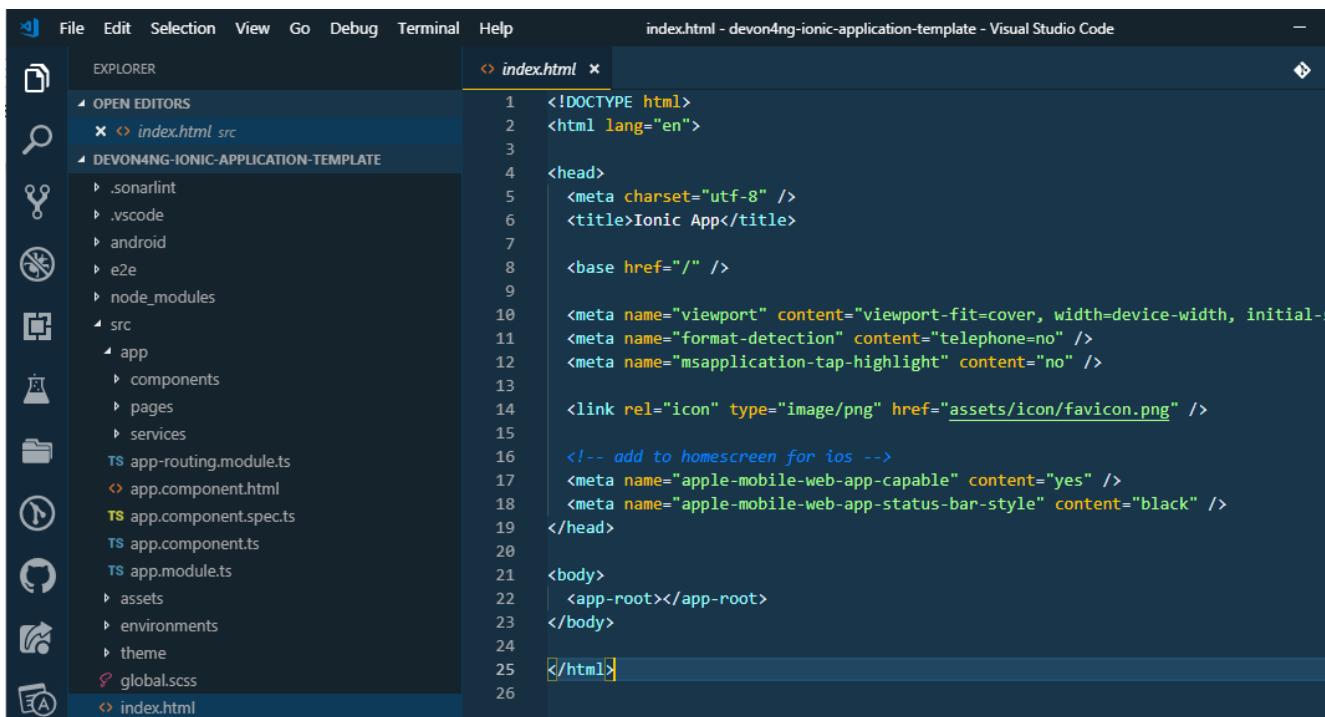
- To generate Ionic structure, download or clone **devon4ng-application-template** from

`https://github.com/devonfw/devon4ng-ionic-application-template`

2. Once done, Right click on the **devonfw.yml** as you already did before in order to use CobiGen.
3. Click on the selected options as seen in the screenshot:



4. Click on Finish
5. The entire ionic structure will be auto generated.



6. Change the server url (with correct serve url) in environment.ts, environment.prod.ts and environment.android.ts files (i.e: itapoc\devon4ng-ionic-application-template\src\environments).

The angular.json file inside the project has already a build configuration for android.

```

1 // This file can be replaced during build by using the 'fileReplacem...
2 // The list of file replacements can be found in 'angular.json'
3
4 export const environment = {
5   production: false,
6 };
7
8 export const SERVER_URL = 'http://localhost:8081/';
9
10 /*
11 * For easier debugging in development mode, you can import the
12 * to ignore zone related error stack frames such as 'zone.run'
13 */
14
15 /* This import should be commented out in production mode because
16 * on performance if an error is thrown.
17 */
18 // Import 'zone.js/dist/zone-error'; // Included with Angular
19

```

```

1 export const environment = {
2   production: false,
3 };
4
5 export const SERVER_URL = 'http://10.0.2.2:8081/';
6

```

```

5 "newProjectRoot": "projects",
6 "projects": {
7   "app": {
8     "root": "",
9     "sourceRoot": "src",
10    "projectType": "application",
11    "prefix": "app",
12    "schematics": {},
13    "architect": {
14      "build": {
15        "builder": "@angular-devkit/build-angular:browser",
16        "options": { ... },
17        "configurations": {
18          "production": { ... }
19        }
20      },
21      "android": {
22        "fileReplacements": [
23          {
24            "replace": "src/environments/environment.ts",
25            "with": "src/environments/environment.android.ts"
26          }
27        ],
28        "i18n": {
29          "progress": false
30        }
31      }
32    },
33    "serve": {
34      "builder": "@angular-devkit/build-angular:dev-server",
35      "options": {
36        "browserTarget": "app:build"
37      },
38      "configurations": {
39        "production": {
40          "browserTarget": "app:build:production"
41        },
42        "ci": {
43          "progress": false
44        }
45      }
46    }
47  }
48 }

```

7. Run npm install in the root folder to download the dependencies

8. Run ionic serve

```

C:\Devon-dist-current\Devon-dist_3.0.0\workspaces\Test\devon4ng-ionic-application-template
λ ionic serve
> ng run app:serve --host=0.0.0.0 --port=8100
[ng] WARNING: This is a simple server for use in testing or debugging Angular applications
[ng] locally. It hasn't been reviewed for security issues.
[ng] Binding this server to an open connection can result in compromising your application or
[ng] computer. Using a different host than the one passed to the "--host" flag might result in
[ng] websocket connection issues. You might need to use "--disableHostCheck" if that's the
[ng] case.
[INFO] Waiting for connectivity with ng...
[INFO] Waiting for connectivity with ng...

[INFO] Development server running!

Local: http://localhost:8100
External: http://10.80.132.29:8100, http://192.168.56.1:8100, http://192.168.99.1:8100

Use Ctrl+C to quit this process

[INFO] Browser window opened to http://localhost:8100!
[ng] i 「wdm」: wait until bundle finished: /

```

11.

Once the execution is successful

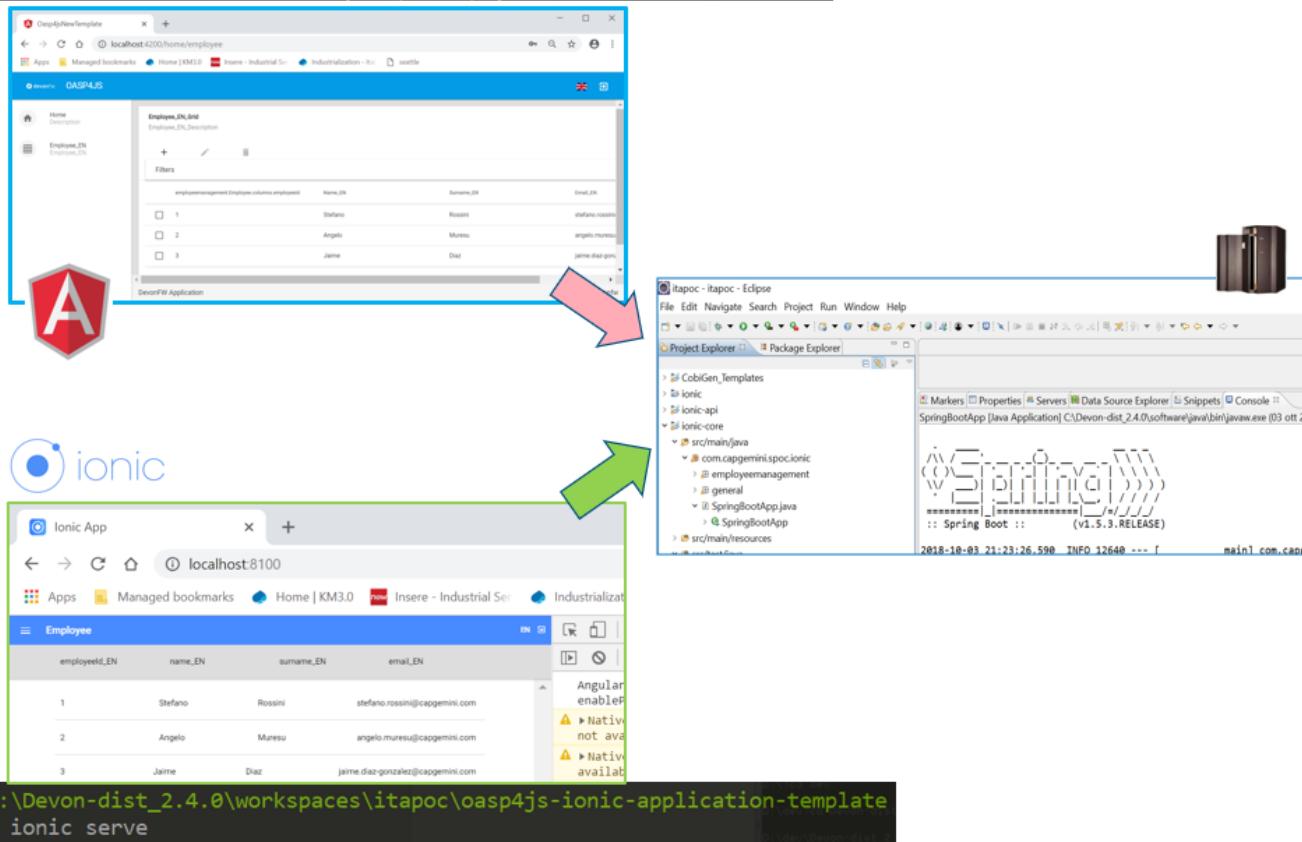
employeeId_EN	name_EN	surname_EN	email_EN
1	Stefano	Rossini	stefano.rossini@capgemini.com
2	Angelo	Muresu	angelo.muresu@capgemini.com
3	Jaime	Gonzalez	jaime.diaz-gonzalez@capgemini.com

- Mobile App DONE*

So: well done

Starting from an Entity class you've successfully generated the Back-End layer (REST, SOAP, DTO, Spring services, Hibernate DAO), the Angular Web App and the Ionic mobile App!

```
C:\Devon-dist_2.4.0\workspaces\itapoc\oasp4js-application-template
λ ng serve -o
```

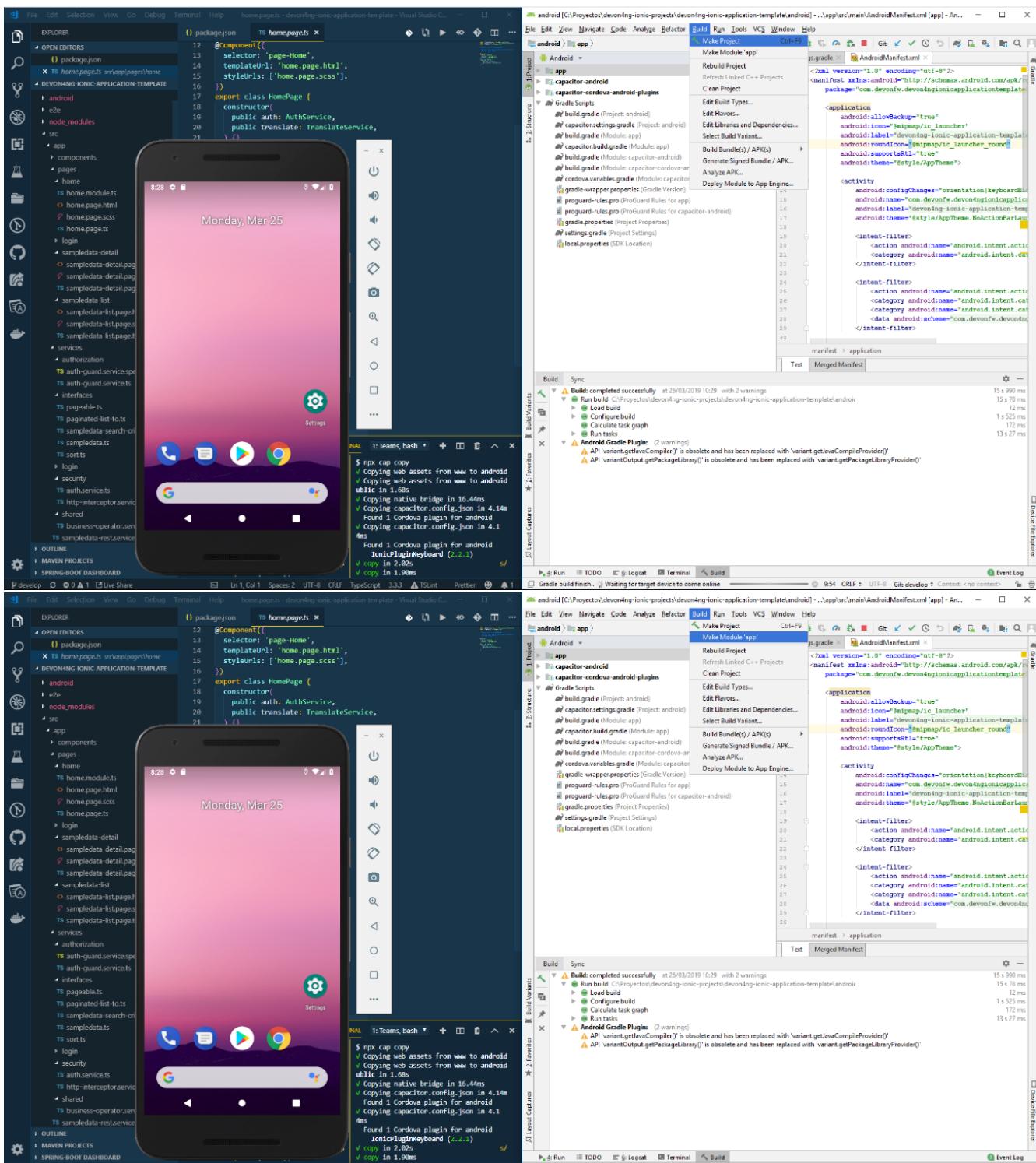


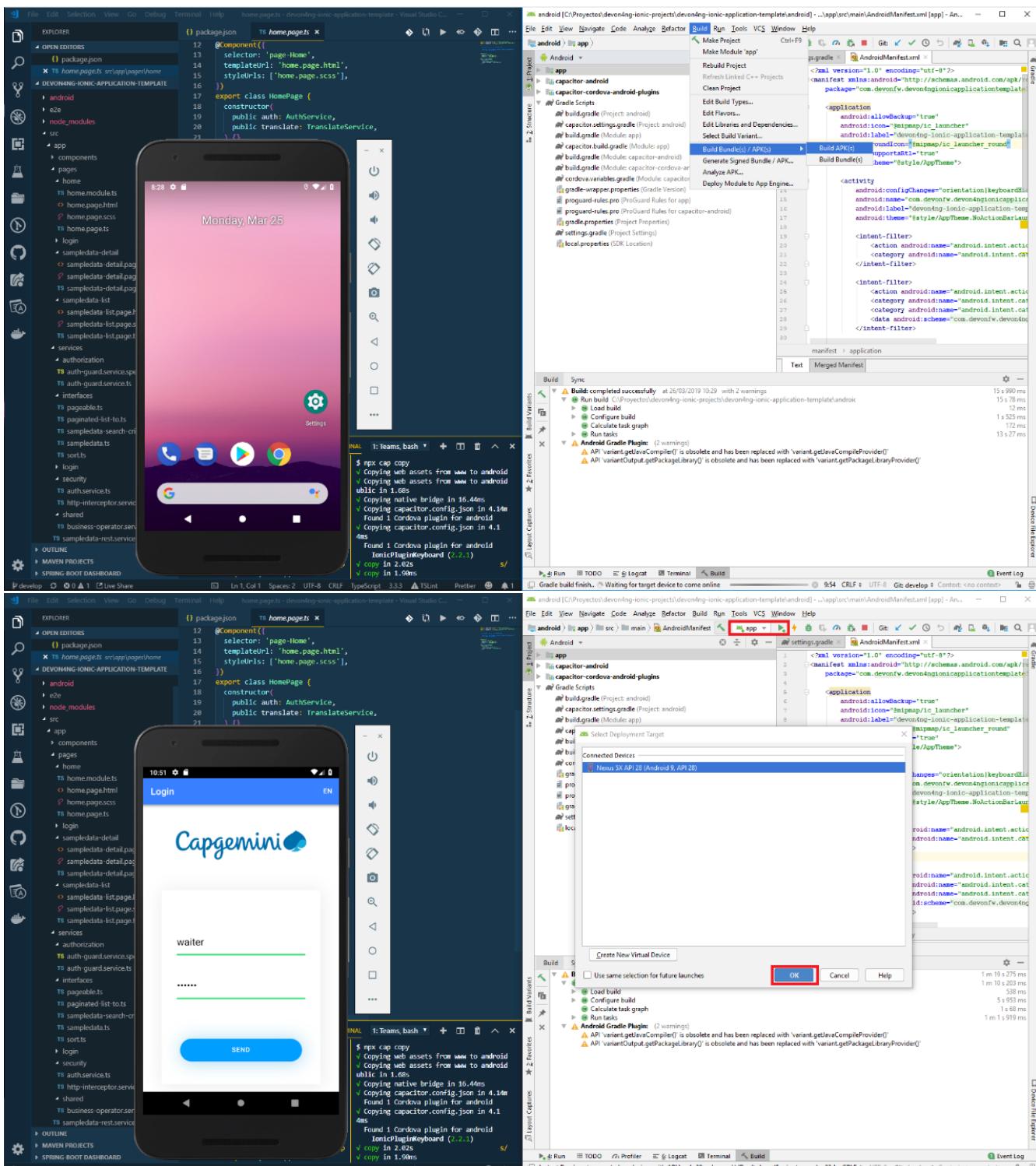
Build APK

Since We're going to create apk remember the following pre-conditions:

- [Gradle](#)
- [Android Studio](#)
- [Android sdk](#)
- [Capacitor](#)

1. Now, open cmd and type the path where your *devon4ng-ionic-application-template* project is present.
2. Run the following commands:
 - a. npx cap init
 - b. ionic build --configuration=android
 - c. npx cap add android
 - d. npx cap copy
 - e. npx cap open android
3. Build the APK using Android studio.





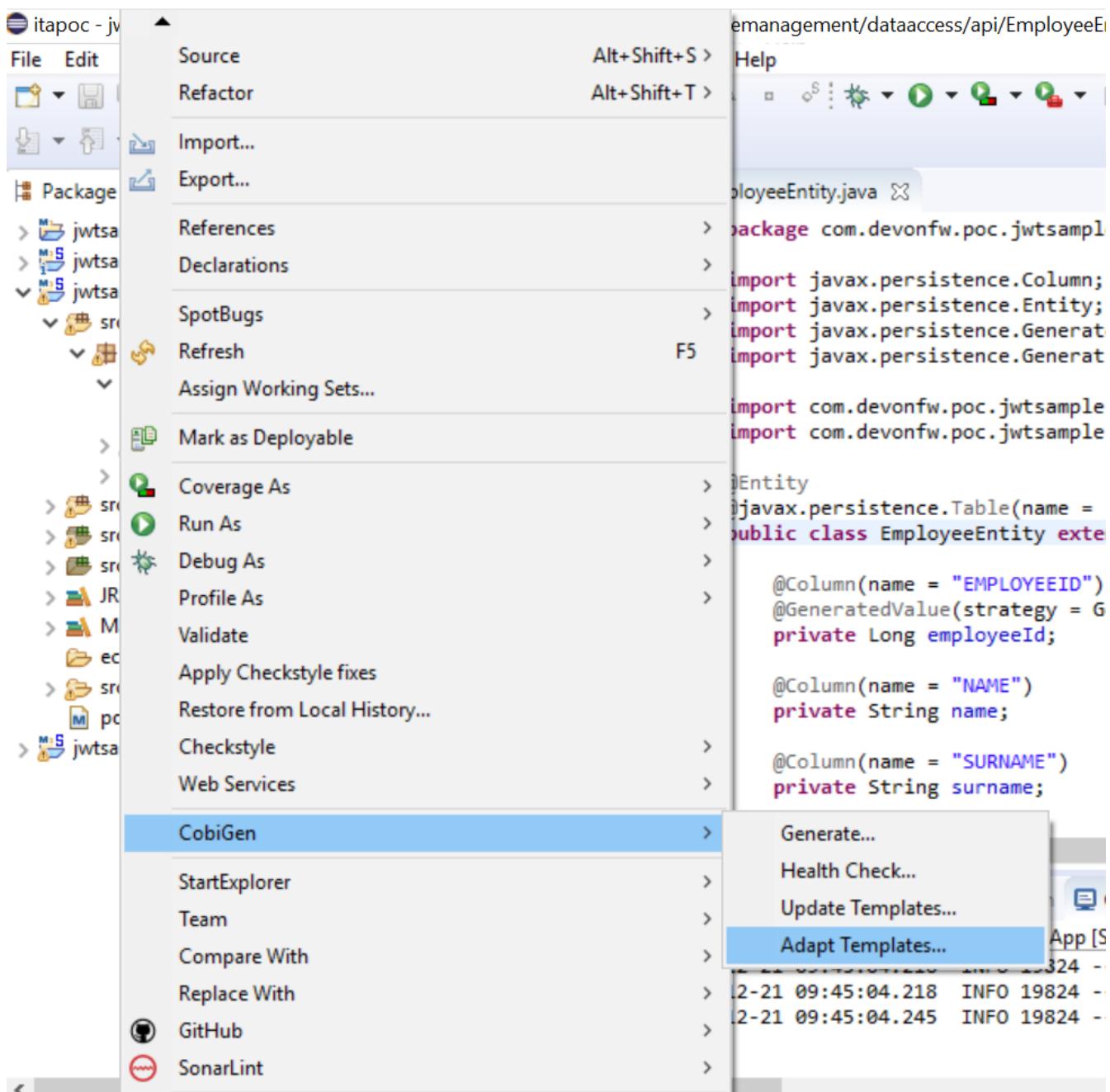
You can find your apk file in

/devon4ng-ionic-application-template/android/app/build/outputs/apk/debug

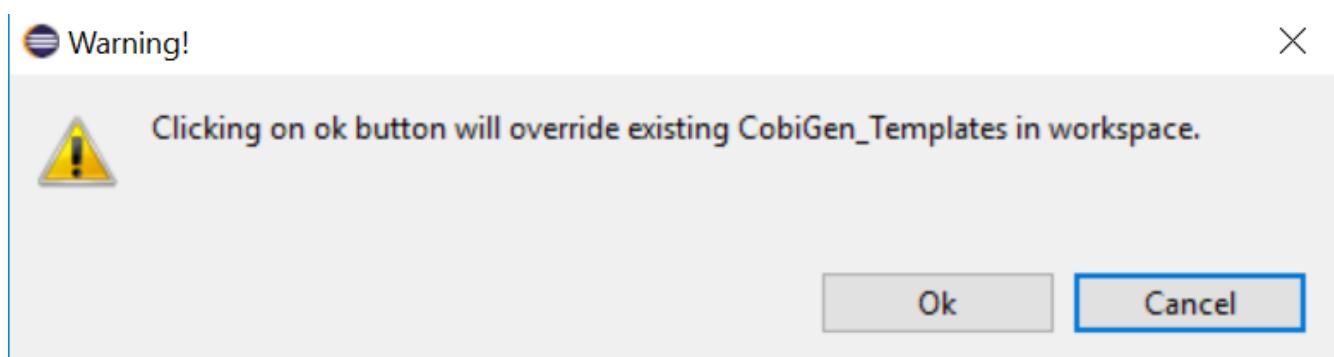
73.7.3. Adapt CobiGen_Templates

After following this tutorial, you will have the CobiGen_Templates downloaded on your local machine. To import these templates you need to do the following:

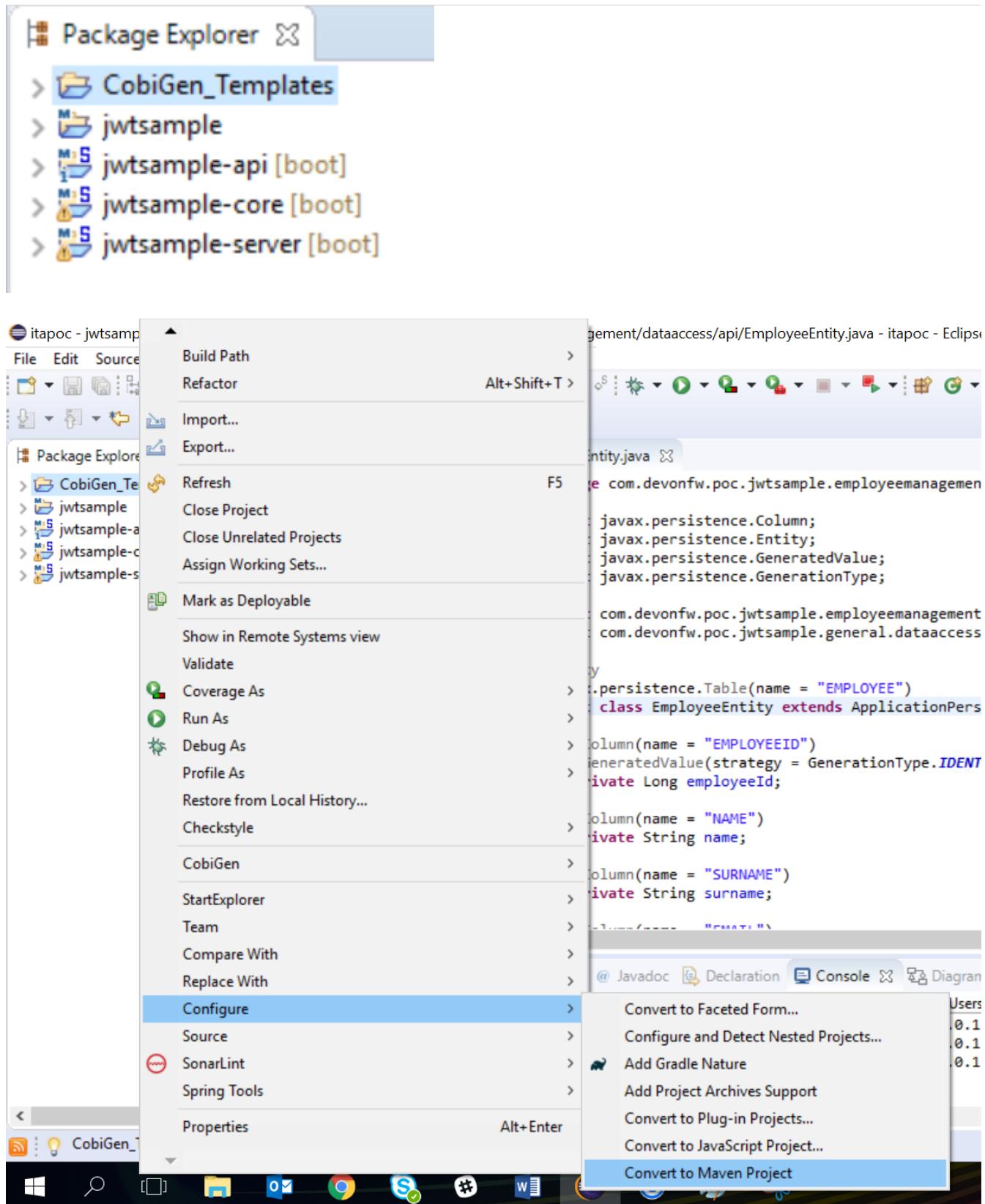
Right click in any part of the package explorer, then click on CobiGen → Adapt templates



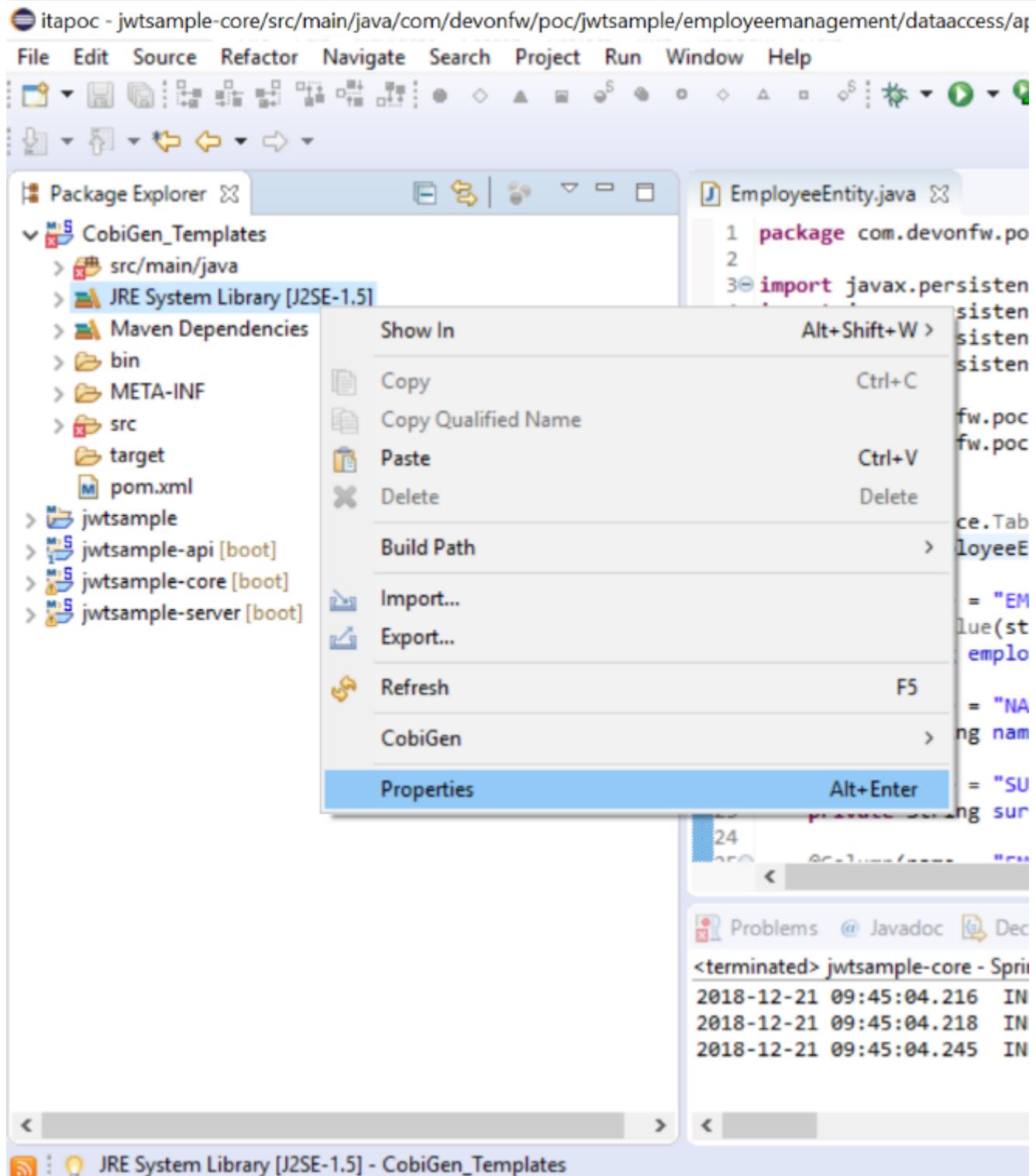
Click *Ok*:



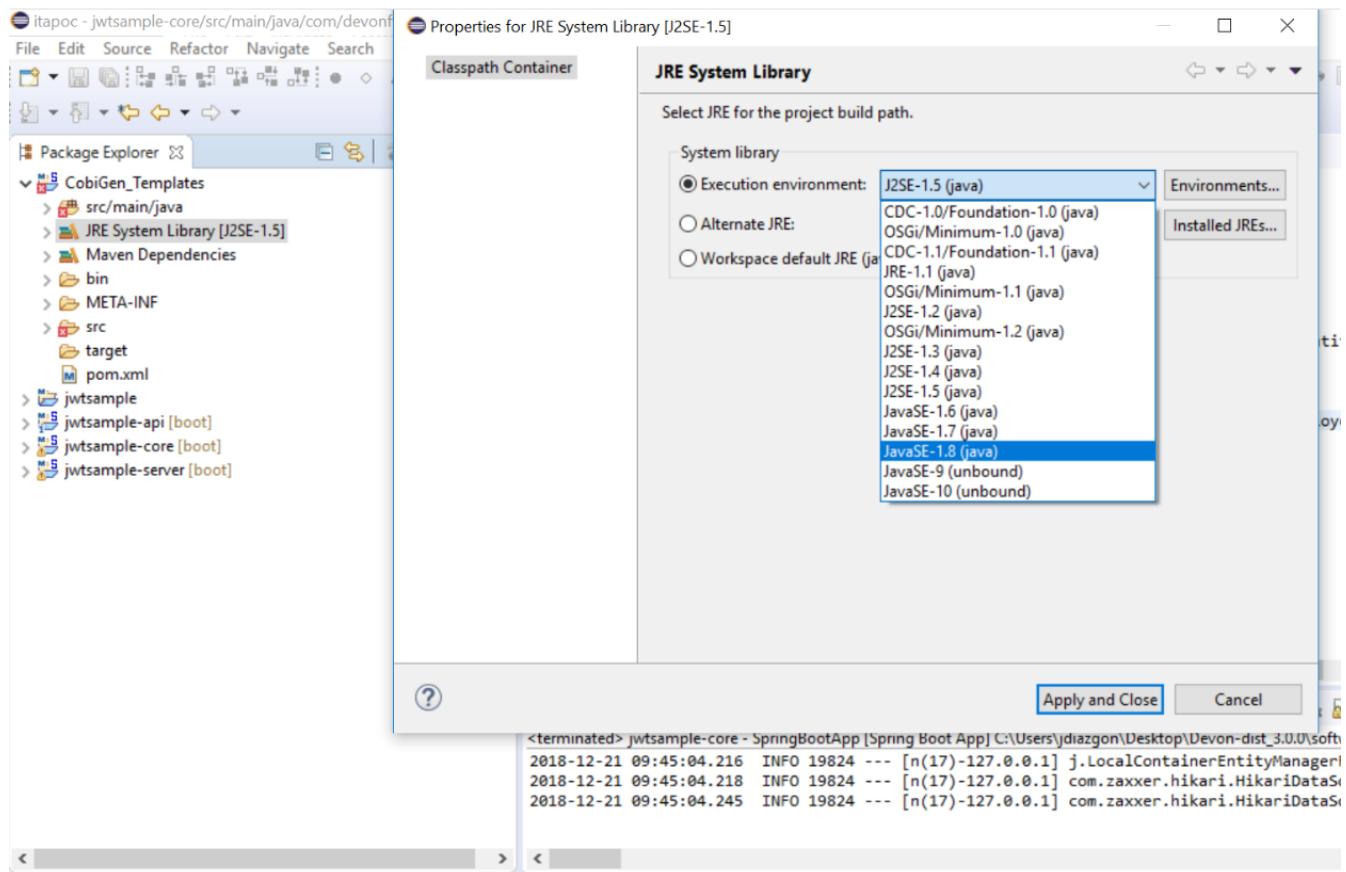
Now the CobiGen_Templates project will be automatically imported into your workspace, as shown on the image below:



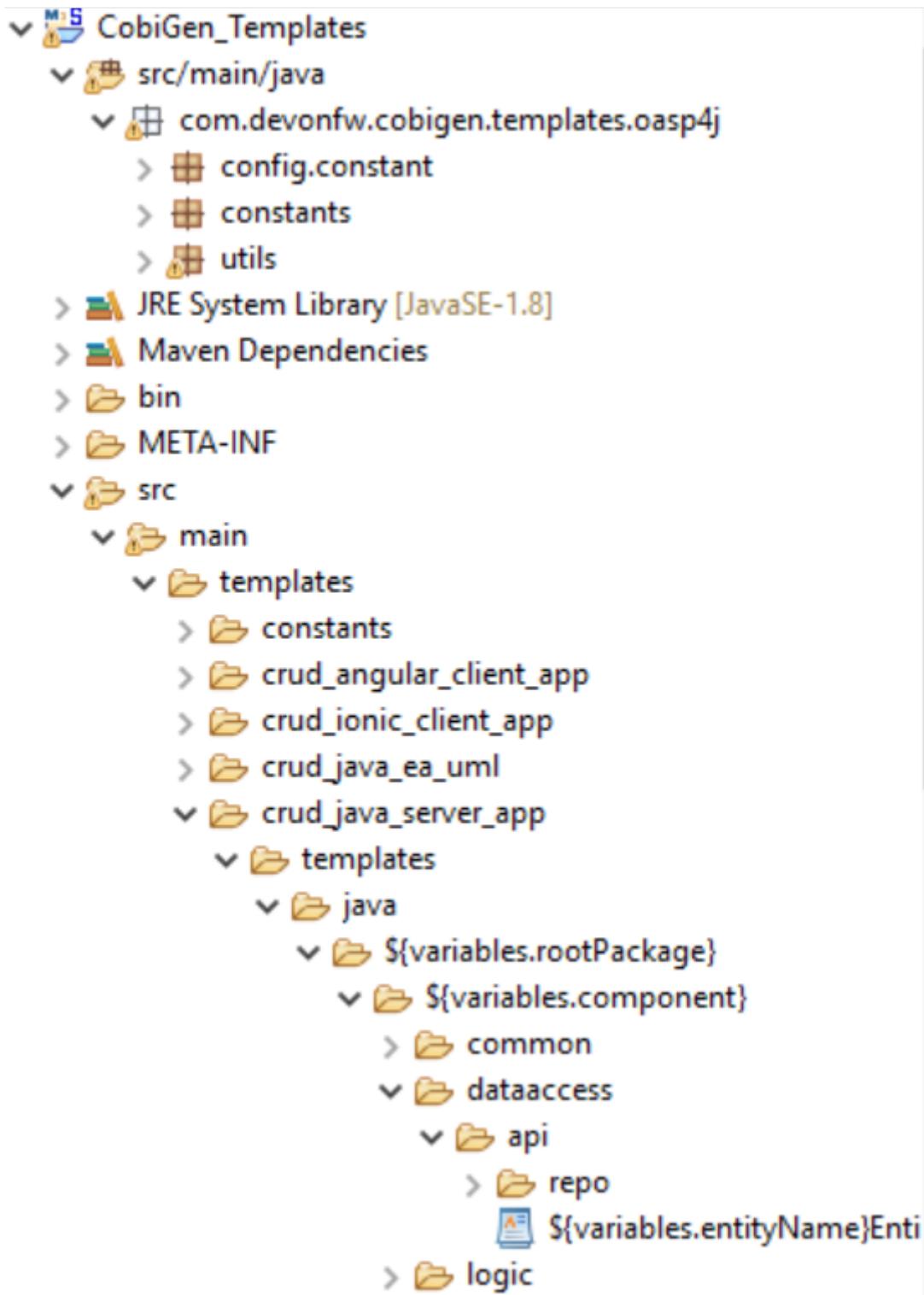
Now you just need to change the Java version of the project to JRE 1.8. Right click on the JRE system library, and then on *Properties*:



Now change the version to Java 1.8



Now you have successfully imported the CobiGen templates. If you want to edit them, you will find them in the folder *src/main/templates*. For instance, the Java templates are located here:



Now you can adapt the templates as much as you want. Documentation about this can be found on:

<https://github.com/devonfw/tools-cobigen/wiki/Guide-to-the-Reader>

73.8. End to End POC Code generation using Entity class

This article helps to create a sample application using cobigen.

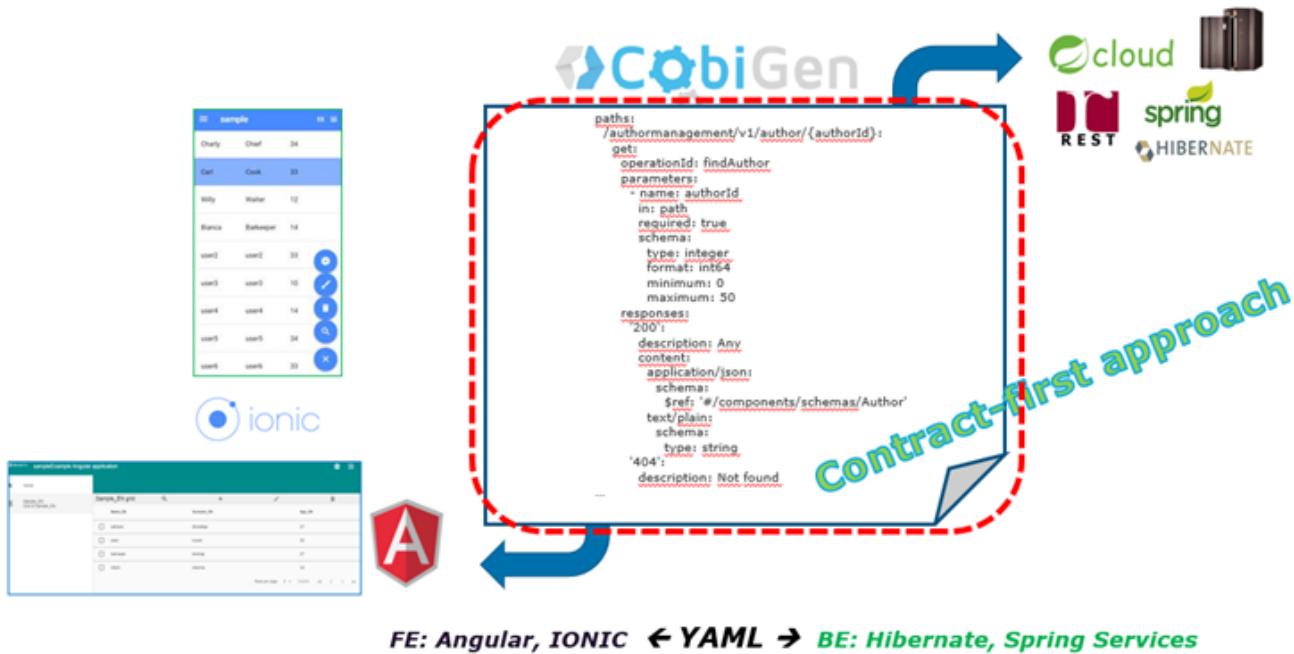
73.8.1. Prerequisites

Download and install devonfw IDE [here](#),

73.8.2. Steps to create a Sample Project using Cobigen

The HOW_TO is divided in 2 parts:

1. BE-Back End generator (DB + DAO + services) – CONTRACT FIRST APPROACH
2. FE-Front End generator (Web App Angular + Ionic App) – CONTRACT FIRST APPROACH



So, ready to go! We're going to start from the BE part ...

Back End

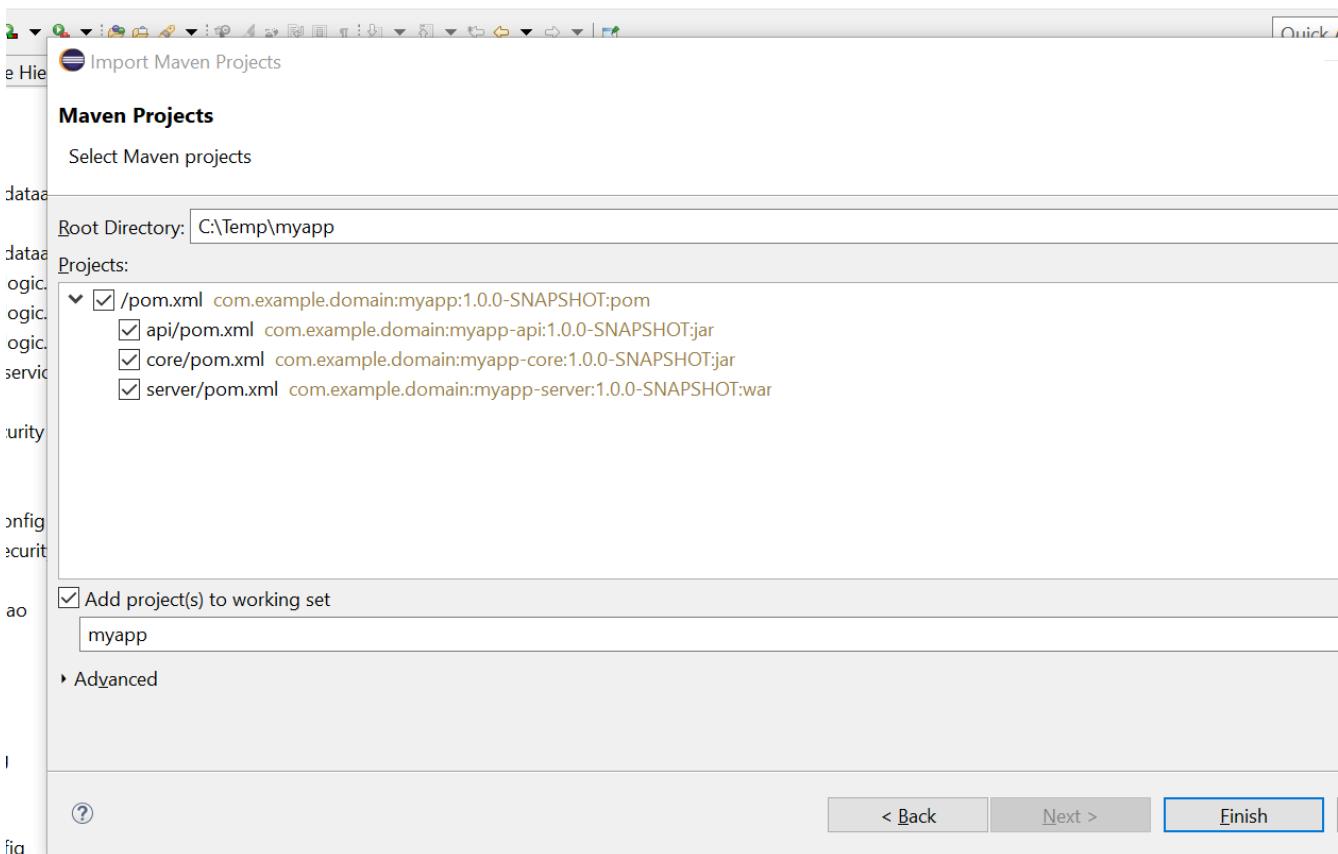
run \devonfw-ide-scripts-3.2.4\.eclipse-main.bat

It will open eclipse

create a project using below command from the command prompt

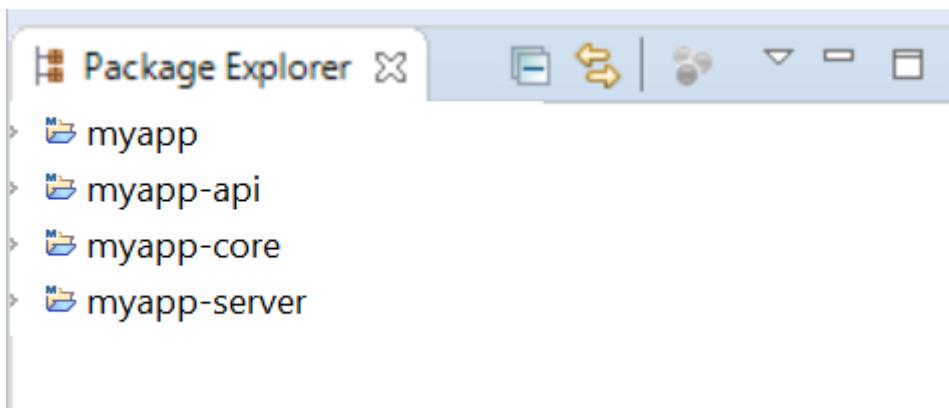
```
devon java create com.example.domain.myapp
```

Import the project to eclipse as maven project



Click **FINISH**

Now We have the following 4 projects.



BEFORE to start to create an Entity class, remember to create the tables !

7. Create a new **SQL file** (i.e: V0005CreateTables-ItaPoc.sql) inside myapp_core and insert the following script:

```
CREATE TABLE EMPLOYEE (
    id BIGINT auto_increment, modificationCounter INTEGER NOT NULL,
    employeeid BIGINT auto_increment,
    name VARCHAR(255),
    surname VARCHAR(255),
    email VARCHAR(255),
    PRIMARY KEY (employeeid)
);
```

WARNING: please note that there are 2 underscore in the name !

The left side shows the project structure:

- myapp-core
- src/main/java
- src/main/resources
 - config
 - db
 - migration
 - type
 - h2
 - V0001_Create_Sequence.sql
 - V0002_Create_RevInfo.sql
 - V0003_Create_BinaryObject.sql
 - V0005_CreateTables_ItaPoc.sql

The right side shows a database editor window with the following details:

 - Type: [dropdown]
 - Name: [dropdown]
 - Database: [dropdown]
 - Status: [dropdown]
 - SQL code:

```
1 CREATE TABLE EMPLOYEE (
2     id BIGINT auto_increment, modificationCounter INTEGER NOT NULL,
3     employeeid BIGINT auto_increment,
4     name VARCHAR(255),
5     surname VARCHAR(255),
6     email VARCHAR(255),
7     PRIMARY KEY (employeeid)
8 );
9 }
```

- Now create another SQL file (i.e: V0006_PopulateTables-ItaPoc.sql) and add following script about the INSERT in order to populate the table created before

WARNING: please note that there are 2 underscore in the name !

```
INSERT INTO EMPLOYEE (id, modificationCounter, employeeid, name, surname, email) VALUES
(1, 1, 1, 'Stefano', 'Rossini', 'stefano.rossini@capgemini.com');
INSERT INTO EMPLOYEE (id, modificationCounter, employeeid, name, surname, email)
VALUES (2, 2, 2, 'Angelo', 'Muresu', 'angelo.muresu@capgemini.com');
INSERT INTO EMPLOYEE (id, modificationCounter, employeeid, name, surname, email) VALUES
(3, 3, 3, 'Jaime', 'Gonzalez', 'jaime.diaz-gonzalez@capgemini.com');
```

The left side shows the project structure, identical to the previous screenshot.

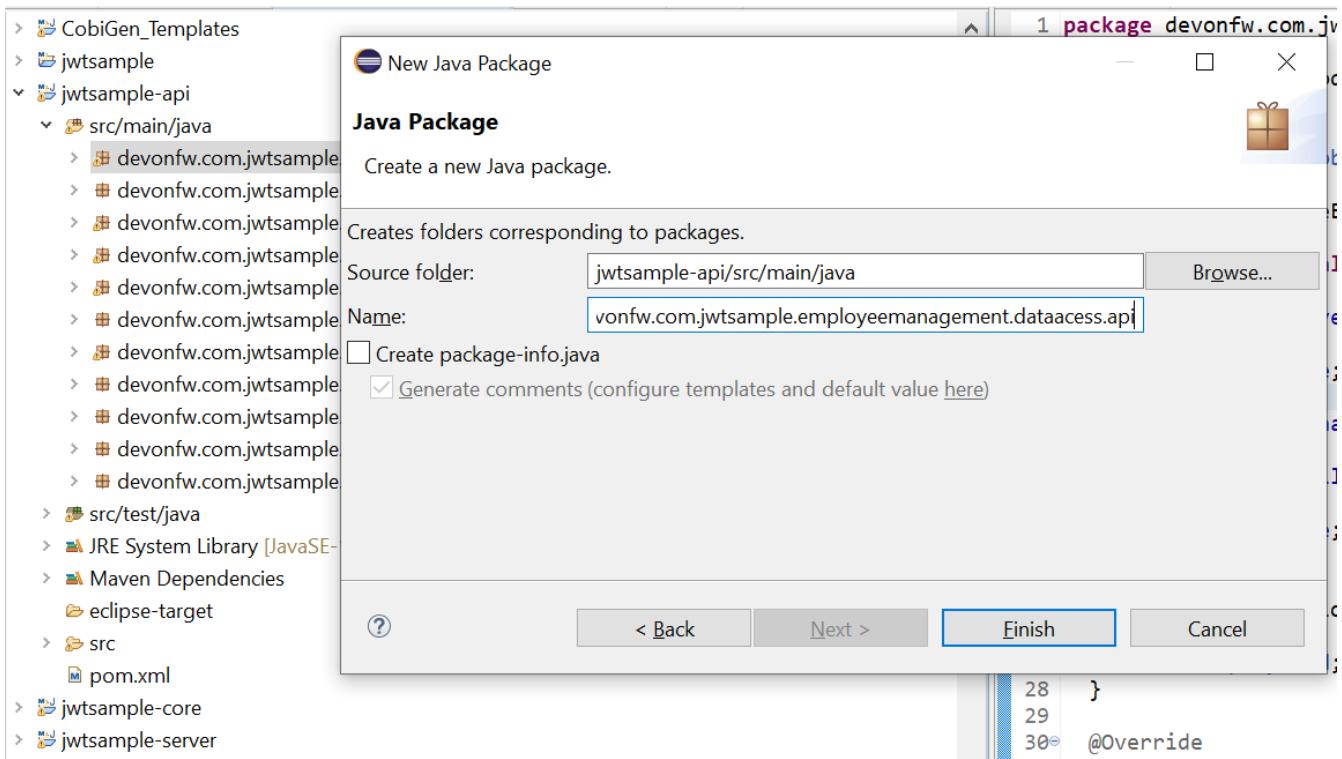
The right side shows a database editor window with the following details:

- Type: [dropdown]
- Name: [dropdown]
- Database: [dropdown]
- Status: Disconnected, Auto Commit
- SQL code:

```
1 INSERT INTO EMPLOYEE (id, modificationCounter, employeeid, name, surname, email) VALUES (1, :^
2 INSERT INTO EMPLOYEE (id, modificationCounter, employeeid, name, surname, email) VALUES (2, :^
3 INSERT INTO EMPLOYEE (id, modificationCounter, employeeid, name, surname, email) VALUES (3, :^
4 |
```

Let's create the Entity Class for the code generation

- Create a package **employeemanagement.dataaccess.api** under the folder myapp-core. Note: It is important to follow this naming convention for CobiGen to work properly.



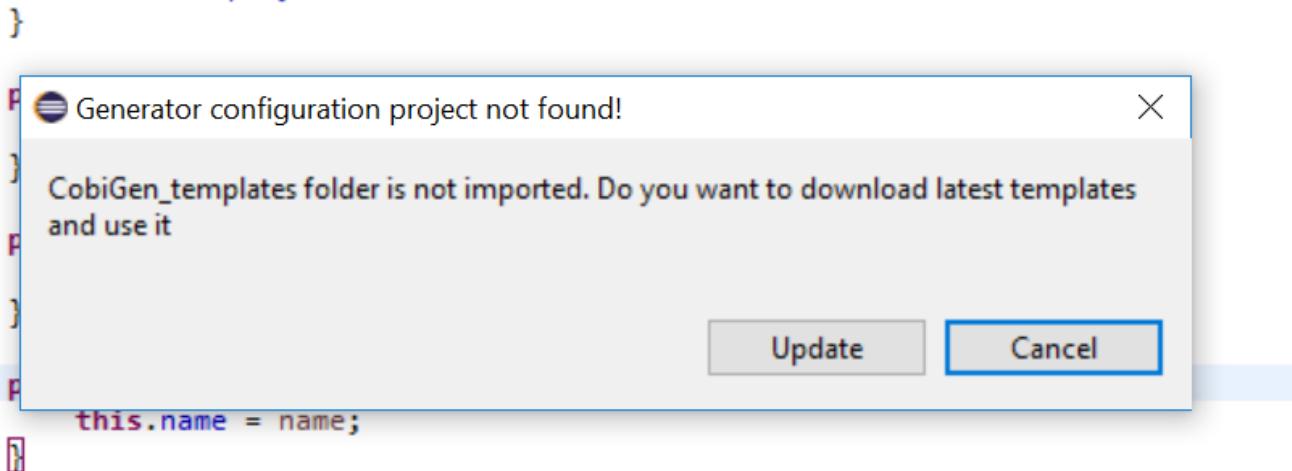
1. Now create a JPA Entity class in this package

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Column;
@Entity
@javax.persistence.Table(name = "EMPLOYEE")
public class EmployeeEntity {
    @Column(name = "EMPLOYEEID")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long employeeId;
    @Column(name = "NAME")
    private String name;
    @Column(name = "SURNAME")
    private String surname;
    @Column(name = "EMAIL")
    private String email;
}
```

then generate getters and setters for all attributes ...

10. Use Cobigen to generate code. Right click on EmployeeEntity. CobiGen → Generate

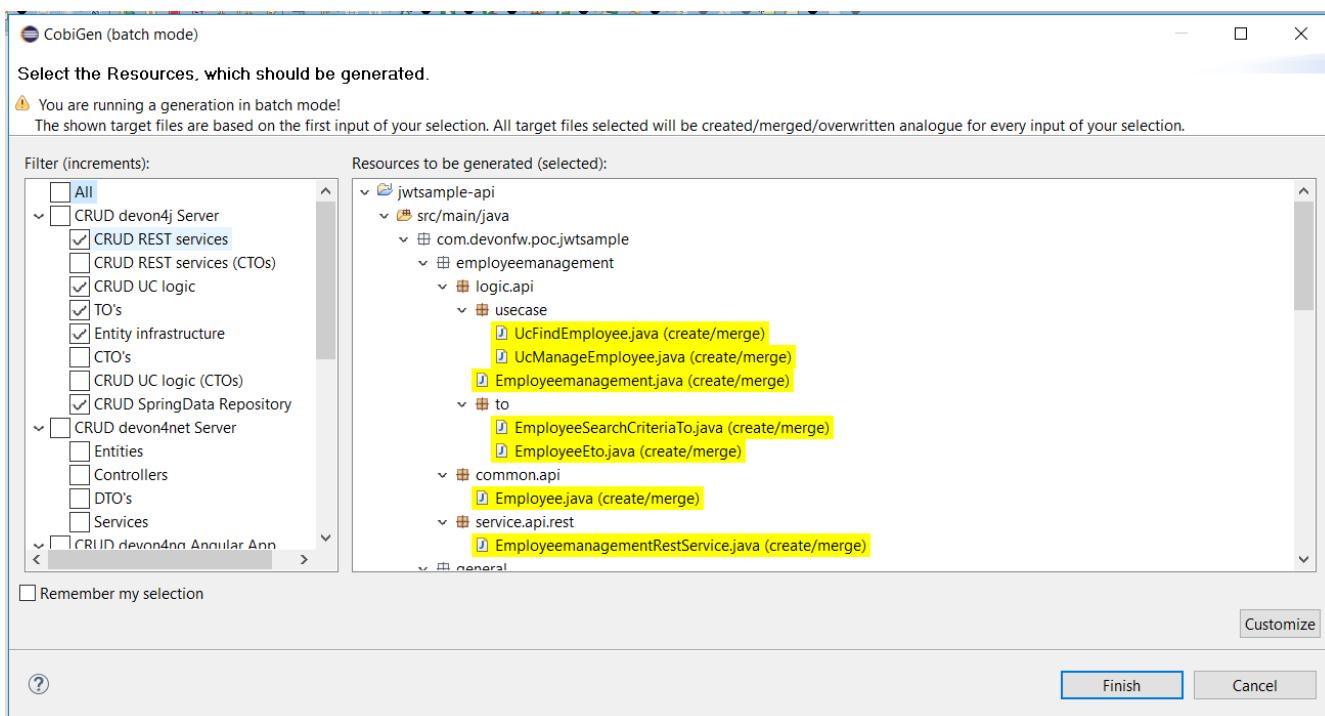
It will ask you to download the templates, click on *update*:



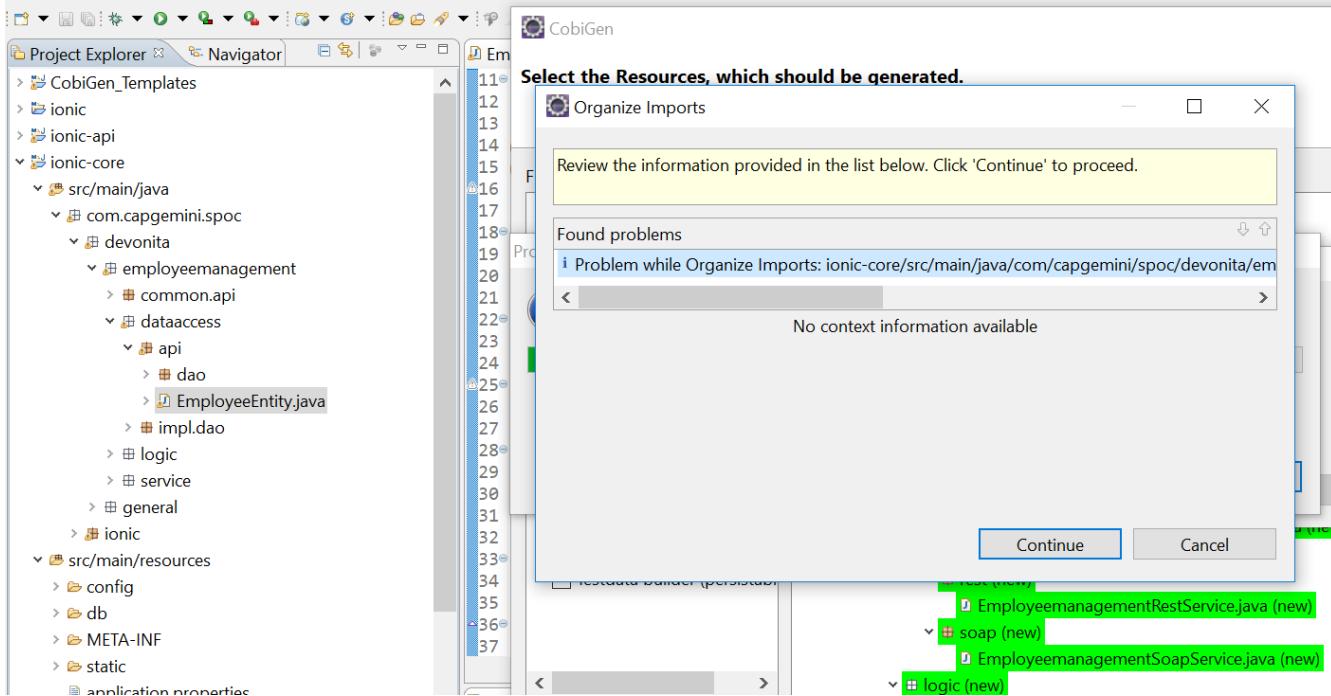
It will automatically download the latest version of *CobiGen_Templates*.

Attention: If you want to adapt the CobiGen_Templates, (normally this is not necessary), you will find at the end of this document a tutorial on how to import them and adapt them!

11. Click on all the option selected as below:

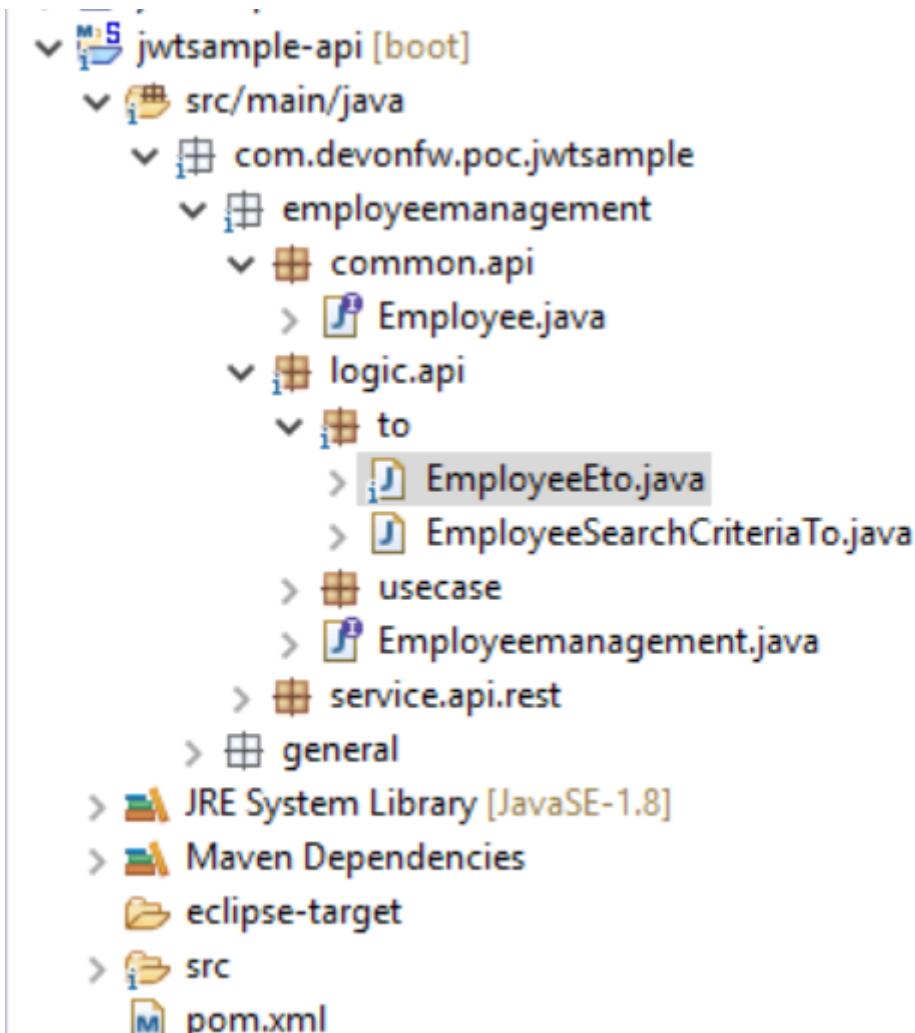


12. Click on finish. Below Screen would be seen. Click on continue

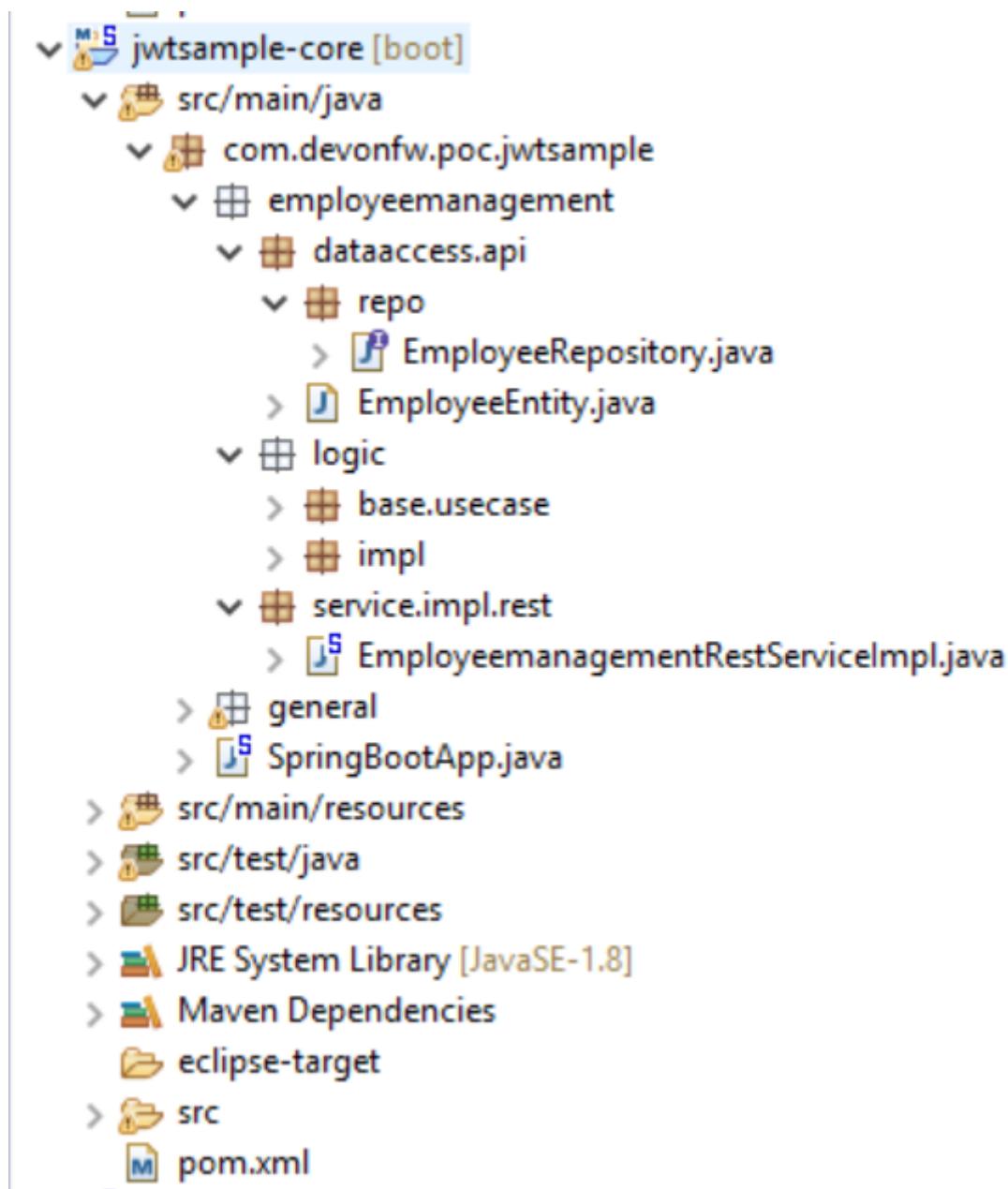


The entire BE layer structure having CRUD operation methods will be auto generated.

Some classes will be generated on the api part (*myapp-api*), normally it will be interfaces, as shown below:

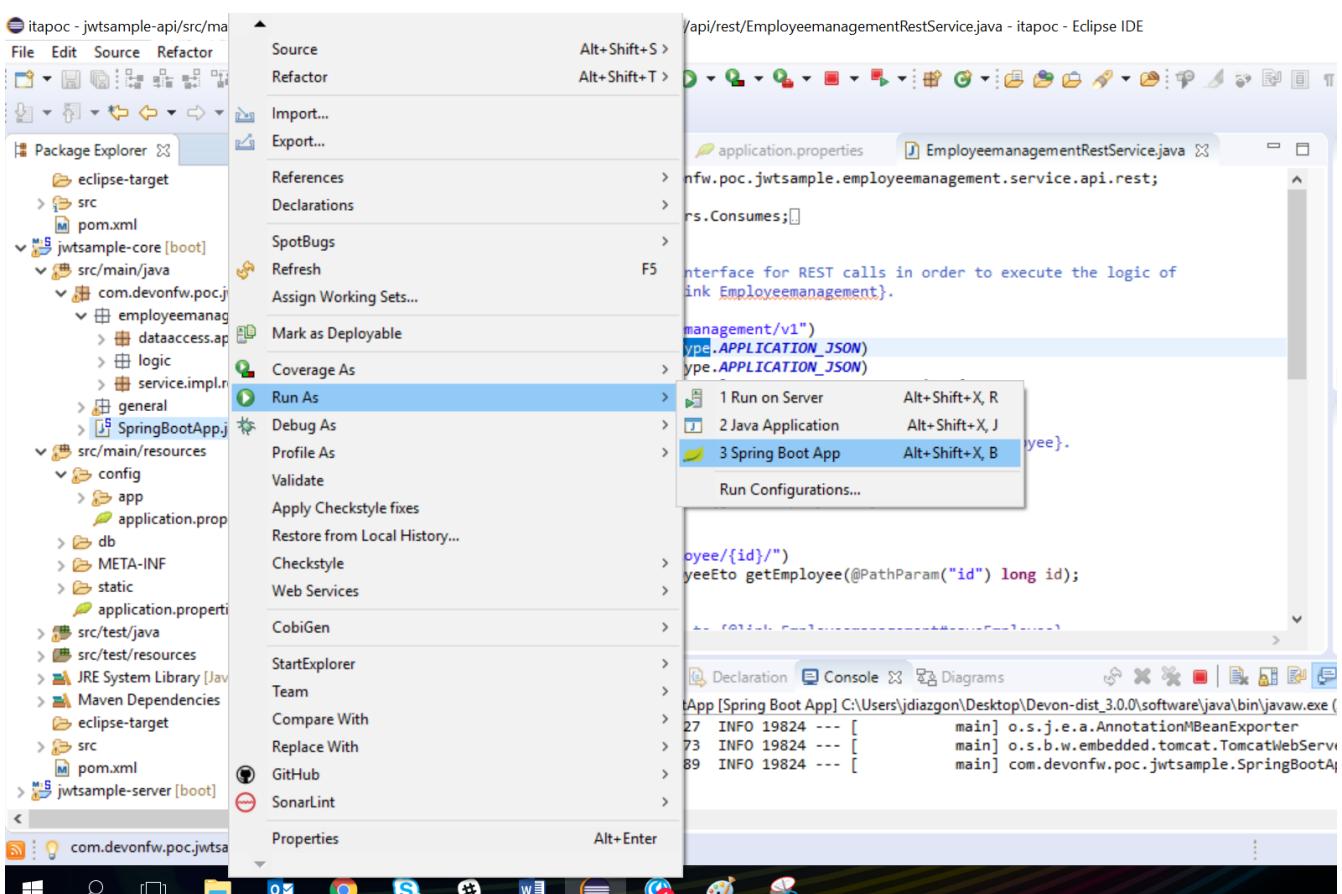
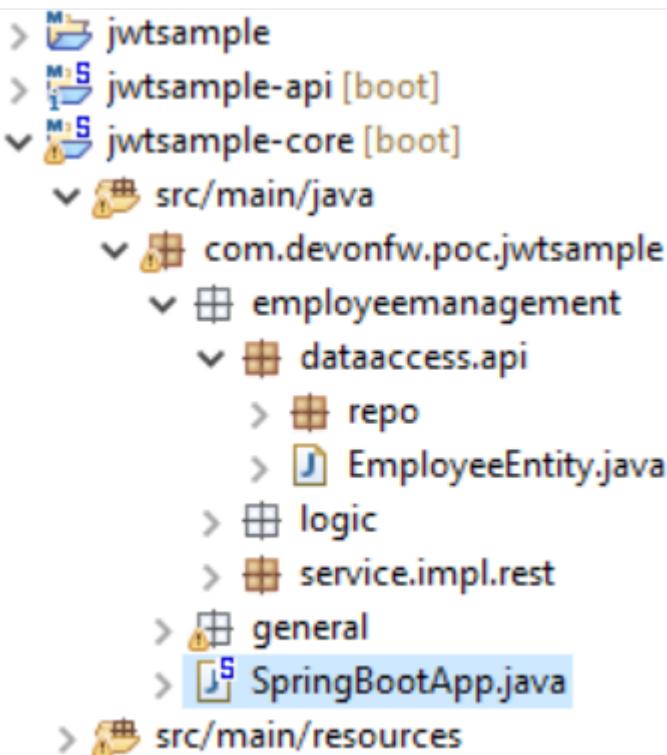


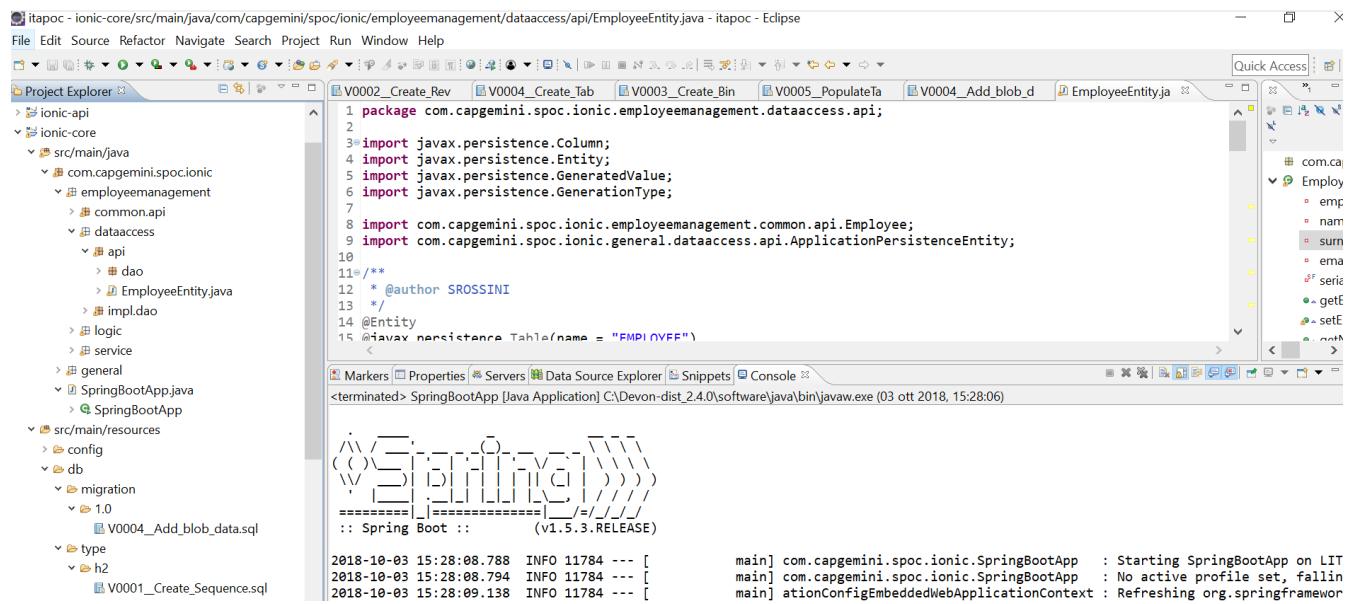
Some other classes will be generated on the core part (*myapp-core*), normally it will be implementations as shown below:



BEFORE to generate the FE, please start the Tomcat server to check that BE Layer has been generated properly.

To start a server you just have to right click on *SpringBootApp.java* → *run as* → *Spring Boot app*

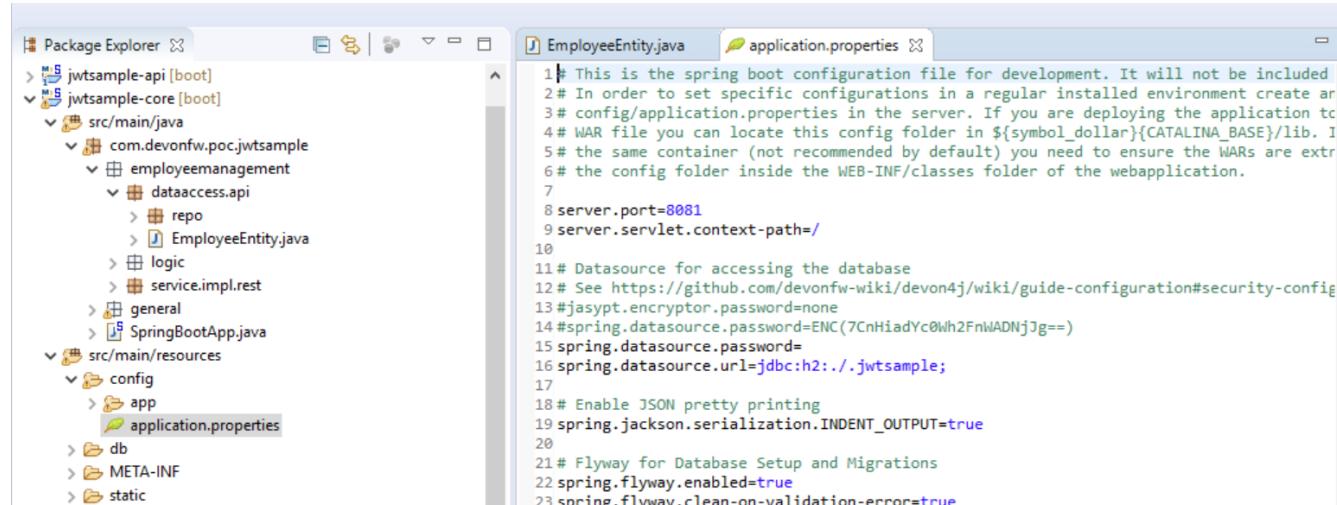




BE DONE

Last but not least: We make a quick REST services test !

See in the application.properties the TCP Port and the PATH



Now compose the Rest service URL:

service class path>/<service method path>

- <server> refers to server with port no. (ie: localhost:8081)
 - <app> is in the application.properties (empty in our case, see above)
 - <rest service class path> refers to EmployeeManagementRestService: (i.e: /employeemanagement/v1)
 - <service method path>/employee/{id} (i.e: for getEmployee method)

The screenshot shows the Eclipse IDE interface with the following details:

- File Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Java development toolbar with icons for file operations, search, and project management.
- Package Explorer:** Shows the project structure:
 - jtapsample
 - jtapsample-api [boot]
 - src/main/java
 - com.devonfw.poc.jtapsample
 - employeemanagement
 - common.api
 - logic.api
 - to
 - EmployeeEto.java
 - EmployeeSearchCriteriaTo.java
 - usecase
 - EmployeeManagement.java
 - service.api.rest
 - EmployeemanagementRestService.java
 - general
 - JRE System Library [JavaSE-1.8]
 - Maven Dependencies
 - eclipse-target
 - src
 - pom.xml
 - EmployeeEntity.java**: Opened in the editor tab.
 - application.properties**: Opened in the editor tab.
 - EmployeemanagementRestService.java**: Opened in the editor tab.

URL of getEmployee for this example is:

for all employees

<http://localhost:8081/services/rest/employeemanagement/v1/employee/search>

for the specific employee

<http://localhost:8081/services/rest/employeemanagement/v1/employee/1>

Now download [Postman](#) to test the rest services.

Once done, you have to create a POST Request for the LOGIN and insert in the body the JSON containing the username and password *waiter*

The screenshot shows the Postman application interface with the following details:

- Header Bar:** Postman, File, Edit, View, Help.
- Toolbar:** New, Import, Runner, Filter, My Workspace, Invite, Examples (0).
- Left Sidebar:** History (selected), Collections.
- Request Details:**
 - Method:** POST
 - URL:** http://localhost:8081/services/rest/login
 - Headers:** (1)
 - Body:** (Green dot) - selected tab, form-data, x-www-form-urlencoded, raw, binary, JSON (application/json).
 - Params:** (Grey dot)
 - Send:** (Blue button)
 - Save:** (Grey button)
 - Cookies:** (Grey button)
 - Code:** (Grey button)
- Body Content:**

```
1 + [
2 "j_username": "waiter",
3 "j_password": "waiter"
4 }
```

Once done with success (**Status: 200 OK**) ...

... We create a NEW GET Request in order to get one employee

The screenshot shows the Postman interface with an 'Untitled Request'. A GET request is made to `http://localhost:8081/services/rest/employeemanagement/v1/employee/1`. The 'Body' tab is selected, displaying a JSON response with the following data:

```

1 {
2   "id": 1,
3   "modificationCounter": 1,
4   "employeeId": 1,
5   "name": "Stefano",
6   "surname": "Rossini",
7   "email": "stefano.rossini@capgemini.com"
8 }

```

Now you can click

Send

Now you've to check that response has got Status: 200 OK and to see the below Employee

The screenshot shows the Postman interface with the 'Body' tab selected. The response status is 200 OK. The JSON response is identical to the one shown in the previous screenshot.

Now that We have successfully tested the BE is time to go to create the FE !

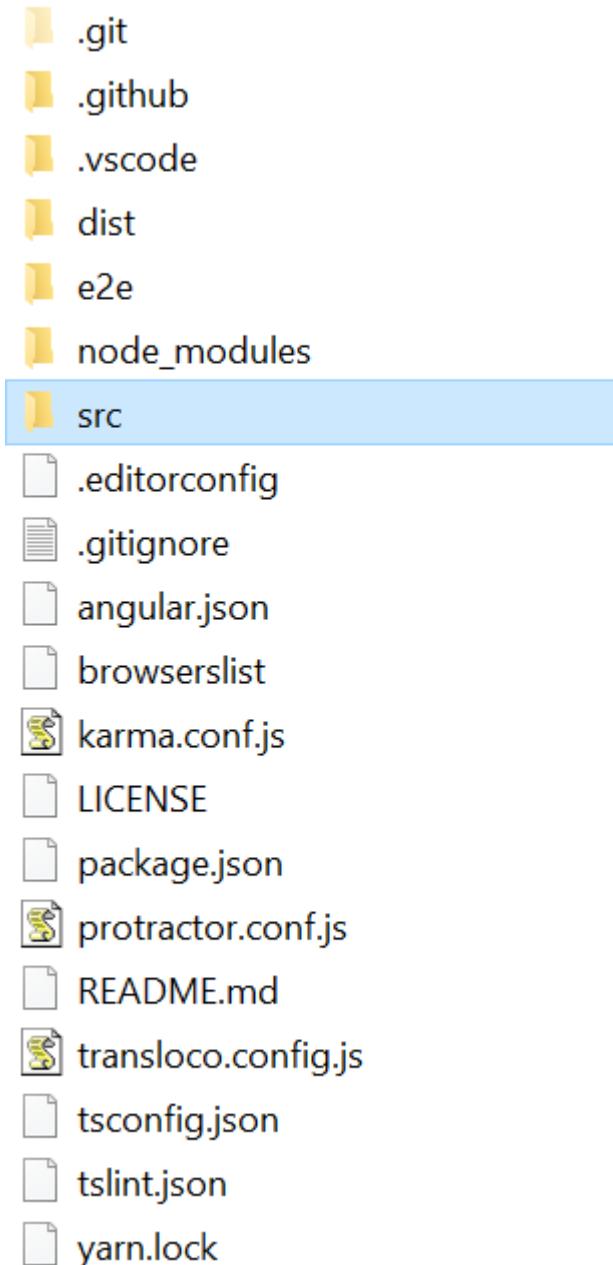
Front End

Let's start now with angular Web and then Ionic app.

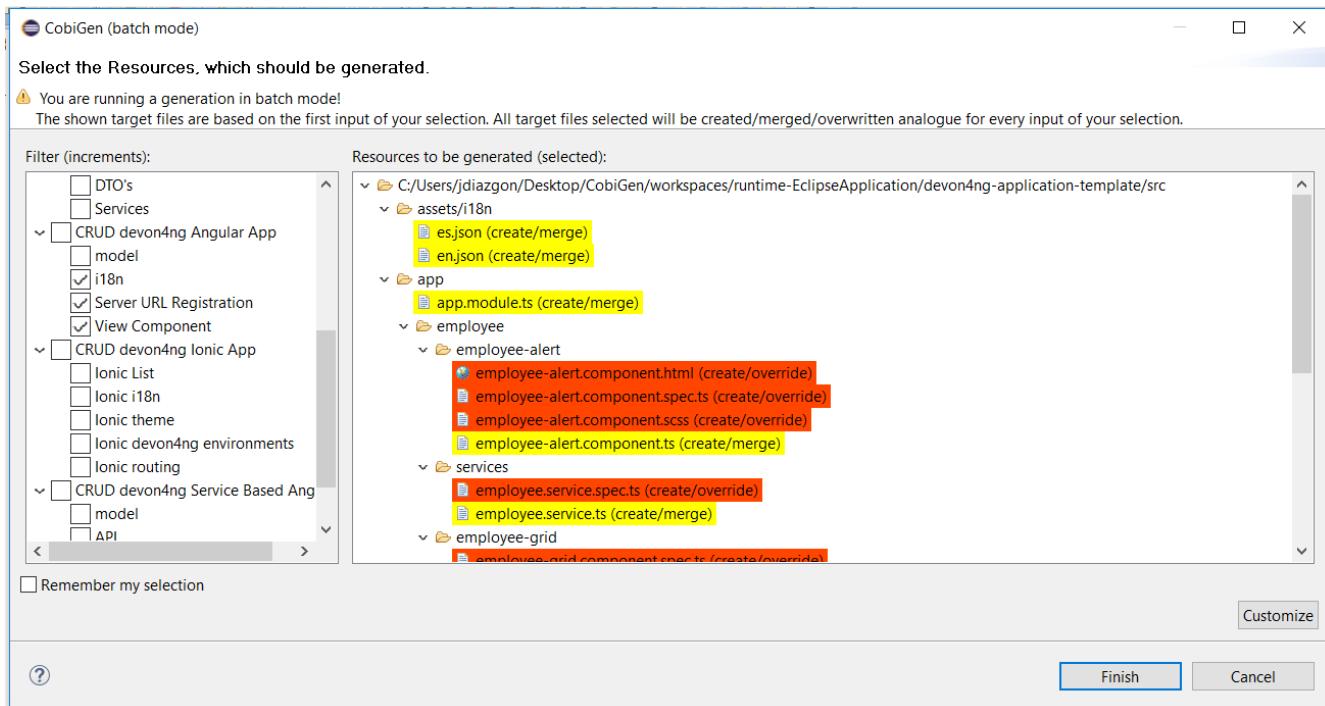
Angular Web App

1. To generate angular structure, download or clone **devon4ng-application-template** from

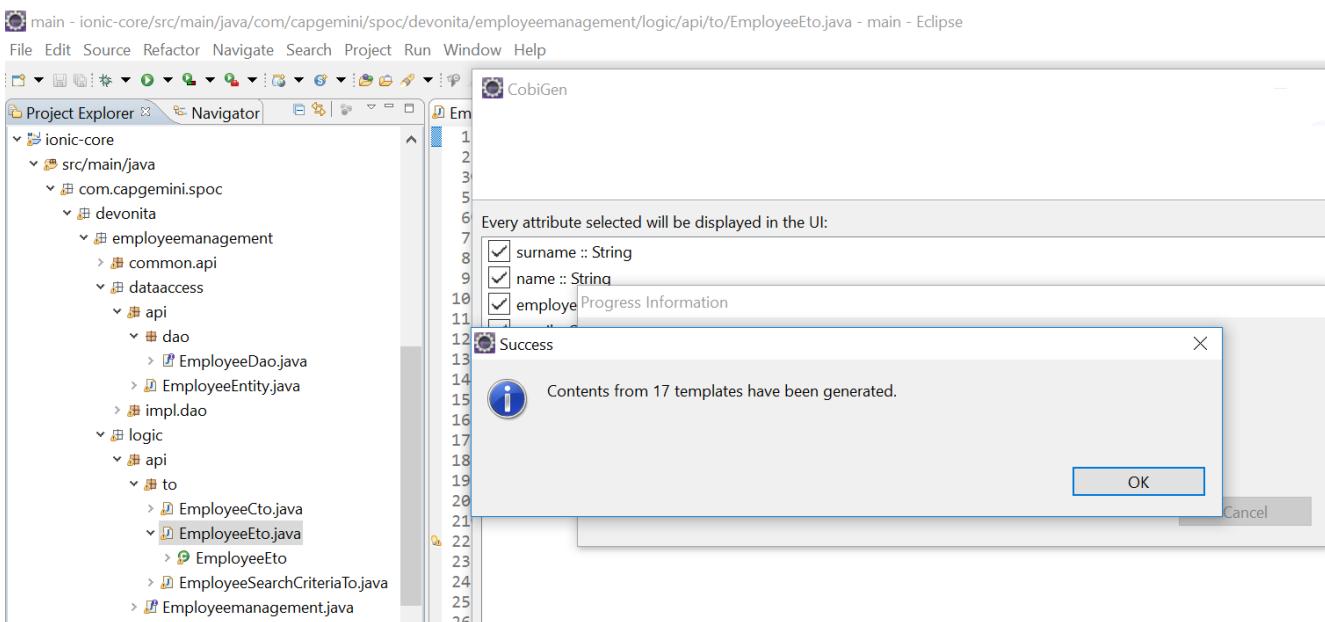
<https://github.com/devonfw/devon4ng-application-template>



- Once done, right click on EmployeeEto.java file present under the package *com.devonfw.poc.employeemanagement.logic.api.to*



4. Click on Finish



5. The entire ANGULAR structure has been auto generated. The generated code will be merged to the existing.

```

1 // Karma configuration file, see link for more information
2 // https://karma-runner.github.io/1.0/config/configuration-file.html
3
4 module.exports = function (config) {
5   config.set({
6     basePath: '',
7     frameworks: ['jasmine', '@angular/cli'],
8     plugins: [
9       require('karma-jasmine'),
10      require('karma-chrome-launcher'),
11      require('karma-jasmine-html-reporter'),
12      require('karma-coverage-istanbul-reporter'),
13      require('@angular/cli/plugins/karma')
14    ],
15    client:{},
16    clearContext: false // leave Jasmine Spec Runner output visible in browser
17  },
18  coverageIstanbulReporter: {
19    reports: [ 'html', 'lcovonly' ],
20    fixWebpackSourcePaths: true
21  },
22  angularCli: {
23    environment: 'dev'
24  },
25  reporters: ['progress', 'kjhtml'],
26  port: 9876,
27  colors: true,
28  logLevel: config.LOG_INFO,
29  autoWatch: true,
30  browsers: ['Chrome'],
31  singleRun: false

```

6. IMPORTANT now you have to add in the **app-routing.module.ts** file the next content, as a child of HomeComponent, in order to enable the route of the new generated component

```

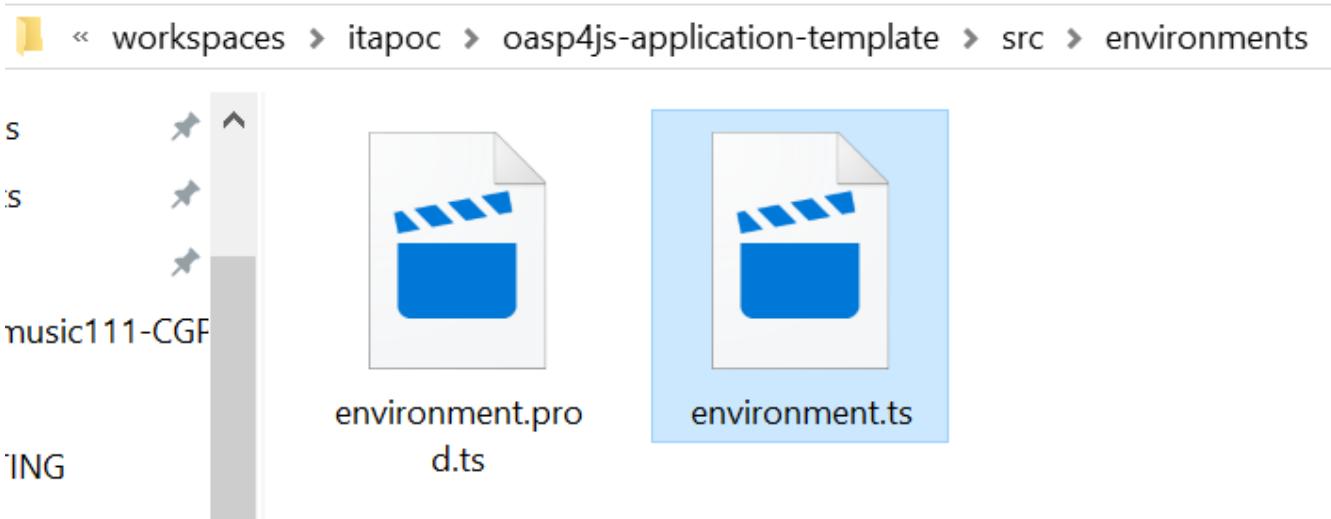
, \{
  path: 'employee',
  component: EmployeeGridComponent,
  canActivate: [AuthGuard],
},

```

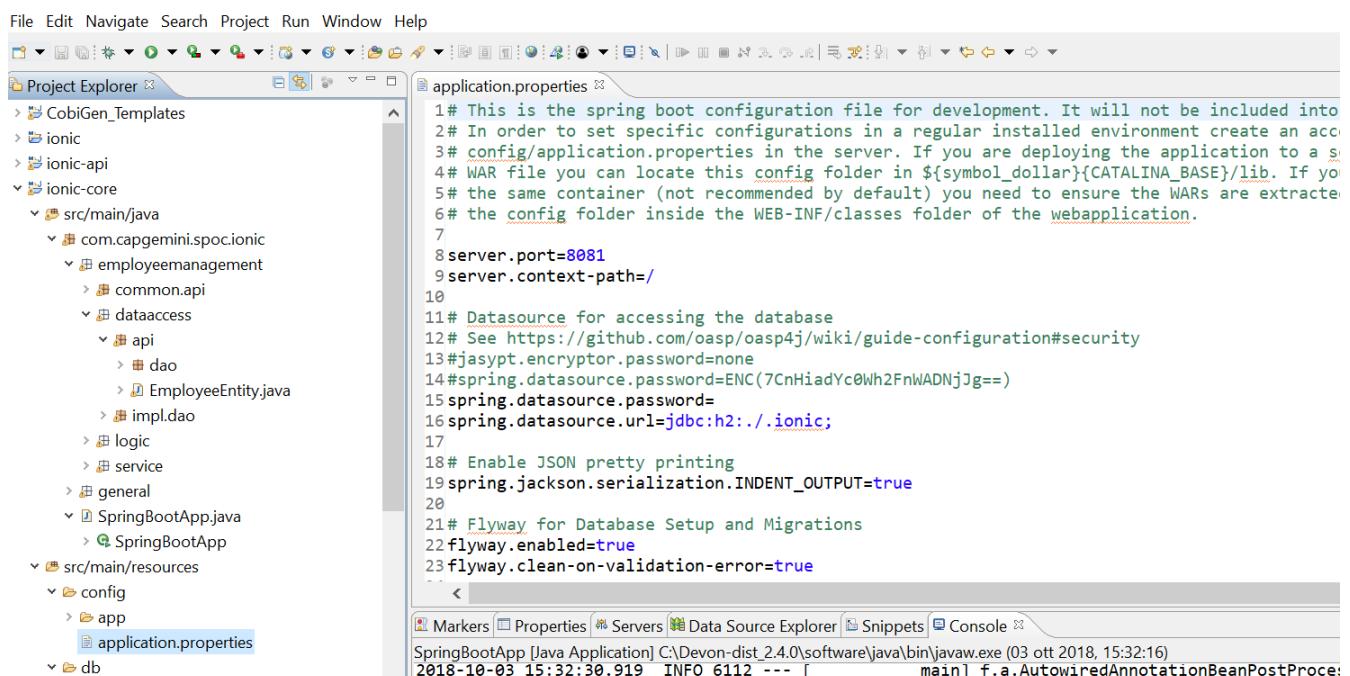
Following picture explain where to place the above content:

```
const routes: Routes = [
  {
    path: 'login',
    component: LoginComponent
  }, {
    path: 'home',
    component: HomeComponent,
    canActivate: [AuthGuard],
    children: [
      {
        path: '',
        redirectTo: '/home/initialPage',
        pathMatch: 'full',
        canActivate: [AuthGuard]
      }, {
        path: 'initialPage',
        component: InitialPageComponent,
        canActivate: [AuthGuard]
      }, {
        path: 'sampleData',
        component: SampleDataGridComponent,
        canActivate: [AuthGuard]
      }, {
        path: 'employee',
        component: EmployeeGridComponent,
        canActivate: [AuthGuard]
      }
    ]
  }, {
    path: '',
    redirectTo: '/login',
    pathMatch: 'full'
  }
]
```

7. Open the command prompt and execute `devon yarn install` from the base folder, which would download all the required libraries..
8. Check the file **environment.ts** if the server path is correct. (for production you will have to change also the environment.prod.ts file)



In order to do that it's important to look at the application.properties to see the values as PATH, TCP port etc ...



For example in this case the URL should be since the context path is empty the server URLs should be like:

```

export const environment = {
  production: false,
  restPathRoot: 'http://localhost:8081/',
  restServiceRoot: 'http://localhost:8081/services/rest/',
  security: 'jwt'
};

```

Warning: REMEMBER to set security filed to **jwt**, if it is not configured already.

- Now run the **ng serve -o** command to run the Angular Application.

```
cmd C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\sadevadi>D:
D:>cd D:\Devon\30Jan_2018\Devon-dist_2.2.0\Devon-dist_2.2.0\workspaces\main\oasp4js-application-template
D:\Devon\30Jan_2018\Devon-dist_2.2.0\Devon-dist_2.2.0\workspaces\main\oasp4js-application-template>ng serve -o
```

```
7601]
corporation. All rights reserved.

n-dist_2.2.0\Devon-dist_2.2.0\workspaces\main\oasp4js-ap
2.2.0\Devon-dist_2.2.0\workspaces\main\oasp4js-ap
```

10. If the command execution is **successful**, the below screen will **appear** and it would be automatically redirected to the url:

<http://localhost:4200/login>

EmployeeID	Name_EN	Surname_EN	Email_EN
1	Stefano	Rossini	stefano.rossini@capgemini.com
2	Angelo	Muresu	angelo.muresu@capgemini.com
3	Jaime	Diaz	jaime.diaz-gonzalez@capgemini.com

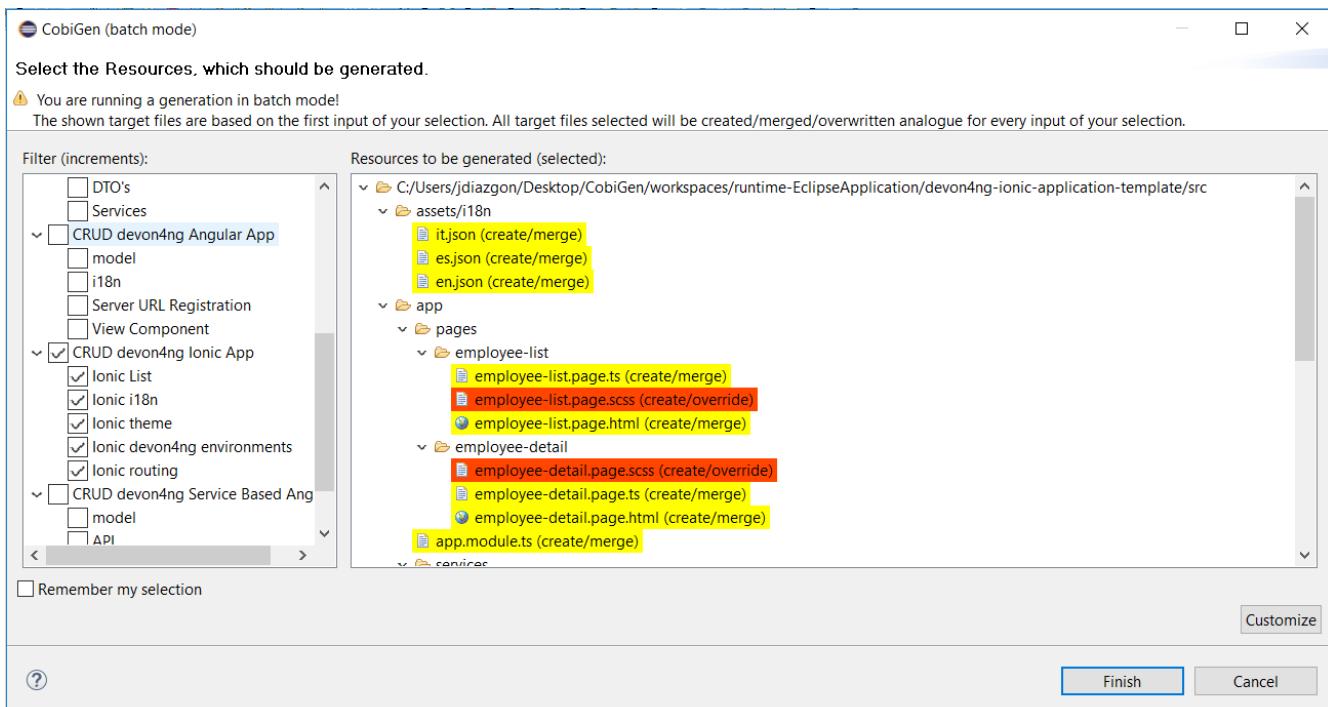
WebApp DONE

Ionic Mobile App

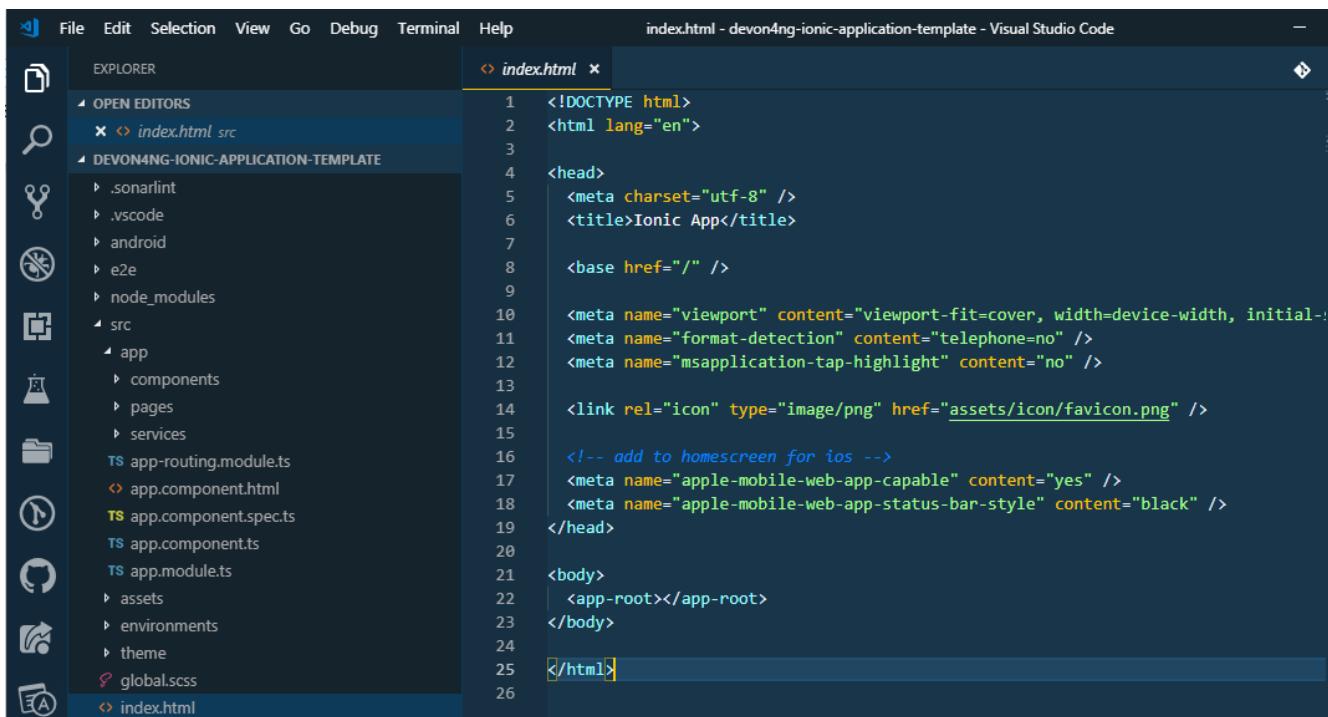
- To generate Ionic structure, download or clone **devon4ng-application-template** from

<https://github.com/devonfw/devon4ng-ionic-application-template>

2. Once done, Right click on the **EmployeeEto** as you already did before in order to use CobiGen.
3. Click on the selected options as seen in the screenshot:



4. Click on Finish
5. The entire ionic structure will be auto generated.



6. Change the server url (with correct serve url) in environment.ts, environment.prod.ts and environment.android.ts files (i.e: itapoc\devon4ng-ionic-application-template\src\environments).

The angular.json file inside the project has already a build configuration for android.

```

1 // This file can be replaced during build by using the 'fileReplacements' field in 'angular.json'.
2 // The list of file replacements can be found in 'angular.json'
3
4 export const environment = {
5   production: false,
6 };
7
8 export const SERVER_URL = 'http://localhost:8081/';
9
10 /*
11 * For easier debugging in development mode, you can import the
12 * to ignore zone related error stack frames such as 'zone.run'
13 */
14
15 /* This import should be commented out in production mode because
16 * on performance if an error is thrown.
17 */
18 // Import 'zone.js/dist/zone-error'; // Included with Angular
19

```

```

1 export const environment = {
2   production: false,
3 };
4
5 export const SERVER_URL = 'http://10.0.2.2:8081/';
6

```

```

5 "newProjectRoot": "projects",
6 "projects": {
7   "app": {
8     "root": "",
9     "sourceRoot": "src",
10    "projectType": "application",
11    "prefix": "app",
12    "schematics": {},
13    "architect": {
14      "build": {
15        "builder": "@angular-devkit/build-angular:browser",
16        "options": { ... },
17        "configurations": {
18          "production": { ... }
19        }
20      },
21      "android": {
22        "fileReplacements": [
23          {
24            "replace": "src/environments/environment.ts",
25            "with": "src/environments/environment.android.ts"
26          }
27        ],
28        "i18n": {
29          "progress": false
30        }
31      }
32    },
33    "serve": {
34      "builder": "@angular-devkit/build-angular:dev-server",
35      "options": { ... },
36      "configurations": {
37        "production": { ... },
38        "browserTarget": "app:build:production"
39      },
40      "ci": {
41        "progress": false
42      }
43    }
44  }
45 }

```

7. Run npm install in the root folder to download the dependencies

8. Run ionic serve

```

C:\Devon-dist-current\Devon-dist_3.0.0\workspaces\Test\devon4ng-ionic-application-template
λ ionic serve
> ng run app:serve --host=0.0.0.0 --port=8100
[ng] WARNING: This is a simple server for use in testing or debugging Angular applications
[ng] locally. It hasn't been reviewed for security issues.
[ng] Binding this server to an open connection can result in compromising your application or
[ng] computer. Using a different host than the one passed to the "--host" flag might result in
[ng] websocket connection issues. You might need to use "--disableHostCheck" if that's the
[ng] case.
[INFO] Waiting for connectivity with ng...
[INFO] Waiting for connectivity with ng...

[INFO] Development server running!

Local: http://localhost:8100
External: http://10.80.132.29:8100, http://192.168.56.1:8100, http://192.168.99.1:8100

Use Ctrl+C to quit this process

[INFO] Browser window opened to http://localhost:8100!

[ng] i 「wdm」: wait until bundle finished: /

```

11.

Once the execution is successful

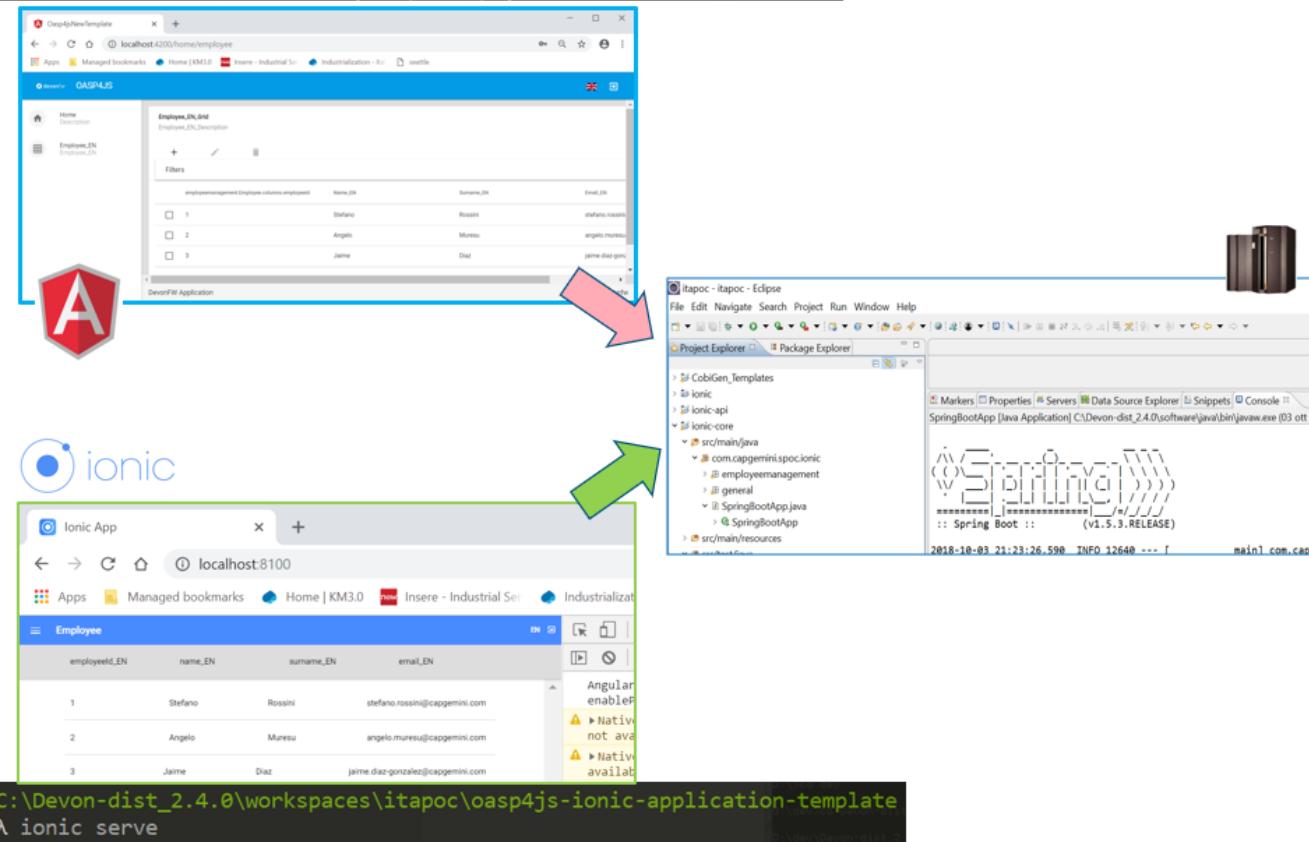
employeeId_EN	name_EN	surname_EN	email_EN
1	Stefano	Rossini	stefano.rossini@capgemini.com
2	Angelo	Muresu	angelo.muresu@capgemini.com
3	Jaime	Gonzalez	jaime.diaz-gonzalez@capgemini.com

- Mobile App DONE*

So: well done

Starting from an Entity class you've successfully generated the Back-End layer (REST, SOAP, DTO, Spring services, Hibernate DAO), the Angular Web App and the Ionic mobile App!

```
C:\Devon-dist_2.4.0\workspaces\itapoc\oasp4js-application-template
λ ng serve -o
```

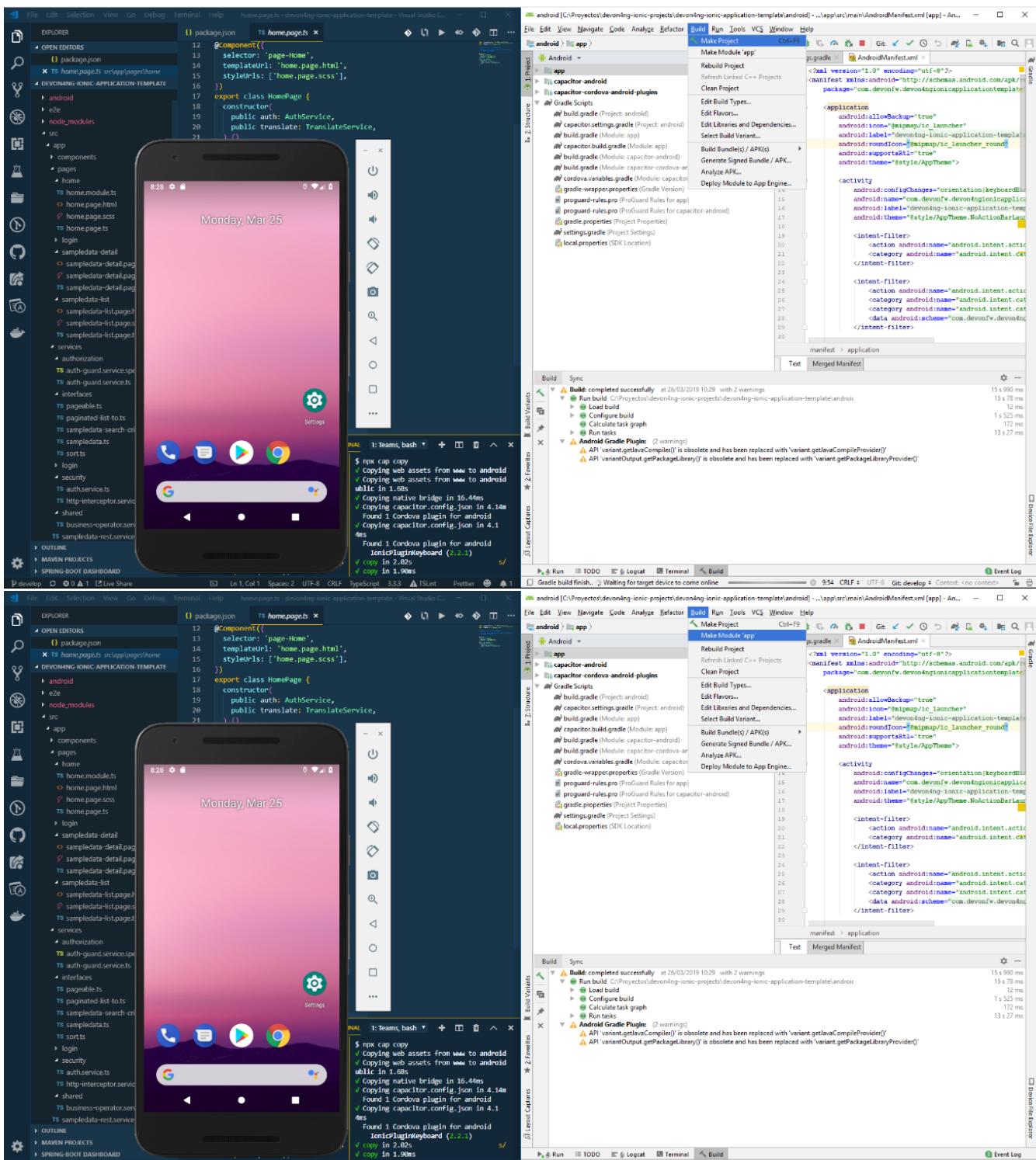


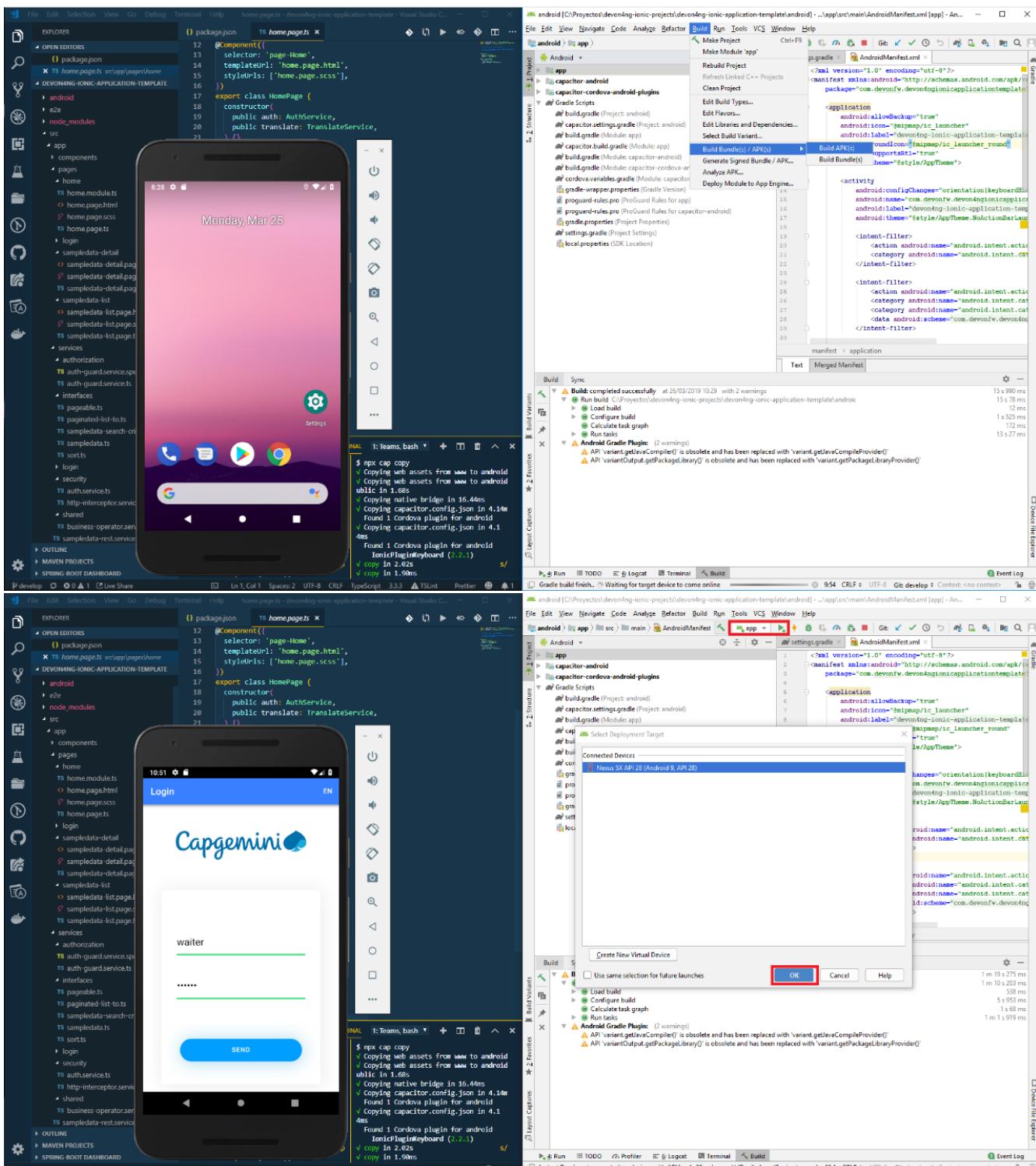
Build APK

Since We're going to create apk remember the following pre-conditions:

- [Gradle](#)
- [Android Studio](#)
- [Android sdk](#)
- [Capacitor](#)

1. Now, open cmd and type the path where your *devon4ng-ionic-application-template* project is present.
2. Run the following commands:
 - a. npx cap init
 - b. ionic build --configuration=android
 - c. npx cap add android
 - d. npx cap copy
 - e. npx cap open android
3. Build the APK using Android studio.





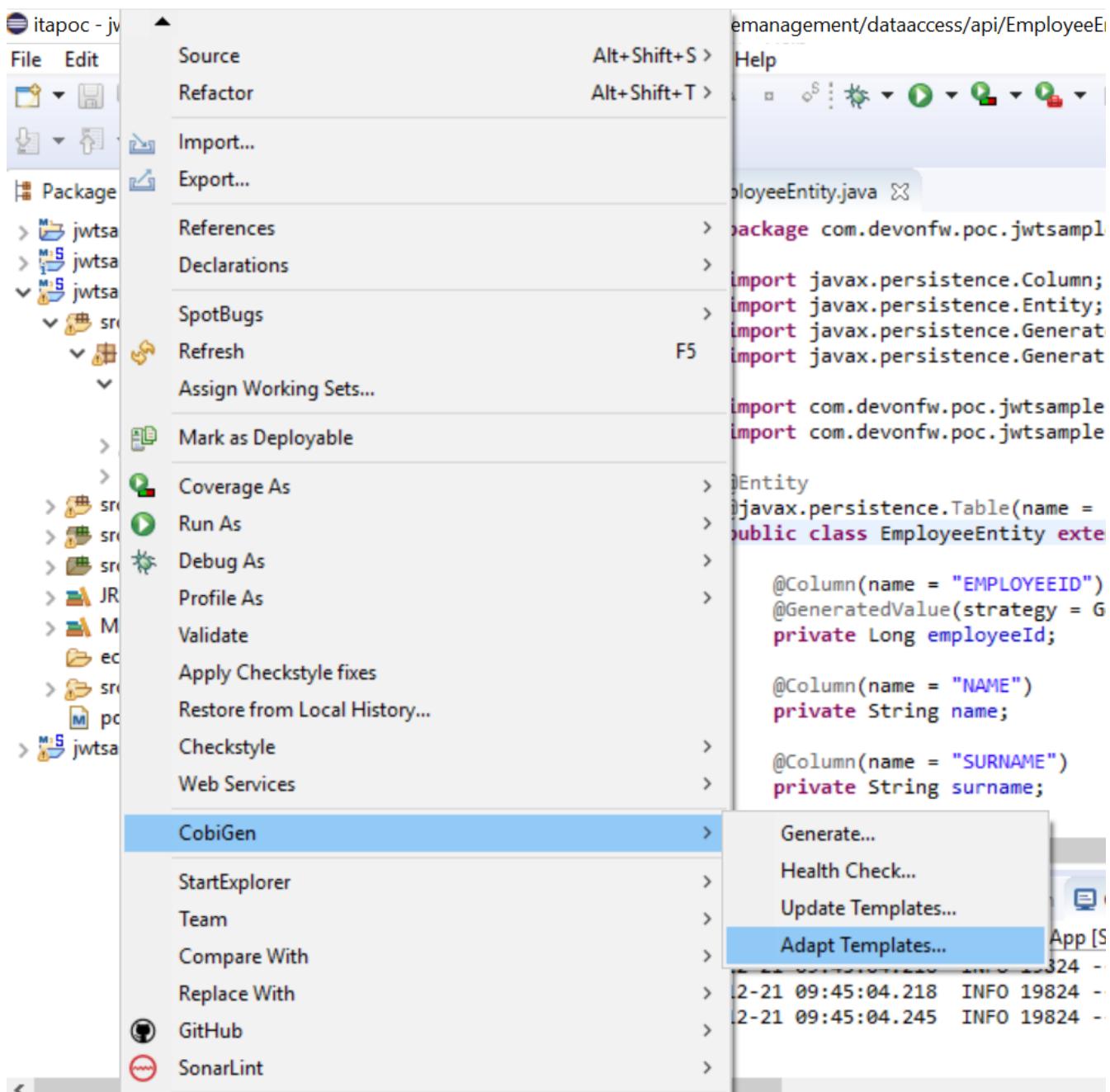
You can find your apk file in

/devon4ng-ionic-application-template/android/app/build/outputs/apk/debug

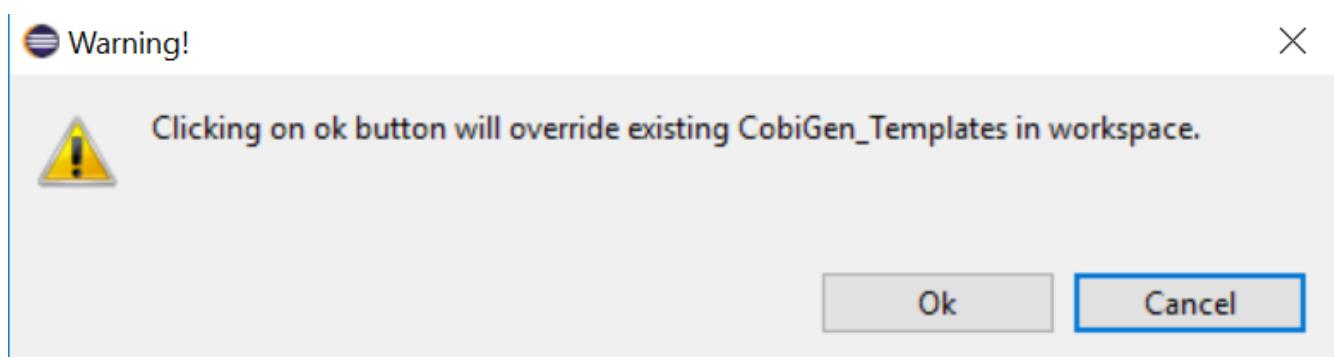
73.8.3. Adapt CobiGen_Templates

After following this tutorial, you will have the CobiGen_Templates downloaded on your local machine. To import these templates you need to do the following:

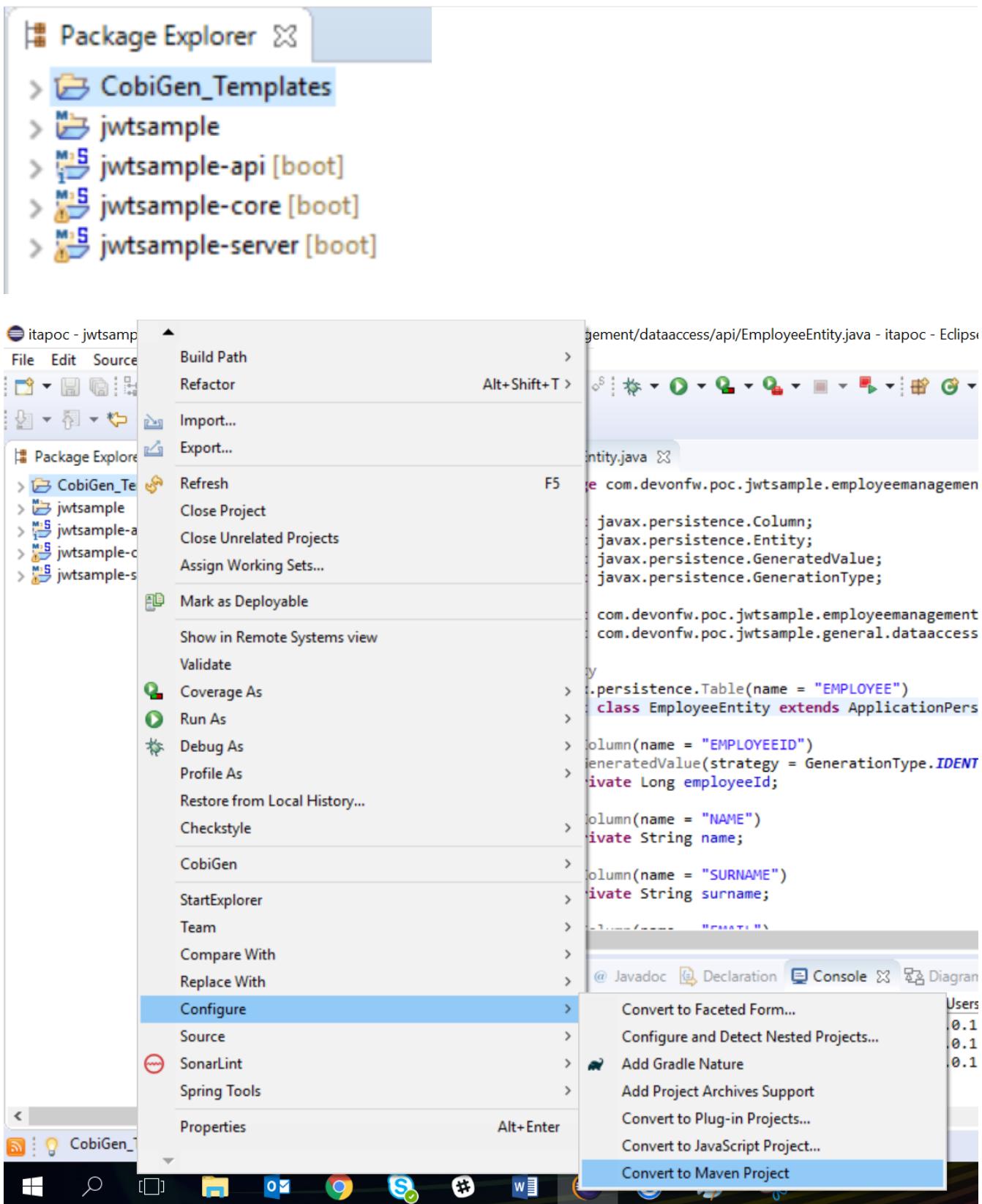
Right click in any part of the package explorer, then click on CobiGen → Adapt templates



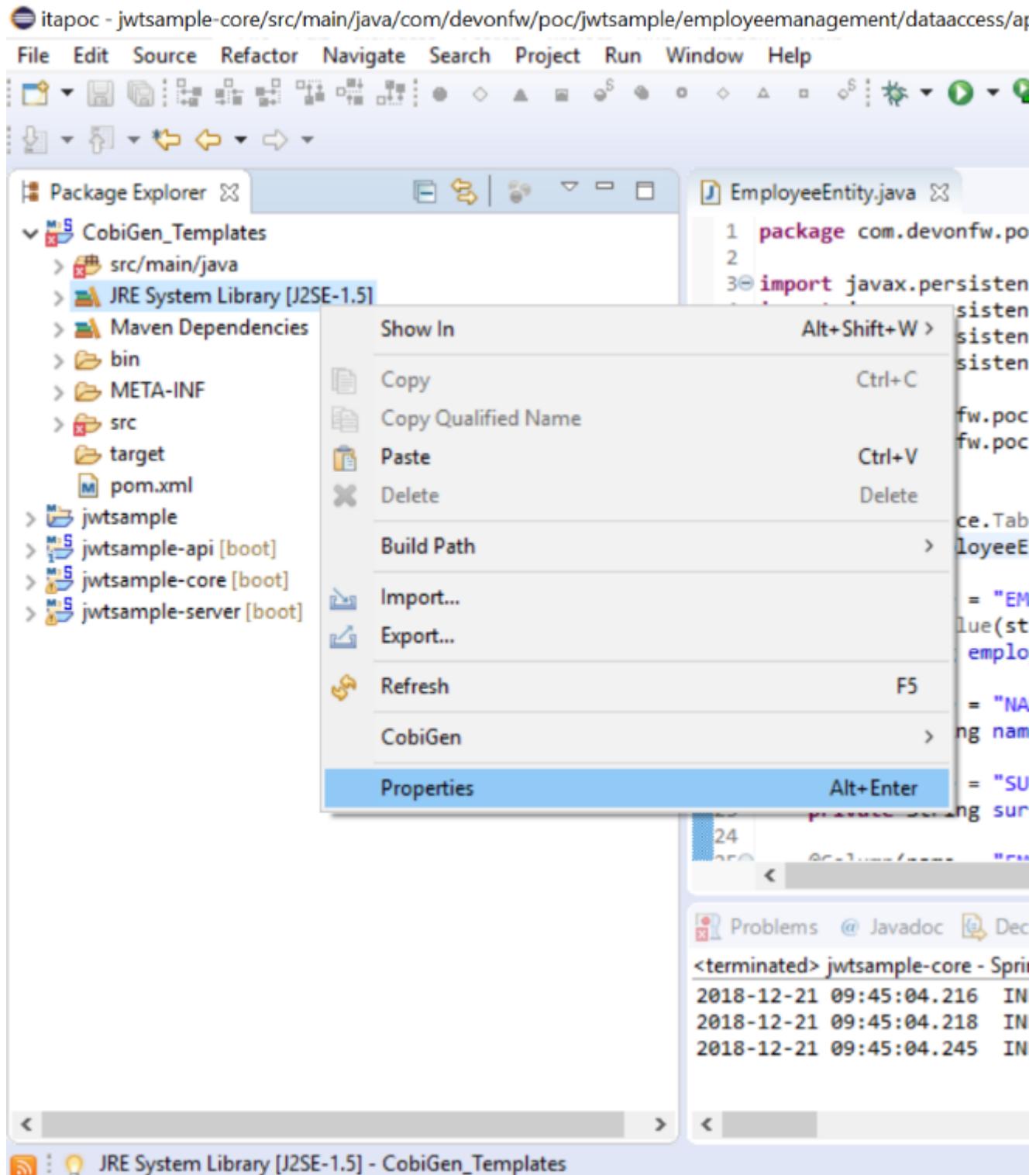
Click Ok:



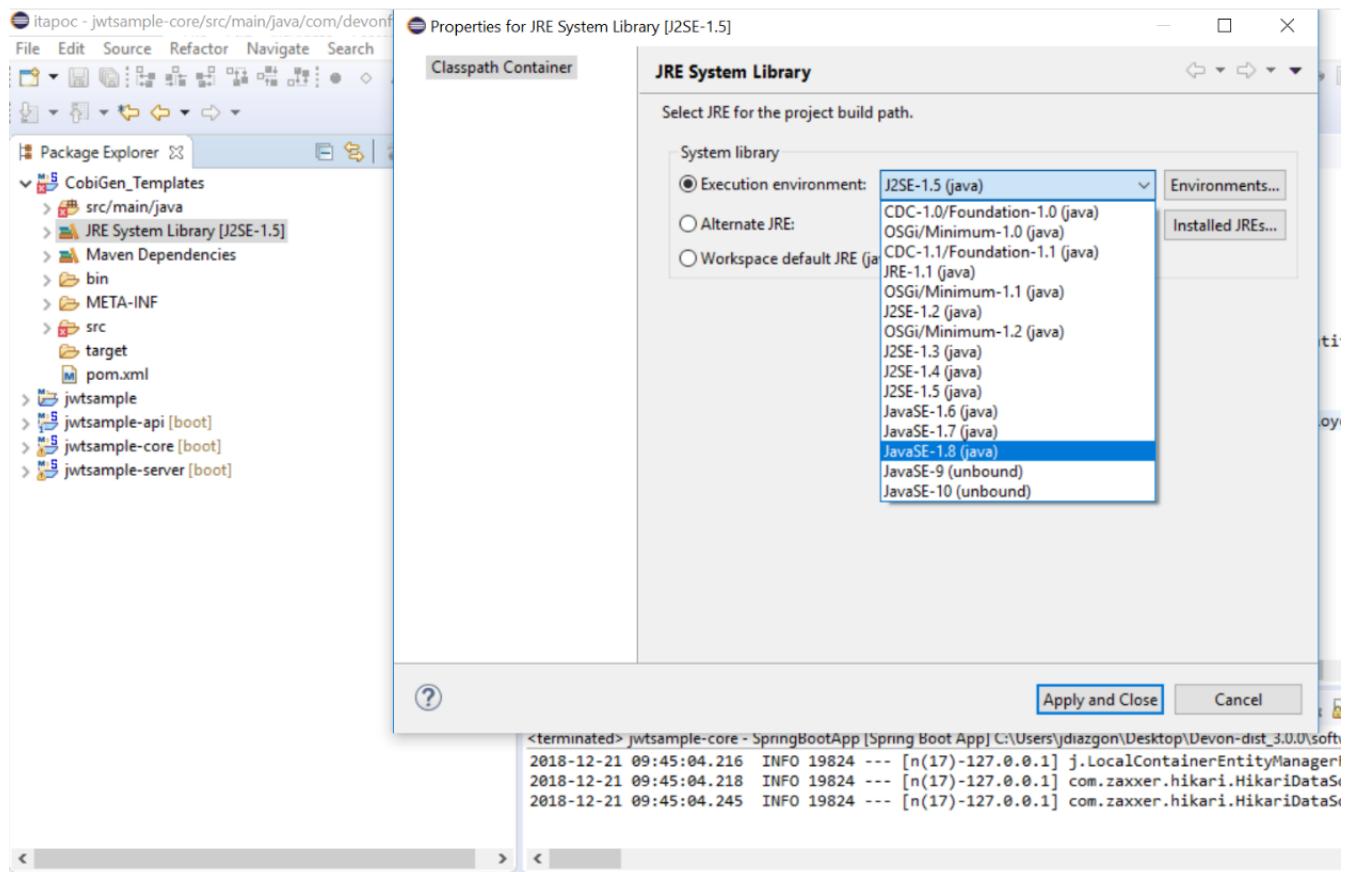
Now the CobiGen_Templates project will be automatically imported into your workspace, as shown on the image below:



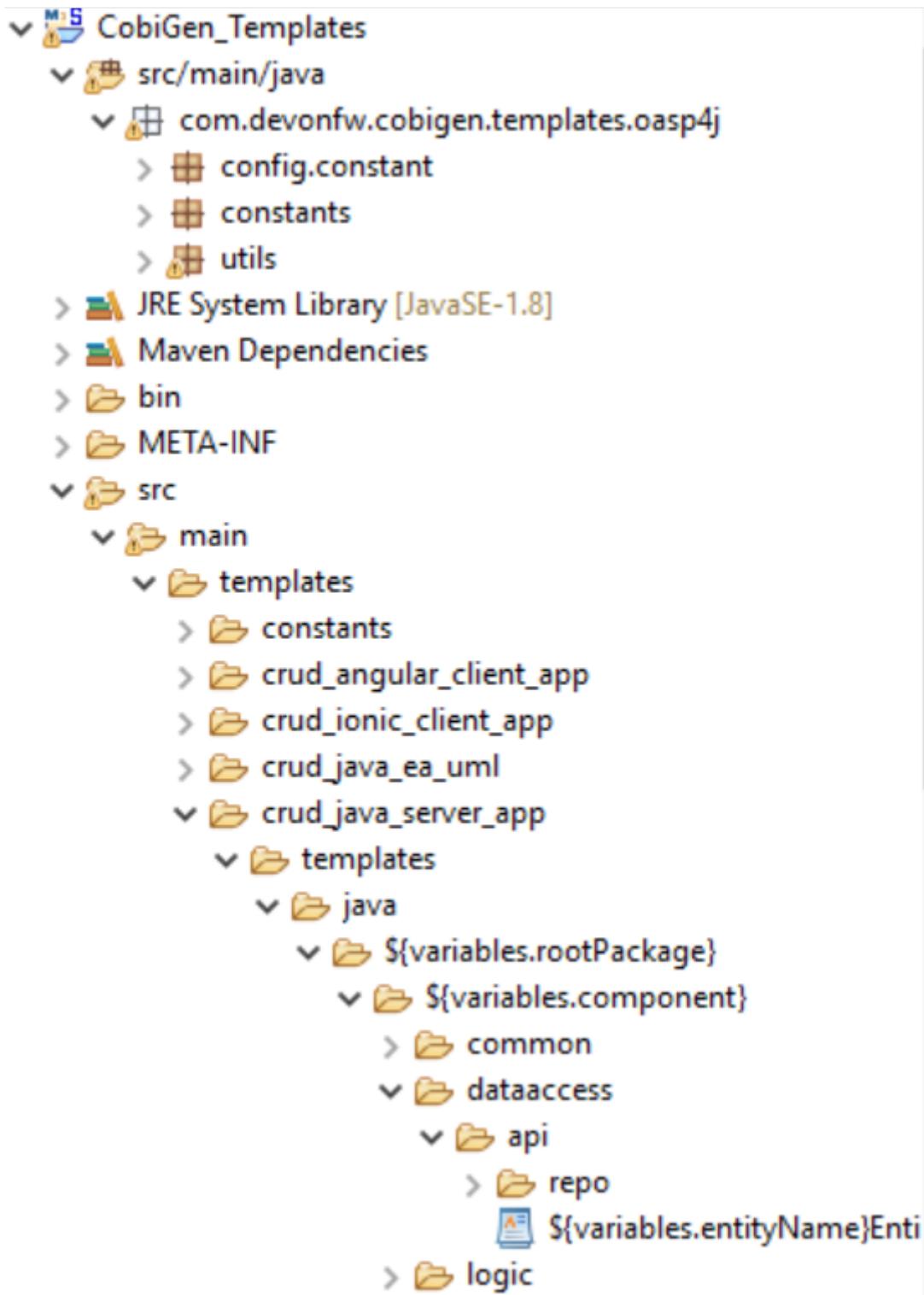
Now you just need to change the Java version of the project to JRE 1.8. Right click on the JRE system library, and then on *Properties*:



Now change the version to Java 1.8



Now you have successfully imported the CobiGen templates. If you want to edit them, you will find them in the folder `src/main/templates`. For instance, the Java templates are located here:



Now you can adapt the templates as much as you want. Documentation about this can be found on:

<https://github.com/devonfw/tools-cobigen/wiki/Guide-to-the-Reader>

73.9. Enable Composite Primary Keys in Entity

In order to enable Composite Primary Keys in entity in CobiGen, the below approach is suggested

The templates in cobigen have been enhanced to support Composite primary keys while still

supporting the default devonfw/Cobigen values with Long id.

Also, the current generation from Entity still holds good - right click from an Entity object, CobiGen → Generate will show the CobiGen wizard relative to the entity generation.

After generating, below example shows how composite primary keys can be enabled.

```
@Entity
@Table(name = "employee")
public class EmployeeEntity {
    private CompositeEmployeeKey id;
    private String name;
    private String lastName;
    @Override
    @EmbeddedId
    public CompositeEmployeeKey getId() {
        return id;
    }
    @Override
    public void setId(CompositeEmployeeKey id) {
        this.id = id;
    }
    ...
}
```

```
public class CompositeEmployeeKey implements Serializable {
    private String companyId;
    private String employeeId;
```

Once the generation is complete, implement PersistenceEntity<ID>.java in the EmployeeEntity and pass the composite primary key object which is CompositeEmployeeKey in this case as the parameter ID.

```
import com.devonfw.module.basic.common.api.entity.PersistenceEntity;
@Entity
@Table(name = "employee")
public class EmployeeEntity implements PersistenceEntity<CompositeEmployeeKey> {
    private CompositeEmployeeKey id;
    private String name;
    private String lastName;
```

Also, the modificationCounter methods needs to be implemented from the interface PersistenceEntity<ID>. The sample implementation of the modification counter can be referred below.

```
@Override
public int getModificationCounter() {
    if (this.persistentEntity != null) {
        // JPA implementations will update modification counter only after the
        transaction has been committed.
        // Conversion will typically happen before and would result in the wrong (old)
        modification counter.
        // Therefore we update the modification counter here (that has to be called
        before serialization takes
        // place).
        this.modificationCounter = this.persistentEntity.getModificationCounter();
    }
    return this.modificationCounter;
}
@Override
public void setModificationCounter(int version) {
    this.modificationCounter = version;
}
```

74. Template Development

Unresolved directive in cobigen.wiki/master-cobigen.asciidoc - include::cobigen-templates_helpful-links.adoc[leveloffset=2]

Part XII: MrChecker - devonfw testing tool

75. Who Is MrChecker

75.1. Who is MrChecker?

MrChecker Test Framework is an end to end test automation framework written in Java. It is an automated testing framework for functional testing of web applications, API web services, Service Virtualization, Security, native mobile apps and, in the near future, databases. All modules have tangible examples of how to build resilient integration test cases based on delivered functions.

75.2. Where does MrChecker apply?

The main goal of MrChecker is to standardize the way we build BlackBox tests. It gives the possibility to have one common software standard in order to build Component, Integration and System tests.

A Test Engineer does not have access to the application source code in order to perform BlackBox tests, but they are able to attach their tests to any application interfaces, such as - IP address - Domain Name - communication protocol - Command Line Interface.

75.3. MrChecker's specification:

- Responsive Web Design application: Selenium Browser
- REST/SOAP: RestAssure
- Service Virtualization: Wiremock
- Database: JDBC drivers for SQL
- Security: RestAssure + RestAssure Security lib
- Standalone Java application: SWING
- Native mobile application for Android: Appium

75.4. Benefits

Every customer may benefit from using MrChecker Test Framework. The main profits for your project are:

- Resilient and robust building and validation process
- Quality gates shifted closer to the software development process
- Team quality awareness increase - including Unit Tests, Static Analysis, Security Tests, Performance in the testing process
- Test execution environment transparent to any infrastructure
- Touch base with the Cloud solution
- Faster Quality and DevOps-driven delivery

-
- Proven frameworks, technologies and processes.

75.5. Test stages

75.5.1. Unit test

A module is the smallest compilable unit of source code. It is often too small to be tested by the functional tests (black-box tests). However, it is the ideal candidate for white-box testing. White-box tests have to be performed as the first static tests (e.g. Lint and inspections), followed by dynamic tests in order to check boundaries, branches and paths. Usually, that kind of testing would require enabling stubs and special test tools.

75.5.2. Component test

This is the black-box test of modules or groups of modules which represent certain functionalities. There are no rules about what could be called a component. Whatever a tester defines as a component, should make sense and be a testable unit. Components can be integrated into bigger components step by step and tested as such.

75.5.3. Integration test

Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered. The software is completed step by step and tested by tests covering a collaboration between modules or classes. The integration depends on the kind of system. For example, the steps could be as follows: run the operating system first and gradually add one component after another, then check if the black-box tests are still running (the test cases will be extended together with every added component). The integration is done in the laboratory. It may be also completed by using simulators or emulators. Additionally, the input signals could be stimulated.

75.5.4. Software / System test

System testing is a type of testing conducted on a complete integrated system to evaluate the system's compliance with its specified requirements. This is a type of black-box testing of the complete software in the target system. The most important factor in successful system testing is that the environmental conditions for the software have to be as realistic as possible (complete original hardware in the destination environment).

76. Test Framework Modules

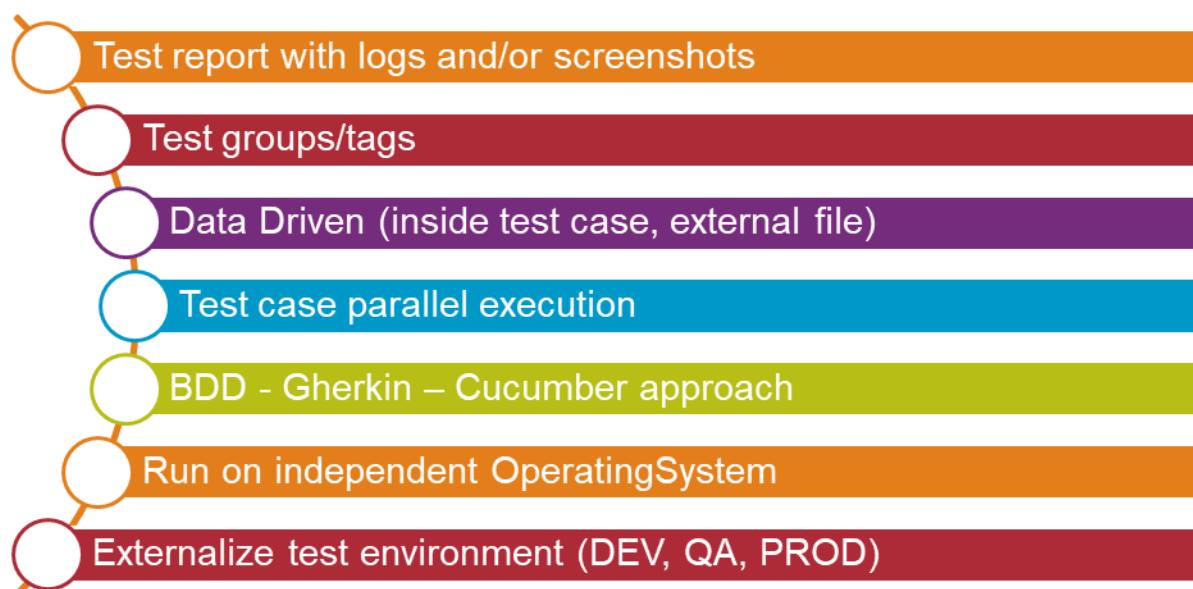
In this section, it is possible to find all the information regarding the main modules of MrChecker:

76.1. Core Test Module

76.1.1. Core Test Module

What is Core Test Module

Core functionality ingredients



Core Test Module Functions

- Test reports with logs and/or screenshots
- Test groups/tags
- Data driven approach
- Test case parallel execution
- BDD - Gherkin - Cucumber approach
- Run on independent Operating Systems
- Externalize test environment (DEV, QA, SIT, PROD)
- Encrypting sensitive data

How to start?

Read: [Framework Test Class](#)

76.1.2. Allure Logger → BFLogger

In Allure E2E Test Framework you have ability to use and log any additional information crucial for:

- test steps
- test execution
- page object actions, and many more.

Where to find saved logs

Every logged information is saved in a separate test file, as a result of parallel tests execution.

The places they are saved:

1. In test folder *C:\Allure_Test_Framework\allure-app-under-test\logs*
2. In every Allure Test report, logs are always embedded as an attachment, according to test run.

How to use logger:

- Start typing **BFLogger**
- Then type . (dot)

Type of logger:

- **BFLogger.logInfo("Your text")** - used for test steps
- **BFLogger.logDebug("Your text")** - used for non official information, either during test build process or in Page Object files
- **BFLogger.logError("Your text")** - used to emphasize critical information

```
import com.capgemini.ntc.test.core.logger.BFLogger;

public class TestCaseFile {

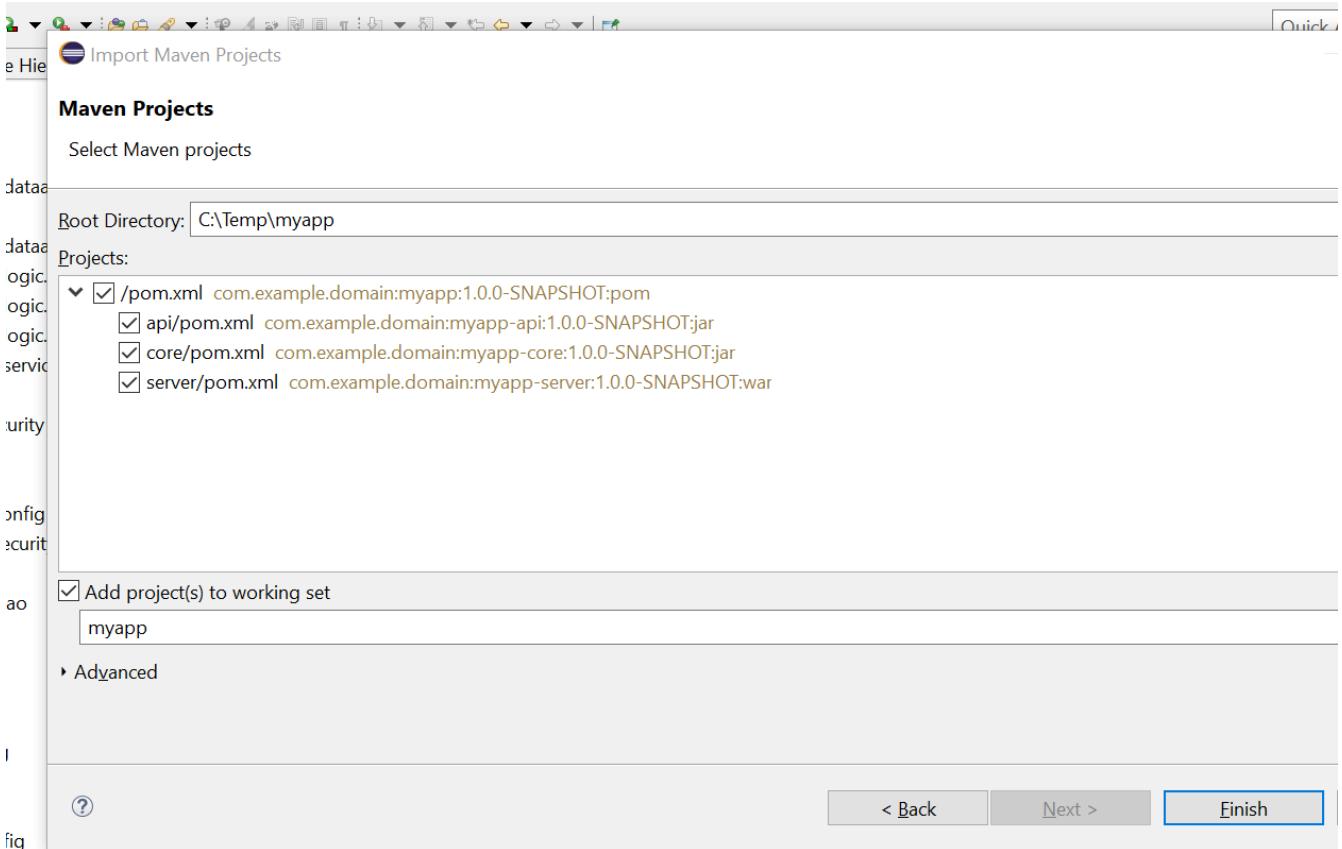
    @Test
    public void test() {

        BFLogger.logInfo("Used for test steps");

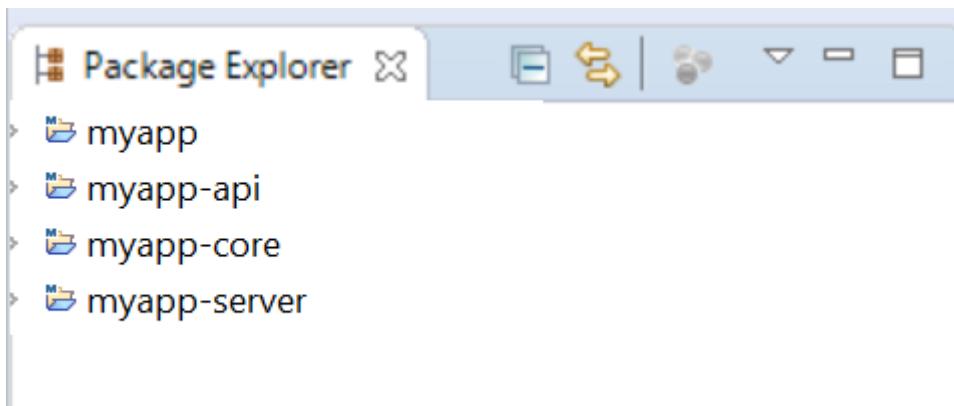
        BFLogger.logDebug("Used for non official information, "
            + "either during test build process or in Page Object files");

        BFLogger.logError("Used to emphasize critical information");
    }
}
```

Console output:



76.1.3. Allure Reports



Allure is a tool designed for test reports.

Generate report - command line

You can generate a report using one of the following commands:

Since mrchecker-core-module version 5.6.2.1:

```
mvn test allure:serve -Dgroups=TestsTag1
```

Prior to mrchecker-core-module version 5.6.2.1:

```
mvn test allure:serve -Dtest=TS_Tag1
```

A report will be generated into temp folder. Web server with results will start. You can additionally configure the server timeout. The default value is "3600" (one hour).

System property `allure.serve.timeout`.

Since mrchecker-core-module version 5.6.2.1:

```
mvn test allure:report -Dgroups=TestsTag1
```

Prior to mrchecker-core-module version 5.6.2.1:

```
mvn test allure:report -Dtest=TS_Tag1
```

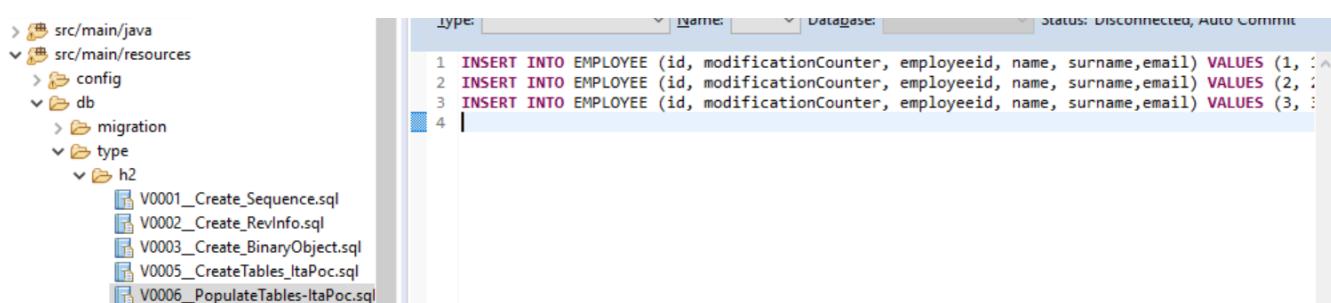
A report will be generated to directory: `target/site/allure-maven/index.html`

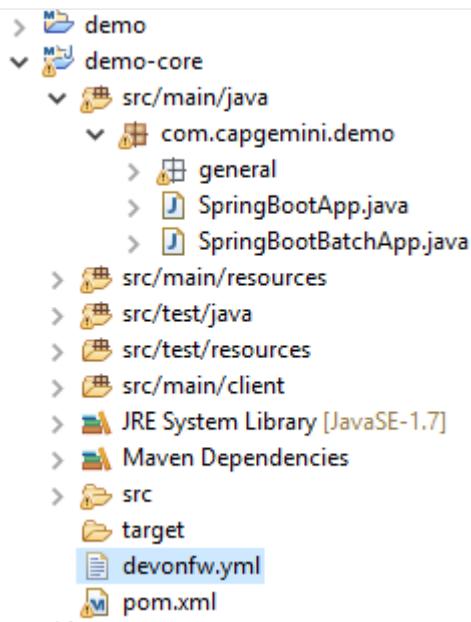
NOTE: Please open `index.html` file under Firefox. Chrome has some limitations to presenting dynamic content. If you want to open a report with a Chromium based Web Browser, you need to launch it first with `--allow-file-access-from-files` argument.

Generate report - Eclipse

A report is created here `allure-app-under-test|target|site|allure-report|index.html`

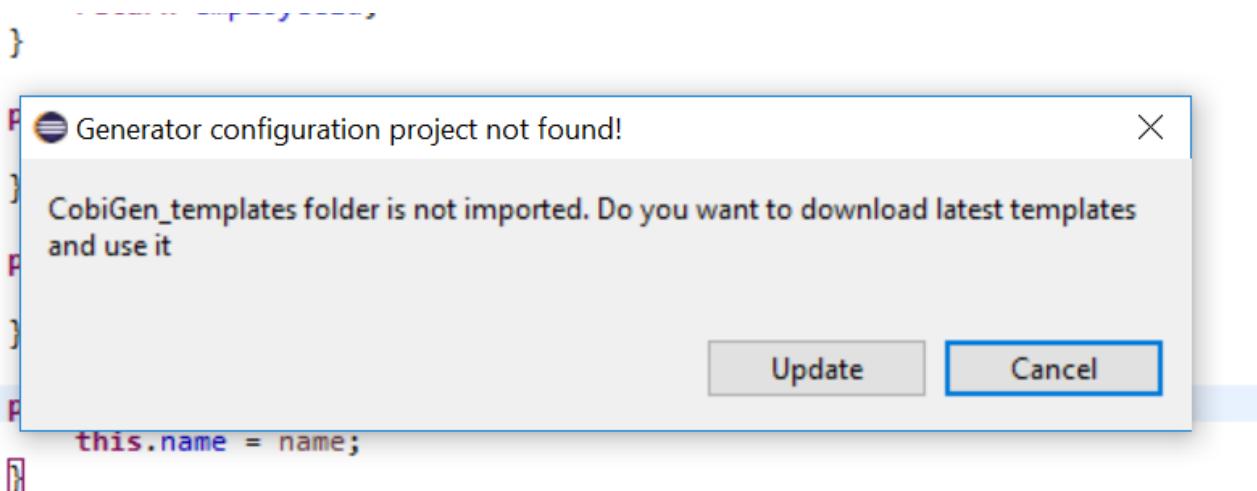
NOTE: Please open `index.html` file under Firefox. Chrome has some limitations to presenting dynamic content. If you want to open a report with a Chromium based Web Browser, you need to launch it first with `--allow-file-access-from-files` argument.





Generate report - Jenkins

In our case, we'll use the Allure Jenkins plugin. When integrating Allure in a Jenkins job configuration, we'll have direct access to the build's test report.

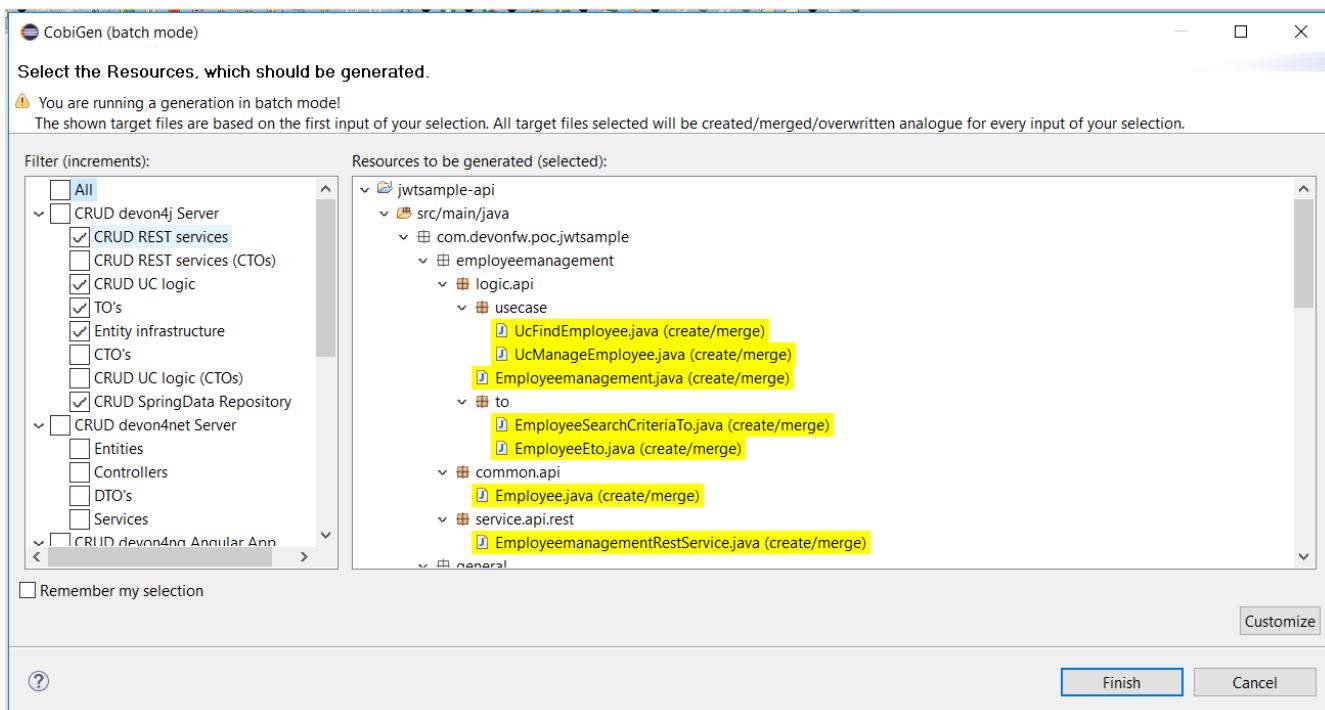


There are several ways to access the Allure Test Reports:

- Using the "Allure Report" button on the left navigation bar or center of the general job overview
- Using the "Allure Report" button on the left navigation bar or center of a specific build overview

Afterwards you'll be greeted with either the general Allure Dashboard (showing the newest build) or the Allure Dashboard for a specific (older) build.

Allure dashboard



The Dashboard provides a graphical overview on how many test cases were successful, failed or broken.

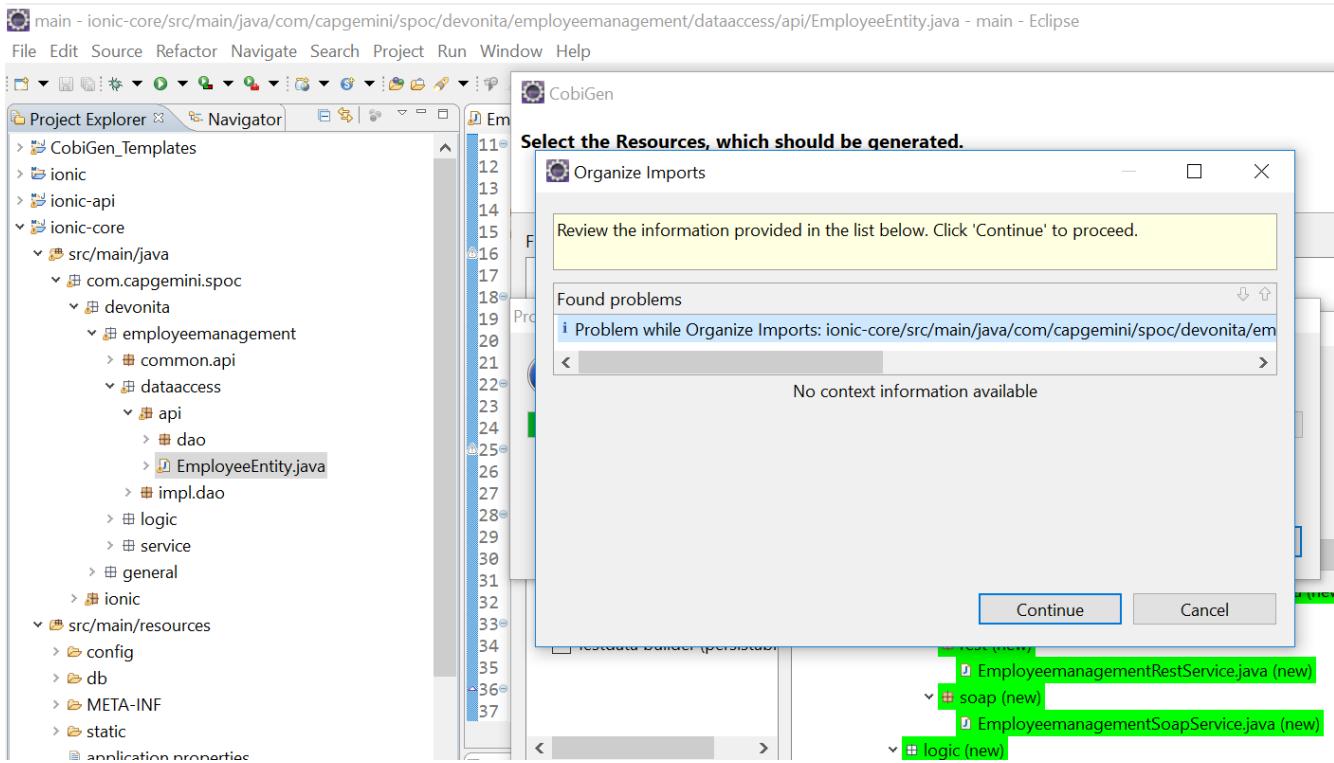
- **Passed** means, that the test case was executed successfully.
- **Broken** means, that there were mistakes, usually inside of the test method or test class. As tests are being treated as code, broken code has to be expected, resulting in occasionally broken test results.
- **Failed** means that an assertion failed.

Defects

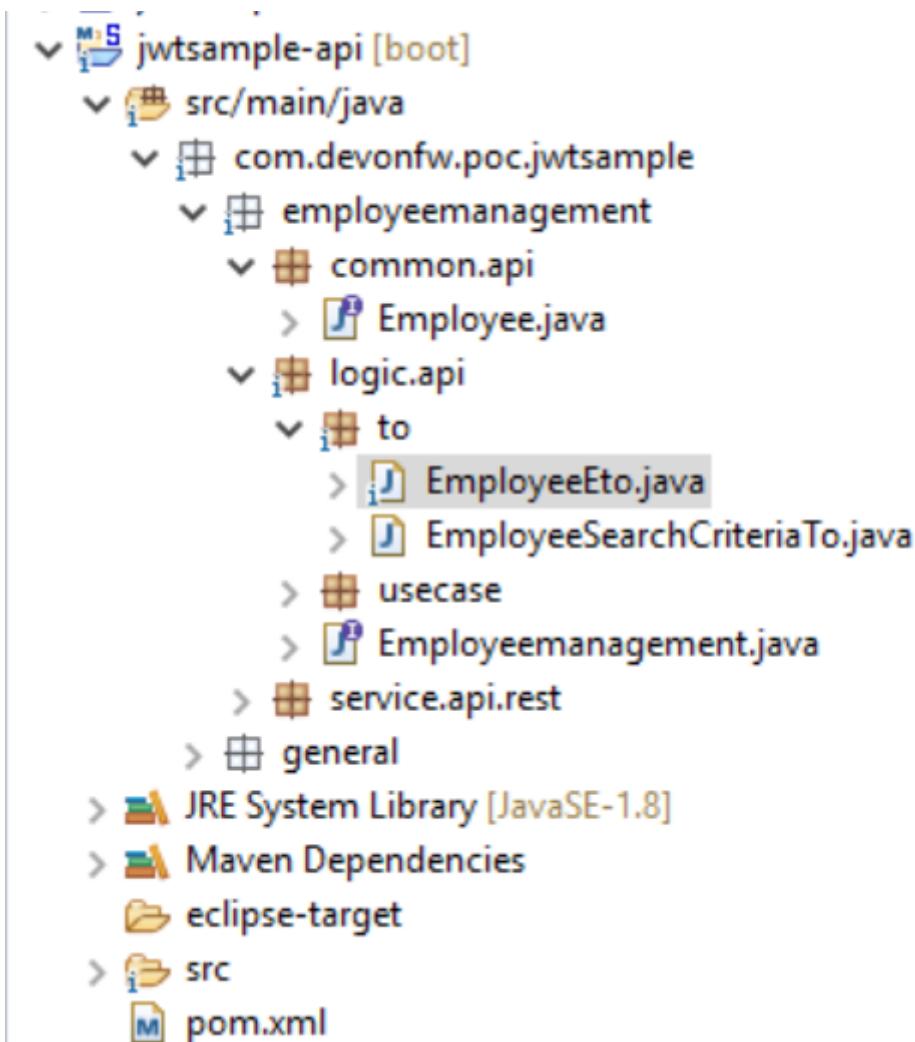
The defects tab lists out all the defects that occurred, and also descriptions thereof. Clicking on a list item displays the test case which resulted in an error. Clicking on a test case allows the user to have a look at the test case steps, as well as Log files or Screenshots of the failure.

Graph

The graph page includes a pie chart of all tests, showing their result status (failed, passed, etc.). Another graph allows insight into the time elapsed during the tests. This is a very useful information to find and eliminate possible bottlenecks in test implementations.



76.1.4. Why join Test Cases in groups - Test Suites



Regression Suite:

Regression testing is a type of [software testing](#) which verifies that software which was previously developed and tested still performs the same way after it was changed or interfaced with another software.

- Smoke
- Business vital functionalities
- Full scope of test cases

Functional Suite:

- Smoke
- Business function A
- Business function B

Single Responsibility Unit:

- Single page
- Specific test case

76.1.5. How to build a Test Suite based on tags

Structure of the Test Suite

Since mrchecker-core-module version 5.6.2.1:

```
package com.capgemini.mrchecker.core.groupTestCases.testSuites;

import org.junit.platform.runner.JUnitPlatform;
import org.junit.platform.suite.api.ExcludeTags;
import org.junit.platform.suite.api.IncludeTags;
import org.junit.platform.suite.api.SelectPackages;
import org.junit.runner.RunWith;

@RunWith(JUnitPlatform.class)
@IncludeTags("TestsTag1")
@ExcludeTags("TagToExclude")
@SelectPackages("com.capgemini.mrchecker.core.groupTestCases.testCases")
public class TS_Tag1 {

}
```

Where:

- `@RunWith(JUnitPlatform.class)` - use Junit5 runner
- `@IncludeTags({"TestsTag1"})` - search all test files with the tag "TestsTag1"

-
- `@ExcludeTags({"TagToExclude"})` - exclude test files with the tag "TagToExclude"
 - `@SelectPackages("com.capgemini.mrchecker.core.groupTestCases.testCases")` - search only test files in "com.capgemini.mrchecker.core.groupTestCases.testCases" package
 - `public class TS_Tag1` - the name of the Test Suite is "TS_Tag1"

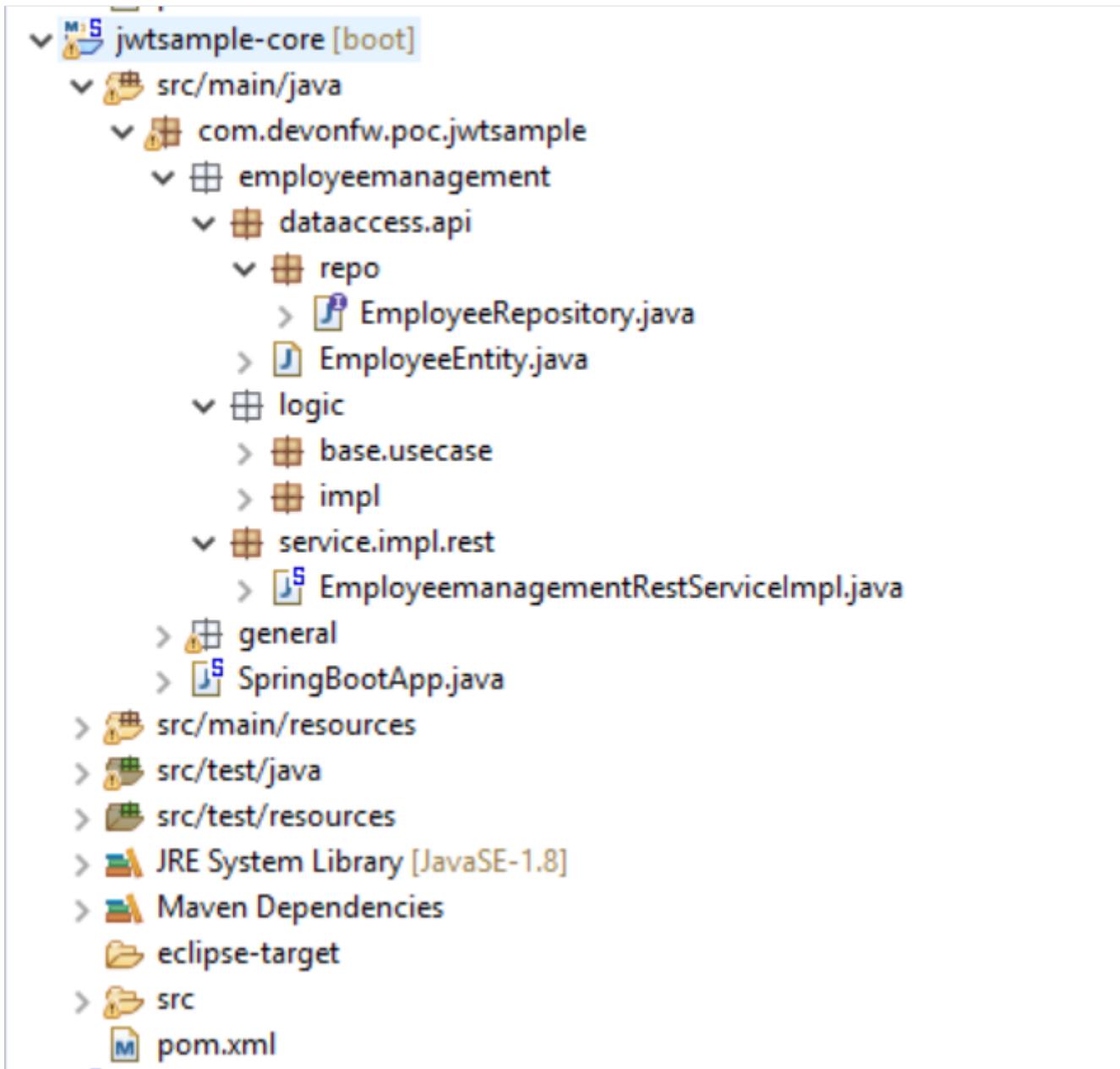
Most commonly used filters to build a Test Suite are ones using:

- `@IncludeTags({ })`
- `@ExcludeTags({ })`

Example:

1. `@IncludeTags({ "TestsTag1" }) , @ExcludeTags({ })` → will execute all test cases with the tag "TestsTag1"
2. `@IncludeTags({ "TestsTag1" }) , @ExcludeTags({ "SlowTest" })` → will execute all test cases with tag "TestsTag1" although it will exclude from this list the test cases with the tag "SlowTest"
3. `@IncludeTags({ }) , @ExcludeTags({ "SlowTest" })` → It will exclude test cases with the tag "SlowTest"

Prior to mrchecker-core-module version 5.6.2.1:



Where:

- `@RunWith(WildcardPatternSuiteBF.class)` - search for test files under `/src/test/java`
- `@IncludeCategories({ TestsTag1.class })` - search for all test files with the tag "TestsTag1.class"
- `@ExcludeCategories({ })` - exclude test files. In this example, there is no exclusion
- `@SuiteClasses({ "**/*Test.class" })` - search only test files, where the file name ends with "`<anyChar/s>Test.class`"
- `public class TS_Tag1` - the name of the Test Suite is "TS_Tag1"

Most commonly used filters to build Test Suite are ones using:

- `@IncludeCategories({ })`
- `@ExcludeCategories({ })`

Example:

1. `@IncludeCategories({ TestsTag1.class }) , @ExcludeCategories({ })` → will execute all test

cases with the tag `TestsTag1.class`

2. `@IncludeCategories({ TestsTag1.class }) , @ExcludeCategories({ SlowTest.class })` → will execute all test cases with the tag "TestsTag1.class" although it will exclude from this list the test cases with the tag "SlowTest.class"
3. `@IncludeCategories({ }) , @ExcludeCategories({ SlowTest.class })` → will execute all test cases from `/src/test/java`, although it will exclude from this list the test cases with the tag "SlowTest.class"

Structure of Test Case

Since `mrchecker-core-module` version 5.6.2.1:

```
package com.capgemini.mrchecker.core.groupTestCases.testCases.tag1;

import org.junit.jupiter.api.Test;

import com.capgemini.mrchecker.core.groupTestCases.testSuites.tags.TestsSelenium;
import com.capgemini.mrchecker.core.groupTestCases.testSuites.tags.TestsSmoke;
import com.capgemini.mrchecker.core.groupTestCases.testSuites.tags.TestsTag1;
import com.capgemini.mrchecker.test.core.BaseTest;

import io.qameta.allure.Feature;

@Feature("TAG1")
@TestsTag1
@TestsSmoke
@TestsSelenium
public class FristTest_tag1_Test extends BaseTest {

    @Override
    public void setUp() {
    }

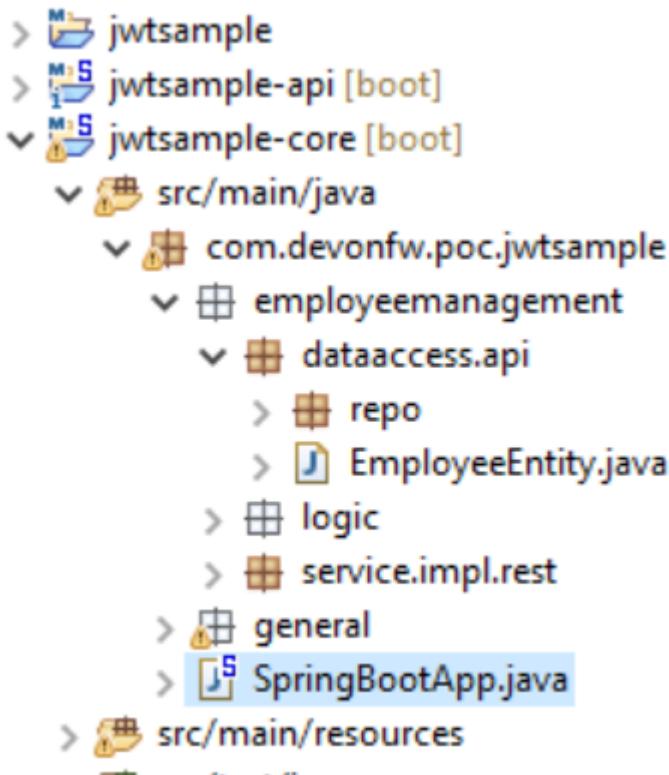
    @Override
    public void tearDown() {
    }

    @Test
    public void testTag1_Frist() {
    }
}
```

Where:

- `@TestsTag1, @TestsSmoke, @TestsSelenium` - list of tags assigned to this test case - "TestsTag1, TestsSmoke, TestSelenium" annotations
- `public class FristTest_tag1_Test` - the name of the test case is "FristTest_tag1_Test"

Prior to mrchecker-core-module version 5.6.2.1:



Where:

- `@Category({ TestsTag1.class, TestsSmoke.class, TestSelenium.class })` - list of tags / categories assigned to this test case - "TestsTag1.class, TestsSmoke.class, TestSelenium.class"
 - `public class FristTest_tag1_Test` - the name of the test case is "FristTest_tag1_Test"

Structure of Tags / Categories

Since mrchecker-core-module version 5.6.2.1:

Tag name: **TestsTag1** annotation

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Tag("TestsTag1")
public @interface TestsTag1 {
}
```

Tag name: **TestsSmoke** annotation

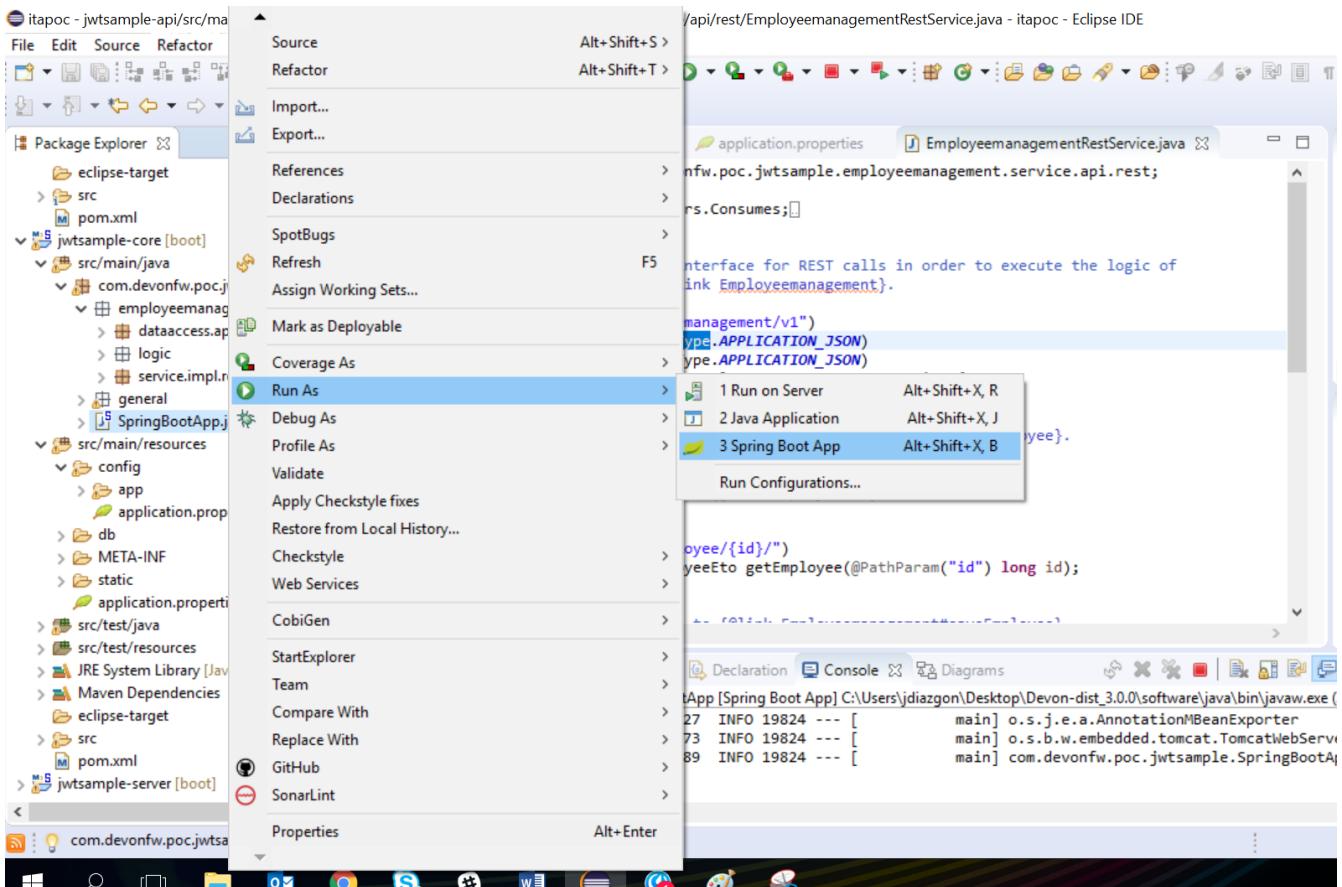
```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Tag("TestsSmoke")
public @interface TestsSmoke {
}
```

Tag name: **TestSelenium** annotation

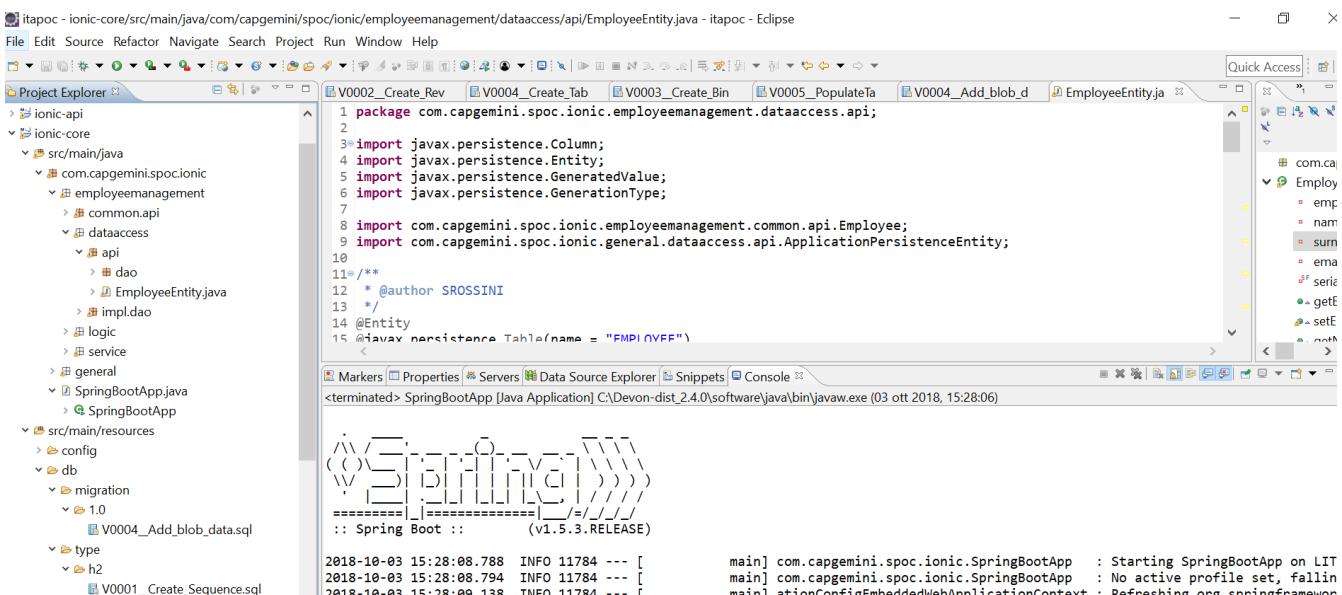
```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Tag("TestsSelenium")
public @interface TestsSelenium {
}
```

Prior to mrchecker-core-module version 5.6.2.1:

Tag name: **TestsTag1.class**



Tag name: **TestsSmoke.class**



Tag name: **TestSelenium.class**

```

1 # This is the spring boot configuration file for development. It will not be included
2 # In order to set specific configurations in a regular installed environment create an
3 # config/application.properties in the server. If you are deploying the application to a
4 # WAR file you can locate this config folder in ${symbol_dollar}{CATALINA_BASE}/lib. In
5 # the same container (not recommended by default) you need to ensure the WARS are extracted
6 # the config folder inside the WEB-INF/classes folder of the webapplication.
7
8 server.port=8081
9 server.servlet.context-path=/
10
11 # Datasource for accessing the database
12 # See https://github.com/devonfw-wiki/devon4j/wiki/guide-configuration#security-config
13 #jasypt.encryptor.password=None
14 #spring.datasource.password=ENC(7CnHiadYc0Wh2FnWADNjJg==)
15 spring.datasource.password=
16 spring.datasource.url=jdbc:h2:./.jwtsample;
17
18 # Enable JSON pretty printing
19 spring.jackson.serialization.INDENT_OUTPUT=true
20
21 # Flyway for Database Setup and Migrations
22 spring.flyway.enabled=true
23 spring.flyway.clean-on-validation-error=true

```

76.1.6. How to run Test Suite

To run a Test Suite you perform the same steps as you do to run a test case

Command line

Since mrchecker-core-module version 5.6.2.1:

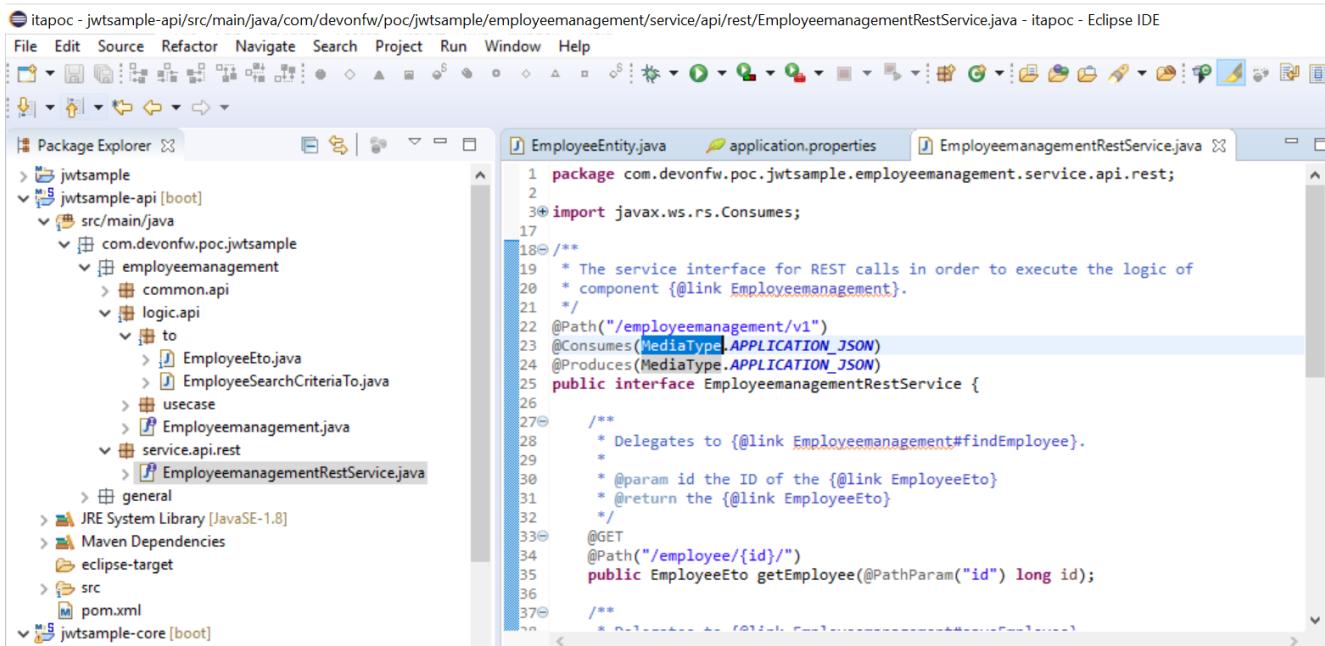
JUnit5 disallows running suite classes from maven. Use -Dgroups=Tag1,Tag2 and -DexcludeGroups=Tag4,Tag5 to create test suites in maven.

```
mvn test site -Dgroups=TestsTag1
```

Prior to mrchecker-core-module version 5.6.2.1:

```
mvn test site -Dtest=TS_Tag1
```

Eclipse



76.1.7. Data driven approach

Data driven approach - External data driven

External data driven - Data as external file injected in test case

Test case - Categorize functionality and severity

You can find more information about data driven here and here

There are a few ways to define parameters for tests.

76.1.8. Internal Data driven approach

Data as part of test case

The different means to pass in parameters are shown below.

Since mrchecker-core-module version 5.6.2.1

Static methods are used to provide the parameters.

A method in the test class:

```

@ParameterizedTest
@MethodSource("argumentsStream")

```

OR

```

@ParameterizedTest
@MethodSource("arrayStream")

```

In the first case the arguments are directly mapped to the test method parameters. In the second case the array is passed as the argument.

```
/**
 * -----
 * --- EXAMPLE 1 ---
 * -----
 * takes parameters from the argumentsStream method
 */
@ParameterizedTest
@MethodSource("argumentsStream")
public void addProducesCorrectValue_usingMethodSourceArgumentsStream(final int a, final int b, final int expectedResult) {
    assertEquals(expectedResult, testSubject.add(a, b));
}

static Stream<Arguments> argumentsStream() {
    return Stream.of(
        Arguments.of(1, 2, 3),
        Arguments.of(3, 4, 7),
        Arguments.of(5, 6, 11),
        Arguments.of(7, 8, 15));
}

@ParameterizedTest
@MethodSource("arrayStream")
public void addProducesCorrectValue_usingMethodSourceArrayStream(final int[] values) {
    assertEquals(values[2], testSubject.add(values[0], values[1]));
}

static Stream<int[]> arrayStream() {
    return Stream.of(new int[] { 1, 2, 3 }, new int[] { 3, 4, 7 }, new int[] { 5, 6, 11 }, new int[] { 7, 8, 15 });
}
```

A method in a different class:

```
@ParameterizedTest
@MethodSource("com.capgemini.mrchecker.core.datadriven.MyContainsTestProvider#provideContainsTrueParameters")
```

```
/**
 * -----
 * --- EXAMPLE 2 ---
 * -----
 * takes parameters from a method in MyContainsTestProvider
 */
@ParameterizedTest
@MethodSource("com.capgemini.mrchecker.core.datadriven.MyContainsTestProvider#provideContainsTrueParameters")
public void testContains_usingSeparateClass(final List<String> list, final String searchString, final boolean expectedResult) {
    assertEquals(expectedResult, testSubject.contains(list, searchString));
}

/unused/
class MyContainsTestProvider {
    public static Stream<Arguments> provideContainsTrueParameters() {
        return Stream.of(
            Arguments.of(NSArray.asList("a", "b", "c", "d", "e"), "c", true),
            Arguments.of(NSArray.asList("a", "b", "c", "d", "e"), "e", true),
            Arguments.of(NSArray.asList("a", "b"), "b", true),
            Arguments.of(Collections.singletonList("a"), "a", true));
    }

    public static Stream<Arguments> provideContainsFalseParameters() {
        return Stream.of(
            Arguments.of(NSArray.asList("a", "b", "c", "d", "e"), "f", false),
            Arguments.of(NSArray.asList("a", "b", "c", "d", "e"), "z", false),
            Arguments.of(NSArray.asList("a", "b"), "e", false),
            Arguments.of(Collections.emptyList(), "e", false));
    }
}
```

Prior to mrchecker-core-module version 5.6.2.1

Parameters that are passed into tests using the `@Parameters` annotation must be `_Object[]_s`

In the annotation:

```
@Parameters({"1, 2, 3", "3, 4, 7", "5, 6, 11", "7, 8, 15"})
```

The screenshot shows the Postman interface with an 'Untitled Request' for a GET method. The URL is `http://localhost:8081/services/rest/employeemanagement/v1/employee/1`. The 'Body' tab is selected, showing an empty JSON object. The 'JSON' button is highlighted.

The parameters must be primitive objects such as integers, strings, or booleans. Each set of parameters is contained within a single string and will be parsed to their correct values as defined by the test method's signature.

In a method named in the annotation:

```
@Parameters(method = "addParameters")
```

The screenshot shows the Postman interface with a response body containing a JSON object:

```

1  {
2      "id": 1,
3      "modificationCounter": 1,
4      "employeeId": 1,
5      "name": "Stefano",
6      "surname": "Rossini",
7      "email": "stefano.rossini@capgemini.com"
8  }

```

The status is 200 OK.

A separate method can be defined and referred to for parameters. This method must return an `Object[]` and can contain normal objects.

In a class:

```
@Parameters(source = MyContainsTestProvider.class)
```

```

1 [{}]
2   "pageable": {
3     "pageSize": 8,
4     "pageNumber": 0
5   }
6 ]

```

A separate class can be used to define parameters for the test. This test must contain at least one static method that returns an Object[], and its name must be prefixed with provide. The class could also contain multiple methods that provide parameters to the test, as long as they also meet the required criteria.

76.1.9. External Data Driven

Data as **external file injected in test case**

Since mrchecker-core-module version 5.6.2.1

Tests use the annotation @CsvFileSource to inject CSVs file.

```
@CsvFileSource(resources = "/datadriven/test.csv", numLinesToSkip = 1)
```

A CSV can also be used to contain the parameters for the tests. It is pretty simple to set up, as it's just a comma-separated list.

Classic CSV

```

/**
 * -----
 * --- EXAMPLE 1 -----
 * -----
 */
@ParameterizedTest
@CsvFileSource(resources = "/datadriven/test.csv", numLinesToSkip = 1)
public void loadParamsFromCsv(String age, String name) {
    BFLogger.logDebug("DataDrivenExampleTest.loadParamsFromCsv()");
    BFLogger.logDebug("\t" + "Name=" + name + " " + "Age=" + age);
}

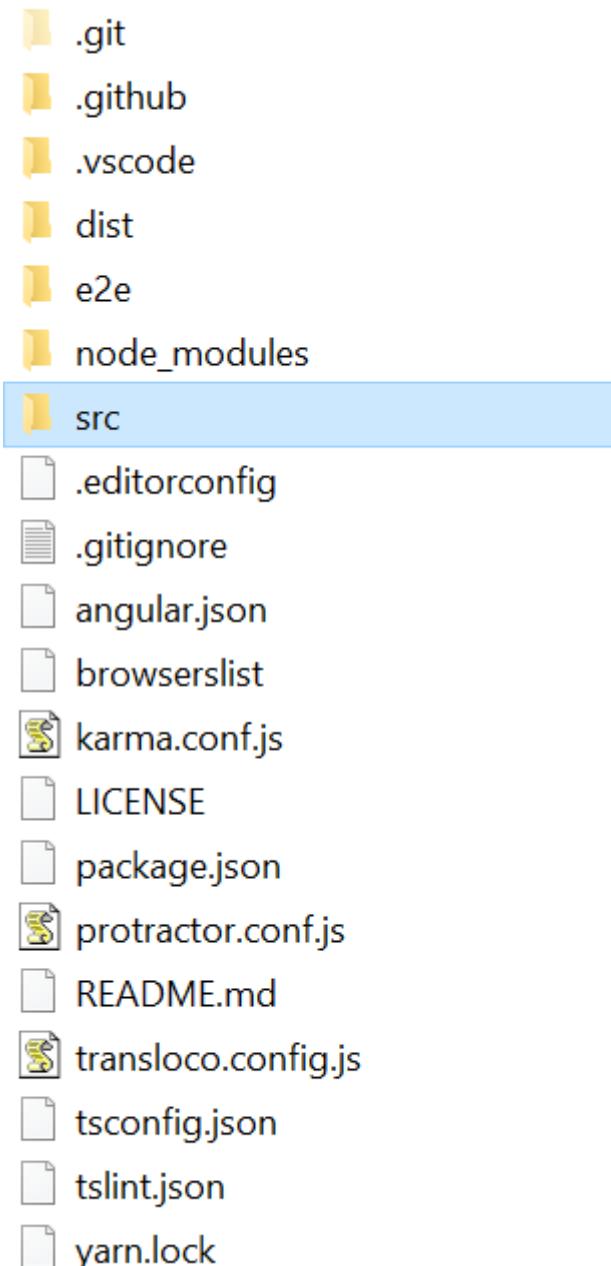
```

and CSV file structure

CSV with headers

```
/**
 * -----
 * --- EXAMPLE 2 -----
 * -----
 * takes parameters from a CSV file with headers
 */
@ParameterizedTest
@CsvFileSource(resources = "/datadriven/with_header.csv")
public void loadParamsFromCsvWithHeader(String age, String name) {
    BFLogger.logDebug("DataDrivenExampleTest.loadParamsFromCsvWithHeader()");
    BFLogger.logDebug("\t" + "Name=" + name + " " + "Age=" + age);
}
```

and CSV file structure



CSV with specific column mapper

```
/** 
 * -----
 * --- EXAMPLE 3
 * -----
 * takes parameters from a CSV file and map them to specific mapper
 */
@ParameterizedTest
@CsvFileSource(resources = "/datadriven/test.csv", numLinesToSkip = 1)
public void loadParamsFromAnyFile(@AggregateWith(PersonAggregator.class) PersonAggregator.Person person) {
    BFLogger.logDebug("DataDrivenExampleTest.loadParamsFromAnyFile()");
    BFLogger.logDebug("\t" + "Name=" + person.getName() + " " + "Age=" + person.getAge());
}
```

and Mapper implementation

```
public class PersonAggregator implements ArgumentsAggregator {

    @Override
    public Object aggregateArguments(ArgumentsAccessor argumentsAccessor, ParameterContext parameterContext) throws ArgumentsAggregationException {
        return new Person(argumentsAccessor.getString(index: 0),
                          argumentsAccessor.getString(index: 1));
    }

    public static class Person {
        private String name;
        private Integer age;

        // Arguments order depends on data in CSV line
        public Person(String age, String name) {
            this.name = name;
            setAge(age);
        }

        // When there is only one argument after CSV line split, than treat this one as it is argument AGE
        public Person(String age) { setAge(age); }

        public String getName() { return name; }

        public boolean isAdult() { return age >= 18; }

        public int getAge() { return age; }

        // When argument AGE is missing, then set default value = 0
        private void setAge(String age) {
            try {
                this.age = Integer.parseInt(age);
            } catch (NumberFormatException e) {
                this.age = 0; // Default value
            }
        }

        @Override
        public String toString() { return "Person of age: " + age; }
    }
}
```

Prior to mrchecker-core-module version 5.6.2.1

Tests use the annotation `@FileParameters` to inject CSVs file.

```
@FileParameters("src/test/resources/datadriven/test.csv")
```

A CSV can also be used to contain the parameters for the tests. It is pretty simple to set up, as it's just a comma-separated list.

Classic CSV

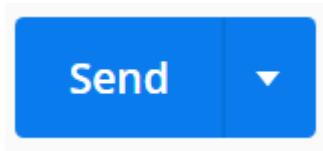
The screenshot shows the Postman interface with the following details:

- Request Method:** POST
- URL:** http://localhost:8081/services/rest/employeemanagement/v1/employee/search
- Body (JSON application/json):**

```

1 < []
2   "pageable": {
3     "pageSize": 8,
4     "pageNumber": 0
5   }
6 >
    
```
- Send Button:** A large blue button labeled "Send" with a dropdown arrow.

and CSV file structure



CSV with headers

The screenshot shows the Postman interface with the following details:

- Status:** 200 OK
- Time:** 551 ms
- Size:** 837 B
- Body (Pretty):**

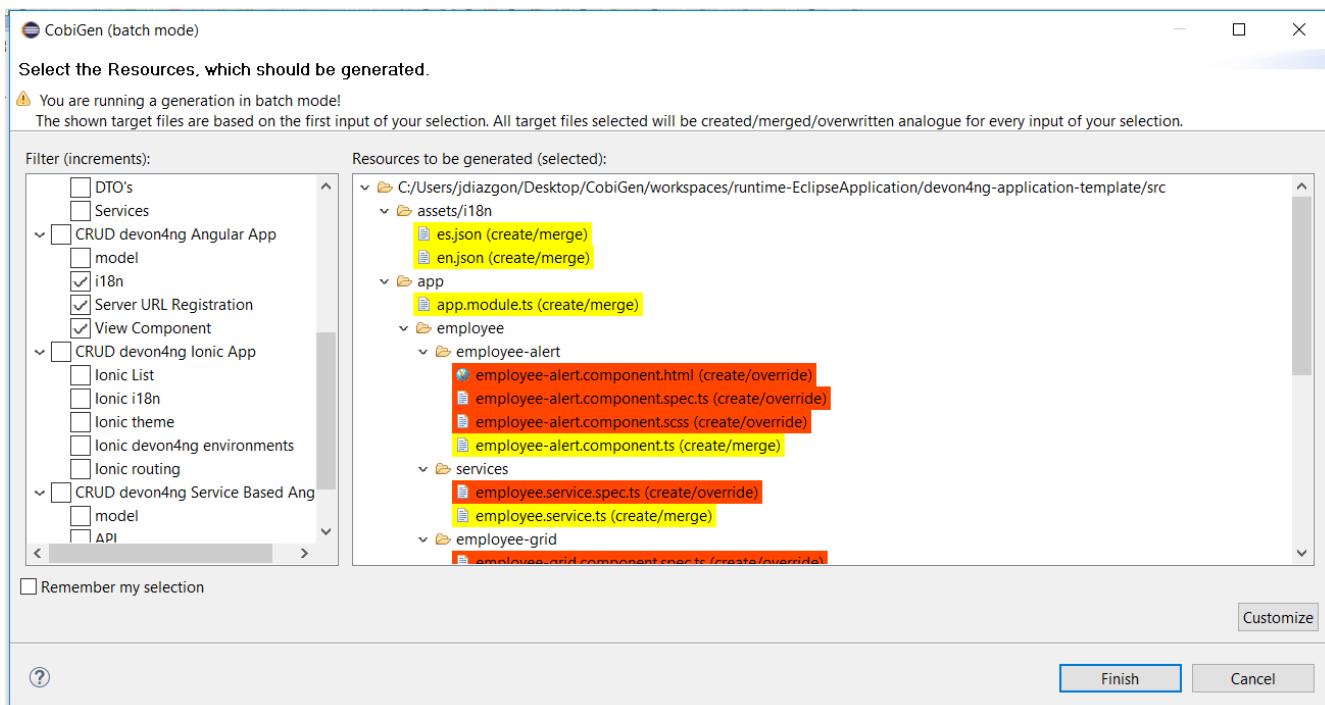
```

1 < {
2   "content": [
3     {
4       "id": 1,
5       "modificationCounter": 1,
6       "employeeId": 1,
7       "name": "Stefano",
8       "surname": "Rossini",
9       "email": "stefano.rossini@capgemini.com"
10      },
11      {
12        "id": 2,
13        "modificationCounter": 2,
14        "employeeId": 2,
15        "name": "Angelo",
16        "surname": "Muresu",
17        "email": "angelo.muresu@capgemini.com"
18      },
19      {
20        "id": 3,
21        "modificationCounter": 3,
22        "employeeId": 3,
23        "name": "Jaime",
24        "surname": "Gonzalez",
25        "email": "jaime.diaz-gonzalez@capgemini.com"
26      }
27    ],
28    "pageable": {
29      "pageNumber": 0,
30      "pageSize": 8
31    },
32    "last": true,
33    "totalElements": 3,
34    "totalPages": 1,
35    "size": 8,
36    "number": 0,
37    "sort": null,
38    "numberOfElements": 3,
39    "first": true
40  }
    
```

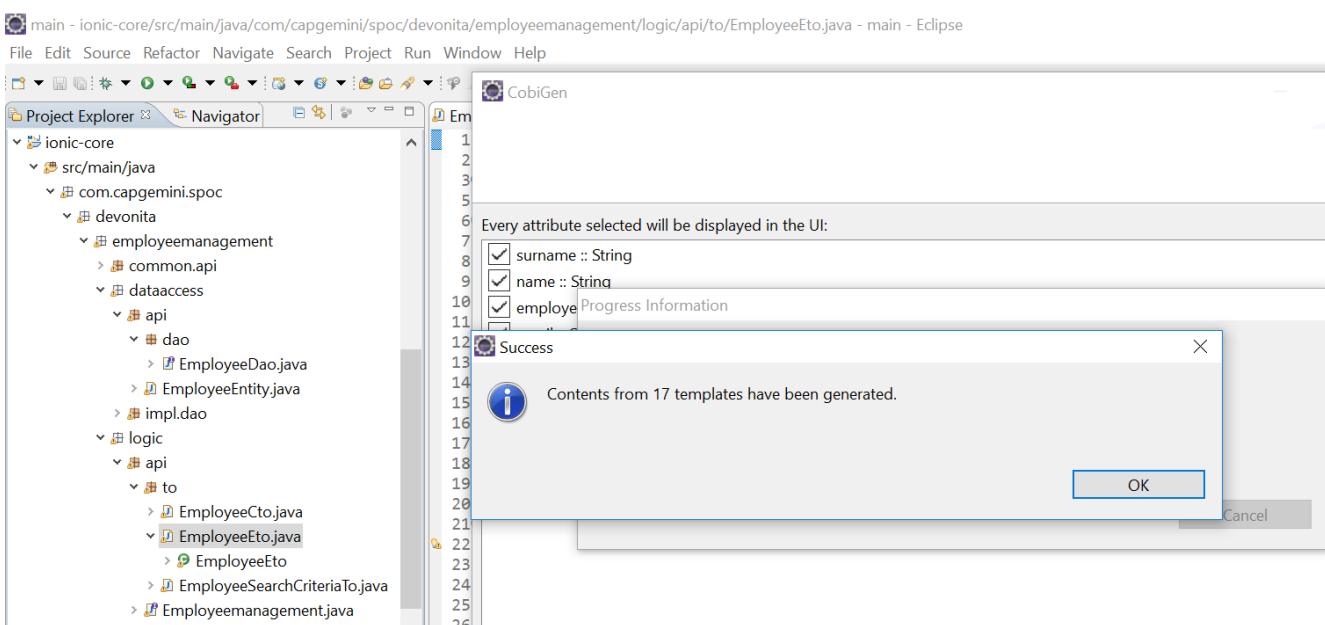
and CSV file structure

	.git
	.github
	.vscode
	dist
	e2e
	node_modules
	src
	.editorconfig
	.gitignore
	angular.json
	browserslist
	karma.conf.js
	LICENSE
	package.json
	protractor.conf.js
	README.md
	transloco.config.js
	tsconfig.json
	tslint.json
	yarn.lock

CSV with specific column mapper



and Mapper implementation



76.1.10. What is "Parallel test execution" ?

Parallel test execution means many "*Test Classes*" can run simultaneously.

"*Test Class*", as this is a JUnit Test class, it can have one or more test cases - "*Test case methods*"

The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** karma.conf.js — casp4js-application-template — Visual Studio Code
- File Menu:** File Edit Selection View Go Debug Tasks Help
- Info Bar:** Code for 64-bit Windows is now available!
- Explorer Panel:**
 - OPEN EDITORS: Welcome, karma.conf.js
 - OASP4JS-APPLICATION-TEMPLATE: app (selected), core, employee, home, layout, login, assets, environments, favicon.ico, DOCKER, COMMITS, COMPARE COMMITS, CODE OUTLINE, PROJECTS
- Code Editor:** Content of karma.conf.js (partial):

```

1 // Karma configuration file, see link for more information
2 // https://karma-runner.github.io/1.0/config/configuration-file.html
3
4 module.exports = function (config) {
5   config.set({
6     basePath: '',
7     frameworks: ['jasmine', '@angular/cli'],
8     plugins: [
9       require('karma-jasmine'),
10      require('karma-chrome-launcher'),
11      require('karma-jasmine-html-reporter'),
12      require('karma-coverage-istanbul-reporter'),
13      require('@angular/cli/plugins/karma')
14    ],
15    client: {
16      clearContext: false // leave Jasmine Spec Runner output visible in browser
17    },
18    coverageIstanbulReporter: {
19      reports: [ 'html', 'lcovonly' ],
20      fixWebpackSourcePaths: true
21    },
22    angularCli: {
23      environment: 'dev'
24    },
25    reporters: ['progress', 'kjhtml'],
26    port: 9876,
27    colors: true,
28    logLevel: config.LOG_INFO,
29    autoWatch: true,
30    browsers: ['Chrome'],
31    singleRun: false

```

76.1.11. How many parallel test classes can run simultaneously?

Since mrchecker-core-module version 5.6.2.1

JUnit5 supports parallelism natively. The feature is configured using a property file located at `src\test\resources\junit-platform.properties`. As per default configuration, concurrent test execution is set to run test classes in parallel using the thread count equal to a number of your CPUs.

```
junit.jupiter.execution.parallel.enabled=true
#junit.jupiter.execution.parallel.mode.default=concurrent
junit.jupiter.execution.parallel.mode.default=same_thread
junit.jupiter.execution.parallel.mode.classes.default=concurrent
#junit.jupiter.execution.parallel.mode.classes.default=same_thread
junit.jupiter.execution.parallel.config.strategy=dynamic
junit.jupiter.execution.parallel.config.dynamic.factor=1
```

Visit [JUnit5 site](#) to learn more about parallel test execution.

Prior to mrchecker-core-module version 5.6.2.1

By default, number of parallel test classes is set to 8.

It can be updated as you please, on demand, by command line:

```
mvn test site -Dtest=TS_Tag1 -Dthread.count=16
```

-*Dthread.count=16* - increase number of parallel Test Class execution to 16.

Overview

Cucumber / Selenium

Business and IT don't always understand each other. Very often misunderstandings between business and IT result in the costly failure of IT projects. With this in mind, Cucumber was developed as a tool to support human collaboration between business and IT.

Cucumber uses executable specifications to encourage a close collaboration. This helps teams to keep the business goal in mind at all times. With Cucumber you can merge specification and test documentation into one cohesive whole, allowing your team to maintain one single source of truth. Because these executable specifications are automatically tested by Cucumber, your single source of truth is always up-to-date.

```
const routes: Routes = [
  {
    path: 'login',
    component: LoginComponent
  }, {
    path: 'home',
    component: HomeComponent,
    canActivate: [AuthGuard],
    children: [
      {
        path: '',
        redirectTo: '/home/initialPage',
        pathMatch: 'full',
        canActivate: [AuthGuard]
      }, {
        path: 'initialPage',
        component: InitialPageComponent,
        canActivate: [AuthGuard]
      }, {
        path: 'sampleData',
        component: SampleDataGridComponent,
        canActivate: [AuthGuard]
      }, {
        path: 'employee',
        component: EmployeeGridComponent,
        canActivate: [AuthGuard]
      }
    ]
  }, {
    path: '',
    redirectTo: '/login',
    pathMatch: 'full'
  }
]
```

Cucumber supports testers when designing test cases. To automate these test cases, several languages can be used. Cucumber also works well with Browser Automation tools such as Selenium Webdriver.

Selenium

Selenium automates browsers and is used for automating web applications for testing purposes.

Selenium offers testers and developers full access to the properties of objects and the underlying tests, via a scripting environment and integrated debugging options.

Selenium consists of many parts. If you want to create robust, browser-based regression automation suites and tests, Selenium Webdriver is most appropriate. With Selenium Webdriver you can also scale and distribute scripts across many environments.

Strengths

Supports BDD

Those familiar with Behavior Driven Development (BDD) recognize Cucumber as an excellent open source tool that supports this practice.

All in one place

With Cucumber / Selenium you can automate at the UI level. Automation at the unit or API level can also be implemented using Cucumber. This means all tests, regardless of the level at which they are implemented, can be implemented in one tool.

Maintainable test scripts

Many teams seem to prefer UI level automation, despite huge cost of maintaining UI level tests compared to the cost of maintaining API or unit tests. To lessen the maintenance of UI testing, when designing UI level functional tests, you can try describing the test and the automation at three levels: business rule, UI workflow, technical implementation.

When using Cucumber combined with Selenium, you can implement these three levels for better maintenance.

Early start

Executable specifications can and should be written before the functionality is implemented. By starting early, teams get most return on investment from their test automation.

Supported by a large community

Cucumber and Selenium are both open source tools with a large community, online resources and mailing lists.

How to run cucumber tests in Mr.Checker

Command line / Jenkins

- Run cucumber tests and generate Allure report. Please use this for Jenkins execution. Report is saved under `./target/site`.

```
mvn clean -P cucumber test site
```

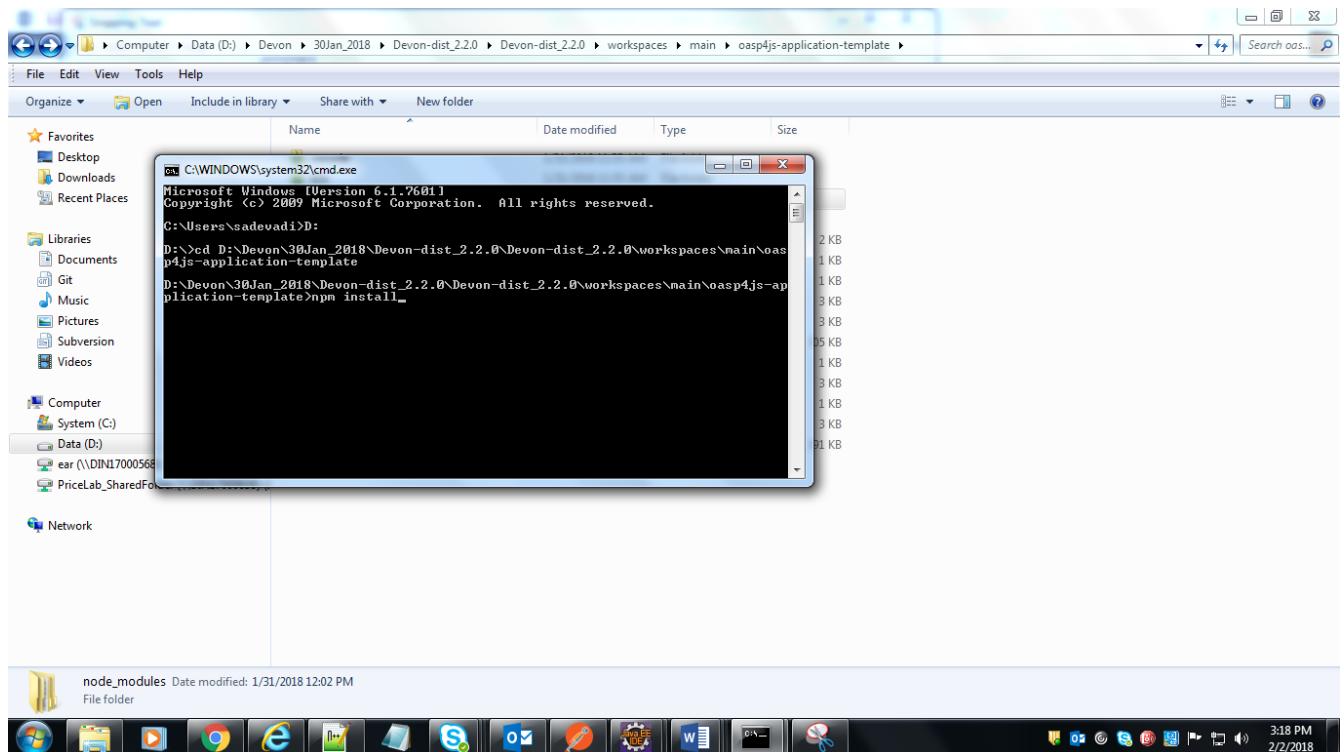
- Run and generate report

```
mvn clean -P cucumber test site allure:report
```

- Run cucumber tests, generate Allure report and start standalone report server

```
mvn clean -P cucumber test site allure:serve
```

Eclipse IDE



Tooling

Cucumber

Cucumber supports over a dozen different software platforms. Every Cucumber implementation provides the same overall functionality, but they also have their own installation procedure and platform-specific functionality. See <https://cucumber.io/docs> for all Cucumber implementations and framework implementations.

Also, IDEs such as IntelliJ offer several plugins for Cucumber support.

Selenium

Selenium has the support of some of the largest browser vendors who have taken (or are taking) steps to make Selenium a native part of their browser. It is also the core technology in countless other browser automation tools, APIs and frameworks.

Automation process

Write a feature file

Test automation in Cucumber starts with writing a feature file. A feature normally consists of several (test)scenarios and each scenario consists of several steps.

Feature: Refund item

Scenario: Jeff returns a faulty microwave

Given Jeff has bought a microwave for \$100

And he has a receipt

When he returns the microwave

Then Jeff should be refunded \$100

Above example shows a feature “Refund item” with one scenario “Jeff returns a faulty microwave”. The scenario consists of four steps each starting with a key word (Given, And, When, Then).

Implementing the steps

Next the steps are implemented. Assuming we use Java to implement the steps, the Java code will look something like this.

```
public class MyStepdefs {
    @Given("Jeff has bought a microwave for $(\\d+)")
    public void Jeff_has_bought_a_microwave_for(int amount) {
        // implementation can be plain java
        // or selenium
        driver.findElement(By.name("test")).sendKeys("This is an example\\n");
        driver.findElement(By.name("button")).click(); // etc
    }
}
```

Cucumber uses an annotation (highlighted) to match the step from the feature file with the function implementing the step in the Java class. The name of the class and the function can be as the developer sees fit. Selenium code can be used within the function to automate interaction with the browser.

Running scenarios

There are several ways to run scenarios with Cucumber, for example the JUnit runner, a command line runner and several third party runners.

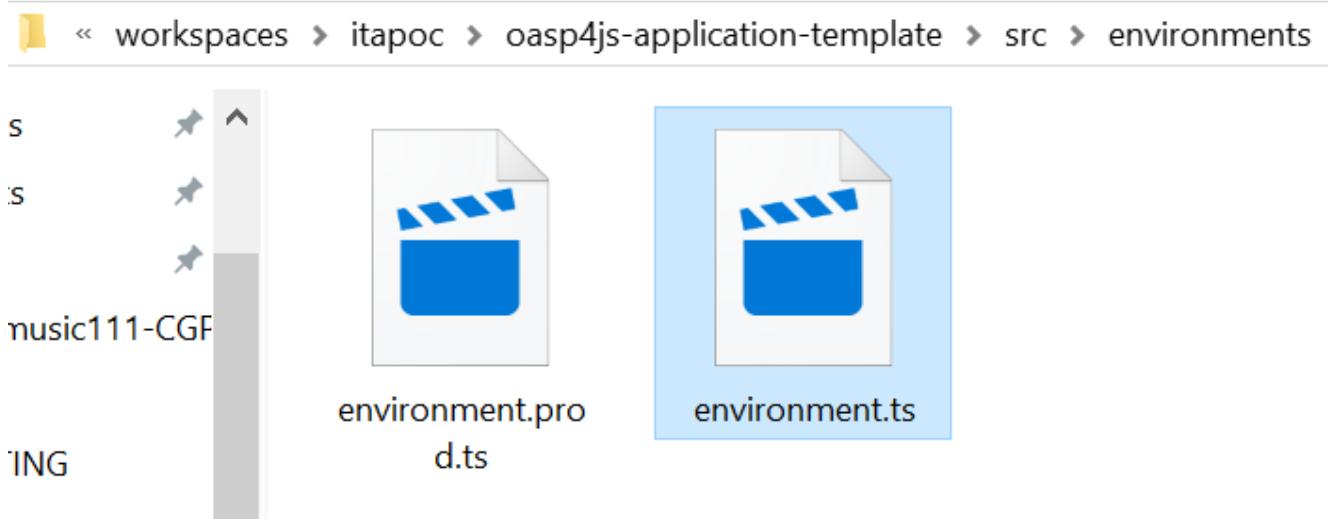
Reporting test results

Cucumber can report results in several different formats, using formatter plugins

Features

Feature files using Gherkin

Cucumber executes your feature files. As shown in the example below, feature files in Gherkin are easy to read so they can be shared between IT and business. Data tables can be used to execute a scenario with different inputs.



Organizing tests

Feature files are placed in a directory structure and together form a feature tree.

Tags can be used to group features based on all kinds of categories. Cucumber can include or exclude tests with certain tags when running the tests.

Reporting test results

Cucumber can report results in several formats, using formatter plugins. Not supported option by Shared Services: The output from Cucumber can be used to present test results in Jenkins or Hudson depending of the preference of the project.

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer view displays a project structure for a Java application named 'ionic'. It includes packages like 'com.capgemini.spoc.ionic' containing 'employeemanagement' and 'dataaccess' sub-packages, and 'SpringBootApp.java'. Below these are 'src/main/resources' and 'db' folders. On the right, the editor view shows the 'application.properties' file with the following content:

```

1# This is the spring boot configuration file for development. It will not be included into
2# In order to set specific configurations in a regular installed environment create an acc
3# config/application.properties in the server. If you are deploying the application to a s
4# WAR file you can locate this config folder in ${symbol_dollar}{CATALINA_BASE}/lib. If yo
5# the same container (not recommended by default) you need to ensure the WARs are extracte
6# the config folder inside the WEB-INF/classes folder of the webapplication.
7
8server.port=8081
9server.context-path=/
10
11# Datasource for accessing the database
12# See https://github.com/oasp/oasp4j/wiki/guide-configuration#security
13#jasrypt.encryptor.password=none
14#spring.datasource.password=ENC(7CnHiadYc0Wh2FnWADNjJg==)
15spring.datasource.password=
16spring.datasource.url=jdbc:h2:./ionic;
17
18# Enable JSON pretty printing
19spring.jackson.serialization.INDENT_OUTPUT=true
20
21# Flyway for Database Setup and Migrations
22flyway.enabled=true
23flyway.clean-on-validation-error=true

```

At the bottom of the screen, the Eclipse status bar shows the path 'SpringBootApp [Java Application] C:\Devon-dist_2.4.0\software\java\bin\javaw.exe (03 ott 2018, 15:32:16)' and the date and time '2018-10-03 15:32:30.919 INFO 6112 --- [main] f.a.AutowiredAnnotationBeanPostProce'.

HOW IS Cucumber / Selenium USED AT Capgemini?

Tool deployment

Cucumber and Selenium are chosen as one of Capgemini's test automation industrial tools. We support the Java implementation of Cucumber and Selenium Webdriver. We can help with creating Cucumber, Selenium projects in Eclipse and IntelliJ.

Application in ATaaS (Automated Testing as a Service)

In the context of industrialisation, Capgemini has developed a range of services to assist and support the projects in process and tools implementation.

In this context a team of experts assists projects using test automation.

The main services provided by the center of expertise are:

- Advise on the feasibility of automation.
- Support with installation.
- Coaching teams in the use of BDD.

76.1.12. Run on independent Operation Systems

As *E2E Allure test framework* is build on top of:

- Java 1.8
- Maven 3.3

This guarantees portability to all operating systems.

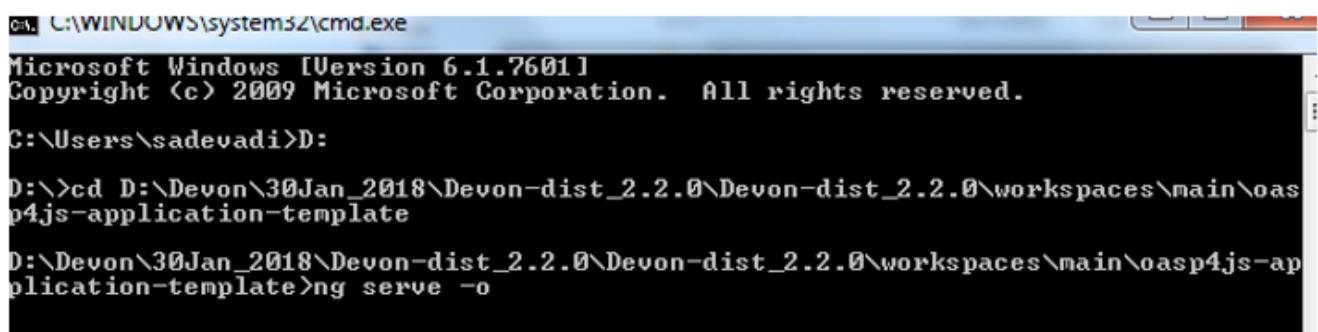
E2E Allure test framework can run on OS:

- Windows,
- Linux and
- Mac.

Test creation and maintenance in *E2E Allure test framework* can be done with any type of IDE:

- Eclipse,
- IntelliJ,
- WebStorm,
- Visual Studio Code,
- many more that support Java + Maven.

76.1.13. System under test environments



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. All rights reserved.

C:\Users\sadevadi>D:
D:\>cd D:\Devon\30Jan_2018\Devon-dist_2.2.0\Devon-dist_2.2.0\workspaces\main\oasp4js-application-template
D:\Devon\30Jan_2018\Devon-dist_2.2.0\Devon-dist_2.2.0\workspaces\main\oasp4js-application-template>ng serve -o
```

- *Quality assurance* or QA is a way of preventing mistakes or defects in manufactured products and avoiding problems when delivering solutions or services to customers; which ISO 9000 defines as "part of quality management focused on providing confidence that quality requirements will be fulfilled".
- *System integration testing* or SIT is a high-level software testing process in which testers verify that all related systems maintain data integrity and can operate in coordination with other systems in the same environment. The testing process ensures that all sub-components are integrated successfully to provide expected results.
- *Development* or *Dev* testing is performed by the software developer or engineer during the construction phase of the software development life-cycle. Rather than replace traditional QA focuses, it augments it. Development testing aims to eliminate construction errors before code is promoted to QA; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and QA process.
- *Prod* If the customer accepts the product, it is deployed to a *production* environment, making it available to all users of the system.

The screenshot shows a web browser window with the URL `localhost:4200/home/employee`. The page title is "OASP4JS". On the left, there's a sidebar with icons for "Home Description" and "Employee_EN Employee_EN". The main content area is titled "Employee_EN_Grid" and "Employee_EN_Description". It features a table with columns: employeeId, Name_EN, Surname_EN, and Email_EN. There are three rows of data:

employeeId	Name_EN	Surname_EN	Email_EN
1	Stefano	Rossini	stefano.rossini@capgemini.com
2	Angelo	Muresu	angelo.muresu@capgemini.com
3	Jaime	Diaz	jaime.diaz-gonzalez@capgemini.com

At the bottom of the page, it says "DevonFW Application" and "Devonfw".

76.1.14. How to use system environment

In Page classes, when you load / start web, it is uncommon to save fixed main url.

Value flexibility is a must, when your web application under test, have different main url, dependence on environment (DEV, QA, SIT, ..., PROD)

Instead of hard coded main url variable, you build your Page classe with dynamic variable.

Example of dynamic variable `GetEnvironmentParam.WWW_FONT_URL`

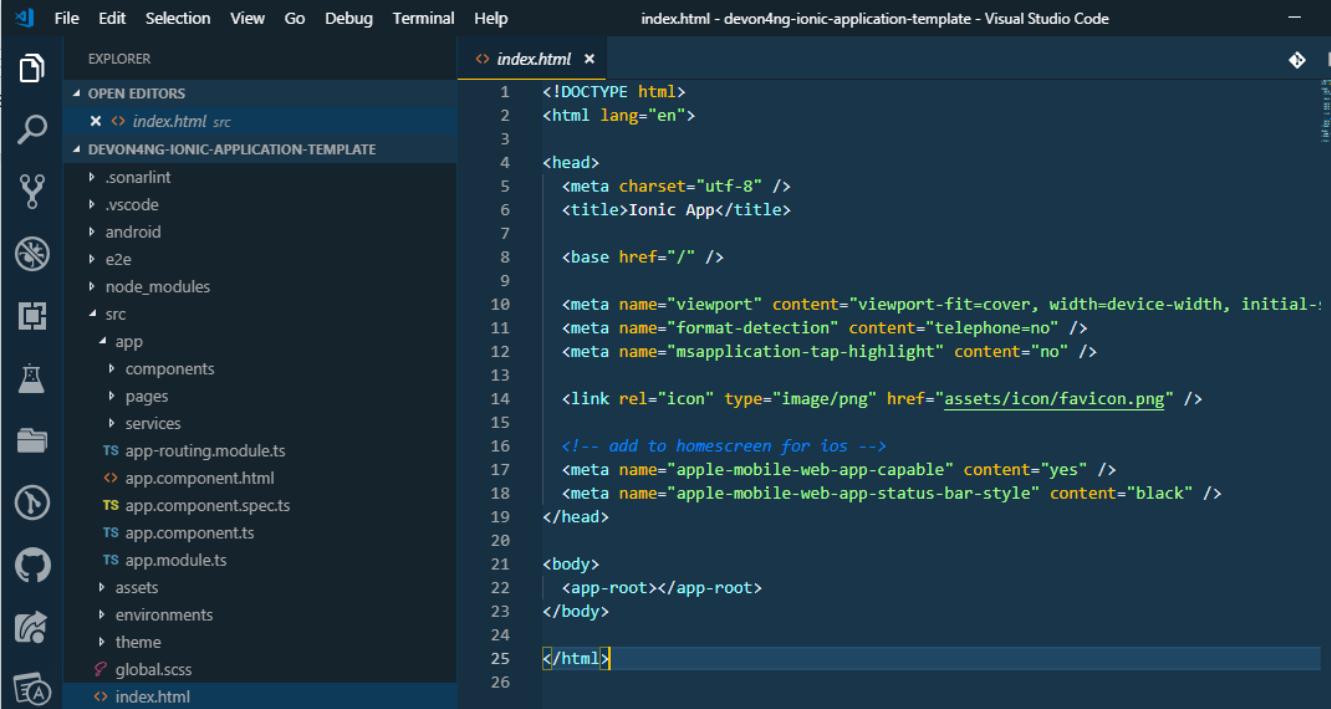
The screenshot shows the "CobiGen (batch mode)" dialog. The left panel lists "Filter (increments):" with several options checked, such as "CRUD devon4ng Ionic App" and "CRUD devon4ng Ionic App". The right panel shows "Resources to be generated (selected):" with a tree view of files under "C:/Users/jdiazgon/Desktop/CobiGen/workspaces/runtime-EclipseApplication/devon4ng-ionic-application-template/src". Some files are highlighted in yellow or red, indicating they are selected for generation. At the bottom, there are buttons for "Finish" and "Cancel".

76.1.15. How to create / update system environment

External file with variable values

Dynamic variable values are stored under path `mrchecker-app-under-test|src|resources|environments|environments.csv`.

NOTE: As environments.csv is Comma-separated file, please be aware of any edition and then save it under Excel.



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** On the left, it shows the project structure for "DEVON4NG-IONIC-APPLICATION-TEMPLATE". It includes folders like .sonarlint, .vscode, android, e2e, node_modules, and src. Under src, there's an app folder containing components, pages, and services. TypeScript files like app-routing.module.ts, app.component.html, app.component.spec.ts, app.component.ts, and app.module.ts are listed. Assets, environments, theme, and global.scss files are also present.
- Code Editor:** The main area displays the content of the index.html file. The code is as follows:

```

1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="utf-8" />
6      <title>Ionic App</title>
7
8      <base href="/" />
9
10     <meta name="viewport" content="viewport-fit=cover, width=device-width, initial-
11     <meta name="format-detection" content="telephone=no" />
12     <meta name="msapplication-tap-highlight" content="no" />
13
14     <link rel="icon" type="image/png" href="assets/icon/favicon.png" />
15
16     <!-- add to homescreen for ios -->
17     <meta name="apple-mobile-web-app-capable" content="yes" />
18     <meta name="apple-mobile-web-app-status-bar-style" content="black" />
19 </head>
20
21     <body>
22         <app-root></app-root>
23     </body>
24
25 </html>
26

```

Encrypting sensitive data

Some types of data you might want to store as environment settings are sensitive in nature (e.g. passwords). You might not want to store them (at least not in their plaintext form) in your repository. To be able to encrypt sensitive data you need to do following:

1. Create a secret (long, random chain of characters) and store it under `mrchecker-app-under-test|src|resources|secretData.txt`. Example: `LhwbTm9V3FUbB05Tt5PiTUEQrXGgWrDLCMthnzLKNy1zA5FVTFiTdHRQAyPRIGXmsAjPUP1]SoSLeSBM`
2. Exclude the file from being checked into the git repository by adding it to `git.ignore`. You will need to pass the file over a different channel among your teammates.
3. Encrypt the values before putting them into the `environments.csv` file by creating following script (put the script where your jasypt library resides, e.g. `C:\MrChecker_Test_Framework|m2|repository\org\jasypt\jasypt|1.9.2`):

```

@ECHO OFF

set SCRIPT_NAME=encrypt.bat
set EXECUTABLE_CLASS=org.jasypt.intf.cli.JasyptPBEStrongEncryptionCLI
set EXEC_CLASSPATH=jasypt-1.9.2.jar
if "%JASYPT_CLASSPATH%" == "" goto computeclasspath
set EXEC_CLASSPATH=%EXEC_CLASSPATH%;%JASYPT_CLASSPATH%

:computeclasspath
IF "%OS%" == "Windows_NT" setlocal ENABLEDELAYEDEXPANSION
FOR %%c in (%~dp0..\lib\*.jar) DO set EXEC_CLASSPATH=!EXEC_CLASSPATH!;%%c
IF "%OS%" == "Windows_NT" setlocal DISABLEDELAYEDEXPANSION

set JAVA_EXECUTABLE=java
if "%JAVA_HOME%" == "" goto execute
set JAVA_EXECUTABLE="%JAVA_HOME%\bin\java"

:execute
%JAVA_EXECUTABLE% -classpath %EXEC_CLASSPATH% %EXECUTABLE_CLASS% %SCRIPT_NAME% %

```

4. Encrypt the values by calling

```

.\encrypt.bat input=someinput password=secret

----ENVIRONMENT-----

Runtime: Oracle Corporation Java HotSpot(TM) 64-Bit Server VM 25.111-b14


----ARGUMENTS-----


input: someinput
password: secret


----OUTPUT-----


JN3n0Fo12GMZoUxR5z2wI2qdipcNH1UD

```

5. Mark the value as encrypted by adding a prefix 'ENC(' and suffix ')' like: ENC(JN3n0Fo12GMZoUxR5z2wI2qdipcNH1UD)

```

1 // This file can be replaced during build using the "fileReplacements" field in "angular.json".
2 // The list of file replacements can be found in "angular.json"
3
4
5 export const environment = {
6   production: false,
7 };
8
9 export const SERVER_URL = 'http://localhost:8081';
10
11 /**
12  * For easier debugging in development mode, you can import the
13  * to ignore zone related error stack frames such as "zone.run"
14  */
15
16 // This import should be commented out in production mode because
17 // on performance if an error is thrown.
18 // import 'zone.js/dist/zone-error'; // Included with Angular

```

```

1 export const environment = {
2   production: false,
3 };
4
5 export const SERVER_URL = 'http://10.0.2.2:8081';

```

```

5 "fileReplacements": [
6   {
7     "app": "src",
8     "root": "src",
9     "sourceRoot": "src",
10    "prefix": "app",
11    "schematics": {},
12    "architect": {
13      "build": {
14        "builder": "@angular-devkit/build-angular:browser",
15        "options": {
16          ...
17        },
18        "configurations": {
19          "production": {
20            ...
21          }
22        },
23        "android": {
24          "fileReplacements": [
25            {
26              "replace": "src/environments/environment.ts",
27              "with": "src/environments/environment.android.ts"
28            }
29          ],
30          ...
31        },
32        "cli": {
33          "progress": false
34        }
35      },
36      "serve": {
37        "builder": "@angular-devkit/build-angular:dev-server",
38        "options": {
39          "browserTarget": "app:build"
40        },
41        "configurations": {
42          "production": {
43            "browserTarget": "app:build:production"
44          },
45          "cli": {
46            "progress": false
47          }
48        }
49      }
50    }
51  }
52 }

```

Bridge between external file nad Page class

To map values from external file with Page class you ought to use class [GetEnvironmentParam](#).

Therefore when you add new variable (row) in `environments.csv` you might need to add this variable to [GetEnvironmentParam](#).

```

C:\Devon-dist-current\Devon-dist_3.0.0\workspaces\Test\devon4ng-ionic-application-template
λ ionic serve
> ng run app:serve --host=0.0.0.0 --port=8100
[ng] WARNING: This is a simple server for use in testing or debugging Angular applications
[ng] locally. It hasn't been reviewed for security issues.
[ng] Binding this server to an open connection can result in compromising your application or
[ng] computer. Using a different host than the one passed to the "--host" flag might result in
[ng] websocket connection issues. You might need to use "--disableHostCheck" if that's the
[ng] case.
[INFO] Waiting for connectivity with ng...
[INFO] Waiting for connectivity with ng...

[INFO] Development server running!

Local: http://localhost:8100
External: http://10.80.132.29:8100, http://192.168.56.1:8100, http://192.168.99.1:8100

Use Ctrl+C to quit this process

[INFO] Browser window opened to http://localhost:8100!

[ng] i 「wdm」: wait until bundle finished: /

```

76.1.16. Run test case with system environment

To run test case with system environment, please use:

- `-Denv=<NameOfEnvironment>`
- `<NameOfEnvironment>` is taken as column name from file `mrchecker-app-under-`

test|src|test|resources|enviroments|environments.csv

Command Line

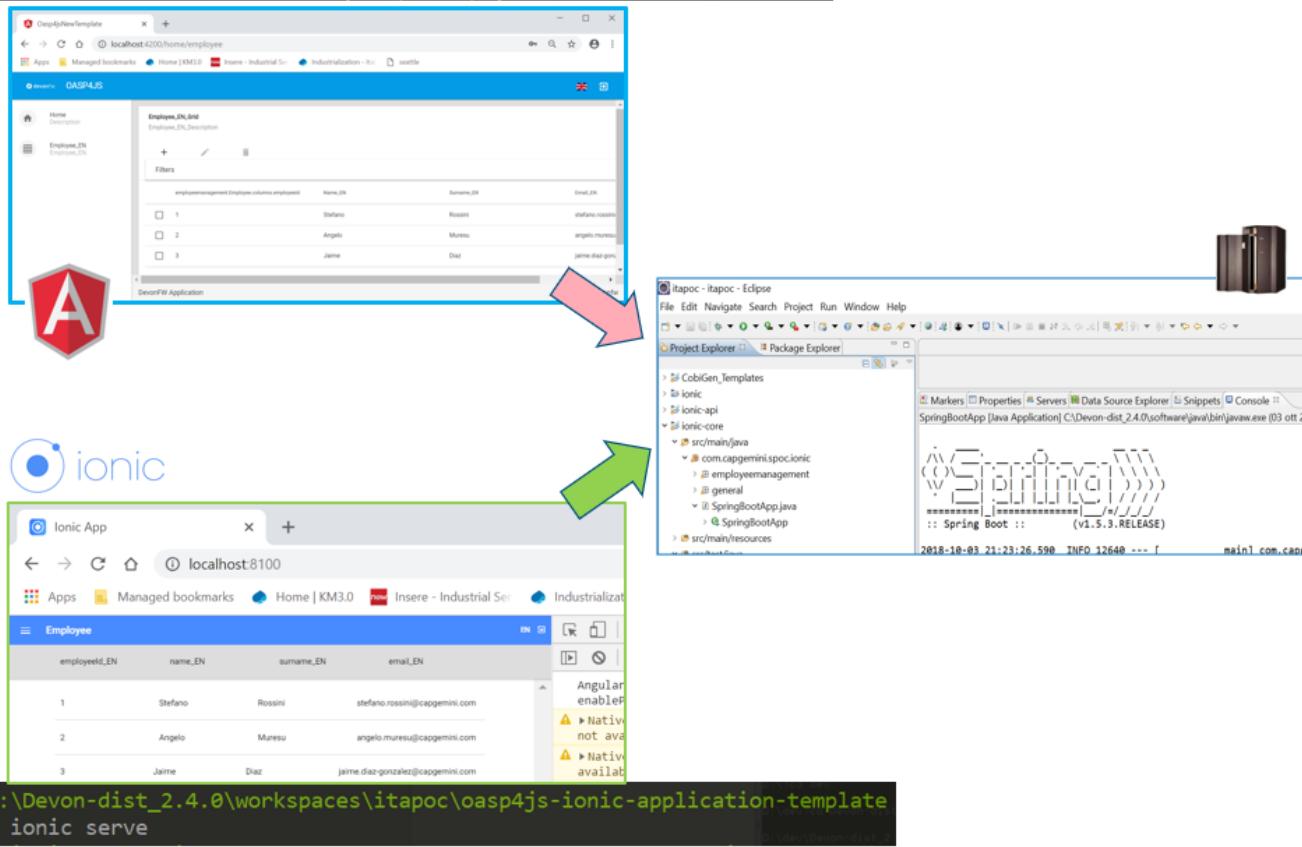
```
mvn test site -Dtest=RegistryPageTest -Denv=DEV
```

Eclipse

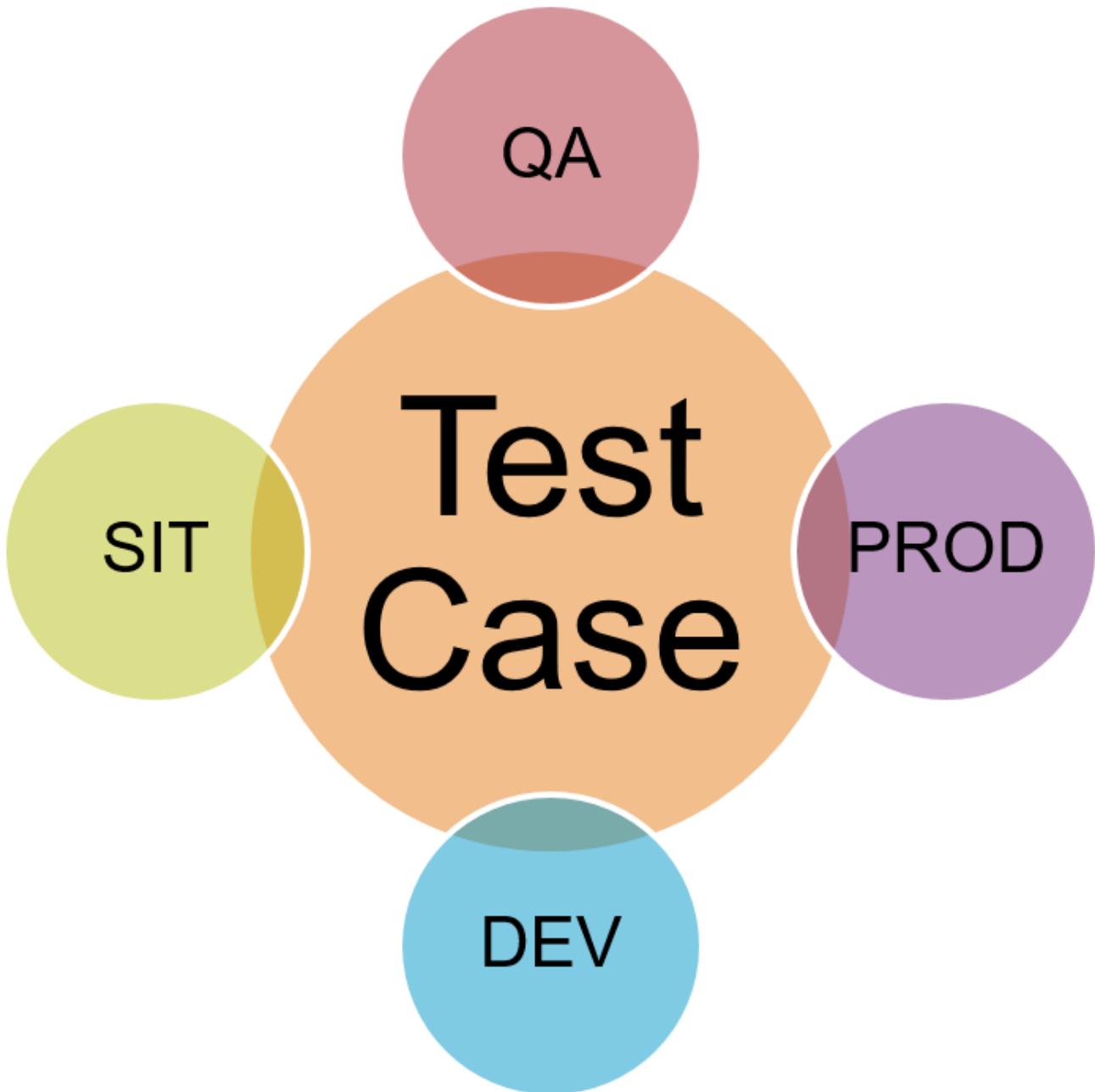
The screenshot shows a mobile application interface titled "Employee". The top navigation bar includes a back arrow, forward arrow, and a refresh icon, followed by the URL "localhost:8100/employee". Below the header, there's a toolbar with icons for responsive mode, screen size (630 x 492), zoom (100%), desktop/online status, and a refresh symbol. The main content area displays a table with three rows of employee data. The columns are labeled "employeeId_EN", "name_EN", "surname_EN", and "email_EN". The first row contains values 1, Stefano, Rossini, and stefano.rossini@capgemini.com. The second row contains values 2, Angelo, Muresu, and angelo.muresu@capgemini.com. The third row contains values 3, Jaime, Gonzalez, and jaime.diaz-gonzalez@capgemini.com. To the right of each row is a vertical stack of five circular icons: a magnifying glass (search), a trash can (delete), a pencil (edit), a plus sign (add), and an X (cancel).

employeeId_EN	name_EN	surname_EN	email_EN
1	Stefano	Rossini	stefano.rossini@capgemini.com
2	Angelo	Muresu	angelo.muresu@capgemini.com
3	Jaime	Gonzalez	jaime.diaz-gonzalez@capgemini.com

```
C:\Devon-dist_2.4.0\workspaces\itapoc\oasp4js-application-template
λ ng serve -o
```



76.1.17. System under test environments



- **Quality assurance** or **QA** is a way of preventing mistakes or defects in manufactured products and avoiding problems when delivering solutions or services to customers which ISO 9000 defines as "part of quality management focused on providing confidence that quality requirements will be fulfilled".
- **System integration testing** or **SIT** is a high-level software testing process in which testers verify that all related systems maintain data integrity and can operate in coordination with other systems in the same environment. The testing process ensures that all sub-components are integrated successfully to provide expected results.
- **Development** or **Dev** testing is performed by the software developer or engineer during the construction phase of the software development life-cycle. Rather than replace traditional QA focuses, it augments it. Development testing aims to eliminate construction errors before code is promoted to QA; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and QA process.
- **Prod** If the customer accepts the product, it is deployed to a **production** environment, making it

available to all users of the system.

[image051] | *images/image051.png*

76.1.18. How to use system environment

In Page classes, when you load / start web, it is uncommon to save fixed main url.

Value flexibility is a must, when your web application under test has different main url, depending on the environment (DEV, QA, SIT, ..., PROD)

Instead of **hard coded** main url variable, you build your Page classes with dynamic variable.

An example of dynamic variable **GetEnvironmentParam.WWW_FONT_URL**

```
import com.capgemini.ntc.selenium.core.BasePage;
import com.capgemini.ntc.selenium.pages.environment.GetEnvironmentParam;
import com.capgemini.ntc.selenium.pages.environment.PageSubURLsEnum;
import com.capgemini.ntc.selenium.pages.environment.PageTitlesEnum;

public class RegistryPage extends BasePage {

    @Override
    public boolean isLoaded() {
        return isUrlAndPageTitleAsCurrentPage(GetEnvironmentParam.WWW_FONT_URL + "" + PageSubURLsEnum.REGISTRATION);
    }

    @Override
    public void load() {
        BasePage.getDriver()
            .get(GetEnvironmentParam.WWW_FONT_URL + "" + PageSubURLsEnum.REGISTRATION);
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return PageTitlesEnum.REGISTRATION + "";
    }
}
```

76.1.19. How to create / update system environment

External file with variable values

Dynamic variable values are stored under **mrchecker-app-under-test\src\resources\environments\environments.csv**.

NOTE: As environments.csv is a comma-separated file, please be careful while editing and then save it under Excel.

service variable	DEV	QA	PROD
WWW_FONT_URL	http://demoga.com/	https://pl.wiktionary.org/wiki/	http://orient-addicts.djamila.pl/
SUB_PAGE			
TOOLS_QA	http://toolsqa.com/		
WEB_SERVICE	http://jsonplaceholder.typicode.com	http://jsonplaceholder.typicode.com	http://jsonplaceholder.typicode.com

Encrypting sensitive data

Some types of data you might want to store as environment settings are sensitive in nature (e.g. passwords). You might not want to store them (at least not in their plaintext form) in your repository. To be able to encrypt sensitive data you need to do following:

1. Create a secret (long, random chain of characters) and store it under **mrchecker-app-under-test\src\resources\secretData.txt**. Example: **LhwbTm9V3FUB05Tt5PiTUEQrXGgWrDLCMthnzLKNy1zA5FVTFiTdHRQAyPRIGXmsAjPUP1JSoSLeSBM**
2. Exclude the file from being checked into the git repository by adding it to **git.ignore**. You will need to pass the file over a different channel among your teammates.
3. Encrypt the values before putting them into the **environments.csv** file by creating following script (put the script where your jasypt library resides, e.g. **C:\MrChecker_Test_Framework\m2\repository\org\jasypt\jasypt\1.9.2**):

```
@ECHO OFF

set SCRIPT_NAME=encrypt.bat
set EXECUTABLE_CLASS=org.jasypt.intf.cli.JasyptPBESStringEncryptionCLI
set EXEC_CLASSPATH=jasypt-1.9.2.jar
if "%JASYPT_CLASSPATH%" == "" goto computeclasspath
set EXEC_CLASSPATH=%EXEC_CLASSPATH%;%JASYPT_CLASSPATH%

:computeclasspath
IF "%OS%" == "Windows_NT" setlocal ENABLEDELAYEDEXPANSION
FOR %%c in (%~dp0..\lib\*.jar) DO set EXEC_CLASSPATH=!EXEC_CLASSPATH!;%%c
IF "%OS%" == "Windows_NT" setlocal DISABLEDELAYEDEXPANSION

set JAVA_EXECUTABLE=java
if "%JAVA_HOME%" == "" goto execute
set JAVA_EXECUTABLE="%JAVA_HOME%\bin\java"

:execute
%JAVA_EXECUTABLE% -classpath %EXEC_CLASSPATH% %EXECUTABLE_CLASS% %SCRIPT_NAME% %*
```

1. Encrypt the values by calling

```
.\encrypt.bat input=someinput password=secret
```

----ENVIRONMENT-----

Runtime: Oracle Corporation Java HotSpot(TM) 64-Bit Server VM 25.111-b14

----ARGUMENTS-----

```
input: someinput
password: secret
```

----OUTPUT-----

JN3n0Fo12GMZoUxR5z2wI2qdipcNH1UD

1. Mark the value as encrypted by adding a prefix 'ENC(' and suffix ')' like: ENC(JN3n0Fo12GMZoUxR5z2wI2qdipcNH1UD)

service	variable	DEV	QA	PROD	DEV1	DEV2
WWW_FONT_URL		http://demoqa.com/		https://pl.wikt	http://orient-	https://myres
DMA_URL		https://dma.company.com				https://myres
MY_RESEARCH_URL_VALUE					https://myres	https://myres
TOOLS_QA		http://toolsqa.com/				
WEB_SERVICE		http://jsonplaceholder.typicode.com		http://jsonpla	http://jsonpla	
USER_PASSWD		ENC(kLihbooTMvRI7/W2yHQC0el/FTzIdVqG)				

Bridge between external file nad Page class

To map values from external file with Page class you ought to use class **GetEnvironmentParam**

Therefore when you add new variable (row) in **environments.csv** you might need to add this variable to **GetEnvironmentParam**.

```

import com.capgemini.ntc.selenium.core.Url;
import com.capgemini.ntc.test.core.BaseTest;
import com.capgemini.ntc.test.core.exceptions.BFInputDataException;

/**
 * @author lucst
 *         Takes values saved in /src/resources/environments/environment.csv.
 *         When -Denv is not set, then it takes default value = DEV
 */
public enum GetEnvironmentParam implements Url {

    // Name if enum must be in line with cell name in /src/resources/environments/environment.csv
    WWW_FONT_URL,
    SPS_WI_URL,
    TOOLS_QA,
    WEB_SERVICE;

    @Override
    public String toString() {
        return this.getAddress();
    }

    public String getAddress() throws BFInputDataException{
        if (null == BaseTest.getEnvironmentService()) {
            throw new BFInputDataException("Environment Parameters class wasn't initialized properly");
        }
        return BaseTest.getEnvironmentService()
            .getServiceAddress(this.name());
    }
}

```

76.1.20. Run test case with system environment

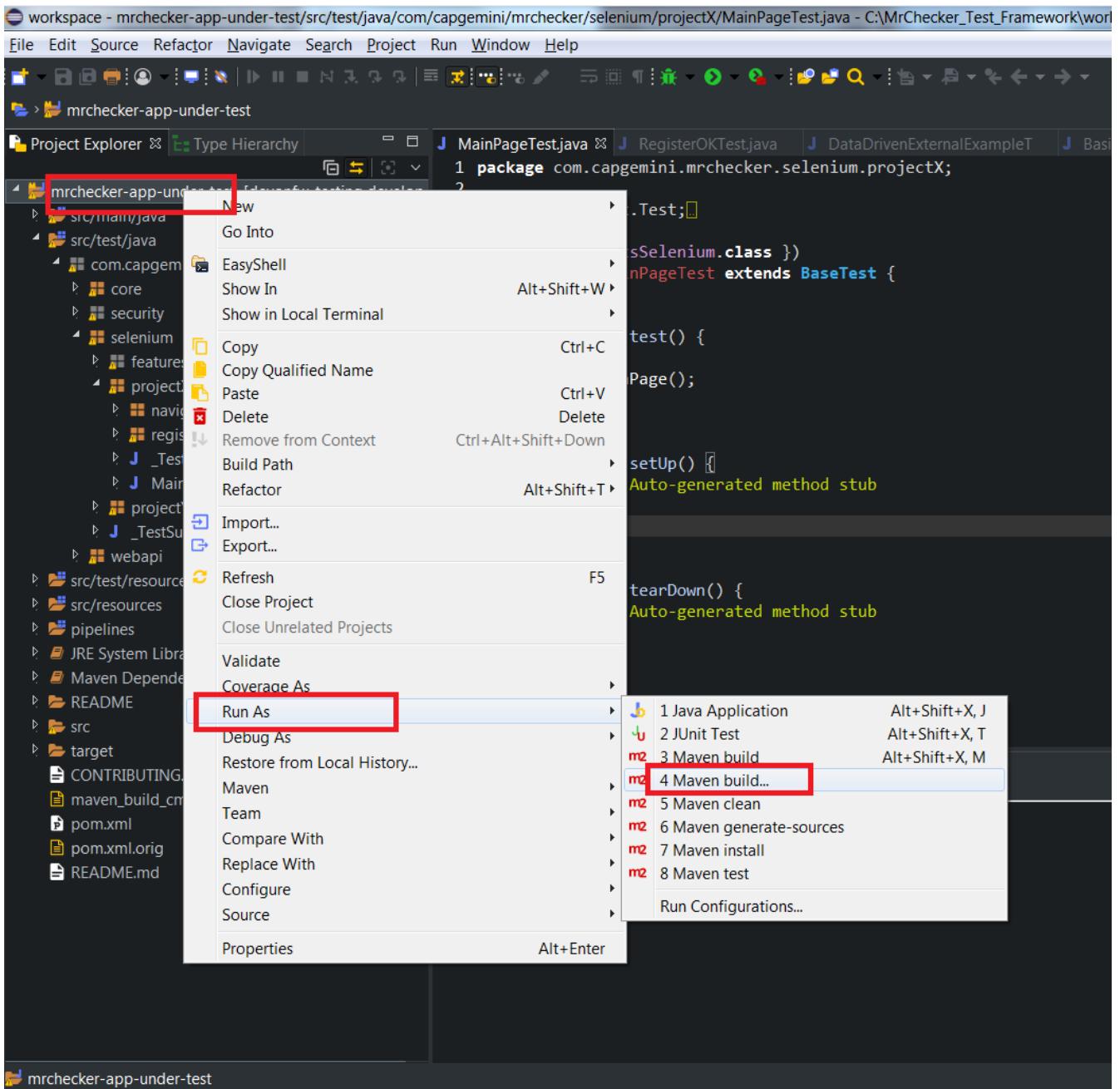
To run test case with system environment, please use: * -Denv=<NameOfEnvironment> * <NameOfEnvironment> is taken as column name from file **mrchecker-app-under-test\src\test\resources\environments\environments.csv**

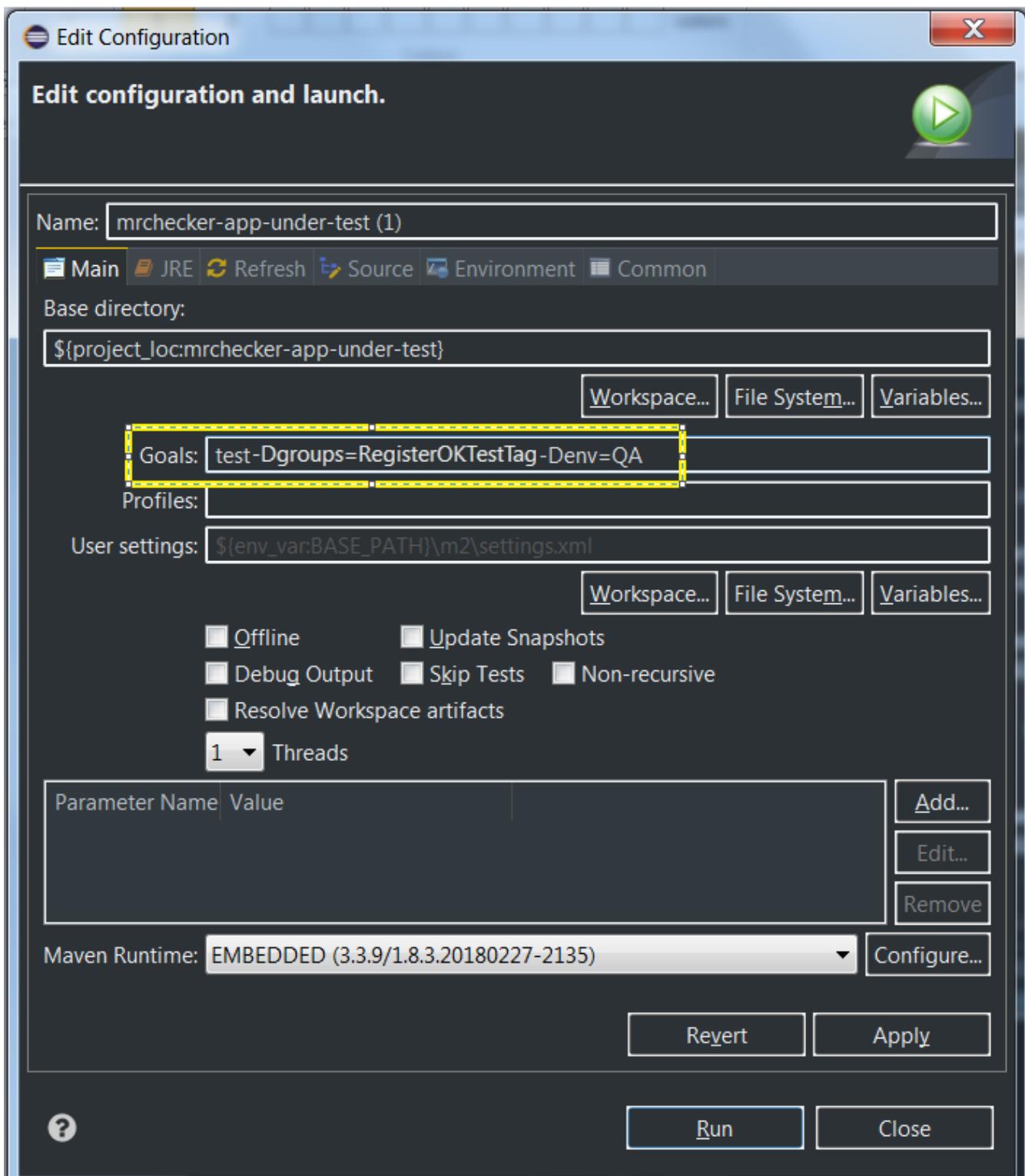
Since mrchecker-core-module version 5.6.2.1

Command Line

```
mvn test site -Dgroups=RegistryPageTestTag -Denv=DEV
```

Eclipse



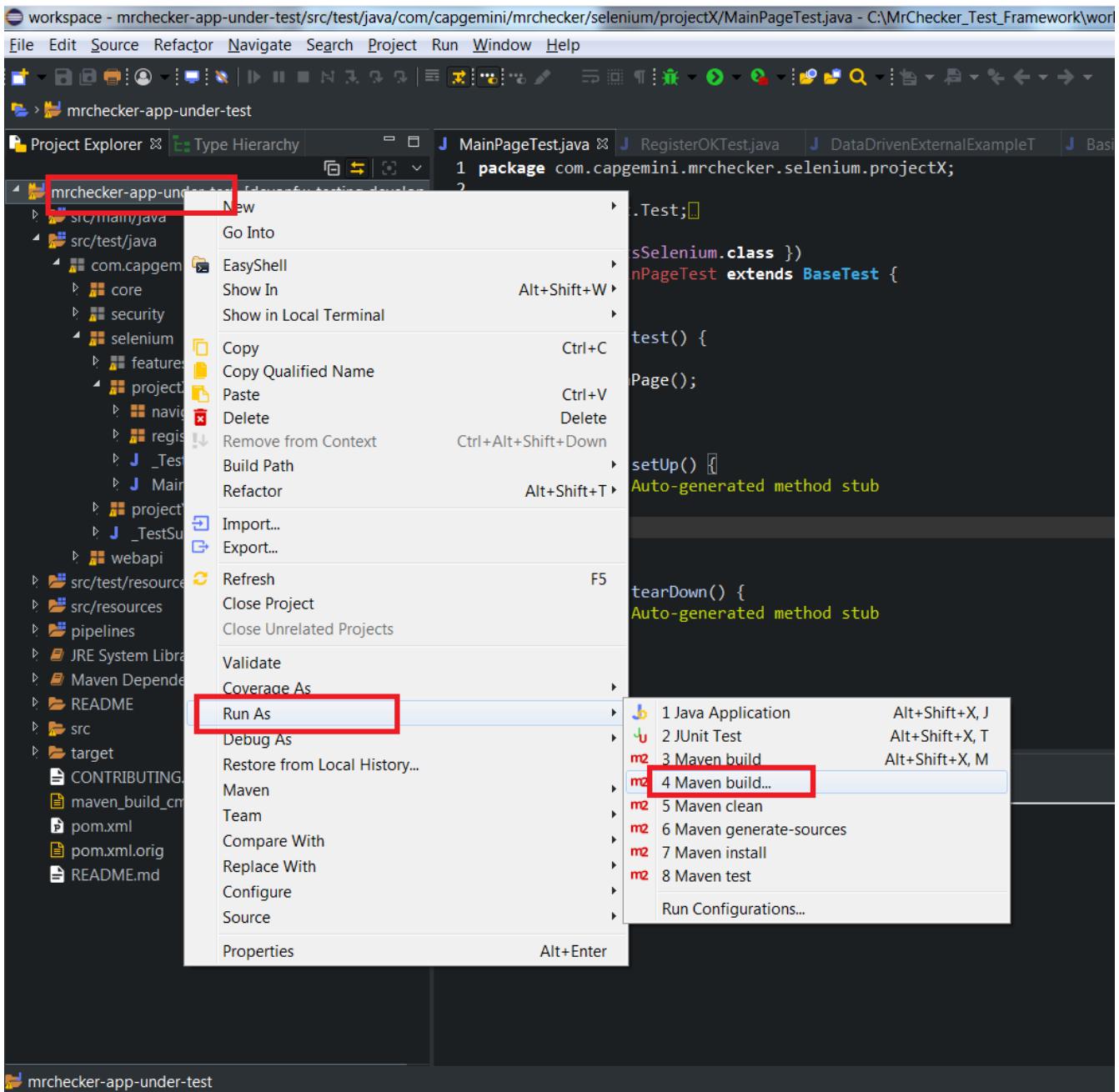


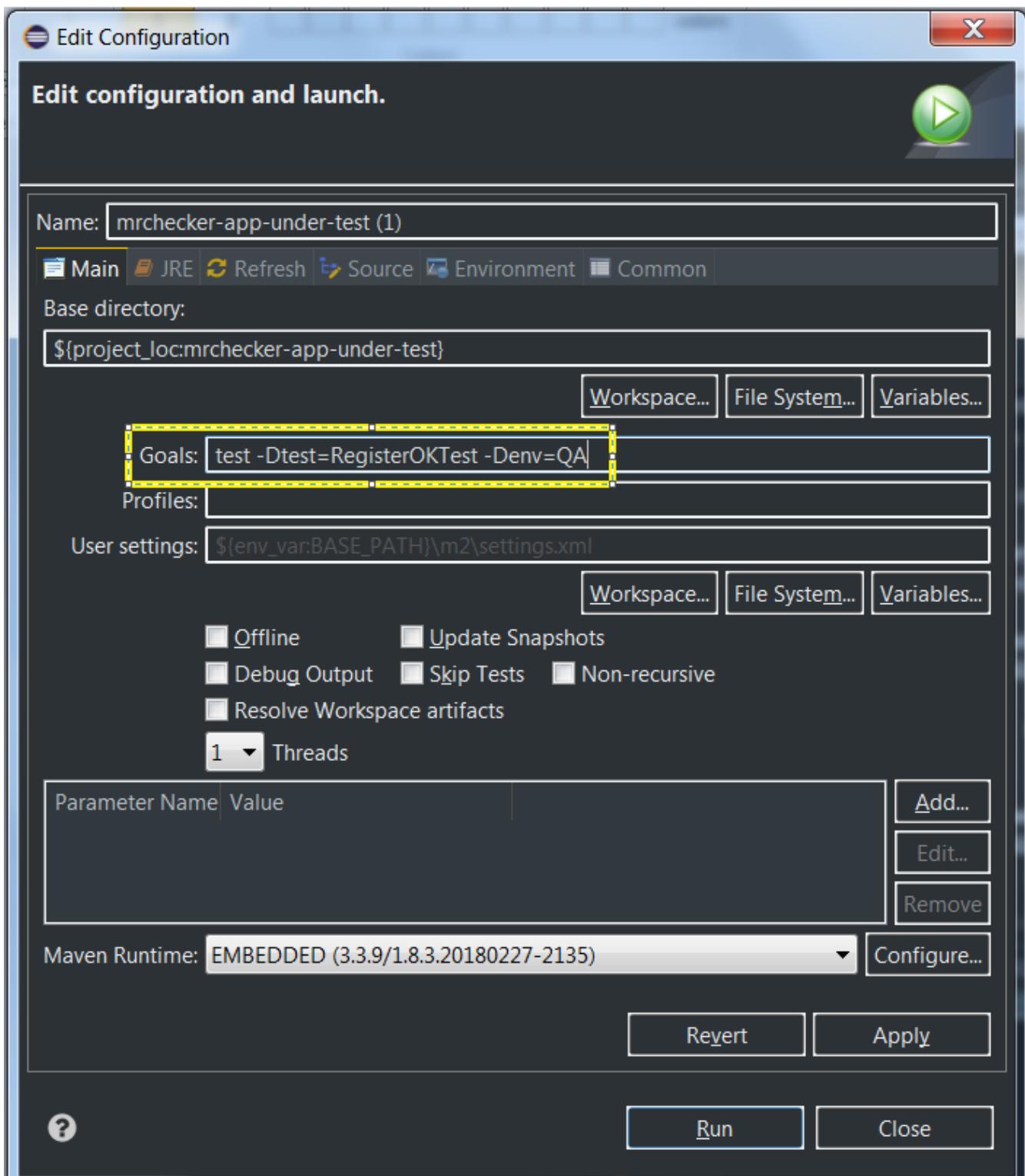
Prior to mrchecker-core-module version 5.6.2.1

Command Line

```
mvn test site -Dtest=RegistryPageTest -Denv=DEV
```

Eclipse





76.2. Selenium Module

76.2.1. Selenium Test Module

What is MrChecker E2E Selenium Test Module



Selenium Structure

- [What is Selenium](#)
- [What is WebDriver](#)
- [What is Page Object Model/Pattern](#)

Framework Features

- [Construction of Framework Page Class](#)
 - Every Page class must extend BasePage
 - What are `isLoaded()`, `load()` and `pageTitle()` for
 - How to create selector variable - `private static final By ButtonOkSelector = By.Css(...)`
 - How to prepare an 'everlasting' selector
 - Method/action naming convention
 - Why we should use `findElementDynamic()` and `findElementQuietly()` instead of classic Selenium `findElement`
 - List of well-rounded groups of user friendly actions (`ElementButton`, `ElementCheckbox`, `ElementInput`, etc.)
 - Verification points of well-defined Page classes and Test classes
- [Run on different browsers: Chrome, Firefox, IE, Safari, Edge](#)
- [Run with different browser options](#)
- [Run with full range of resolution \(mobile, desktop\): Testing responsible Design Webpage](#)

How to start?

Read: [My first Selenium Test](#)

Selenium Best Practices

- [Table of best practices](#)

Selenium UFT Comparison

- [Selenium UFT Comparison](#)

76.3. Selenium Structure

76.3.1. What is Selenium

Selenium is a framework for testing browser applications. The test automation supports:

- Frequent regression testing
- Repeating test case executions
- Documentation of test cases
- Finding defects
- Multiple Browsers

The Selenium testing framework consists of multiple tools:

- **Selenium IDE**

The Selenium Integrated Development Environment is a prototyping tool for building test scripts. It is a Firefox Plugin and provides an easy-to-use interface for developing test cases. Additionally, Selenium IDE contains a recording feature, that allows the user to record user inputs that can be automatically re-executed in future.

- **Selenium 1**

Selenium 1, also known as Selenium RC, commands a Selenium Server to launch and kill browsers, interpreting the Selenese commands passed from the test program. The Server acts as an HTTP proxy. **This tool is deprecated.**

- **Selenium 2**

Selenium 2, also known as Selenium WebDriver, is designed to supply a well-designed, object-oriented API that provides improved support for modern advanced web-app testing problems.

- **Selenium 3.0**

The major change in Selenium 3.0 is removing the original Selenium Core implementation and replacing it with one backed by WebDriver. There is now a W3C specification for browser automation, based on the Open Source WebDriver.

- **Selenium Grid**

Selenium Grid allows the scaling of Selenium RC test cases, that must be run in multiple and potentially variable environments. The tests can be run in parallel on different remote machines.

Selenium on the Production Line

More information on Selenium on the Production Line can be found [here](#).

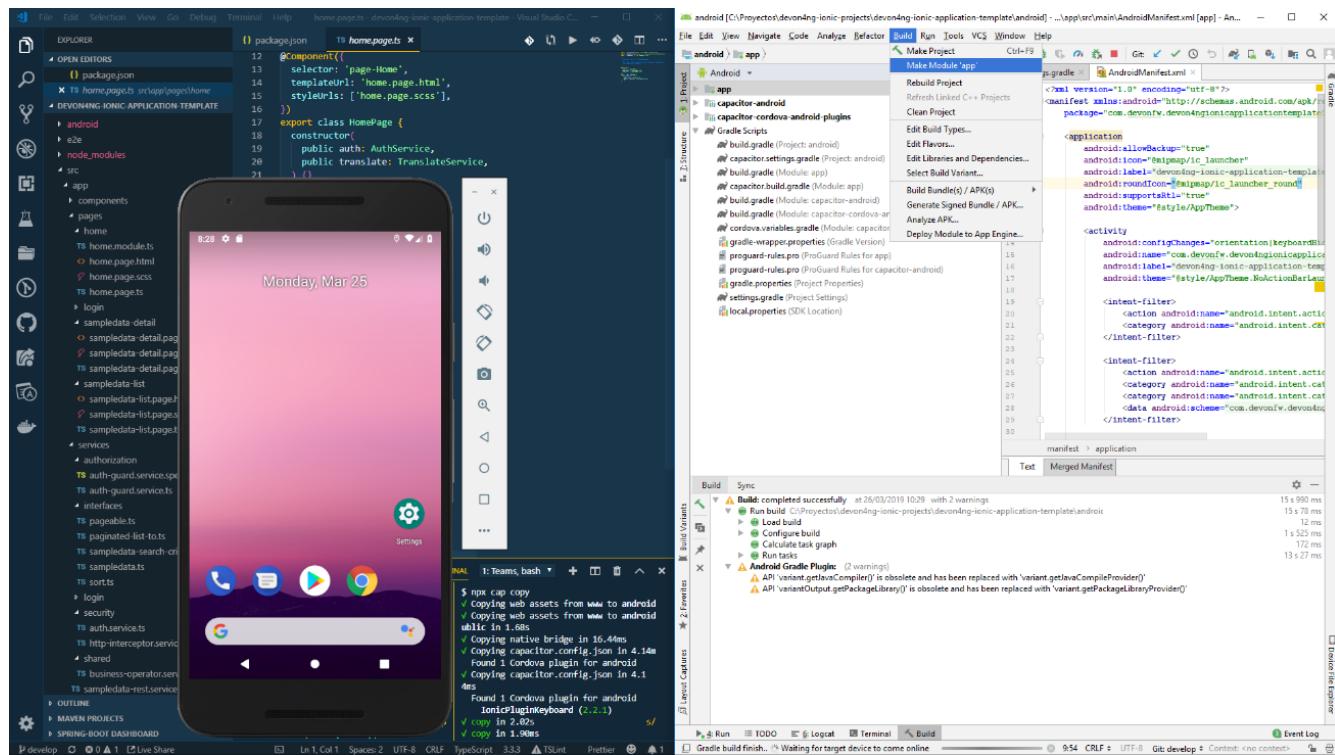
tl;dr

The Production Line has containers running Chrome and Firefox Selenium Nodes. The communication with these nodes is accomplished using Selenium Grid.

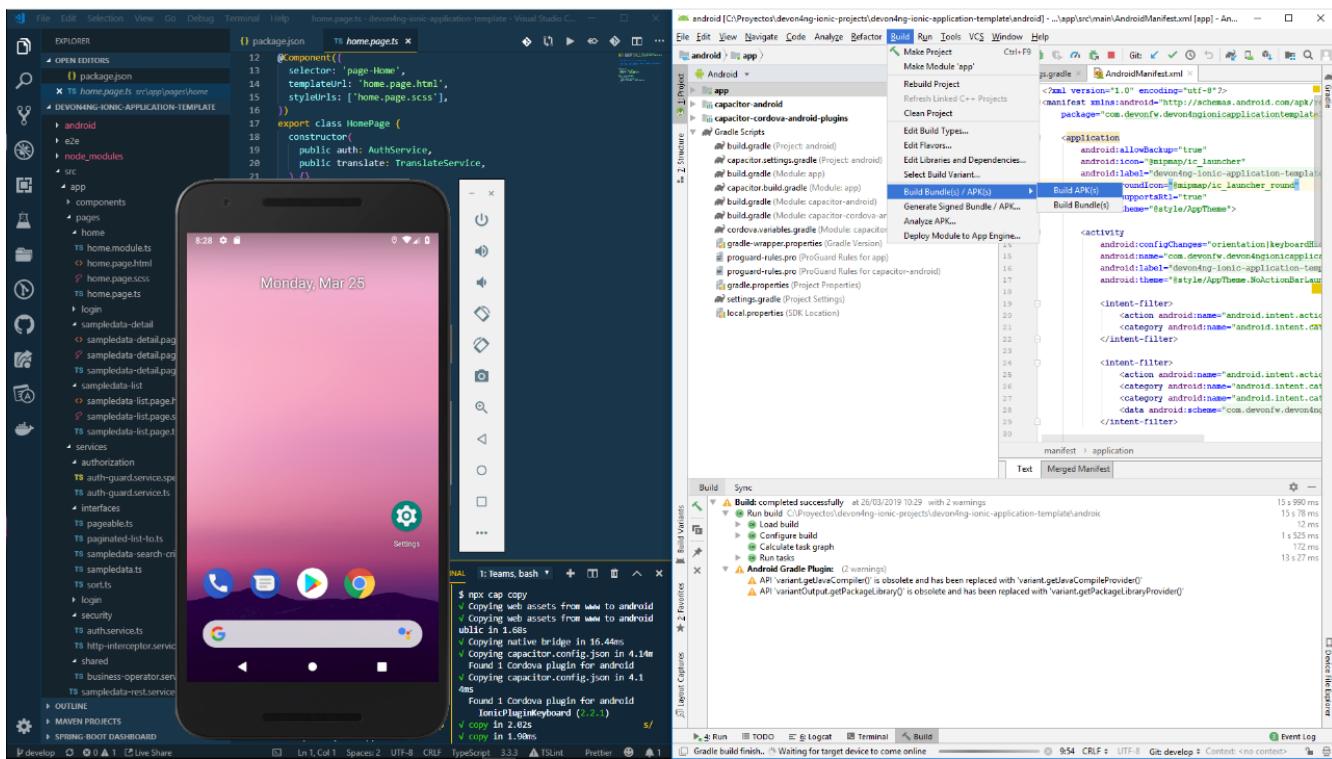
Having issues using Selenium on the Production Line? Check the Production Line [issue list](#), maybe it's a known issue that can be worked around.

76.3.2. What is WebDriver

On the one hand, it is a very convenient API for a programmer that allows for interaction with the browser, on the other hand it is a driver concept that enables this direct communication.



How does it work?

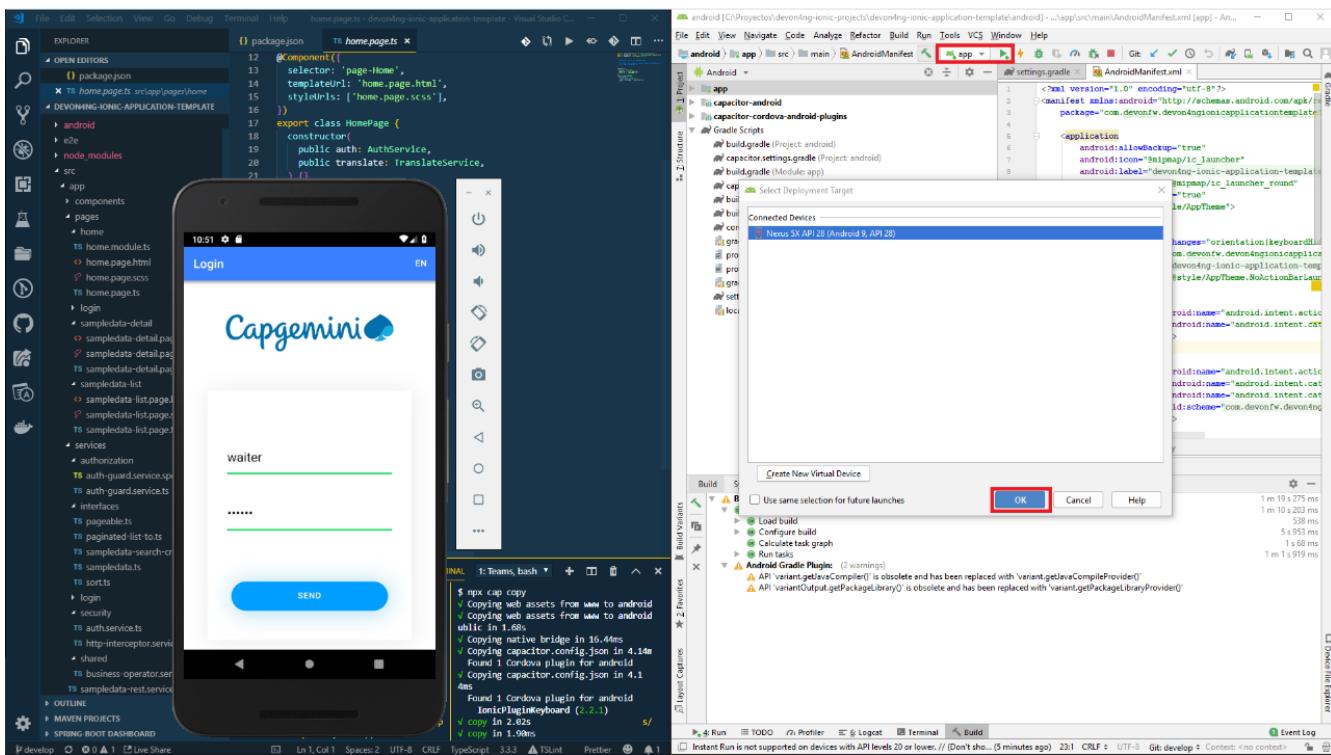


A tester, through their test script, can command WebDriver to perform certain actions on the WAUT on a certain browser. The way the user can command WebDriver to perform something is by using the client libraries or language bindings provided by WebDriver.

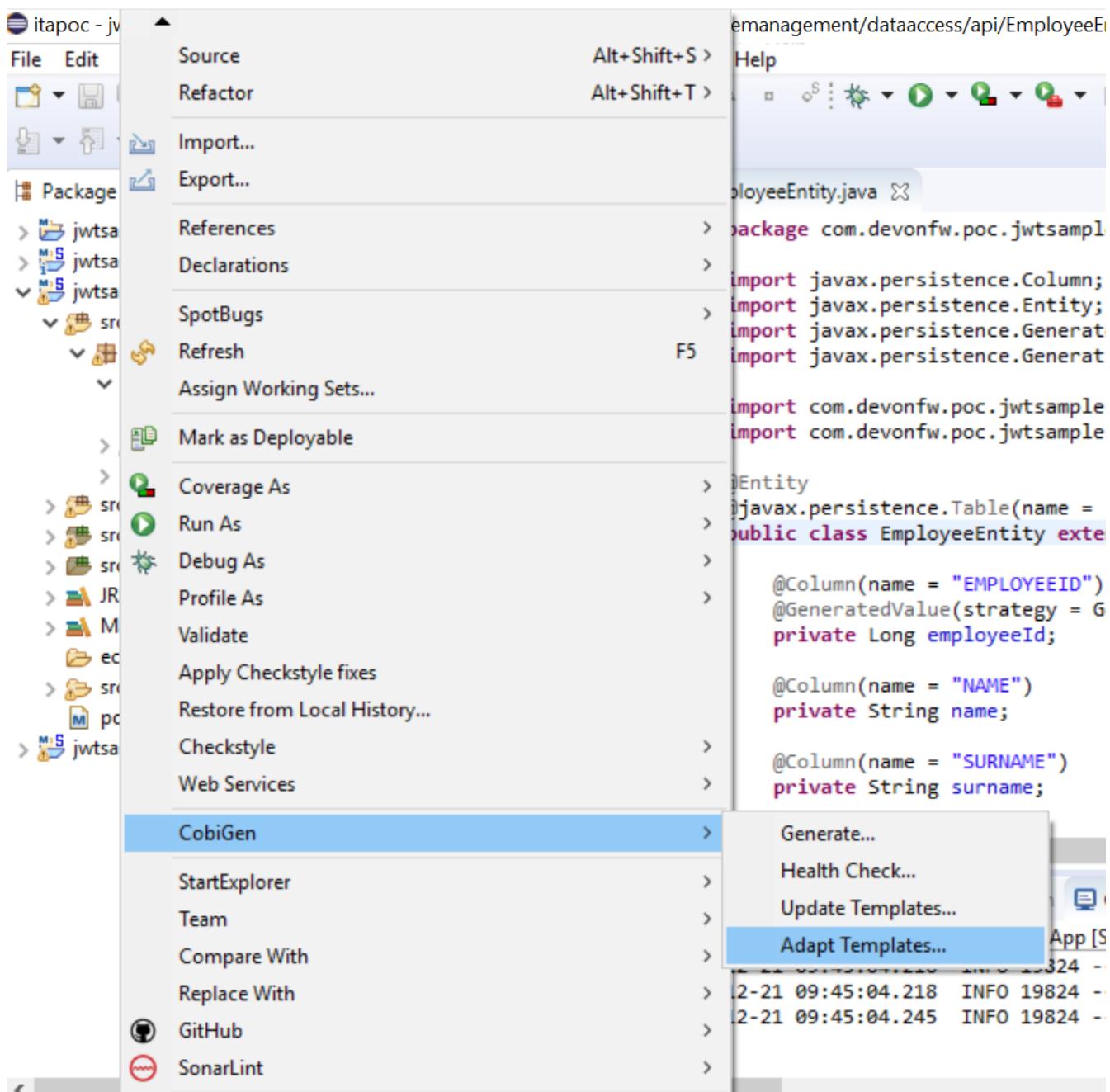
By using the language-binding client libraries, a tester can invoke browser-specific implementations of WebDriver, such as Firefox Driver, IE Driver, Opera Driver, and so on, to interact with the WAUT of the respective browser. These browser-specific implementations of WebDriver will work with the browser natively and execute commands from outside the browser to simulate exactly what the application user does.

After execution, WebDriver will send the test result back to the test script for developer's analysis.

76.3.3. What is Page Object Model?

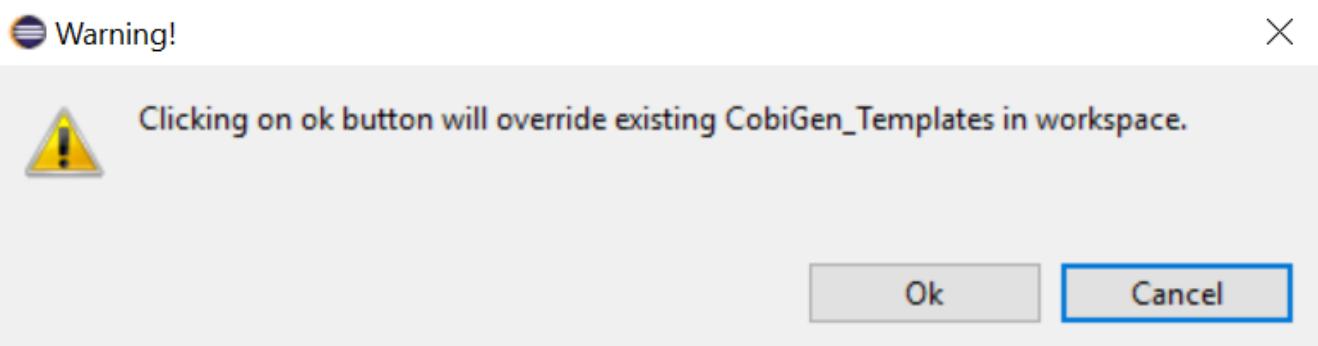


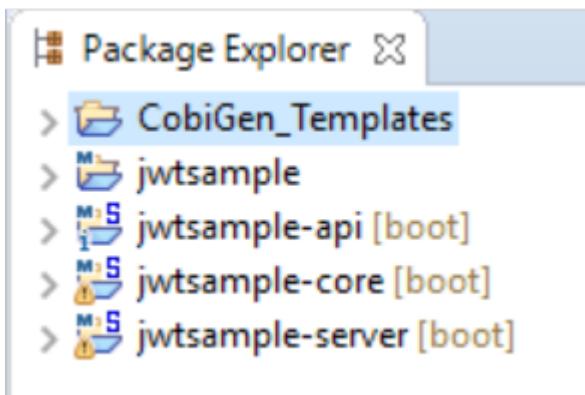
Creating Selenium test cases can result in an unmaintainable project. One of the reasons is that too much duplicated code is used. Duplicated code could result from duplicated functionality leading to duplicated usage of locators. The main disadvantage of duplicated code is that the project is less maintainable. If a locator changes, you have to walk through the whole test code to adjust locators where necessary. By using the page object model we can make non-brittle test code and reduce or eliminate duplicate test code. In addition, it improves the readability and allows us to create interactive documentation. Last but not least, we can create tests with less keystroke. An implementation of the page object model can be achieved by separating the abstraction of the test object and the test scripts.



76.3.4. Basic Web elements

This page will provide an overview of basic web elements.





Name	Method to use element
Form: Input Text	elementInputText()
Form: Label	elementLabel()
Form: Submit Button	elementButton()
Page: Button	elementButton()
Checkbox	elementCheckbox()
Radio	elementRadioButton()
Elements (Tabs, Cards, Account, etc.)	elementTab()
Dropdown List	elementDropdownList()
Link	-
Combobox	elementList()

Comparision how picking value from **checkbox** can be done:

- by classic Selenium atomic actions
- by our enhanced Selenium wrapper

Classic Selenium atomic actions

```
List<WebElement> checkboxesList = getDriver()
    .findElements(selectorHobby);
WebElement currentElement;
for (int i = 0; i < checkboxesList.size(); i++) {
    currentElement = checkboxesList.get(i);
    if (currentElement.getAttribute("value")
        .equals(hobby.toString()) && currentElement.isSelected() != true)
    {
        currentElement.click();
    }
}
```

Enhanced Selenium in E2E test framework

```
getDriver().elementCheckbox(selectorHobby)
    .setCheckBoxByValue(hobby.toString());
```

76.4. Framework Features

76.4.1. Page Class

Page Object Models allow for the representation of a webpage as a Java Class. The class contains all required web elements like buttons, textfields, labels, etc. When initializing a new project, create a new package to store the Page Object Models in.

Initialization

Source folder: *allure-app-under-test/src/main/java*

Name: *com.example.selenium.pages.YOUR_PROJECT*

Classes being created inside of this new package have to extend the **BasePage** class. As a result, a few abstract methods from **BasePage** have to be implemented.

```
public class DemoPage extends BasePage {

    @Override
    public boolean isLoading() {
        }

    @Override
    public void load() {
        }

    @Override
    public String pageTitle() {
        }
}
```

The example above demonstrates a minimum valid Page Object class with all required methods included.

BasePage method: **isLoading**

The inherited method **isLoading()** can be used to check if the current Page Object Model has been loaded correctly. There are multiple ways to verify a correctly loaded page. One example would be to compare the actual page title with the expected page title.

```
public boolean isLoaded() {
    if(getDriver().getTitle().equals("EXPECTED_TITLE")) {
        return true;
    }
    return false;
}
```

BasePage method: load

The method `load()` can be used to tell the webdriver to load a specific page.

```
public void load() {
    getDriver().get("http://SOME_PAGE");
}
```

BasePage method: pageTitle

The `pageTitle()` method returns a String containing the page title.

Creating a selector variable

To initialize web elements, a large variety of selectors can be used.

We recommend creating a private and constant field for every web element you'd like to represent in Java. Use the guide above to find the preferred selector and place it in the code below at "WEB_ELEMENT_SELECTOR".

```
private static final By someWebElementSelector = By.CSS("WEB_ELEMENT_SELECTOR");
```

As soon as you create the selector above, you can make use of it to initialize a WebElement object.

```
WebElement someWebElement = getDriver().findDynamicElement(someWebElementSelector);
```

Note: The examples displayed in the `cssSelector.docx` file use the Selenium method `driver.findElement()` to find elements. However, using this framework we recommend `findDynamicElement()` or `findQuietlyElement().findDynamicElement()` allows waiting for dynamic elements, for example buttons that pop up.

Creating a page method

To interact with the page object, we recommend creating methods for each action.

```
public void enterGoogleSearchInput(String query) {
    ...
}
```

Creating a method like the one above allows the test case to run something like `googleSearchPage.enterGoogleSearchInput("Hello")` to interact with the page object.

Naming Conventions

For code uniformity and readability, we provide a few method naming conventions.

Element	Action	Name (example)
Form: Input text	enter	<code>enterUsernameInput()</code>
	is (label)	<code>isUsernameInputPresent()</code>
	is (value)	<code>isUsernameEmpty()</code>
	get	<code>getUsernameValue()</code>
Form: Label	get	<code>getCashValue()</code>
	is (value)	<code>isCashValueEmpty()</code>
	is (label)	<code>isCashLabelPresent()</code>
Form: Submit Button	submit	<code>submitLoginForm()</code>
	is	<code>isLoginFormPresent()</code>
Page: Button	click	<code>clickInfoButton()</code>
	is	<code>isInfoButtonpresent()</code>
Checkbox	set	<code>setRememberMeCheckbox()</code>
	unset	<code>unsetRememberMeCheckbox()</code>
	is (present)	<code>isRememberMeCheckboxPresent()</code>
	is (value)	<code>isRememberMeCheckboxSet()</code>
Radio	set	<code>setMaleRadioValue("Woman")</code>
	is (present)	<code>isMaleRadioPresent()</code>
	is (visible)	<code>isMaleRadioVisible()</code>
	get	<code>getSelectedMaleValue()</code>
Elements (Tabs, Cards, Account, etc.)	click	<code>clickPositionTab() / clickMyBilanceCard()</code>
	is	<code>isMyBilanceCardPresent()</code>
Dropdown List	select	<code>selectAccountTypeValue(typeName)</code>
	unselect	<code>unselectAccountTypeValue(typeName)</code>
	multiple select	<code>selectAccountTypesValues(List typeNames)</code>
	is (list)	<code>isAccountTypeDropdownListPresent()</code>

Element	Action	Name (example)
	is (element present)	isAccountTypeElementPresent(typeName)
	is (element selected)	isAccountTypeSelected(typeName)
Link	click	clickMoreLink()
	is	isMoreLinkPresent()
Combobox	select	selectSortCombobox()
	is (present)	isSortComboboxPresent(name)
	is (contain)	selectSortComboboxContain(name)
Element Attribute	get	getPositionTabCss()
	get	getMoreLinkHref() / getRememberMeCheckboxName()

A css selector is used to select elements from an HTML page.

Selection by element tag, class or id are the most common selectors.

```
<p class='myText' id='123'>
```

This text element (p) can be found by using any one of the following selectors:

The HTML element: "p". Note: in practical use this will be too generic, if a preceding text section is added, the selected element will change.

The class attribute preceded by ".": ".myText"

The id attribute preceded by "#": "#123"

Using other attributes

When a class or an id attribute is not sufficient to identify an element, other attributes can be used as well, by using "[attribute=value]": For example:

```
<a href='https://ns.nl/example.html'>
```

This can be selected by using the entire value: "a[href='https://ns.nl/example.html']". For selecting links starting with, containing, ending with see the list below.

Using sub-elements

The css selectors can be stacked, by appending them:

```
<div id='1'><a href='ns.nl'></div>
<div id='2'><a href='nsinternational.nl'></div>
```

In the example above, the link element to nsinternational can be obtained with: "#2 a".

When possible avoid

- Using paths of commonly used HTML elements within the containers (HTML: div). This will cause failures when a container is added, a common occurrence during development, e.g. "div div p". Use class or id instead, if those are not available, request them to be added in the production code.
- Magic order numbers. It is possible to get the second text element in its parent container by using the selector "p:nth-child(2)". If the items are representing different items, ask the developer to add specific attributes. It is also possible to request all items, with a selector similar to ".myList li", and iterate through them later.

List

A good list with CSS Selectors can be found at W3Schools:

https://www.w3schools.com/cssref/css_selectors.asp

76.4.2. Selenium UFT Comparison

Subject	HP UFT	HP LeanFT	Selenium	Selenium IDE
Language	VBScript	Same as Selenium	Supports several languages. Java	Javascript
Learning curve	Based on VBScript which is relatively easy to learn	Less intuitive, more coding knowledge necessary	Less intuitive, more coding skills necessary	Record/playback possible. Generated code difficult to maintain
Project type	Traditional	Agile	Agile	Agile
User oriented	More Tester	More Developer	More Developer	More Tester
Object recognition	Test object identification and storage in object repository	Same as UFT	With Firebug	Same as SE
Customizations	Only the available standard. No customization	Same as UFT	Lots of customizations possible	Fewer than SE
Framework	Needed. Exists in ATaaS		Needed. Integration with Fitnesse, Cucumber, Gauche	No Framework. Limited capabilities of the tool.

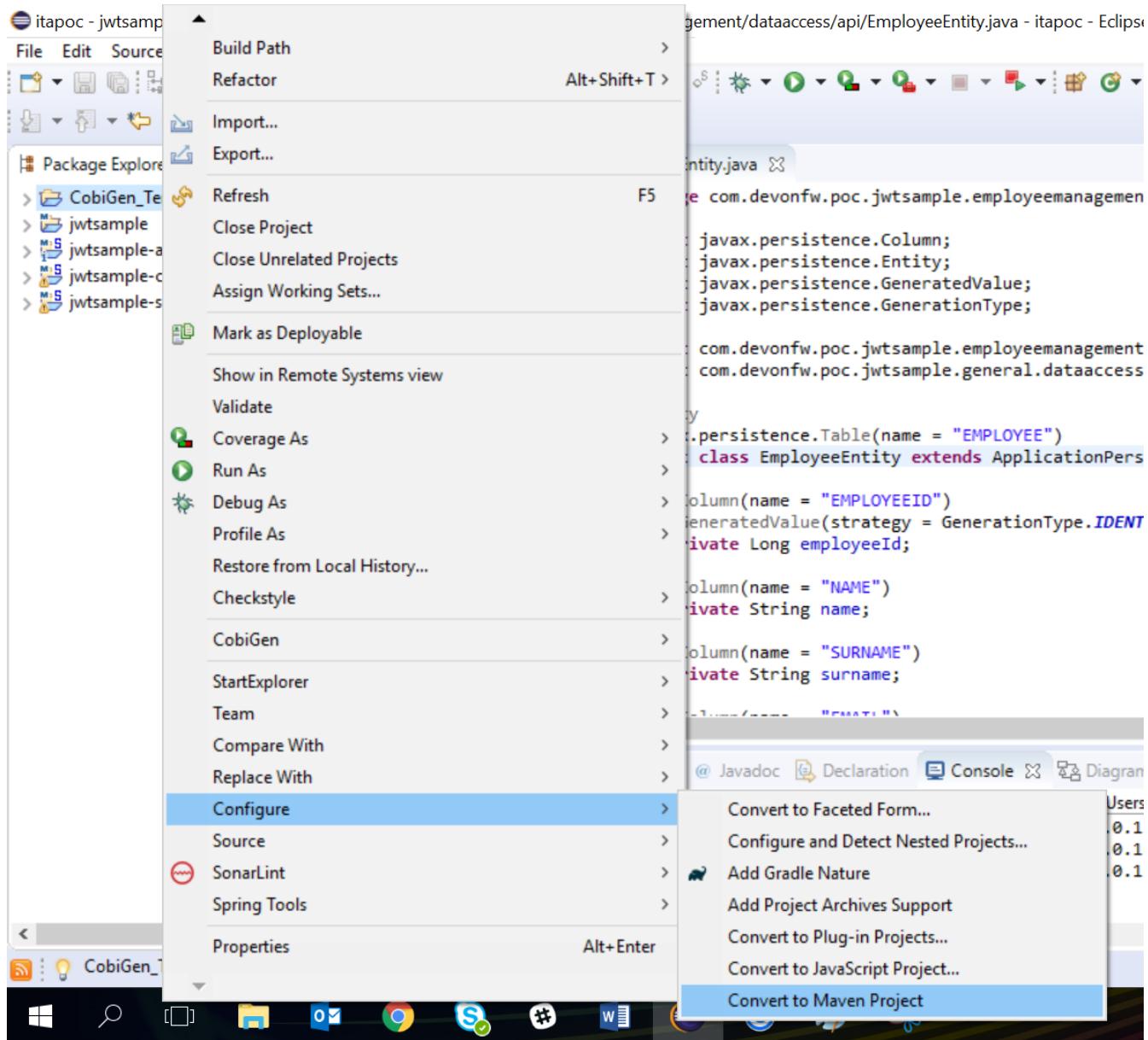
Subject	HP UFT	HP LeanFT	Selenium	Selenium IDE
Operating System support	Runs on Windows	Runs on Windows	Multiple OS support. With Grid: testing on multiple devices at same time	Plugin for Firefox
Application coverage	Many	Many	Web only	Web only
Multiple browsers	In UFT 12.5 available	In 12.5 available	Multiple tests in multiple browser windows at once and faster support for new browser versions	Multiple tests in multiple browser windows at once and faster support for new browser versions
System Load	High system load (RAM & CPU usage)	Lower load than HP UFT?	Lower load than HP UFT	Lower load than HP UFT
ALM integration	With HP ALM – full integration		Jira, Jenkins Not with ALM tool	Same as SE
Integration with other tools	A lot can be built, but many are already covered.	More than UFT.	Freeware and can be integrated with different open source tools	Freeware and can be integrated with different open source tools
Addins	Add-ins necessary to access all capabilities of the tool – license related	Same as UFT	See integration with other tools	See integration with other tools
Reporting	Complete, link to ALM	Same as UFT	No native mechanism for generating reports, but multiple plugins available for reporting	No native mechanism for generating reports, but multiple plugins available for reporting
Support	HP full support	Same as UFT	Limited support as it is open source	Limited support as it is open source
License costs	About 17K – Capgemini price 5K. Included in the S2 service charge	Same price as HP UFT	Free	Free limited functionality (no iterations / conditional statements)
iVAL Service	ATaaS	Not in a S2 service	Not in a S2 service	Not in a S2 service

Bold for key differentiators.

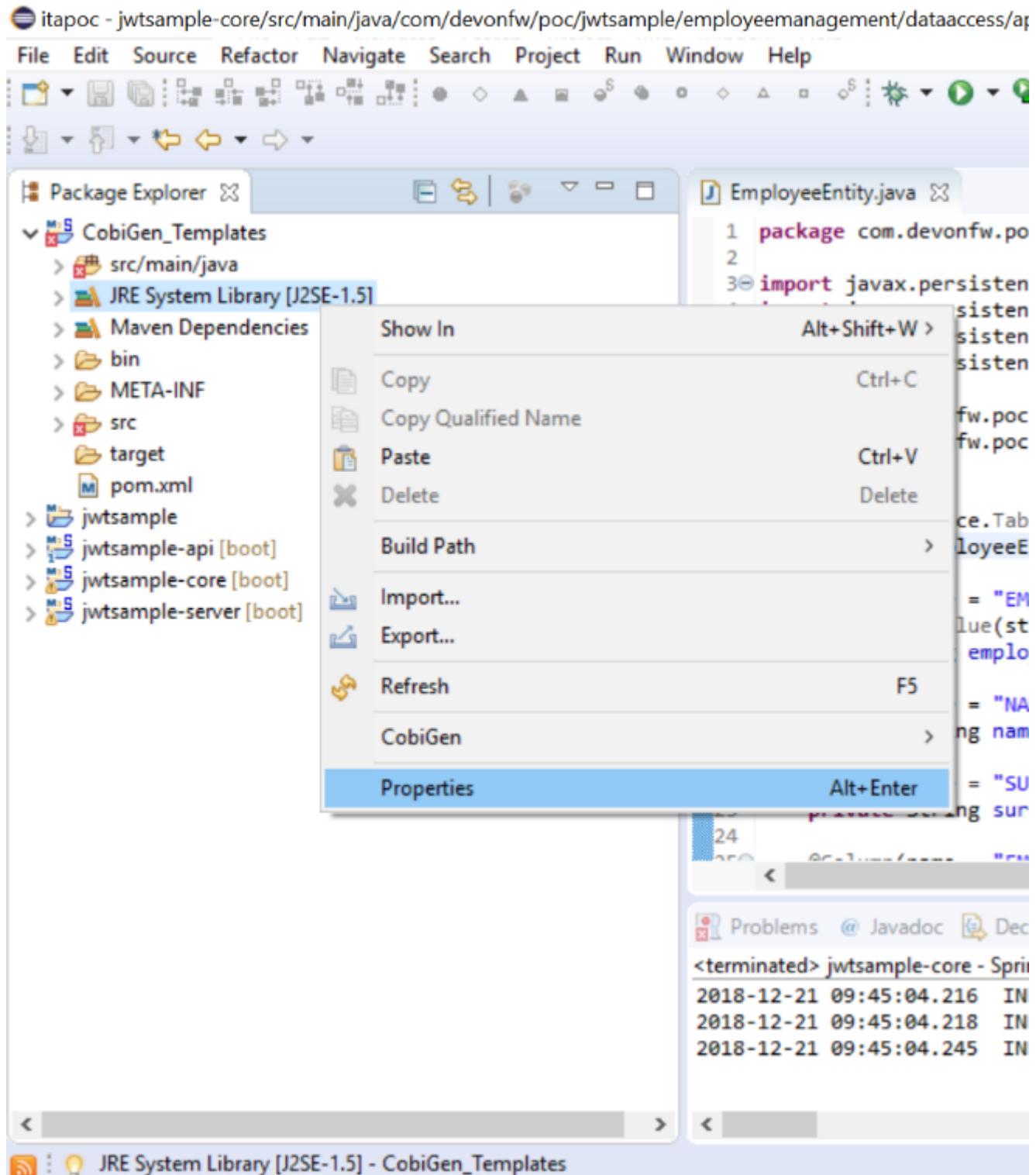
Projects also choose an available resource and the knowledge of that resource.

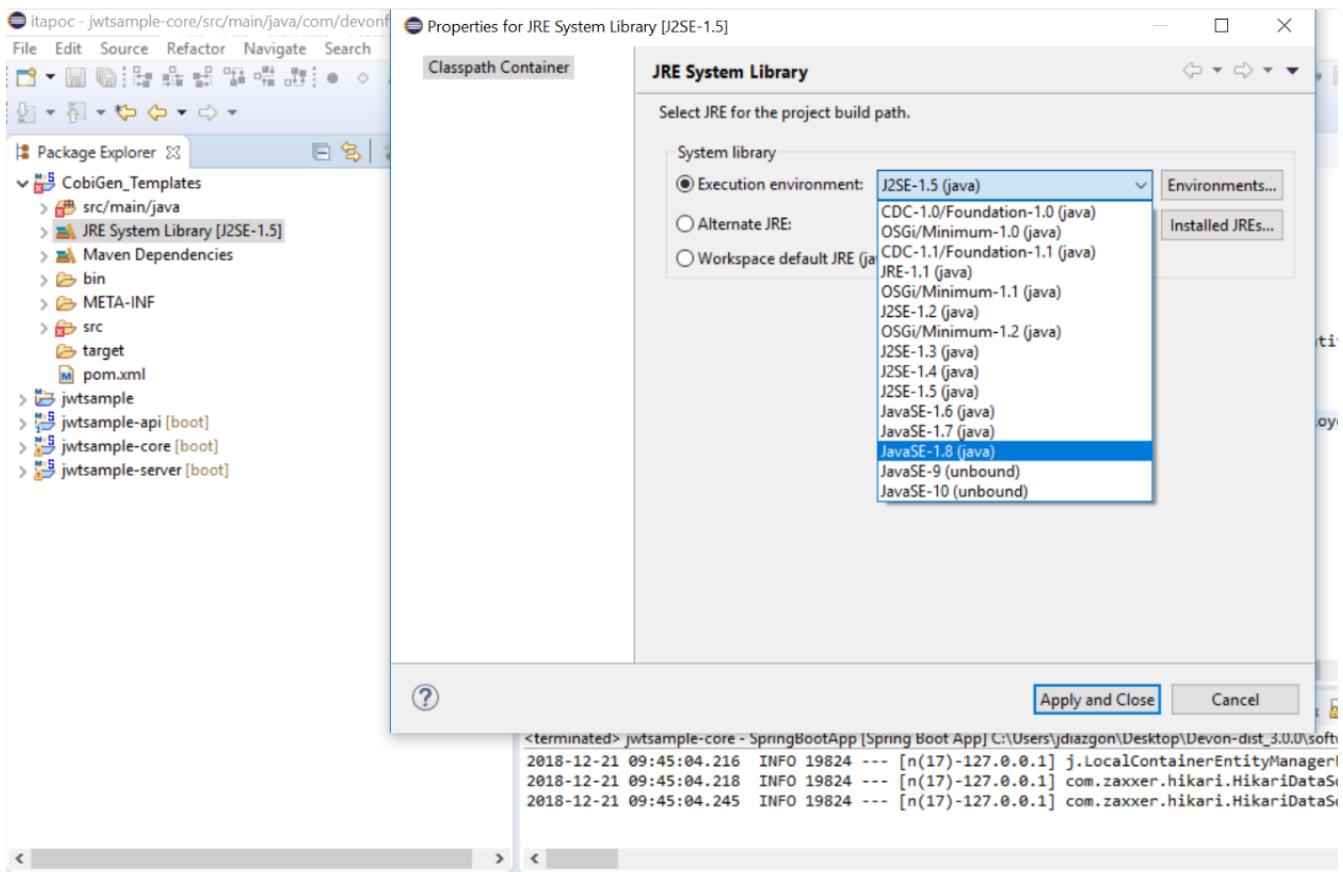
Both: Framework determines the quality of automation. Needs to be set up by someone with experience with the tool

76.4.3. Run on different browsers



To execute each test with a chosen installed browser, specific arguments are required in Run configuration.





It is necessary to enter `-Dbrowser=` with browser parameter name as an argument (in 'Arguments' tab):

firefox ie phantomjs chrome chromeheadless For example: `-Dbrowser=ie`

`_ea_` should be entered as an argument to restore default settings.

76.4.4. Browser options

To run a browser with specific options during runtime, please use

`-DbrowserOptions=<options>`

```
> mvn test -DbrowserOptions="param1"
> mvn test -DbrowserOptions="param1=value1"
```

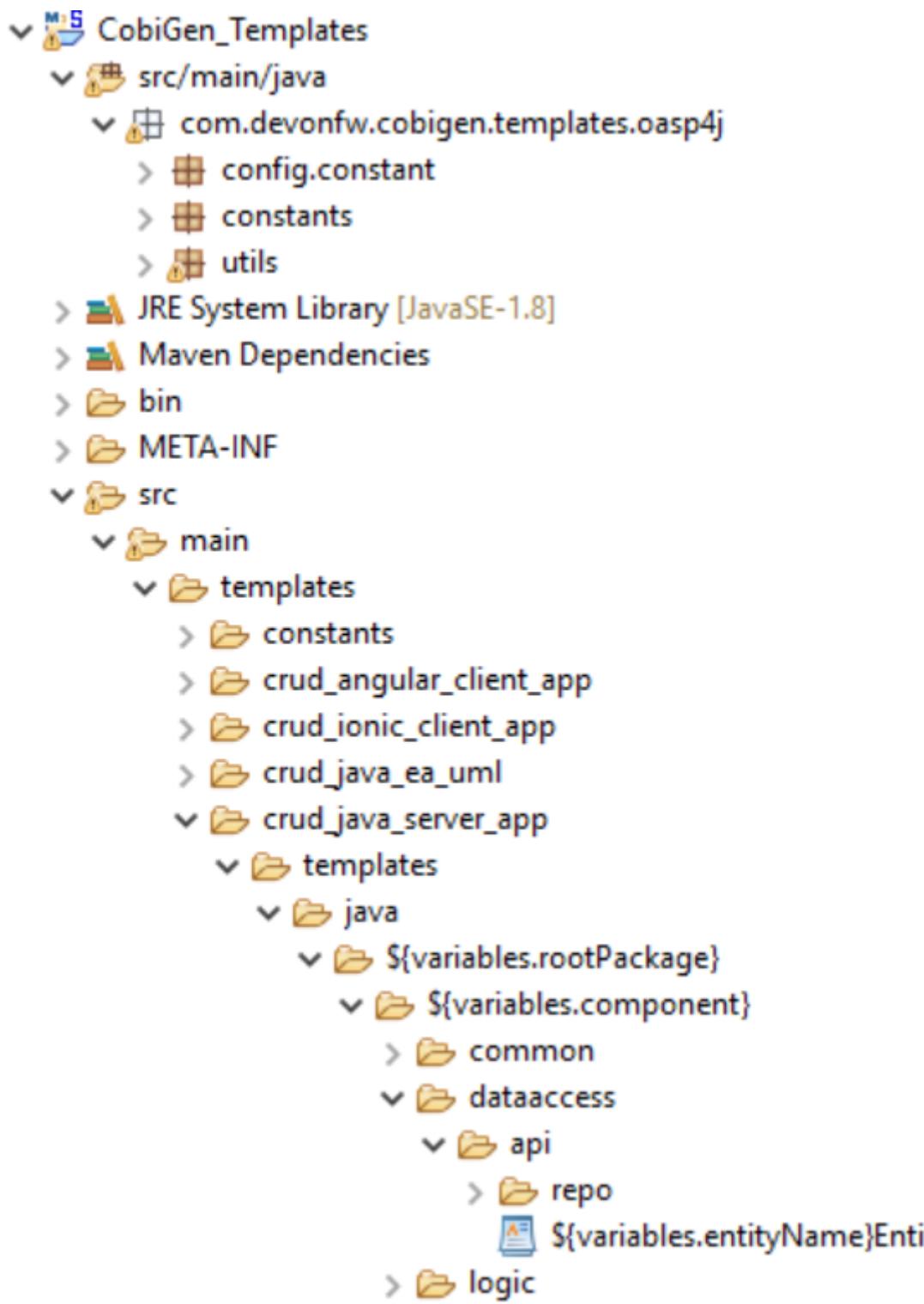
examples:

- One parameter `-DbrowserOptions="headless"`
- One parameter `-DbrowserOptions="--incognito"`
- Many parameters
`-DbrowserOptions="headless;param1=value1;testEquals=FirstEquals=SecondEquals;--testMe"`

List of options/capabilities supported by:

- Selenium Grid
- Chrome Driver

76.4.5. Run with full range of resolution



In order to execute tests in different browser resolutions, it is required to provide these resolutions as a test parameter.

Test example with resolutions included may be found in *ResolutionTest* test class

```

 ResolutionTest.java ✘
 1 package com.capgemini.ntc.selenium.tests.samples.resolutions;
 2
 3 import static org.hamcrest.CoreMatchers.*;
 4
 5 @Features("Resolution")
 6 @Category({ TestsResolution.class, TestsSelenium.class })
 7 @RunWith(ParallelParameterized.class)
 8 public class ResolutionTest extends BaseTest {
 9     private RegistryPage registryPage;
10
11     private static Object[] getResolutions() {
12         return new Object[] {
13             ResolutionEnum.w768,
14             ResolutionEnum.w960,
15             ResolutionEnum.w1920 };
16     }
17
18     @Override
19     public void setUp() {
20         registryPage = new RegistryPage();
21     }
22
23     @junitparams.Parameters(method = "getResolutions")
24     @Test
25     public void resolution_test(ResolutionEnum resolutionEnum) throws InterruptedException {
26         ResolutionUtils.setResolution(BasePage.getDriver(), resolutionEnum);
27
28         assertThat(true, is(registryPage.isButtonSubmitDisplayed()));
29         TimeUnit.SECONDS.sleep(1); //This is for demo. Do not do it at home
30
31     }
32
33     @Override
34     public void tearDown() {
35     }
36
37 }
38

```

Example of resolution notation is available in *ResolutionEnum* class

```

 ResolutionTest.java ✘ J ResolutionEnum.java ✘
 1 package com.capgemini.ntc.selenium.core.enums;
 2
 3 public enum ResolutionEnum implements IResolutionList {
 4     w320(320, 240),
 5     w480(480, 320),
 6     w568(568, 320),
 7     w768(768, 576),
 8     w960(960, 720),
 9     w1024(1024, 768),
10     w1140(1140, 760),
11     w1280(1280, 1024),
12     w1366(1366, 768),
13     w1600(1600, 1200),
14     w1800(1800, 1440),
15     w1920(1920, 1080);
16
17     private int width;
18     private int height;
19
20     private ResolutionEnum(int width, int height) {
21         this.width = width;
22         this.height = height;
23     }
24
25     public int getWidth() {
26         return width;
27     }
28
29     public int getHeight() {
30         return height;
31     }
32
33     public String toString() {
34         return "Width:" + getWidth() + " Height:" + getHeight();
35     }
36 }
37

```

Test with given resolution parameters will be launched as many times as the number of resolutions provided.

76.4.6. Selenium Best Practices

The following table displays a few best practices that should be taken into consideration when developing Selenium test cases.

Best Practices	Description
"Keep it Simple"	Do not force use every Selenium feature available - Plan before creating the actual test cases
Using Cucumber	Cucumber can be used to create initial testcases for further decision making
Supporting multiple browsers	Test on multiple browsers (in parallel, if applicable) if the application is expected to support multiple environments
Test reporting	Make use of test reporting modules like Junit which is included in the framework
Maintainability	Always be aware of the maintainability of tests - You should always be able to adapt to changes
Testing types	Which tests should be created? Rule of thumb: 70% Unit test cases, 20% Integration test cases and 10% UI Test cases
Test data	Consider before actually developing tests and choosing tools: Where to get test data from, how to reset test data

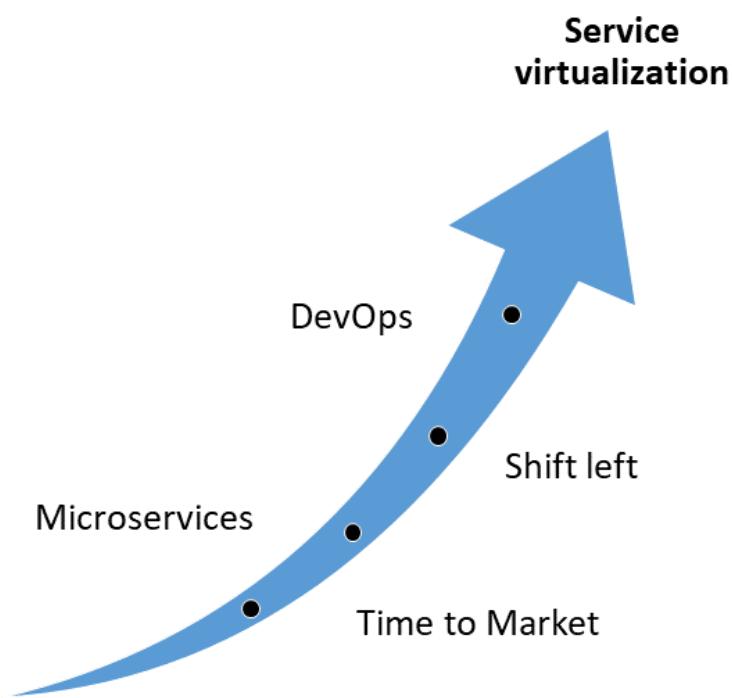
76.5. Web API Module

Service Virtualization

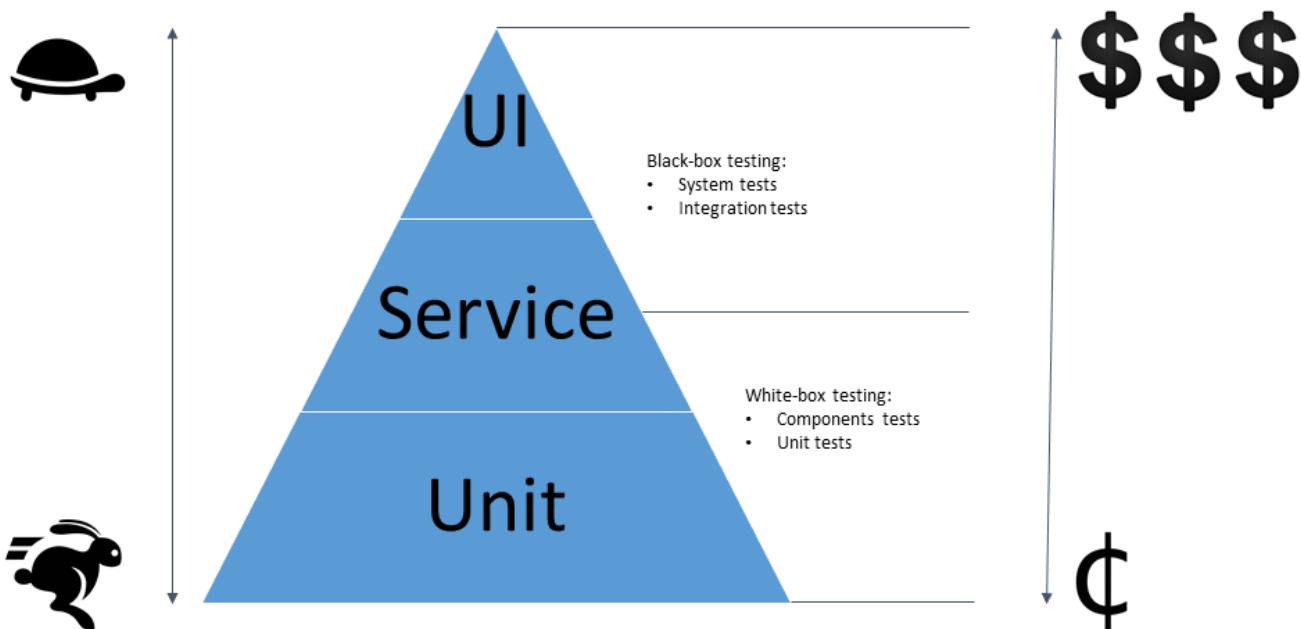
- [What is service virtualization](#)
- [How to plug in service virtualization into Application Under Test](#)
- [How to make a virtual asset](#)
- [Smoke Tests virtualization](#)

76.5.2. Is it doable to keep pace in QA with today's software agile approach?

DevOps + Microservices + Shift left + Time to Market == ? Service virtualization ?



Test pyramid



What is service virtualization

Service Virtualization has become recognized as one of the best ways to speed up testing and accelerate your time to market.

Service virtualization lets you automatically execute tests even when the application under test's dependent system components (APIs, third-party applications, etc.) cannot be properly accessed or configured for testing. By simulating these dependencies, you can ensure that your tests will encounter the appropriate dependency behaviour and data each and every time that they execute.

Service virtualization is the simulation of interfaces – not the virtualization of systems.

According to [Wikipedia's service virtualization](#) entry: *Service virtualization emulates the behaviour of software components to remove dependency constraints on development and testing teams. Such constraints occur in complex, interdependent environments when a component connected to the application under test is:*

- *Not yet completed*
- *Still evolving*
- *Controlled by a third-party or partner*
- *Available for testing only in a limited capacity or at inconvenient times*
- *Difficult to provision or configure in a test environment*

- *Needed for simultaneous access by different teams with varied test data setup and other requirements*
- *Restricted or costly to use for load and performance testing*

For instance, instead of virtualizing an entire database (and performing all associated test data management as well as setting up the database for every test session), you monitor how the application interacts with the database, then you emulate the related database behaviour (the SQL queries that are passed to the database, the corresponding result sets that are returned, and so forth).

Mocks, stubs and virtual services

The most commonly discussed categories of test doubles are mocks, stubs and virtual services.

Stub: a minimal implementation of an interface that normally returns hardcoded data that is tightly coupled to the test suite. It is most useful when the suite of tests is simple and keeping the hardcoded data in the stub is not an issue. Some stubs are handwritten; some can be generated by tools. A stub is normally written by a developer for personal use. It can be shared with testers, but wider sharing is typically limited by interoperability issues related to software platform and deployment infrastructure dependencies that were hardcoded. A common practice is when a stub works in-process directly with classes, methods, and functions for the unit, module, and acceptance testing. Some developers will say that a stub can also be primed, but you cannot verify an invocation on a stub. Stubs can also be communicating "over the wire", for example, HTTP, but some would argue that they should be called virtual services in that case.

Mock: a programmable interface observer, that verifies outputs against expectations defined by the test. It is frequently created using a third party library, for example in Java that is Mockito, JMock or WireMock. It is most useful when you have a large suite of tests and a stub will not be sufficient because each test needs a different data set up and maintaining them in a stub would be costly. The mock lets us keep the data set-up in the test. A mock is normally written by a developer for personal use but it can be shared with testers. However, wider sharing is typically limited by interoperability issues related to software platform and deployment infrastructure dependencies that were hardcoded. They are most often work-in-progress directly with classes, methods, and functions for a unit, module, and acceptance testing. Mock provides responses based on a given request satisfying predefined criteria (also called request or parameter matching). A mock also focuses on interactions rather than state so mocks are usually stateful. For example, you can verify how many times a given method was called or the order of calls made to a given object.

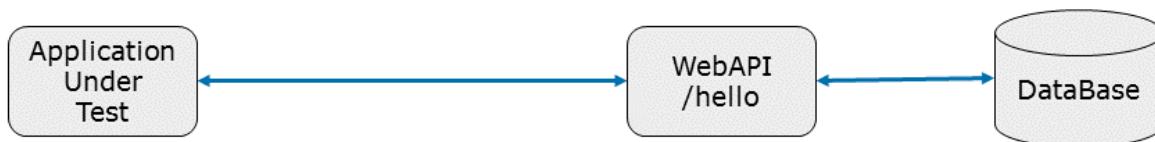
Virtual service: a test double often provided as a Software-as-a-Service (SaaS), is always called remotely, and is never working in-process directly with methods or functions. A virtual service is often created by recording traffic using one of the service virtualization platforms instead of building the interaction pattern from scratch based on interface or API documentation. A virtual service can be used to establish a common ground for teams to communicate and facilitate artefact sharing with other development teams as well as testing teams. A virtual service is called remotely (over HTTP, TCP, etc.) normally supports multiple protocols (e.g. HTTP, MQ, TCP, etc.), while a stub or mock normally supports only one. Sometimes virtual services will require users to authorize, especially when deployed in environments with enterprise-wide visibility. Service virtualization tools used to create virtual services will most often have user interfaces that allow less tech-savvy

software testers to hit the ground running, before diving into the details of how specific protocols work. They are sometimes backed by a database. They can also simulate non-functional characteristics of systems such as response times or slow connections. You can sometimes find virtual services that provide a set of stubbed responses for given request criteria and pass every other request to a live backend system (partial stubbing). Similar to mocks, virtual services can have quite complex request matchers, that allow having one response returned for many different types of requests. Sometimes, virtual services simulate system behaviours by constructing parts of the response based on request attributes and data.

It is often difficult to say definitely which of the following categories a test double fits into. They should be treated as a spectrum rather than strict definitions.

76.5.3. Plug in service virtualization

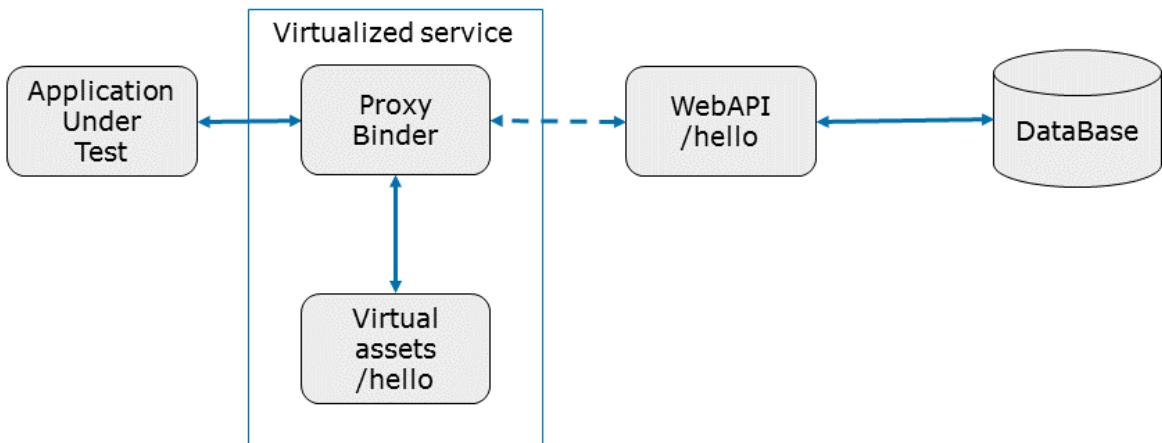
Classic application structure



This is a quite common application structure, where we have any of the following in Application Under Test (AUT):

- UI / GUI
- WebAPI
- 3rd party service

Classic application structure with virtualization



This classic application is quite fragile for development and/or test process. Especially so, if the component (WebAPI) connected to the Application Under Test is:

- Not yet completed
- Still evolving
- Controlled by a third-party or partner
- Available for testing only in limited capacity or at inconvenient times
- Difficult to provision or configure in a test environment
- Needed for simultaneous access by different teams with varied test data setup and other requirements
- Restricted or costly to use for load and performance testing

You can find the full list of such "classic application structure" limitations here [What-is-service-virtualization](#).

*Service virtualization is the key solution to address such a list of impediments. *

For simplicity, AUT connects to other components by TCP/IP protocol. Therefore AUT has an IP address and port number where given components operate. *To plug in virtualization server, the author of AUT ought to switch IP and port to "proxy server" instead of real endpoint component*

(WebAPI) . Finally, "proxy server" maps requests come from AUT with either virtual assets or real endpoint component (WebAPI). How do maps work in such a "proxy server"? Have a look here [How-to-make-virtual-asset](#)

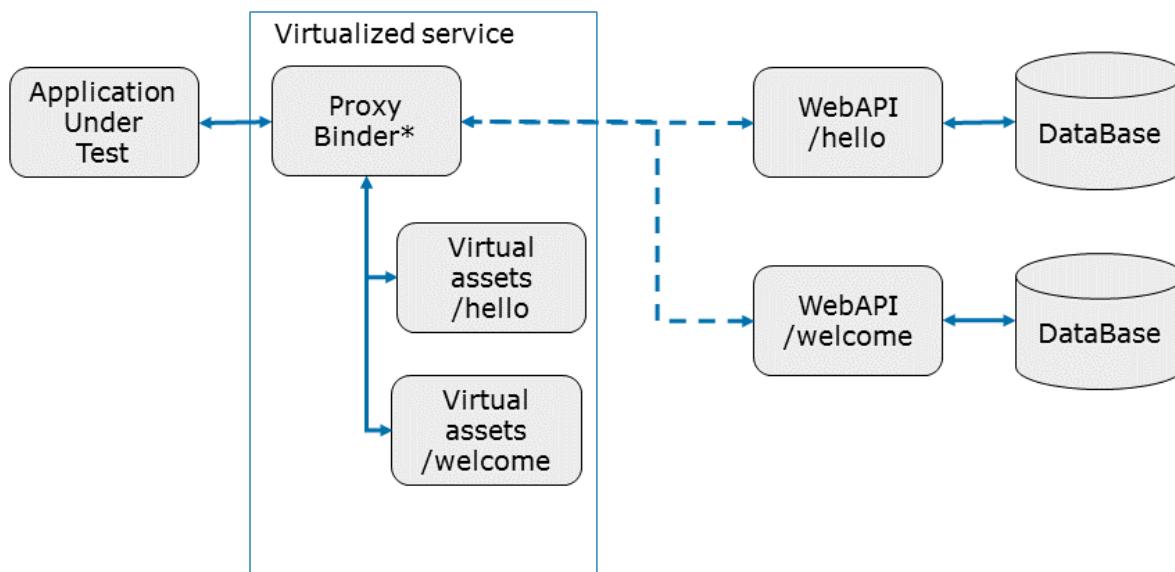
Therefore AUT is build either with:

- switchable property file acquired on startup

or

- "on the fly" operation to change IP and ports of connected components.

Classic APP structure with full scope - Binding in service virtualization



*) Bind request message with either virtual asset or real endpoint

76.5.4. How to make a virtual asset

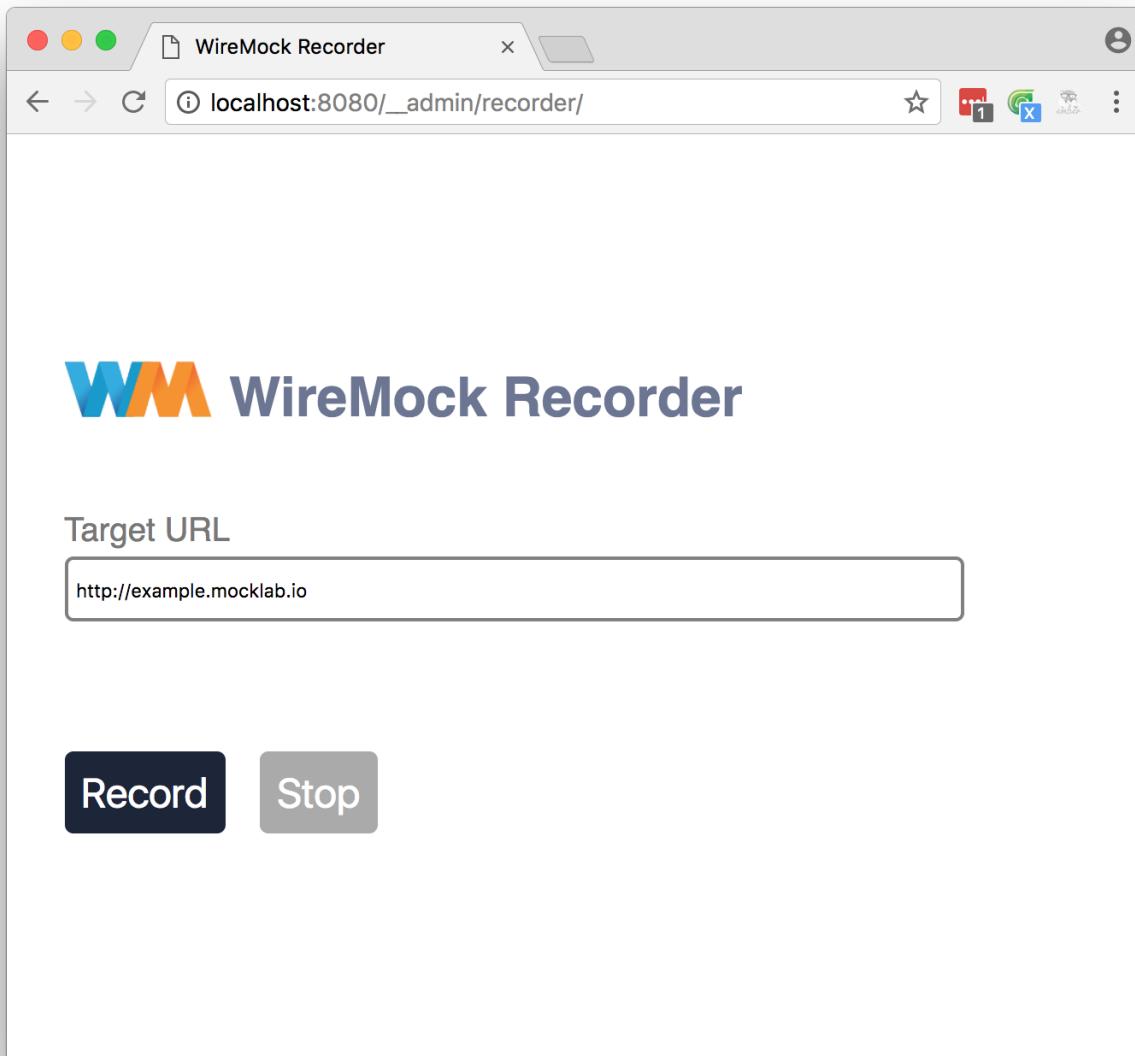
This can be done in four ways:

- Record all traffic (Mappings and Responses) that comes through proxy - by UI
- Record all traffic (Mappings and Responses) that comes through proxy - by Code
- Create Mappings and Responses manually by text files
- Create Mappings and Responses manually by code

Record all traffic (Mappings and Responses) that comes through proxy - UI

Full article here [Wiremock record-playback](#).

First, start an instance of [WireMock running standalone](#). Once that's running, visit the recorder UI page at http://localhost:8080/_admin/recorder (assuming you started WireMock on the default port of 8080).



Enter the URL you wish to record from in the target URL field and click the Record button. You can use <http://example.mocklab.io> to try it out.

Now you need to make a request through WireMock to the target API so that it can be recorded. If you're using the example URL, you can generate a request using curl:

```
$ curl http://localhost:8080/recordables/123
```

Now click stop. You should see a message indicating that one stub was captured.

You should also see that a file has been created called something like *recordables_123-40a93c4a-d378-4e07-8321-6158d5dbcb29.json* under the mappings directory created when WireMock started up, and that a new mapping has appeared at http://localhost:8080/_admin/mappings.

Requesting the same URL again (possibly disabling your wifi first if you want a firm proof) will now serve the recorded result:

```
$ curl http://localhost:8080/recordables/123

{
  "message": "Congratulations on your first recording!"
}
```

Record all traffic (Mappings and Responses) that comes through proxy - by Code

An example of how such a record can be achieved

```
@Test
public void startRecording() {

    SnapshotRecordResult recordedMappings;

    DriverManager.getDriverVirtualService()
        .start();
    DriverManager.getDriverVirtualService()
        .startRecording("http://example.mocklab.io");
    recordedMappings = DriverManager.getDriverVirtualService()
        .stopRecording();

    BFLogger.logDebug("Recorded messages: " + recordedMappings.toString());
}
```

Create Mappings and Responses manually by text files

EMPTY

Create Mappings and Responses manually by code

Link to full file structure: [REST_FarenheitToCelsiusMethod_Test.java](#)

Start up Virtual Server

```

public void startVirtualServer() {

    // Start Virtual Server
    WireMockServer driverVirtualService = DriverManager.getDriverVirtualService();

    // Get Virtual Server running http and https ports
    int httpPort = driverVirtualService.port();
    int httpsPort = driverVirtualService.httpsPort();

    // Print is Virtual server running
    BFLogger.logDebug("Is Virtual server running: " +
driverVirtualService.isRunning());

    String baseURI = "http://localhost";
    endpointBaseUri = baseURI + ":" + httpPort;
}

```

Plug in a virtual asset

[REST_FarenheitToCelsiusMethod_Test.java](#)

```

public void activateVirtualAsset() {
    /*
    * -----
    * Mock response. Map request with virtual asset from file
    * -----
    */
    BFLogger.logInfo("#1 Create Stub content message");
    BFLogger.logInfo("#2 Add resource to virtual server");
    String restResourceUrl = "/some/thing";
    String restResponseBody = "{\\"FahrenheitToCelsiusResponse\\":{\\\"FahrenheitToCelsiusResult\\":37.777777777778}}";

    new StubREST_Builder //For active virtual server ...
        .StubBuilder(restResourceUrl) //Activate mapping, for this Url AND
        .setResponse(restResponseBody) //Send this response AND
        .setStatusCode(200) // With status code 200 FINALLY
        .build(); //Set and save mapping.

}

```

Link to full file structure: [StubREST_Builder.java](#)

Source link to [How to create Stub](#).

[StubREST_Builder.java](#)

```

public class StubREST_Builder {

```

```
// required parameters
private String endpointURI;

// optional parameters
private int statusCode;

public String getEndpointURI() {
    return endpointURI;
}

public int getStatusCode() {
    return statusCode;
}

private StubREST_Builder(StubBuilder builder) {
    this.endpointURI = builder.endpointURI;
    this.statusCode = builder.statusCode;
}

// Builder Class
public static class StubBuilder {

    // required parameters
    private String endpointURI;

    // optional parameters
    private int statusCode = 200;
    private String response = "{ \"message\": \"Hello\" }";

    public StubBuilder(String endpointURI) {
        this.endpointURI = endpointURI;
    }

    public StubBuilder setStatusCode(int statusCode) {
        this.statusCode = statusCode;
        return this;
    }

    public StubBuilder setResponse(String response) {
        this.response = response;
        return this;
    }

    public StubREST_Builder build() {

        // GET
        DriverManager.getDriverVirtualService()
            .givenThat(
                // Given that request with ...
                get(urlMatching(this.endpointURI))
    }
}
```

```
        .withHeader("Content-Type",
equalTo.ContentType.JSON.toString()))
            // Return given response ...
            .willReturn(aResponse()
                .withStatus(this.statusCode)
                .withHeader("Content-Type",
ContentType.JSON.toString())
                    .withBody(this.response)
                    .withTransformers("body-transformer")));
    }

    // POST
    DriverManager.getDriverVirtualService()
        .givenThat(
            // Given that request with ...
            post(urlMatching(this.endpointURI))
                .withHeader("Content-Type",
equalTo.ContentType.JSON.toString()))
            // Return given response ...
            .willReturn(aResponse()
                .withStatus(this.statusCode)
                .withHeader("Content-Type",
ContentType.JSON.toString())
                    .withBody(this.response)
                    .withTransformers("body-transformer")));

    // PUT
    DriverManager.getDriverVirtualService()
        .givenThat(
            // Given that request with ...
            put(urlMatching(this.endpointURI))
                .withHeader("Content-Type",
equalTo.ContentType.JSON.toString()))
            // Return given response ...
            .willReturn(aResponse()
                .withStatus(this.statusCode)
                .withHeader("Content-Type",
ContentType.JSON.toString())
                    .withBody(this.response)
                    .withTransformers("body-transformer")));

    // DELETE
    DriverManager.getDriverVirtualService()
        .givenThat(
            // Given that request with ...
            delete(urlMatching(this.endpointURI))
                .withHeader("Content-Type",
equalTo.ContentType.JSON.toString()))
            // Return given response ...
            .willReturn(aResponse()
                .withStatus(this.statusCode)
                .withHeader("Content-Type",
```

```

ContentType.JSON.toString())
    .withBody(this.response)
    .withTransformers("body-transformer")));

// CATCH any other requests
DriverManager.getDriverVirtualService()
    .givenThat(
        any(anyUrl())
            .atPriority(10)
            .willReturn(aResponse()
                .withStatus(404)
                .withHeader("Content-Type",
ContentType.JSON.toString())

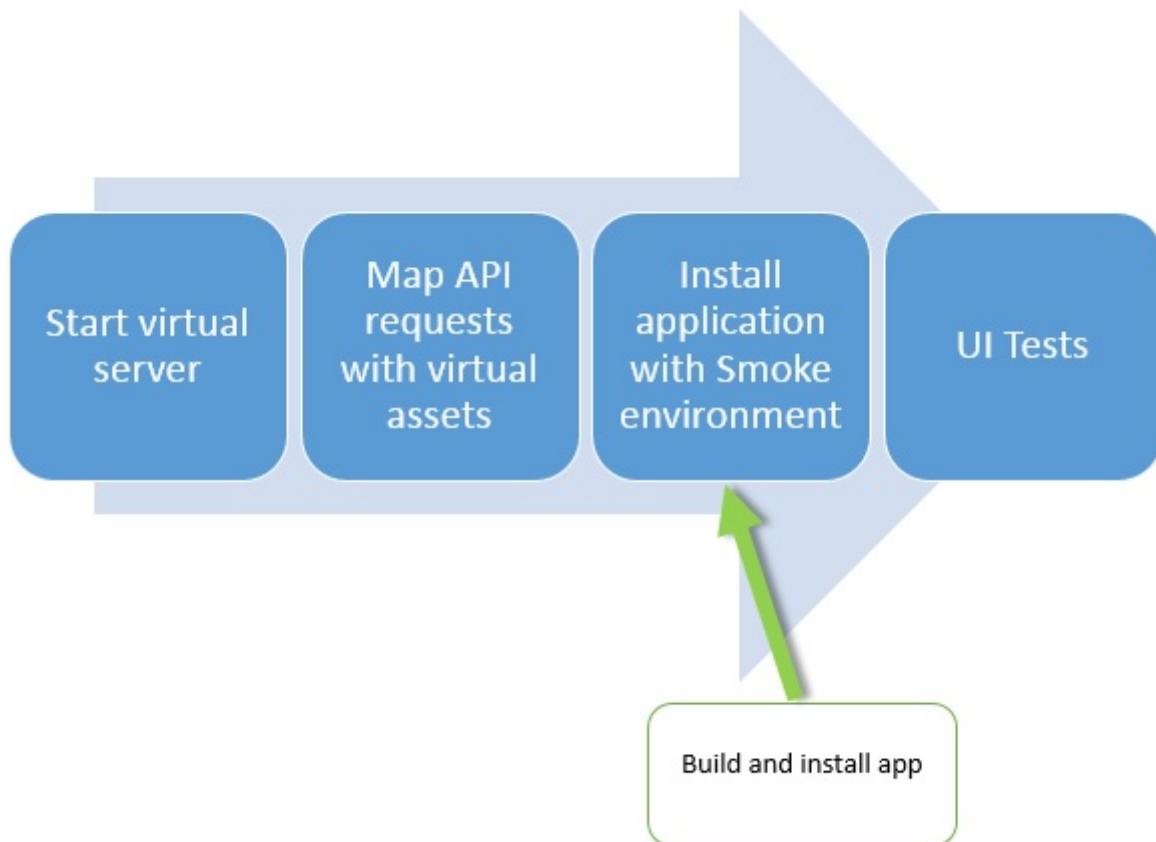
.withBody("{\"status\":\"Error\",\"message\":\"Endpoint not found\"}")
    .withTransformers("body-transformer")));

    return new StubREST_Builder(this);
}
}
}

```

76.5.5. Start a virtual server

The following picture presents the process of executing Smoke Tests in a virtualized environment:



Install docker service

If docker is not already installed on machine (this should be checked during C2C creation), install docker, docker-compose, apache2-utils, openssl (You can use **script** to install docker & docker-compose OR refer to this **post** and add Alias for this machine <C2C_Alias_Name>):

- run the script
- sudo apt-get install -y apache2-utils

Build a docker image

Dockerfile:

```
FROM docker.xxx.com/ubuntu:16.04
MAINTAINER Maintainer Name "maintainer@email.address"
LABEL name=ubuntu_java \
      version=v1-8.0 \
      base="ubuntu:16.04" \
      build_date="03-22-2018" \
      java="1.8.0_162" \
      wiremock="2.14.0" \
      description="Docker to use with Ubuntu, JAVA and WIREMOCK"

# Update and install the applications needed
COPY 80proxy /etc/apt/apt.conf.d/80proxy
RUN apt-get update
RUN apt-get install -y \
    wget \
    libfontconfig \
    unzip \
    zip \
    ksh \
    curl \
    git

COPY wgetrc /etc/wgetrc

#Env parameters

### JAVA PART ###

#TO UPDATE:please verify url link to JDK
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
##Download and install JAVA JDK8
RUN mkdir /opt/jdk
RUN wget -qq --header "Cookie: oraclelicense=accept-securebackup-cookie"
http://download.oracle.com/otn-pub/java/jdk/8u162-
b12/0da788060d494f509bf8624735fa2f1/jdk-8u162-linux-x64.tar.gz && tar -zxf jdk-8u162-
linux-x64.tar.gz -C /opt/jdk && rm jdk-8u162-linux-x64.tar.gz && update-alternatives
--install /usr/bin/javac javac /opt/jdk/jdk1.8.0_162/bin/javac 100 && java -version &&
chmod 755 -R /opt/jdk/jdk1.8.0_162/
```

```

RUN java -version

##Add user
RUN useradd -u 29001 -g 100 svrpwiredev

##Add app
RUN mkdir -p -m 777 /app
COPY wiremock-standalone-2.14.0.jar /app/wiremock-standalone-2.14.0.jar

##Expose port
EXPOSE 8080

##Set workdir
WORKDIR /App

##Run app
CMD java -jar /app/wiremock-standalone-2.14.0.jar

```

Execute the following steps with a specified version to build a docker image and push it to the repository :

```

## Build image
sudo docker build -t docker.xxx.com/app/build/wiremock:v2.14.0.

## Push image
sudo docker login docker.xxx.com
sudo docker push docker.xxx.com/app/build/wiremock:v2.14.0.

```

Run docker image

To run a docker image, execute the following command:

```

sudo docker run -td -p 8080:8080 -v
/home/wiremock/repo/app/docker/QA/mappings:/app/mappings -v
/home/wiremock/repo/app/docker/QA/_files:/app/_files --restart always
docker.xxx.com/app/build/wiremock:v2.14.0.

```

Where:

-p - publish a container's port to the host

-v - bind mount a volume. WireMock server creates two directories under the current one: mappings and _files. It is necessary to mount directories with already created mappings and responses to make it work.

-restart always - restart policy to apply when a container exists

All of the parameters are described in: [official docker documentation](#)

76.5.6. Map requests with virtual assets

What is WireMock?

WireMock is an HTTP mock server. At its core it is a web server that can be primed to serve canned responses to particular requests (stubbing) and that captures incoming requests so that they can be checked later (verification). It also has an assortment of other useful features including record/playback of interactions with other APIs, injection of faults and delays, simulation of stateful behaviour.

Full documentation can be found under the following link: [WireMock](#)

Record / create virtual assets mappings

Record

WireMock can create stub mappings from requests it has received. Combined with its proxying feature, this allows you to "record" stub mappings from interaction with existing APIs.

Record and playback (Legacy): [documentation](#)

```
java -jar wiremock-standalone-2.16.0.jar --proxy-all="http://search.twitter.com"  
--record-mappings --verbose
```

Once it's started and request is sent to it, it will be redirected to "http://search.twitter.com" and traffic (response) is saved to files in mappings and __files directories for further use.

Record and playback (New): [documentation](#)

Enable mappings in a virtual server

When the WireMock server starts, it creates two directories under the current one: mappings and __files. To create a stub, it is necessary to drop a file with a .json extension under mappings.

Run docker with mounted volumes

Mappings are in a repository. It is necessary to mount directories with already created mappings and responses to make it work:

```
sudo docker run -td -p 8080:8080 -v  
/home/wiremock/repo/app/docker/QA/mappings:/app/mappings -v  
/home/wiremock/repo/app/docker/QA/__files:/app/__files --restart always  
docker.xxx.com/app/build/wiremock:v2.14.0.
```

The description of how to build and run docker is available under: [Docker run command description](#)

Recorded mappings

Recorded mappings are kept in the project repository.

Create a user and map them to docker user

To enable the connection from Jenkins to Virtual Server (C2C), it is necessary to create a user and map them to docker group user. It can be done using the following command:

```
adduser -G docker -m wiremock
```

To set the password for a wiremock user:

```
passwd wiremock
```

Create SSH private and public keys for a wiremock user

SSH keys serve as a means of identifying yourself to an SSH server using [public-key cryptography](#) and [challenge-response authentication](#). One immediate advantage this method has over traditional password is that you can be authenticated by the server without ever having to send your password over the network.

To create an SSH key, log in as wiremock (previously created user).

```
su wiremock
```

The .ssh directory is not by default created below user home directory. Therefore, it is necessary to create it:

```
mkdir ~/.ssh
```

Now we can proceed with creating an RSA key using ssh-keygen (a tool for creating new authentication key pairs for SSH):

```
ssh-keygen -t rsa
```

A key should be created under /.ssh/id_rsa Appending the public keys to authorized_keys:

```
wiremock@vc2crptXXXXXXXX:~/ssh$ cat id_rsa.pub >> authorized_keys
```

Install an SSH key in Jenkins

To add an SSH key to Jenkins, go to credentials in your job location. Choose the folder within credentials, then 'global credentials', 'Add credentials'. Fill in the fields. Finally, the entry should be created.

Build a Jenkins Groovy script

The description of how to use SSH Agent plugin in Jenkins pipeline can be found under: <https://www.karthikeyan.tech/2017/09/ssh-agent-blue-ocean-via-jenkins.html>

Example of use:

```
sshagent (credentials: [env.WIREMOCK_CREDENTIALS]) {
    sh """
        ssh -T -o StrictHostKeyChecking=no -l ${env.WIREMOCK_USERNAME}
        ${env.WIREMOCK_IP_ADDRESS} "docker container restart ${env.WIREMOCK_CONTAINER_NAME}"
    """
}
```

Where: env.WIREMOCK_CREDENTIALS is a credential id of previously created wiremock credentials. Now that it is present, we can execute commands on a remote machine, where in ssh command: env.WIREMOCK_USERNAME - user name of user connected with configured private key env.WIREMOCK_IP_ADDRESS - ip address of the machine where this user with this private key exists

Pull repository with virtual assets

To pull the repository on a remote machine, it is necessary to use the previously described SSH Agent plugin. An example of use:

```
sshagent (credentials: [env.WIREMOCK_CREDENTIALS]) {
    withCredentials([usernamePassword(credentialsId: env.STASH_CREDENTIALS,
        passwordVariable: 'PASS', usernameVariable: 'USER')]) {
        sh """
            ssh -T -o StrictHostKeyChecking=no -l ${env.WIREMOCK_USERNAME}
            ${env.WIREMOCK_IP_ADDRESS} "cd
            ~/${env.APPLICATION_DIRECTORY_WIREMOCK}/${env.PROJECT_HOME}; git fetch
            https://$USER:$PASS@$env.GIT_WITHOUT_HTTPS ${env.GIT_BRANCH}; git reset --hard
            FETCH_HEAD; git clean -df"
        """
    }
}
```

Where:

withCredentials allows various kinds of credentials (secrets) to be used in idiosyncratic ways. Each binding will define an environment variable active within the scope of the step. Then the necessary commands are executed:

cd ... - command will change from current directory to the specified directory with git repository

git fetch ... ;**git reset** ... ;**git clean** ... - pull from GIT branch. Git pull or checkout are not used here to prevent the situation with wrong coding between Mac OSX/Linux etc.

PLEASE remember that when using this script for the first time, the code from previous block should be changed to:

```
stage("ssh-agent"){
    sshagent (credentials: [env.WIREMOCK_CREDENTIALS]) {
        withCredentials([usernamePassword(credentialsId: env.STASH_CREDENTIALS,
passwordVariable: 'PASS', usernameVariable: 'USER')]) {
            sh """
                ssh -T -o StrictHostKeyChecking=no -l ${env.WIREMOCK_USERNAME}
${env.WIREMOCK_IP_ADDRESS} "cd ~/${env.APPLICATION_DIRECTORY_WIREMOCK} ;git clone
--depth=1 --branch=develop https://$USER:$PASS@${env.GIT_WITHOUT_HTTPS}";
"""
        }
    }
}
```

76.5.7. Install an application with Smoke environment

Update properties settings file

New settings file is pushed to the repository. Example configuration:

```
...
<key>autocomplete</key>
<string>http://server:port</string>
<key>benefitsummary</key>
<string>http://server:port</string>
<key>checkscan</key>
<string>http://server:port</string>
<key>dpesb</key>
<string>http://server:port</string>
...
...
```

Address of service (backend) should be changed to wiremock address as it is shown on listing to change the default route.

Build an application with updated properties file

New versions of application are prepared by Jenkins job.

Install an application on target properties file

Installation of an application is actually executed in a non-automated way using SeeTest environment.

76.5.8. UI tests

Run Jenkins job

Jenkinsfile:

```
// Jenkins parameters are overriding the properties below
def properties = [
    JENKINS_LABELS : 'PWI_LINUX_DEV',
    APPLICATION_FOLDER : 'app_dir',
    PROJECT_HOME : 'app_home_folder',

    //WIREMOCK
    WIREMOCK_CREDENTIALS : 'vc2crptXXXXXXn',
    WIREMOCK_USERNAME : 'wiremock',
    WIREMOCK_ADDRESS : '',
    'http://vc2crptXXXXXXn.xxx.com:8080',
    WIREMOCK_IP_ADDRESS : '10.196.67.XXX',
    WIREMOCK_CONTAINER_NAME : 'wiremock',
    APPLICATION_DIRECTORY_WIREMOCK : 'repo',

    //GIT
    GIT_CREDENTIALS : 'e47742cc-bb66-4321-2341-a2342er24f2',
    GIT_BRANCH : 'develop',
    GIT_SSH : '',
    'ssh://git@stash.xxx.com/app/app.git'
    GIT_HTTPS : '',
    'HTTPS://git@stash.xxx.com/app/app.git',
    STASH_CREDENTIALS : 'e47742cc-bb66-4321-2341-a2342er24f2',

    //DOCKER
    ARTIFACTORY_USER_CREDENTIALS : 'e47742cc-bb66-4321-2341-a2342er24f2',
    SEETEST_DOCKER_IMAGE : '',
    'docker.xxx.com/project/images/app:v1-8.3',
    SEETEST_DOCKER_IMAGE : '',
    SEETEST_APPLICATION_FOLDER : 'seetest_dir',
    SEETEST_PROJECT_HOME : 'Automated Scripts',
    SEETEST_GIT_SSH : '',
    'ssh://git@stash.xxx.com/pr/seetest_automation_cucumber.git',
    SEETEST_GIT_BRANCH : 'develop',
    SEETEST_GRID_USER_CREDENTIALS : 'e47742cc-bb66-4321-2341-a2342er24f2',
    SEETEST_CUCUMBER_TAG : '@Virtualization',
    SEETEST_CLOUD_NAME : 'Core Group',
    SEETEST_IOS_VERSION : '11',
    SEETEST_IOS_APP_URL : ''
]
```

```

SEETEST_INSTALL_APP : 'No',
SEETEST_APP_ENVIRONMENT : 'SmokeTests',
SEETEST_DEVICE_QUERY : '',
]

node(properties.JENKINS_LABELS) {
    try {
        prepareEnv(properties)
        gitCheckout()
        stageStartVirtualServer()
        stageMapApiRequests()
        stageInstallApplication()
        stageUITests()
    } catch(Exception ex) {
        currentBuild.result = 'FAILURE'
        error = 'Error' + ex
    }
}

//=====END OF
PIPELINE=====

private void prepareEnv(properties) {
    cleanWorkspace()
    overrideProperties(properties)
    setWorkspace()
}

private void gitCheckout() {
    dir(env.APPLICATION_FOLDER) {
        checkout([$class: 'GitSCM', branches: [[Web-API-Test-Module-Smoke-Tests-virtualization.asciidoc_name: env.GIT_BRANCH]], doGenerateSubmoduleConfiguration: false, extensions: [[Web-API-Test-Module-Smoke-Tests-virtualization.asciidoc_$class: 'CloneOption', depth: 0, noTags: false, reference: '', shallow: false, timeout: 50]], gitTool: 'Default', submoduleCfg: [], userRemoteConfigs: [[Web-API-Test-Module-Smoke-Tests-virtualization.asciidoc_credentialsId: env.GIT_CREDENTIALS, url: env.GIT_SSH]]])
    }
}

private void stageStartVirtualServer() {
    def module = load "${env.SUBMODULES_DIR}/stageStartVirtualServer.groovy"
    module()
}

private void stageMapApiRequests() {
    def module = load "${env.SUBMODULES_DIR}/stageMapApiRequests.groovy"
    module()
}

private void stageInstallApplication() {
    def module = load "${env.SUBMODULES_DIR}/stageInstallApplication.groovy"
}

```

```

    module()
}

private void stageUITests() {
    def module = load "${env.SUBMODULES_DIR}/stageUITests.groovy"
    module()
}

private void setWorkspace() {
    String workspace = pwd()
    env.APPLICATION_DIRECTORY = "/${env.APPLICATION_DIRECTORY}"
    env.WORKSPACE_LOCAL - workspace + env.APPLICATION_DIRECTORY
    env.SEETEST_PROJECT_HOME_ABSOLUTE_PATH =
"${workspace}/${env.SEETEST_APPLICATION_FOLDER}/${env.SEETEST_PROJECT_HOME}"
    env.SUBMODULES_DIR = env.WORKSPACE_LOCAL + "/pipelines/SmokeTests.submodules"
    env.COMMONS_DIR      = env.WORKSPACE_LOCAL + "/pipelines/commons"
}

/*
    function overrides env values based on provided properties
*/
private void overrideProperties(properties) {
    for (param in properties) {
        if (env.(param.key) == null) {
            echo "Adding parameter '${param.key}' with default value: '$param.value'"
            env.(param.key) = param.value
        } else {
            echo "Parameter '${param.key}' has overridden value: '${env.(param.key)}'"
        }
    }

    echo sh(script: "env | sort", returnStdout: true)
}

private void cleanWorkspace() {
    sh 'rm-rf *'
}

```

stageStartVirtualServer.groovy:

```

def call () {
    stage("Check virtual server") {
        def statusCode

        try {
            def response = httpRequest "${env.WIREMOCK_ADDRESS}/__admin/"
            statusCode = response.status
        } catch(Exception ex) {
            currentBuild.result = 'FAILURE'
            error 'WireMock server is unreachable.'
        }

        if(statusCode !=200) {
            currentBuild.result = 'FAILURE'
            error 'WireMock server is unreachable. Return code: ${statusCode}'
        }
    }
}

```

stageMapApiRequests.groovy:

```

def call() {
    stage("Map API requests with virtual assets") {
        checkoutRepository()
        restartWiremock()
        checkWiremockStatus()
    }
}

private checkoutRepository() {
    extractHTTPSUrl()
    sshagent (credentials: [env.WIREMOCK_CREDENTIALS]) {
        withCredentials([usernamePassword(credentialsId: env.STASH_CREDENTIALS,
passwordVariable: 'PASS', usernameVariable: 'USER')]) {
            sh """
                ssh -T -o StrictHostKeyChecking=no -l ${env.WIREMOCK_USERNAME}
${env.WIREMOCK_IP_ADDRESS}
"cd~/${env.APPLICATION_DIRECTORY_WIREMOCK}/${env.PROJECT_HOME}; git fetch
https://$USER:$PASS@${env.GIT_WITHOUT_HTTPS} ${env.GIT_BRANCH}; git reset --hard
FETCH_HEAD; git clean -df"
"""
        }
    }
}

private restartWiremock() {
    sshagent (credentials: [env.WIREMOCK_CREDENTIALS]) {
        sh """
            ssh -T -o StrictHostKeyChecking=no -l ${env.WIREMOCK_USERNAME}

```

```

${env.WIREMOCK_IP_ADDRESS} "docker container restart ${env.WIREMOCK_CONTAINER_NAME}"
"""
}

private checkWiremockStatus() {
    int wiremockStatusCheckCounter =6
    int sleepTimeInSeconds = 10
    def wiremockStatus

    for (i = 0; i < wiremockStatusCheckCounter; i++) {
        try {
            wiremockStatus = getHttpRequestStatus()
            echo "WireMock server status code: ${wiremockStatus}"
        } catch(Exceprion ex) {
            echo "Exception when checking connection to WireMock"
        }
        if(wiremockStatus == 200) break
        else sh "sleep ${sleepTimeInSeconds}"
    }

    if(wiremockStatus != 200) {
        currentBuild.result = 'FAILURE'
        error 'WireMock server is unreachable. Return code: ${wiremockStatus}'
    }
}

private def getHttpRequestStatus() {
    def response = httpRequest "${env.WIREMOCK_ADDRESS}/__admin"
    return response.status

private extractHTTPSUrl() {
    env.GIT_WITHOUT_HTTPS = env.GIT_HTTPS.replace("https://", "")
}

return this

```

stageInstallApplication.groovy:

```

def call() {
    stage('Install application with smoke tests environment') {
        dir(env.SEETEST_APPLICATION_FOLDER) {
            checkout([$class: 'GitSCM', branches: [[Web-API-Test-Module-Smoke-Tests-
virtualization.asciidoc_name: env.SEETEST_GIT_BRANCH]], 
doGenerateSubmoduleConfigurations: false, extensions: [], gitTool: 'default',
submoduleCfg: [], userRemoteConfigs: [[Web-API-Test-Module-Smoke-Tests-
virtualization.asciidoc_credentialsId: env.GIT_CREDENTIALS, url:
env.SEETEST_GIT_SSH]]])
        }
    }
}

return this

```

stageUITests.groovy:

```

def call() {
    stage('UI tests') {
        def utils = load "${env.SUBMODULES_DIR}/utils.groovy"

        try {
            utils.generateUserIDVariable(); //Generate USER_ID and USER_GROUP
            docker.image(env.SEETEST_DOCKER_IMAGE).inside("-u
${env.USER_ID}:${env.USER_GROUP}") {
                withCredentials([[Web-API-Test-Module-Smoke-Tests-
virtualization.asciidoc_$class: 'UsernamePasswordMultiBinding', credentialsId:
"${env.ARTIFACTORY_USER_CREDENTIALS}", passwordVariable: 'ARTIFACTORY_PASSWORD',
usernameVariable: 'ARTIFACTORY_USERNAME']]) {
                    executeTests()
                    compressArtifacts()
                    publishJUnitResultReport()
                    archiveArtifacts()
                    publishHTMLReports()
                    publishCucumberReports()
                }
            }
        } catch (Exception exc) {
            throw exc
        }
    }
}

private executeTests() {
    withCredentials([usernamePassword(credentialsId:
env.SEETEST_GRID_USER_CREDENTIALS, passwordVariable: 'GRID_USER_PASSWORD',
usernameVariable: 'GRID_USER_NAME')]) {
        sh """
        cd ${env.SEETEST_PROJECT_HOME_ABSOLUTE_PATH}

```

```

        mvn clean test -B -Ddriver="grid" -Dtags="${env.SEETEST_CUCUMBER_TAG}"
-DcloudName="${env.SEETEST_CLOUD_NAME}" -DdeviceQuery="${env.SEETEST_DEVICE_QUERY}"
-DgridUser="${GRID_USER_NAME}" -DgridPassword="${GRID_USER_PASSWORD}"
-Dinstall="${env.SEETEST_INSTALL_APP}" -DiosUrl="${env.SEETEST_IOS_APP_URL}"
-DdeviceType="iPhone" -DiosVersion="${env.SEETEST_IOS_VERSION}"
-DparallelMode="allOnAll" -Denv="${env.SEETEST_APP_ENVIRONMENT}" site
"""

}

}

private compressArtifacts() {
    echo "Compressing artifacts from /target/site"
    sh """
        zip -r allure_report.zip **/${env.SEETEST_PROJECT_HOME}/target/site
"""
}

private publishJUnitResultReport() {
    echo "Publishing JUnit reports from
${env.SEETEST_APPLICATION_FOLDER}/${env.SEETEST_PROJECT_HOME}/target/surefire-
reports/junitreporters/*.xml"

    try {
        junit
        "${env.SEETEST_APPLICATION_FOLDER}/${env.SEETEST_PROJECT_HOME}/target/surefire-
reports/junitreporters/*.xml"
    } catch(e) {
        echo("No JUnit report found")
    }
}

private archiveArtifacts() {
    echo "Archiving artifacts"

    try {
        archiveArtifacts allowEmptyArchive: true, artifacts: "**/allure_report.zip"
    } catch(e) {
        echo("No artifacts found")
    }
}

private publishHTMLReports() {
    echo "Publishing HTML reports from
${env.SEETEST_APPLICATION_FOLDER}/${env.SEETEST_PROJECT_HOME}/target/site/allure-
maven-plugin"

    try {
        publishHTML([allowMissing: false, alwaysLinkToLastBuild: true, keepAll: true,
reportDir:
"${env.SEETEST_APPLICATION_FOLDER}/${env.SEETEST_PROJECT_HOME}/target/site/allure-
maven-plugin", reportFiles: 'index.html', reportName: 'Allure report', reportTitles:
'Allure report'])
    }
}

```

```
    } catch(e) {
        echo("No artifacts found")
    }
}

private publishCucumberREPORTS() {
    echo "Publishing Cucumber reports from
${env.SEETEST_APPLICATION_FOLDER}/${env.SEETEST_PROJECT_HOME}/target/cucumber-
parallel/*.json"

    try {
        step([$class: 'CucumberReportPublisher', fileExcludePattern '',
fileIncludePattern:
"${env.SEETEST_APPLICATION_FOLDER}/${env.SEETEST_PROJECT_HOME}/target/cucumber-
parallel/*.json", ignoreFailedTests: false, jenkinsBasePath: '', jsonReportDirectory:
'', missingFails: false, parallelTesting: false, pendingFails: false, skippedFails:
false, undefinedFails: false])
    } catch(e) {
        echo("No Cucumber report found")
    }
}

return this
```

Configuration

It is possible to configure Jenkins job in two ways. First one is to edit the Jenkinsfile. All of the properties are in properties collection as below:

```

def properties = [
    JENKINS_LABELS : 'PWI_LINUX_DEV'
    ...
    //Docker
    ARTIFACTORY_USER_CREDENTIALS : 'ba2e4f46-56f1-4467-ae97-17b356d6s643',
    SEETEST_DOCKER_IMAGE : 'docker.XXX.com/app/base-images/seetest:v1-8.3',
    ...
    //SeeTest
    SEETEST_APPLICATION_FOLDER : 'seetest_dit',
    SEETEST_PROJECT_HOME : 'Automated_Scripts',
    SEETEST_GIT_SSH :
    'ssh://stash.XXX.com/app/seetest_automation_cucumber.git',
    SEETEST_GIT_BRANCH : 'develop',
    ...
]

```

Second way is to add properties in 'Configure job'. All of the properties there are overriding properties from Jenkinsfile (they have the highest priority). They can then be set during 'Build with Parameters' process.

Reports

After a job execution 'Allure report' and 'Cucumber-JVM' reports should be visible. If any tests fail, You can check on which screen (printscreens from failures are attached, why and etc.)

76.6. Security Module

76.6.1. Security Test Module

What is Security

Application Security is concerned with **Integrity**, **Availability** and **Confidentiality** of data processed, stored and transferred by the application.

Application Security is a cross-cutting concern which touches every aspect of the Software Development Lifecycle. You can introduce some SQL injection flaws in your application and make it exploitable, but you can also expose your secrets (which will have nothing to do with code itself) due to poor secret management process, and fail as well.

Because of this and many other reasons, not every aspect of security can be automatically verified. Manual tests and audits will still be needed. Nevertheless, every security requirement which is automatically verified will prevent code degeneration and misconfiguration in a continuous

manner.

How to test Security

Security tests can be performed in many different ways, such as:

- **Static Code Analysis** - improves the security by (usually) automated code review. A good way to search for vulnerabilities, which are 'obvious' on the code level (e.g. SQL injection). The downside of this approach is that professional tools to perform such scans are very expensive and still produce many false positives.
- **Dynamic Code Analysis** - tests are run against a working environment. A good way to search for vulnerabilities, which require all client- and server-side components to be present and running (like e.g. Cross-Site Scripting). Tests are performed in a semi-automated manner and require a proxy tool (like e.g. OWASP ZAP)
- **Unit tests** - self-written and self-maintained tests. They usually work on the HTTP/REST level (this defines the trust boundary between the client and the server) and run against a working environment. Unit tests are best suited for verifying requirements which involve business knowledge of the system or which assure secure configuration on the HTTP level.

In the current release of the Security Module, the main focus will be **Unit Tests**.

Although the most common choice of environment for running security tests on will be **integration**(the environment offers the right stability and should mirror the production closely), it is not uncommon for some security tests to run on production as well. This is done for e.g. TLS configuration testing to ensure proper configuration of the most relevant environment in a continuous manner.

76.7. Database Module

76.7.1. Database Test Module

What is MrChecker Database Test Module

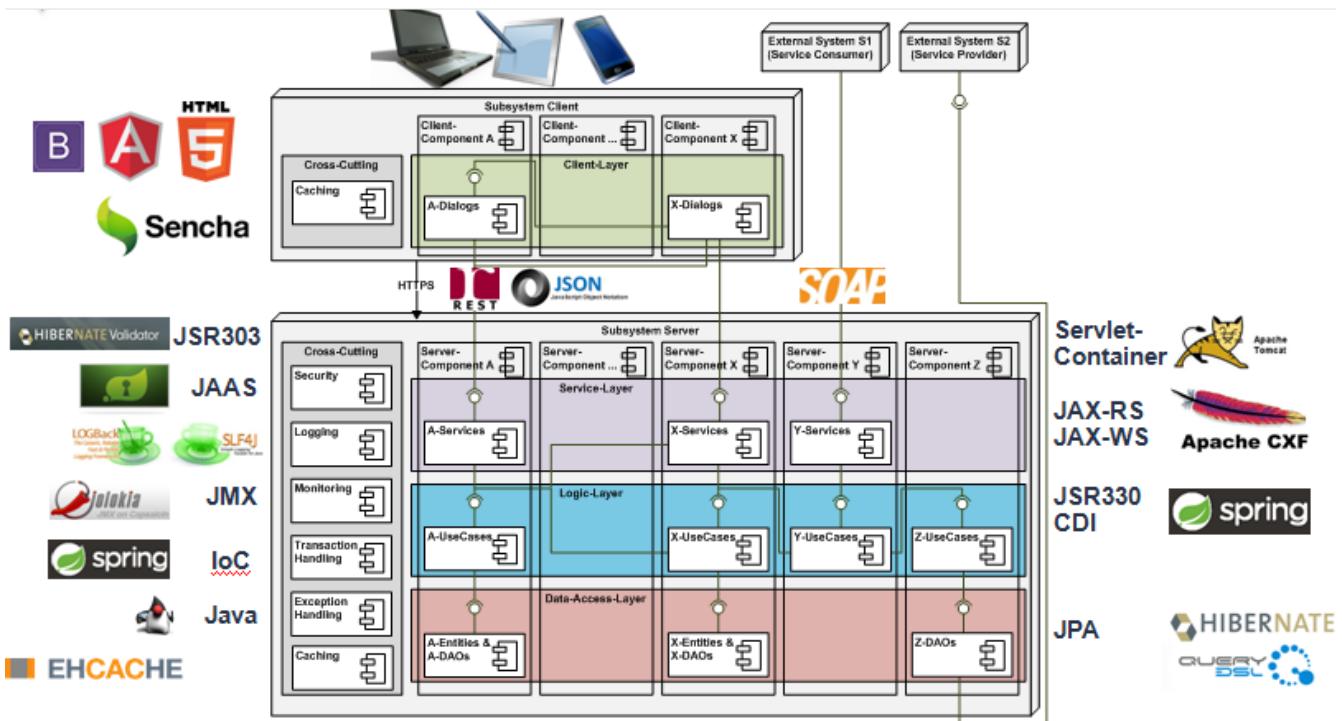
Database module is based on Object-Relational Mapping programming technique. All functionalities are built using Java Persistence API but examples use Hibernate as a main provider.

JPA structure schema

This module was written to allow the use of any JPA provider. The structure is represented in the schema below.



ORM representation applied in Framework



76.8. Mobile Test Module

76.8.1. Mobile Test Module

What is MrChecker E2E Mobile Test Module

MrChecker E2E Mobile test Module is a suitable solution for testing Remote Web Design, Mobile Browsers and application. A user can write tests suitable for all mobile browsers with a full range of resolution. The way of working is similar to Selenium and uses the same rules and patterns as the Web Driver. For more information please look in the [Selenium test module](#).

What is Page Object Architecture

Creating Selenium test cases can result in an unmaintainable project. One of the reasons is that too many duplicated code is used. Duplicated code could be caused by the duplicated functionality and this will result in duplicated usage of locators. The disadvantage of duplicated code is that the project is less maintainable. If some locator will change, you have to walk through the whole test code to adjust locators where necessary. By using the page object model we can make non-brittle test code and reduce or eliminate duplicate test code. Beside of that it improves the readability and allows us to create interactive documentation. Last but not least, we can create tests with less keystroke. An implementation of the page object model can be achieved by separating the abstraction of the test object and the test scripts.

Page Object Pattern

Page Object Pattern

Common automation pattern

Maintainable

Less amount of code

Class is a page reflection

Object is used only if test enter its page reflection

Mobile Structure

It is build on the top of the Appium library. Appium is an open-source tool for automating native, mobile web, and hybrid applications on iOS mobile, Android mobile, and Windows desktop platforms. Native apps are those written using iOS, Android, or Windows SDKs. Mobile web apps are web apps accessed using a mobile browser (Appium supports Safari on iOS and Chrome or the built-in 'Browser' app on Android). Hybrid apps have a wrapper around a "webview" - a native control that enables interaction with web content.

Run on different mobile devices

To execute each test with chosen connected mobile devices, it is required to use specific arguments in Run configuration.

Run Window Help

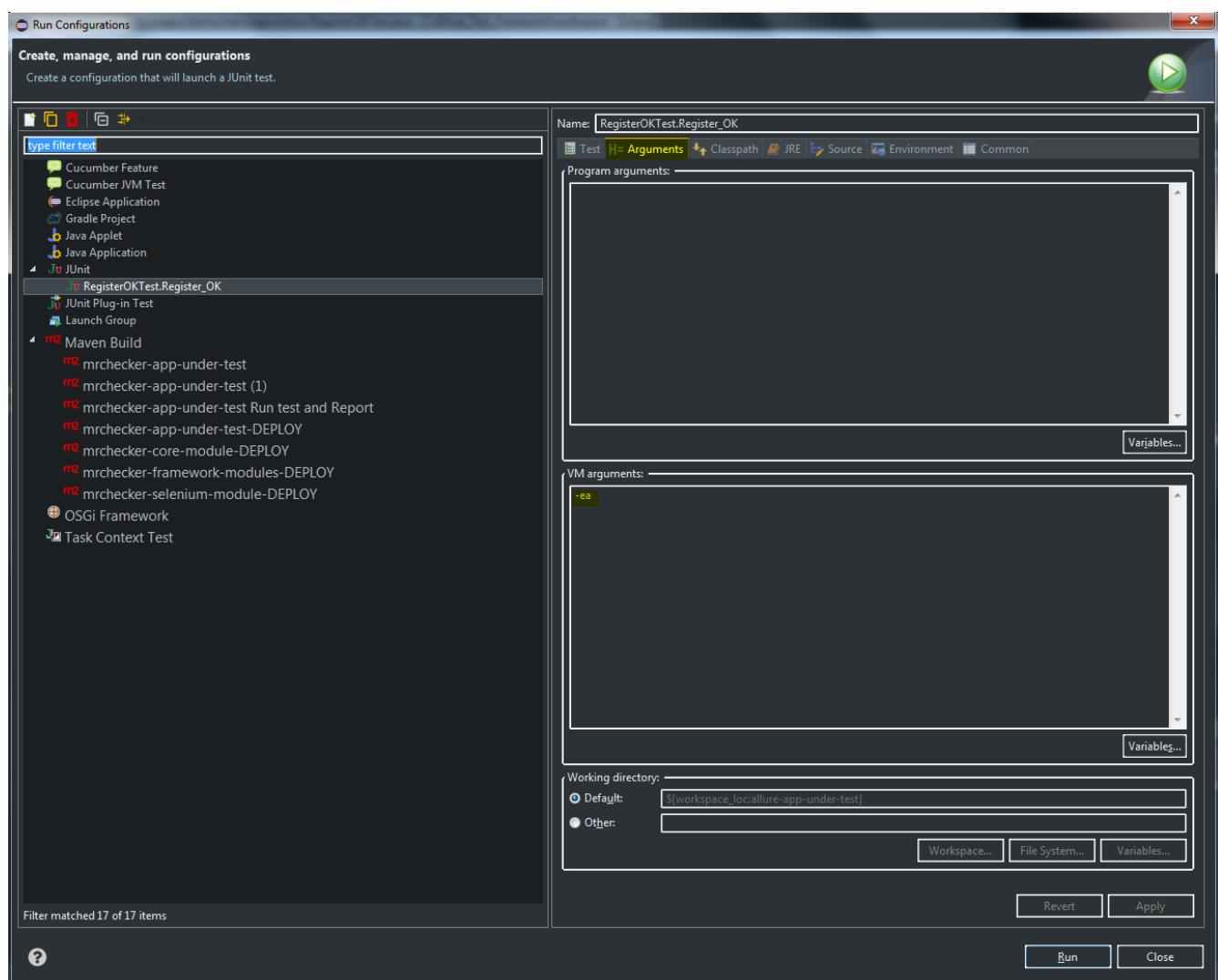
JU 1 RegisterOKTest.Register_OK
m2 2 allure-app-under-test-DEPLOY
m2 3 allure-framework-modules-DEPLOY

Run As Run Configurations... Organize Favorites...

```

34     private static String confirmPassword = password;
35
36@    @Override
37    public void setUp() {
38        registrationPage = new RegistrationPage();
39    }
40
41@    @Override
42    public void tearDown() {
43        // TODO Auto-generated method stub
44    }
45
46@    @Test
47    public void Register_OK() {
48        assertTrue("Site title: " + registrationPage.getActualPageTitle(),
49                    registrationPage.getActualPageTitle().equals(PageTitlesEnum.REGISTRATION.toSt
50

```



Default supported arguments in MrChecker:

- **deviceUrl** - http url to Appium Server, default value "http://127.0.0.1:4723"
- **automationName** - which automation engine to use , default value "Appium"

- **platformName** - which mobile OS platform to use , default value "Appium"
- **platformVersion** - mobile OS version , default value ""
- **deviceName** - the kind of mobile device or emulator to use , default value "Android Emulator"
- **app** - the absolute local path or remote http URL to a .ipa file (IOS), .app folder (IOS Simulator), .apk file (Android) or .apks file (Android App Bundle), or a .zip file, default value ":"
- **browserName** - name of mobile web browser to automate. Should be an empty string if automating an app instead, default value ""
- **newCommandTimeout** - how long (in seconds) Appium will wait for a new command from the client before assuming the client quit and ending the session, default value "4000"
- **deviceOptions** - any other capabilities not covered in essential ones, default value none

Example usage:

```
mvn clean test -Dtest=MyTest -DdeviceUrl="http://192.168.0.1:1234"
-DplatformName="iOS" -DdeviceName="iPhone Simulator" -Dapp=".\\Simple_App.ipa"
```

```
mvn clean test -Dtest=MyTest -Dapp=".\\Simple_App.apk"
-DdeviceOptions="orientation=LANDSCAPE;appActivity=MainActivity;chromeOptions=['
--disable-popup-blocking']"
```

Check also:

- + [My First Selenium Test](#)
- + [How to use Mobile test Module](#)
- + [Example TestCase](#)
- + Full list of [Generic Capabilities](#)
- + List of additional capabilities for [Android](#)
- + List of additional capabilities for [iOS](#)

76.8.2. How to use mobile test Module

1. Install [IDE with MrChecker](#)
2. Switch branch to 'feature/Create-mobile-module-#213' - by default it is 'develop'

```
git checkout feature/Create-mobile-module-#213
```

1. Install and setup git checkout feature/Create-mobile-module-#213[Appium Server]
2. Connect to local Device by Appium Server

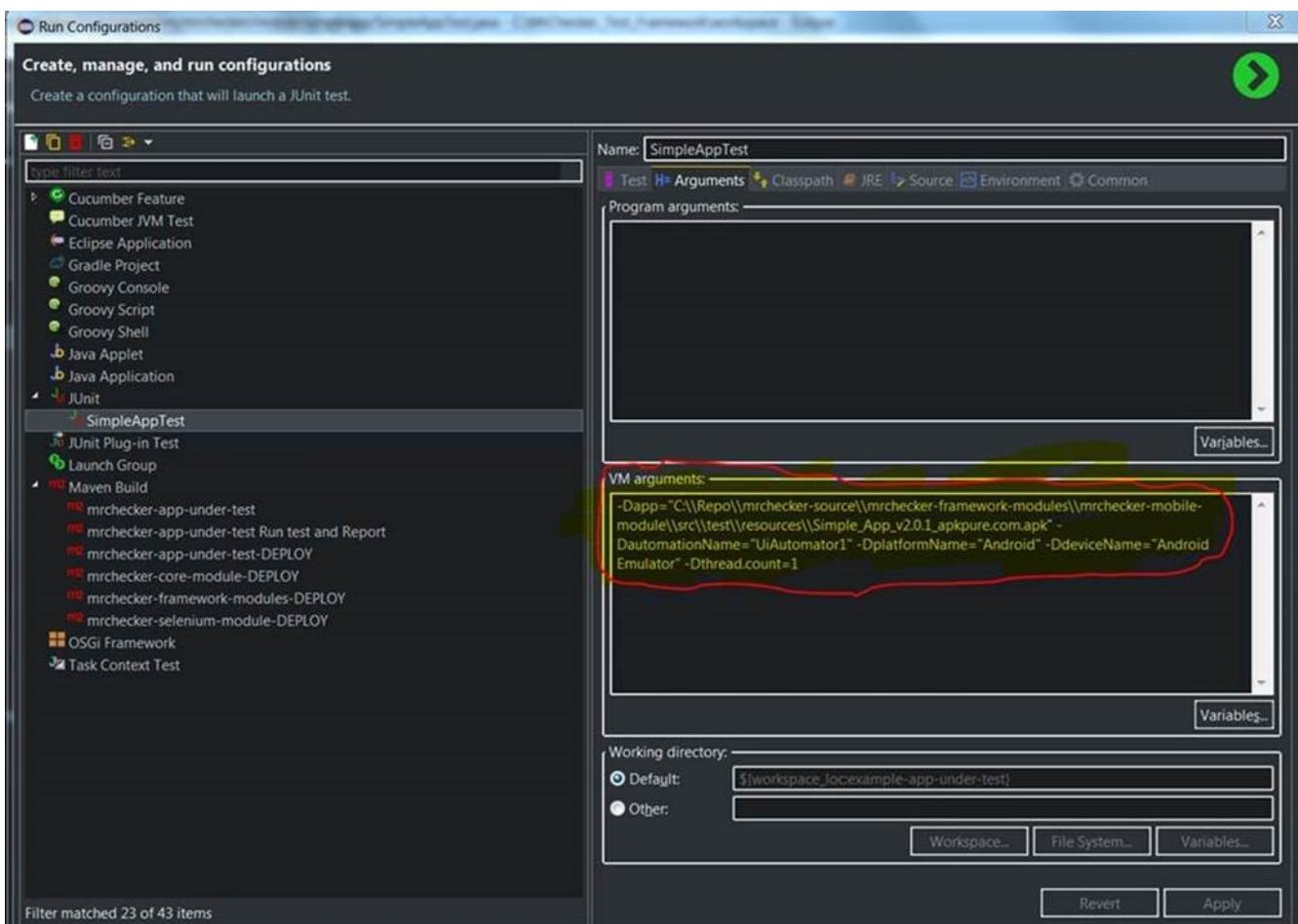
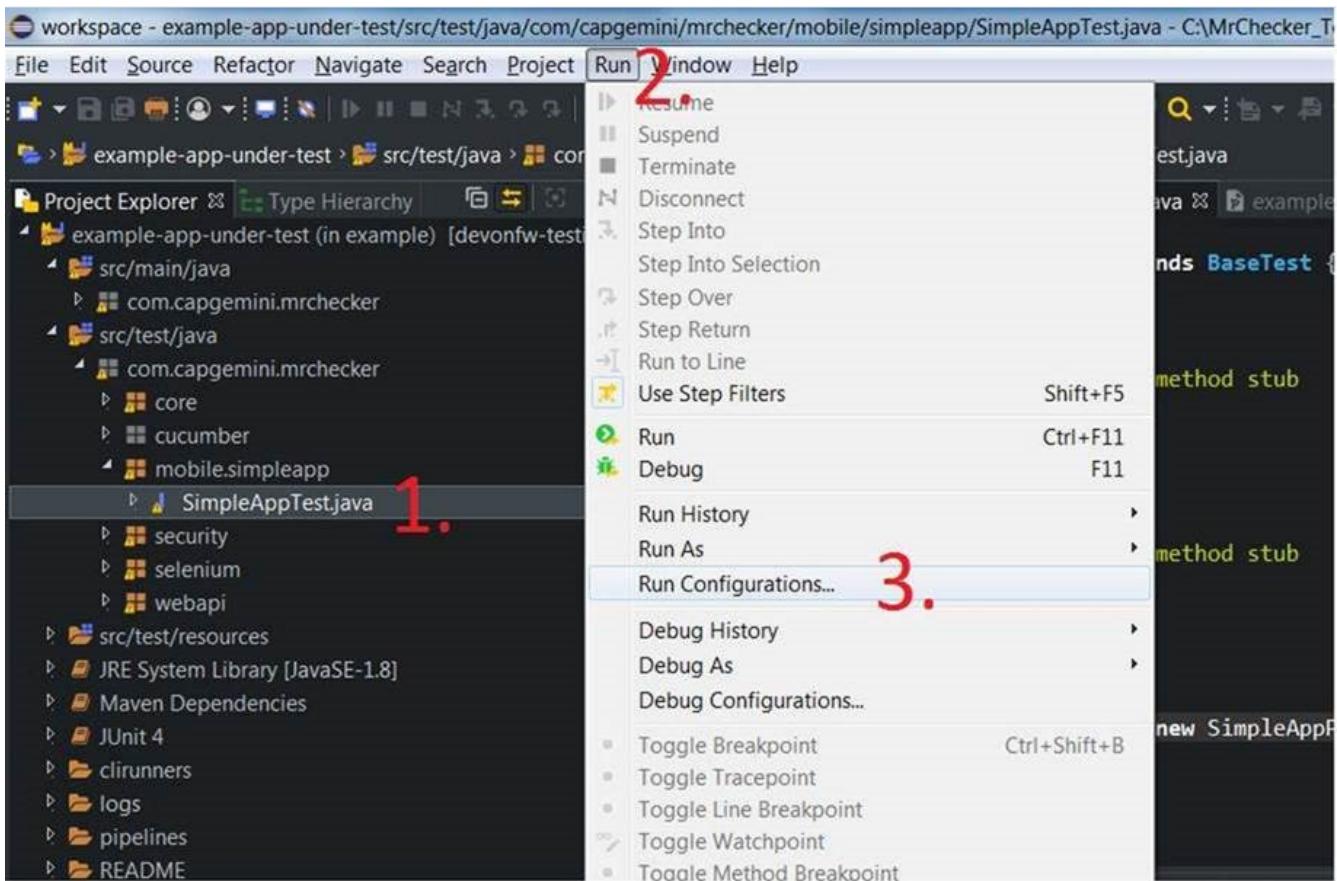
- 1.
- Install Android SDK <https://developer.android.com/studio/index.html#command-tools> ->
- 2.
- Download Platform and Build-Tools (Android versions - > https://en.wikipedia.org/wiki/Android_version_history)
 * sdkmanager "platform-tools" "platforms;android-19"
 * sdkmanager "build-tools;19.0.0"
 * copy from /build-tools file "aapt.exe" to /platform-tools
- 3.
- Set Environment:
 ANDROID_SDK_ROOT = D:\sdk-tools-windows-4333796
 PATH = %PATH%; %ANDROID_SDK_ROOT%
- 4.
- Start Appium Server
- 5.
- Start Session in Appium Server, capabilities
 {
 "platformName": "Android",
 "deviceName": "Android Emulator",
 "app": "D:\\Repo\\mrchecker-source\\mrchecker-framework-modules\\mrchecker-mobile-module\\src\\test\\resources\\SimpleApp_v2.0.1_apkpure.com.apk",
 "automationName": "UiAutomator1"
 }

3. Run Mobile tests with runtime parameters. List of supported parameters could be found [here](#)

- From command line (as in Jenkins):

```
mvn clean compile test -Dapp=".\\Simple_App_v2.0.1_apkpure.com.apk"
-DautomationName="UiAutomator1" -Dthread.count=1
```

- from IDE:



76.9. DevOps Test Module

76.9.1. DevOPS Test Module

What does DevOps mean for us?

DevOps consists of a mixture of three key components in a technical project:

- People's skills and mindset
- Processes
- Tools

Using E2E MrChecker Test Framework it is possible to cover the majority of these areas.

QA Team Goal

For QA engineers, it is essential to take care of the product code quality.

Therefore, we have to understand, that a **test case is also code which has to be validated** against quality gates. As a result, we must **test our developed test case** like it is done during standard Software Delivery Life Cycle.

Well rounded test case production process

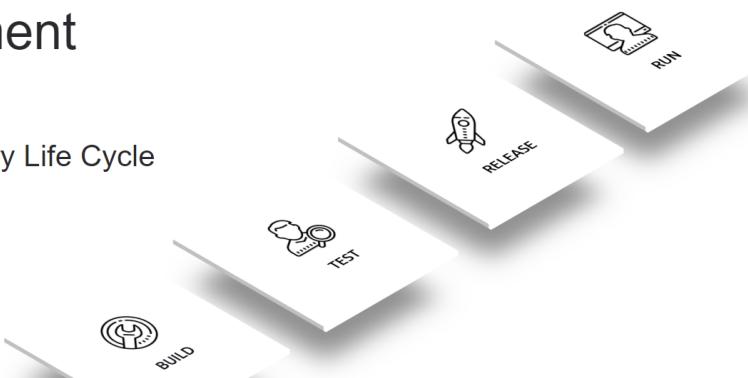
- How do we define top-notch test cases development process in **E2E MrChecker Test Framework**

The screenshot shows the top navigation bar of the devonfw website. It includes the devonfw logo, a search bar with placeholder text "Search by keyword(s)...", and four main navigation links: RESOURCES, EXPLORE, DOCS, and COMMUNITY. A small icon representing user profile or account is also present.

The standard open source software development platform

Serving the full Software Delivery Life Cycle

[GET STARTED](#) [DOWNLOAD](#)



Continuous Integration (CI) and Continuous Delivery (CD)

- **Continuous Integration (CI)** - a procedure where quality gates validate test case creation process
- **Continuous Delivery (CD)** - a procedure where we include created test cases, validated against CI, as smoke/regression/security

The Open Application Standard Platform for Java (devon4j) provides a standardized architecture blueprint, an open best-of-breed technology stack as well as industry proven best practices and code conventions for a cloud ready SPRING based server.

From developers to developers!

Architecture basics

Coding conventions

Samples

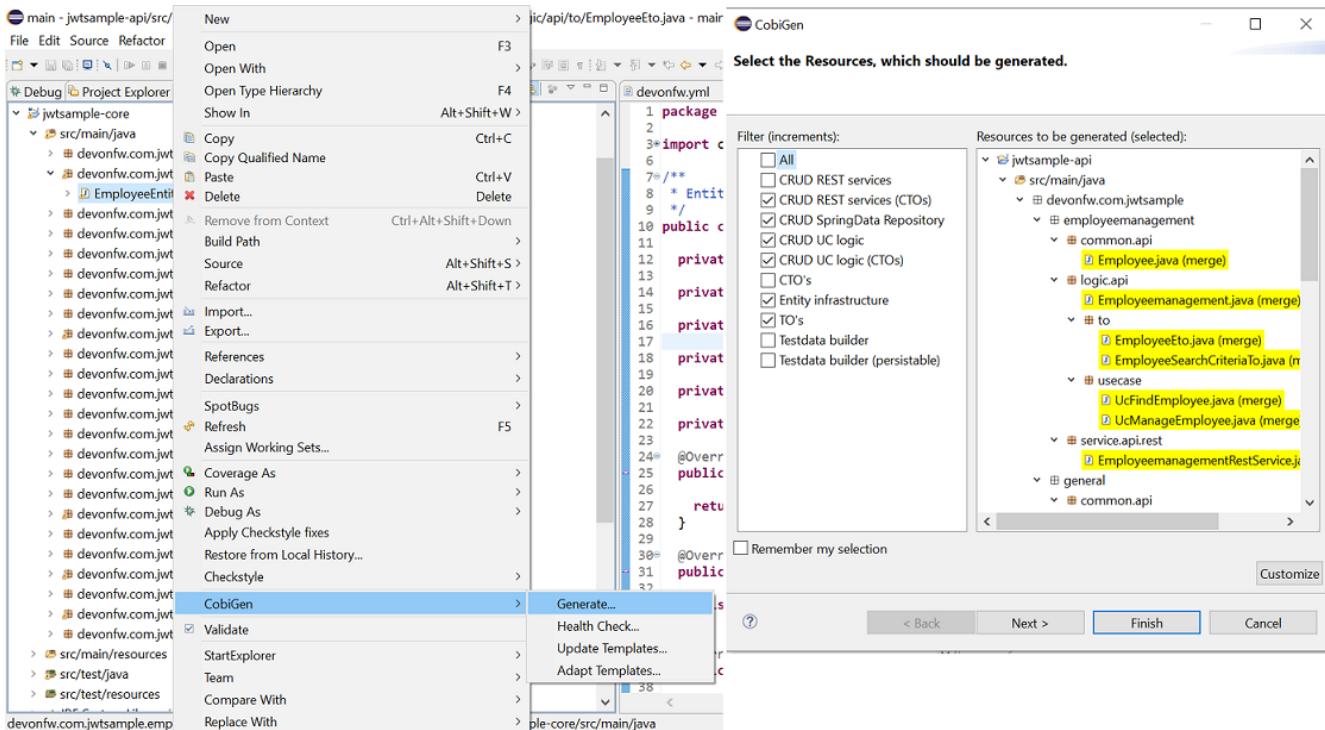
Product owners

devon4j guide

Architecture basics

Coding conventions

What should you receive from this DevOps module



What will you gain with our DevOps module

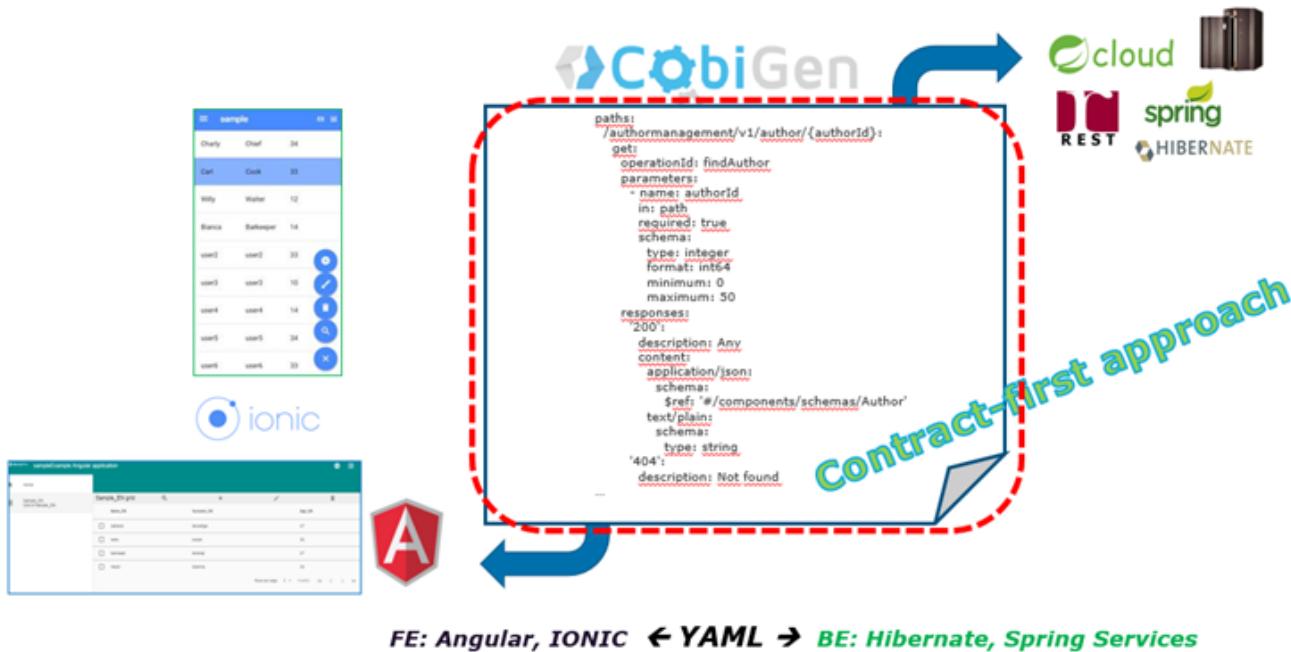
The CI procedure has been divided into transparent modules. This solution makes configuration and maintenance very easy because everyone is able to manage versions and customize the configuration independently for each module. A separate security module ensures the protection of your credentials and assigned access roles regardless of changes in other modules.

^

Name

-  conf
-  doc
-  scripts
-  settings
-  software
-  system
-  workspaces
-  workspaces_vs
-  Configurator.log
-  console.bat
-  create-or-update-workspace.bat
-  create-or-update-workspace-vs.bat
-  Devonfw3.0.0-Release-Notes.txt
-  ps-console.bat
-  s2-create.bat
-  s2-init.bat
-  update-all-workspaces.bat
-  variables.bat
-  vscode-workspaces_vs.bat

Your CI process will be matched to the current project. You can easily go back to the previous configuration, test a new one or move a selected one to other projects.



DevOps module supports a delivery model in which executors are made available to the user as needed. It has such advantages as:

- Saving computing resources
- Eliminating guessing on your infrastructure capacity needs
- Not spending time on running and maintaining additional executors

```
C:\WINDOWS\system32\cmd.exe
IDE environment has been initialized.
Copied workspaces\main\development\settings\maven\settings.xml to conf\.m2\settings.xml
Copied workspaces\main\development\settings\version\settings.json to conf\settings.json
lug 16, 2018 4:12:14 PM io.oasp.ide.eclipse.configurator.core.Configurator logCall
INFO: io.oasp.ide.eclipse.configurator.core.Configurator -u
lug 16, 2018 4:12:14 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Updating workspace
lug 16, 2018 4:12:14 PM io.oasp.ide.eclipse.configurator.core.Configurator collectWorkspaceFiles
INFO: Collected 78 configuration files.
lug 16, 2018 4:12:15 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Completed
Eclipse preferences for workspace: "main" have been created/updated
Created eclipse-main.bat
Finished creating/updating workspace: "main"

Press any key to continue . . .
```

```
C:\WINDOWS\system32\cmd.exe
IDE environment has been initialized.
IDE environment has been initialized.
Copied workspaces\main\development\settings\version\settings.json to conf\settings.json
lug 16, 2018 4:12:47 PM io.oasp.ide.eclipse.configurator.core.Configurator logCall
INFO: io.oasp.ide.eclipse.configurator.core.Configurator -u
lug 16, 2018 4:12:47 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Updating workspace
lug 16, 2018 4:12:47 PM io.oasp.ide.eclipse.configurator.core.Configurator collectWorkspaceFiles
INFO: Collected 78 configuration files.
lug 16, 2018 4:12:47 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Completed
Eclipse preferences for workspace: "examples" have been created/updated
Created eclipse-examples.bat
Finished creating/updating workspace: "examples"

IDE environment has been initialized.
Copied workspaces\main\development\settings\version\settings.json to conf\settings.json
lug 16, 2018 4:12:48 PM io.oasp.ide.eclipse.configurator.core.Configurator logCall
INFO: io.oasp.ide.eclipse.configurator.core.Configurator -u
lug 16, 2018 4:12:48 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Updating workspace
lug 16, 2018 4:12:48 PM io.oasp.ide.eclipse.configurator.core.Configurator collectWorkspaceFiles
INFO: Collected 78 configuration files.
lug 16, 2018 4:12:48 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Completed
Eclipse preferences for workspace: "main" have been created/updated
Finished creating/updating workspace: "main"

Finished updating all workspaces
Press any key to continue . . .
```

System (C:) > Devon-dist_2.4.0

Name

 conf
 doc
 scripts
 settings
 software
 system
 workspaces
 workspaces_vs
 console.bat
 create-or-update-workspace.bat
 create-or-update-workspace-vs.bat
 Devonfw2.4.0-Release-Notes.txt
 EclipseConfigurator.log
 eclipse-examples.bat
 eclipse-itapoc.bat
 eclipse-main.bat
 ns-console.bat

How to build this DevOps module

Once you have implemented the module, you can learn more about it here:

- [Docker commands](#)

76.9.2. Continuous Integration

Embrace quality with Continuous Integration while you produce test case(s).

Overview

There are two ways to set up your Continuous Integration environment:

1. Create a Jenkins instance from scratch (e.g. by using the Jenkins Docker image)

Using a clean Jenkins instance requires the installation of additional plugins. The plugins required and their versions can be found on [this page](#).

2. Use three pre-configured custom Docker image provided by us

No more additional configuration is required (but optional) using this custom Docker image. Additionally, this Jenkins setup allows dynamical scaling across multiple machines and even cloud (AWS, Azure, Google Cloud etc.).

Jenkins Overview

Jenkins is an Open Source Continuous Integration Tool. It allows the user to create automated build jobs which will run remotely on so called Jenkins Slaves. A build job can be triggered by several events, for example on new pull request on specified repositories or timed (e.g. at midnight).

Jenkins Configuration

Tests created by using the testing framework can easily be implemented on a Jenkins instance. The following chapter will describe such a job configuration. If you're running your own Jenkins instance, you may have to install additional plugins listed on the page [Jenkins Plugins](#) for a trouble-free integration of your tests.

Initial Configuration

The test job is configured as a so-called *parameterized* job. This means, after starting the job, parameters can be specified, which will then be used in the build process. In this case, *branch* and *testname* will be expected when starting the job. These parameters specify which branch in the code repository should be checked out (possibly feature branch) and the name of the test that should be executed.

This screenshot shows the Jenkins job configuration interface for defining parameters. The 'This project is parameterized' checkbox is checked. Two parameters are defined:

- String Parameter** (Name: BRANCH, Default Value: empty, Description: Branch name, where test's will be executed. In example: origin/develop, feature/lustefan-1)
- String Parameter** (Name: TESTNAME, Default Value: empty, Description: This is test class name with full package name, example com.example.selenium.tests.tests.demo.main.registration.RegisterOKTest)

An 'Add Parameter' button is visible at the bottom left.

Build Process Configuration

- The first step inside the build process configuration is to get the author of the commit that was made. The mail will be extracted and gets stored in a file called *build.properties*. This way, the author can be notified if the build fails.

```
echo "$(git rev-parse HEAD)" > gitCommitId.txt
echo GIT_COMMIT=$(head -n 1 gitCommitId.txt) >> build.properties
GIT_AUTHOR=$(git --no-pager show -s --format='%an' $GIT_COMMIT)
GIT_AUTHOR_EMAIL=$(git --no-pager show -s --format='%ae' $GIT_COMMIT)
echo GIT_AUTHOR=${GIT_AUTHOR} >> build.properties
echo GIT_AUTHOR_EMAIL=${GIT_AUTHOR_EMAIL} >> build.properties
```

See [the list of available environment variables](#)

Advanced...

- Next up, Maven will be used to check if the code can be compiled, without running any tests.

Invoke top-level Maven targets

Maven Version: v3.3.9

Goals: clean install test-compile -DskipTests=true

Advanced...

After making sure that the code can be compiled, the actual tests will be executed.

Invoke top-level Maven targets

Maven Version: v3.3.9

Goals: surefire:test -Dtest=\${TESTNAME} -DseleniumGrid='http://10.40.232.61:4444/wd/hub' -Dos=LINUX -Dbrowser=chrome

Advanced...

- Finally, reports will be generated.

Invoke top-level Maven targets

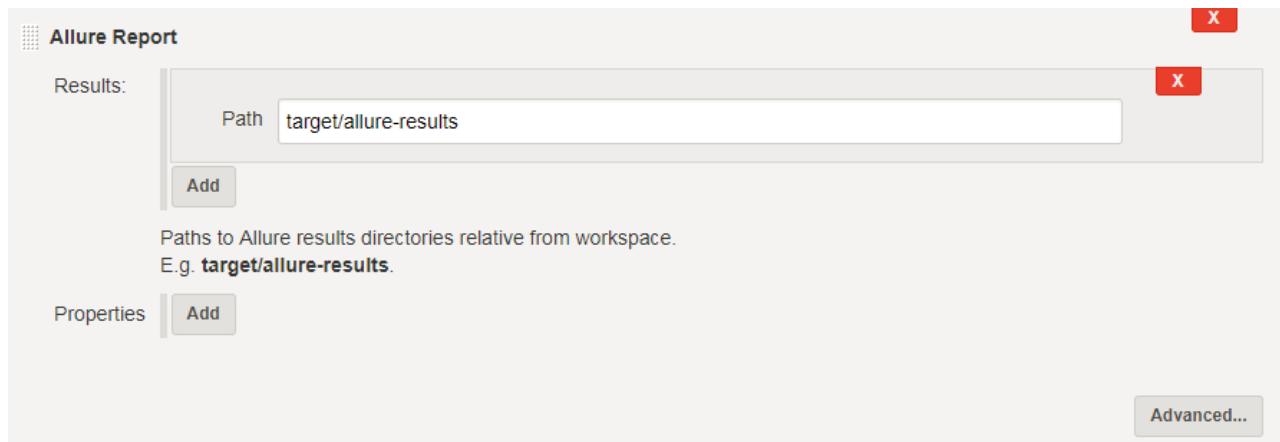
Maven Version: v3.3.9

Goals: site

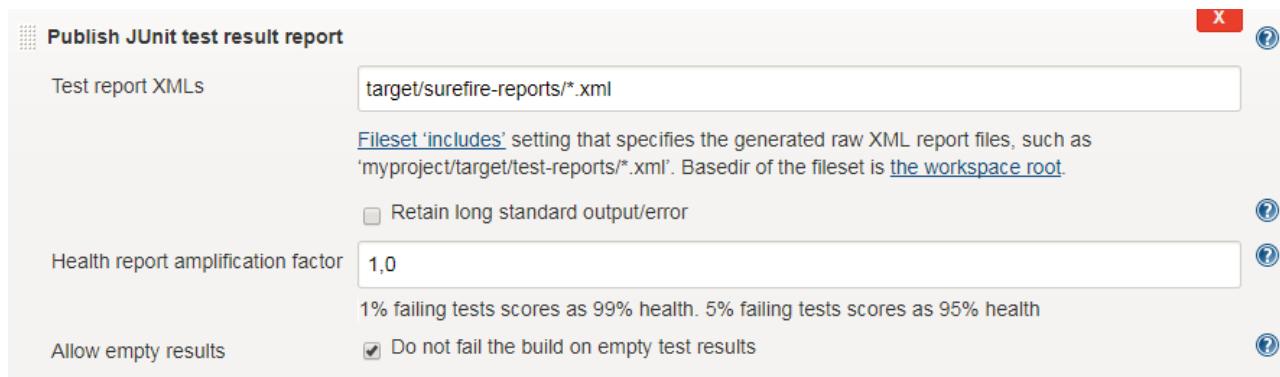
Advanced...

Post Build Configuration

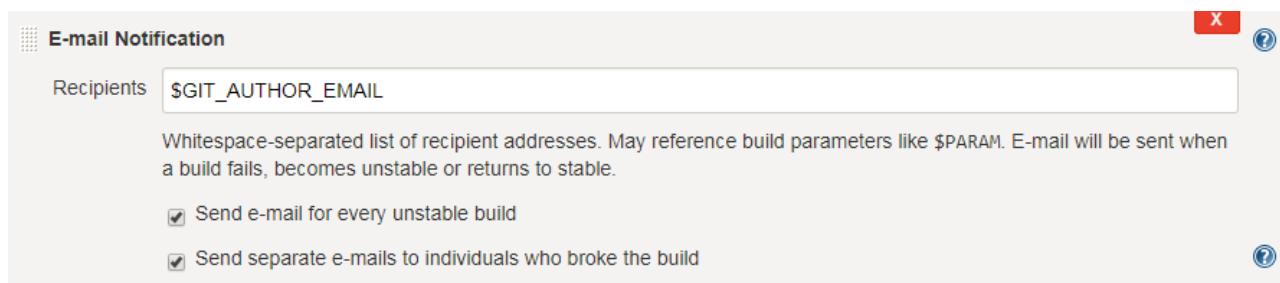
- At first, the results will be imported to the [Allure System](#)



- JUnit test results will be reported as well. Using this step, the test result trend graph will be displayed on the Jenkins job overview.



- Finally, an E-Mail will be sent to the previously extracted author of the commit.



Using the Pre-Configured Custom Docker Image

If you are starting a new Jenkins instance for your tests, we'd suggest using the pre-configured Docker image. This image already contains all the configurations and additional features.

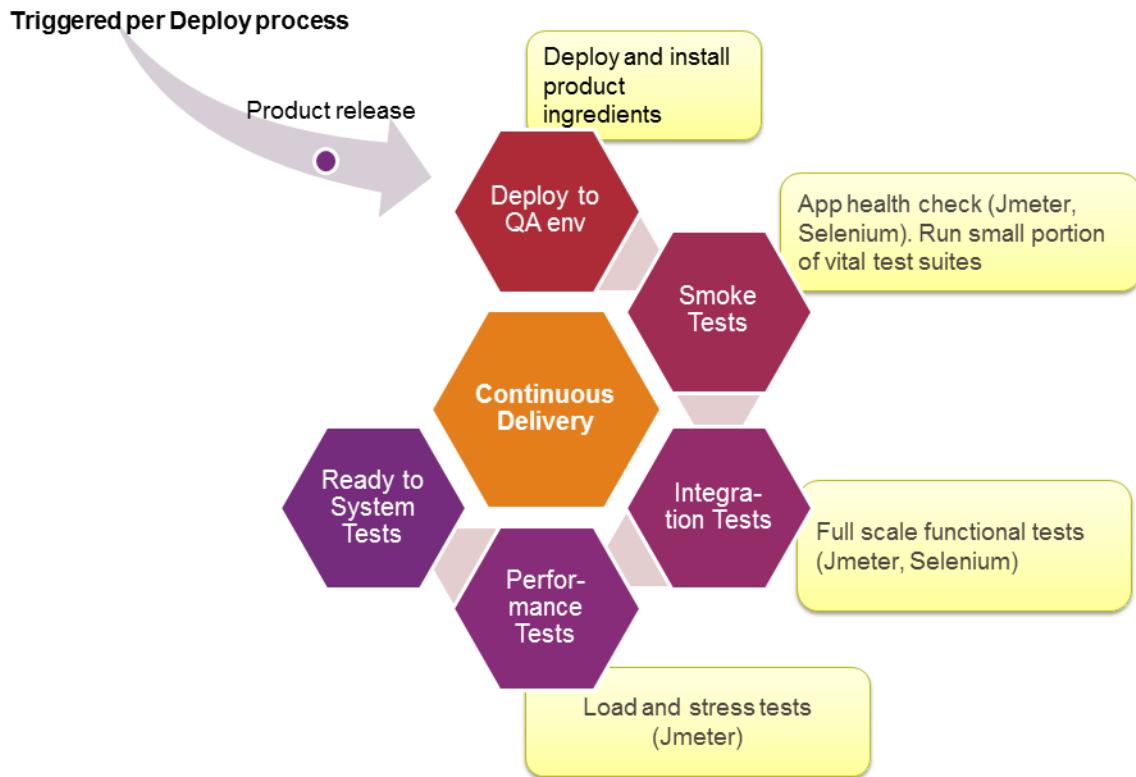
The configurations are e.g. Plugins and Pre-Installed job setup samples. This way, you don't have to set up the entire CI-Environment from the ground up.

Additional features from this docker image allow dynamic creation and deletion of Jenkins slaves, by creating Docker containers. Also, Cloud Solutions can be implemented to allow wide-spread load balancing.

76.9.3. Continuous Delivery

Include quality with Continuous Delivery during product release.

Standardize CD build pipeline



Overview

CD from Jenkins point of view does not change a lot from Continuous Integration one.

Jenkins Overview

Use the same Jenkins settings for Jenkins CD setup as for CI, please. [link](#). The only difference is:

- What type of test you will execute. Before, we have been choosing test case(s), now we will choose test suite(s)
- Who will trigger the given Smoke/Integration/Performance job
- What is the name of official branch. This branch ought always to be used in every CD execution. It will be either **master** or **develop**.

Jenkins for Smoke Tests

In the \$TESTNAME variable, where we input the test name([link](#)), please input the name of a test suite assembled together of tests tagged as smoke tests -([link](#)) thus running all the smoke tests.

Jenkins for Performance Tests

Under construction - added when WebAPI module is included.

76.9.4. Pipeline structure

Pipeline configuration:

The default interaction with Jenkins required manual jobs. This keeps configuration of a job in Jenkins separate from source code. With Pipeline plugin users can implement a pipeline procedure in Jenkinsfile and store it in repository with other code. This approach is used in Mr Checker framework. More info: <https://jenkins.io/solutions/pipeline/>

Our CI & CD processes are divided into a few separate files: Jenkins_node.groovy is the file to manage all processes. It defines all operations executed on a Jenkins node, so all code in this file is closed in node closure. Workflow in Jenkinsfile:

- Read all parameters from a Jenkins job
- Execute stage to prepare the environment
- Execute git pull command
- Set Jenkins job description
- Execute compilation of the project in a special prepared docker container
- Execute unit tests
- Execute integration tests
- Deploy artifacts to a local repository
- Deploy artifacts to an external repository (nexus/arifactory)

Not all the steps must be present in the Jenkins files. This should be configured for particular job requirements.

Description of stages:

Stage “Prepare environment”

First thing to do in this stage is overwriting properties loaded from Jenkins job. It is defined in “overrideProperties” function. The next function, “setJenkinsJobVariables” defines environment variables such as :

- JOB_NAME_UPSTREAM
- BUILD_DISPLAY_NAME_UPSTREAM
- BUILD_URL_UPSTREAM
- GIT_CREDENTIALS
- JENKINS_CREDENTIALS

The last function in the stage – “setWorkspace” -creates an environment variable with path to local workspace. This is required because when using pipeline plugin, Jenkins does not create the WORKSPACE env variables.

Stage "Git pull"

It pulls sources from the repository and loads “git pull” file which contains additional methods:

- setGitAuthor – setting properties about git author to the file “build.properties” and loading created file
- tryMergeWithBranch – checking if actual branch can be merged with default main branch

Stage “Build compile”

Verify with maven that code builds without errors

Stage “Unit test”

Execute unit tests with mvn surefire test and publish reports in junit and allure format

Stage “Integration test”

Execute integration tests with mvn surefire test and publish reports in junit and allure format

[[devops-test-module-pipeline-structure.asciidoc_stage-deploy—local-repo]] === Stage “Deploy – local repo”

Archive artifacts as a jar file in the local repository

[[devops-test-module-pipeline-structure.asciidoc_stage-deploy—nexu-repo]] === Stage ”Deploy – nexu repo”

Deploy to the external repository with maven release deploy command with credentials stored in Jenkins machine. Additional files:

- mailSender.groovy – contains methods for sending mail with generated content
- stashNotification.groovy – send job status for bitbucket by a curl command
- utils.groovy - contains additional functions to load properties, files and generate additional data

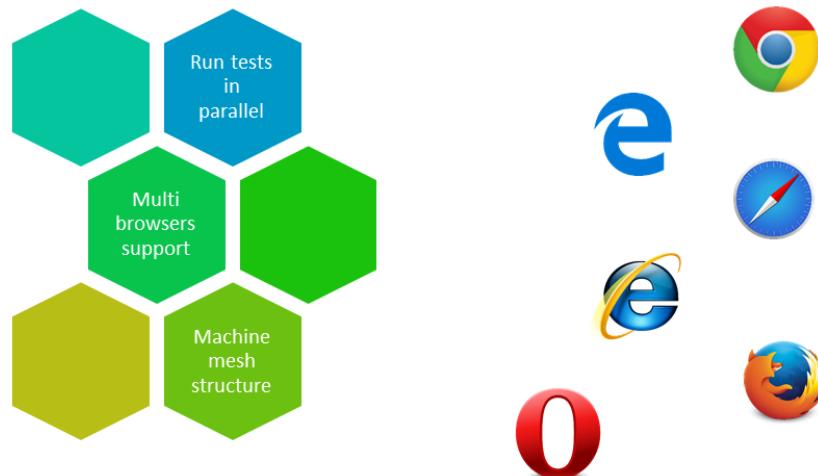
76.9.5. Selenium Grid

What is Selenium Grid

Selenium Grid allows running web/mobile browsers test cases to fulfill basic factors, such as:

- Independent infrastructure, similar to end-users'
- Scalable infrastructure (~50 simultaneous sessions at once)
- Huge variety of web browsers (from mobile to desktop)
- Continuous Integration and Continuous Delivery process
- Supporting multi-type programming languages (java, javascript, python, ...).

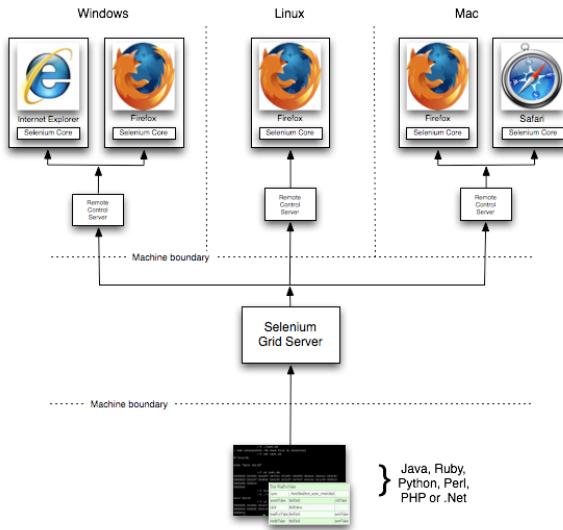
Selenium Grid – where we use it



On a daily basis, a test automation engineer uses their local environments for test case execution/development. However, a created browser test case has to be able to run on any infrastructure. Selenium Grid enables this portability.

Selenium Grid Structure

Selenium Grid - structure



Full documentation of Selenium Grid can be found [here](#) and [here](#).

'Vanilla flavour' Selenium Grid is based on two, not very complicated ingredients:

1. **Selenium Hub** - as one machine, accepting connections to grid from test cases executors. It also

plays a managerial role in connection to/from Selenium Nodes

2. **Selenium Node** - from one to many machines, where on each machine a browser used during test case execution is installed.

How to setup

There are two options of Selenium Grid setup:

- Classic, static solution - [link](#)
- Cloud, scalable solution - [link](#)

Advantages and disadvantages of both solutions:

Single Selenium Grid - comparison

	Cost to setup (20 instances/browsers)	Cost to scale up (down) new 20 instances/browsers	Team responsibility	Resilient and robust	Portability
Classic solution	<ul style="list-style-type: none"> • 200 \$ /month (VMs) 	<ul style="list-style-type: none"> • + 20 \$ /month (VM) + 4 hour SeleniumGrid specialist 	<ul style="list-style-type: none"> • Works as centralized solution used across all Test departments 	<ul style="list-style-type: none"> • No replication. Manual actions needed to recover 	<ul style="list-style-type: none"> • Manual infrastructure setup, for each browser (Chrome, Firefox, IE, Safari, Edge)
Cloud solution	<ul style="list-style-type: none"> • 10 \$ /month (VMs) 	<ul style="list-style-type: none"> • + 10 \$ /month (VMs) + 0.25 hour junior team member 	<ul style="list-style-type: none"> • Works as centralized per department or decentralized solution per team 	<ul style="list-style-type: none"> • Auto recovery. Balance number of active instances/ browsers through swarm of active VMs 	<ul style="list-style-type: none"> • Auto deploy script for Chrome and Firefox. Open interface to add manually IE, Safari, Edge

*) Classic solution – selenium grid run as a Java standalone application

*) Cloud solution – selenium grid run as a Docker container

VM – Virtual Machine

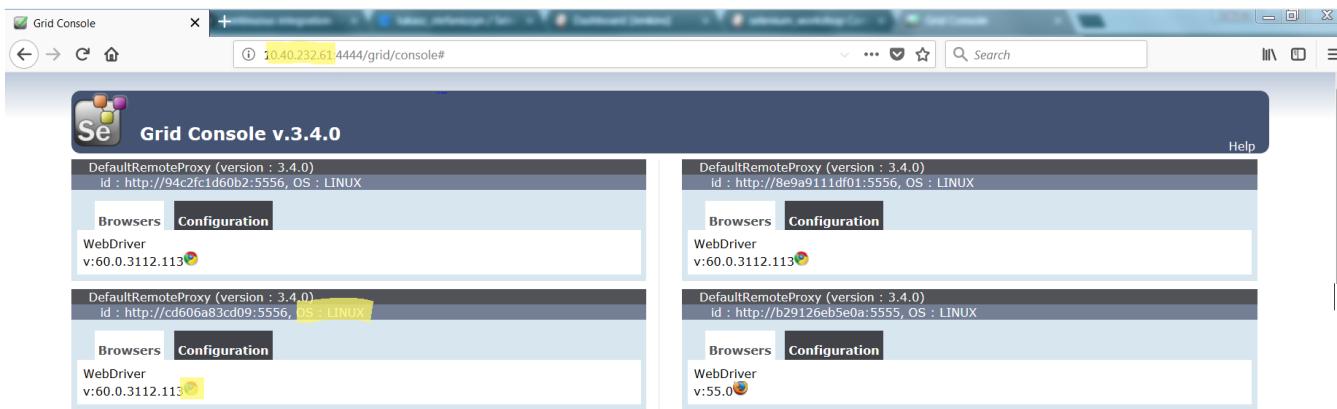
How to use Selenium Grid with E2E Mr Checker Test Frameworks

Run the following command either in Eclipse or in Jenkins:

```
> mvn test -Dtest=com.capgemini.ntc.selenium.tests.samples.resolutions.ResolutionTest
-DseleniumGrid="http://10.40.232.61:4444/wd/hub" -Dos=LINUX -Dbrowser=chrome
```

As a result of this command:

- **-Dtest=com.capgemini.ntc.selenium.features.samples.resolutions.ResolutionTest** - name of test case to execute
- **-DseleniumGrid="http://10.40.232.61:4444/wd/hub"** - IP address of Selenium Hub
- **-Dos=LINUX** - what operating system must be assumed during test case execution
- **-Dbrowser=chrome** - what type of browser will be used during test case execution



76.9.6. List of Jenkins Plugins

Plugin Name	Version
blueocean-github-pipeline	1.1.4
blueocean-display-url	2.0
blueocean	1.1.4
workflow-support	2.14
workflow-api	2.18
plain-credentials	1.4
pipeline-stage-tags-metadata	1.1.8
credentials-binding	1.12
git	3.5.1
maven-plugin	2.17
workflow-durable-task-step	2.12
job-dsl	1.64
git-server	1.7
windows-slaves	1.3.1
github	1.27.0
blueocean-personalization	1.1.4
jackson2-api	2.7.3
momentjs	1.1.1
workflow-basic-steps	2.6
workflow-aggregator	2.5
blueocean-rest	1.1.4
gradle	1.27.1
pipeline-maven	3.0.0
blueocean-pipeline-editor	0.2.0

Plugin Name	Version
durable-task	1.14
scm-api	2.2.2
pipeline-model-api	1.1.8
config-file-provider	2.16.3
github-api	1.85.1
pam-auth	1.3
workflow-cps-global-lib	2.8
github-organization-folder	1.6
workflow-job	2.12.1
variant	1.1
git-client	2.5.0
sse-gateway	1.15
script-security	1.29.1
token-macro	2.1
jquery-detached	1.2.1
blueocean-web	1.1.4
timestampper	1.8.8
greenballs	1.15
handlebars	1.1.1
blueocean-jwt	1.1.4
pipeline-stage-view	2.8
blueocean-i18n	1.1.4
blueocean-git-pipeline	1.1.4
ace-editor	1.1
pipeline-stage-step	2.2
email-ext	2.58
envinject-api	1.2
role-strategy	2.5.1
structs	1.9
locale	1.2
docker-workflow	1.13
ssh-credentials	1.13
blueocean-pipeline-scm-api	1.1.4

Plugin Name	Version
metrics	3.1.2.10
external-monitor-job	1.7
junit	1.21
github-branch-source	2.0.6
blueocean-config	1.1.4
cucumber-reports	3.8.0
pipeline-model-declarative-agent	1.1.1
blueocean-dashboard	1.1.4
subversion	2.9
blueocean-autofavorite	1.0.0
pipeline-rest-api	2.8
pipeline-input-step	2.7
matrix-project	1.11
pipeline-github-lib	1.0
workflow-multibranch	2.16
docker-plugin	0.16.2
resource-disposer	0.6
icon-shim	2.0.3
workflow-step-api	2.12
blueocean-events	1.1.4
workflow-scm-step	2.6
display-url-api	2.0
favorite	2.3.0
build-timeout	1.18
mapdb-api	1.0.9.0
pipeline-build-step	2.5.1
antisamy-markup-formatter	1.5
javadoc	1.4
blueocean-commons	1.1.4
cloudbees-folder	6.1.2
ssh-slaves	1.20
pubsub-light	1.10
pipeline-graph-analysis	1.4

Plugin Name	Version
allure-jenkins-plugin	2.23
mailer	1.20
ws-cleanup	0.33
authentication-tokens	1.3
blueocean-pipeline-api-impl	1.1.4
ldap	1.16
docker-commons	1.8
branch-api	2.0.10
workflow-cps	2.36.1
pipeline-model-definition	1.1.8
blueocean-rest-impl	1.1.4
ant	1.7
credentials	2.1.14
matrix-auth	1.7
pipeline-model-extensions	1.1.8
pipeline-milestone-step	1.3.1
jclouds-jenkins	2.14
bouncycastle-api	2.16.1

76.9.7. What is Docker

Docker is an open source software platform to create, deploy and manage virtualized application containers on a common operating system (OS), with an ecosystem of allied tools.

76.9.8. Where do we use Docker

DevOps module consists of Docker images

1. Jenkins image
2. Jenkins job image
3. Jenkins management image
4. Security image

in addition, each new node is also based on Docker

76.9.9. Exploring basic Docker options

Let's show some of the most important commands that are needed when working with our DevOps module based on the Docker platform. Each command given below should be preceded by a sudo

call by default. If you don't want to use `sudo` command create a Unix group called docker and add a user to it.

```
$ sudo groupadd docker  
$ sudo usermod -aG docker $USER
```

Build an image from a Dockerfile

```
# docker build [OPTIONS] PATH | URL | -  
#  
# Options:  
# --tag , -t : Name and optionally a tag in the 'name:tag' format  
  
$ docker build -t vc_jenkins_jobs .
```

Container start

```
# docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]  
#  
# Options:  
# -d : To start a container in detached mode (background)  
# -it : interactive terminal  
# --name : assign a container name  
# --rm : clean up  
# --volumes-from="" : Mount all volumes from the given container(s)  
# -p : explicitly map a single port or range of ports  
# --volume : storage associated with the image  
  
$ docker run -d --name vc_jenkins_jobs vc_jenkins_jobs
```

Remove one or more containers

```
# docker rm [OPTIONS] CONTAINER  
#  
# Options:  
# --force , -f : Force the removal of a running container  
  
$ docker rm -f jenkins
```

List containers

```
# docker ps [OPTIONS]
# --all, -a : Show all containers (default shows just running)

$ docker ps
```

Pull an image or a repository from a registry

```
# docker pull [OPTIONS] NAME[:TAG|@DIGEST]

$ docker pull jenkins/jenkins:2.73.1
```

Push the image or a repository to a registry

Pushing new image takes place in two steps. First save the image by adding container ID to the commit command and next use push:

```
# docker push [OPTIONS] NAME[:TAG]

$ docker ps
# copy container ID from the result
$ docker commit b46778v943fh vc_jenkins_mng:project_x
$ docker push vc_jenkins_mng:project_x
```

Return information on Docker object

```
# docker inspect [OPTIONS] NAME|ID [NAME|ID...]
#
# Options:
# --format , -f : output format

$ docker inspect -f '{{ .Mounts }}' vc_jenkins_mng
```

List images

```
# docker images [OPTIONS] [REPOSITORY[:TAG]]
#
# Options:
--all , -a : show all images with intermediate images

$ docker images
$ docker images jenkins
```

Remove one or more images

```
# docker rmi [OPTIONS] IMAGE [IMAGE...]
#
# Options:
#   --force , -f : Force removal of the image
$ docker rmi jenkins/jenkins:latest
```

Run a command in a running container

```
# docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
# -d : run command in the background
# -it : interactive terminal
# -w : working directory inside the container
# -e : Set environment variables

$ docker exec vc_jenkins_jobs sh -c "chmod 755 config.xml"
```

76.9.10. Advanced commands

Remove dangling images

```
$ docker rmi $(docker images -f dangling=true -q)
```

Remove all images

```
$ docker rmi $(docker images -a -q)
```

Removing images according to a pattern

```
$ docker images | grep "pattern" | awk '{print $2}' | xargs docker rm
```

Remove all exited containers

```
$ docker rm $(docker ps -a -f status=exited -q)
```

Remove all stopped containers

```
$ docker rm $(docker ps --no-trunc -aq)
```

Remove containers according to a pattern

```
$ docker ps -a | grep "pattern" | awk '{print $1}' | xargs docker rmi
```

Remove dangling volumes

```
$ docker volume rm $(docker volume ls -f dangling=true -q)
```

77. MrChecker download

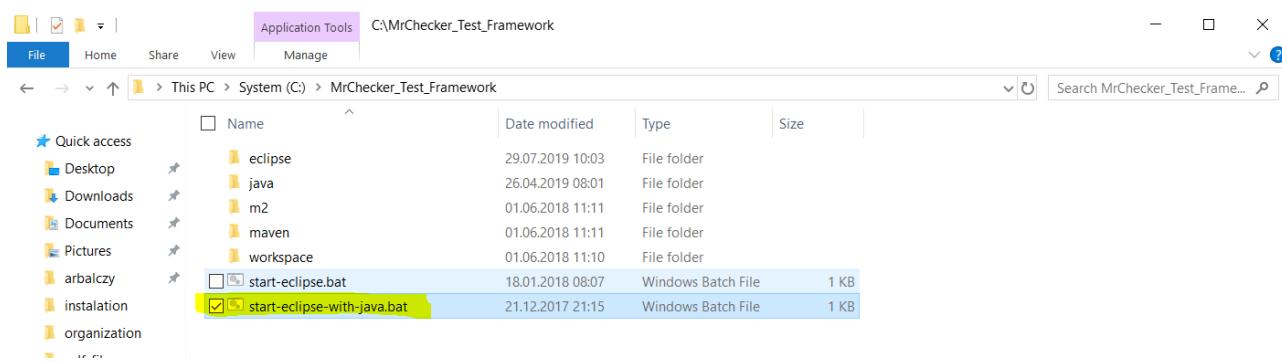
77.1. Windows

77.1.1. Easy out of the Box

1. Click on the link [Ready to use MrChecker_Test_Environment for Junit4](#) or [Ready to use MrChecker_Test_Environment for Junit5](#) and download the package
2. Unzip the downloaded MrChecker Test Framework into the folder C:\ on your PC - recommended tool: [7z](#) All the necessary components, such as Eclipse, Java and Maven will be pre-installed for you. There is no need for any additional installations.

Note: Please double check the place into which you have unzipped MrChecker_Test_Framework

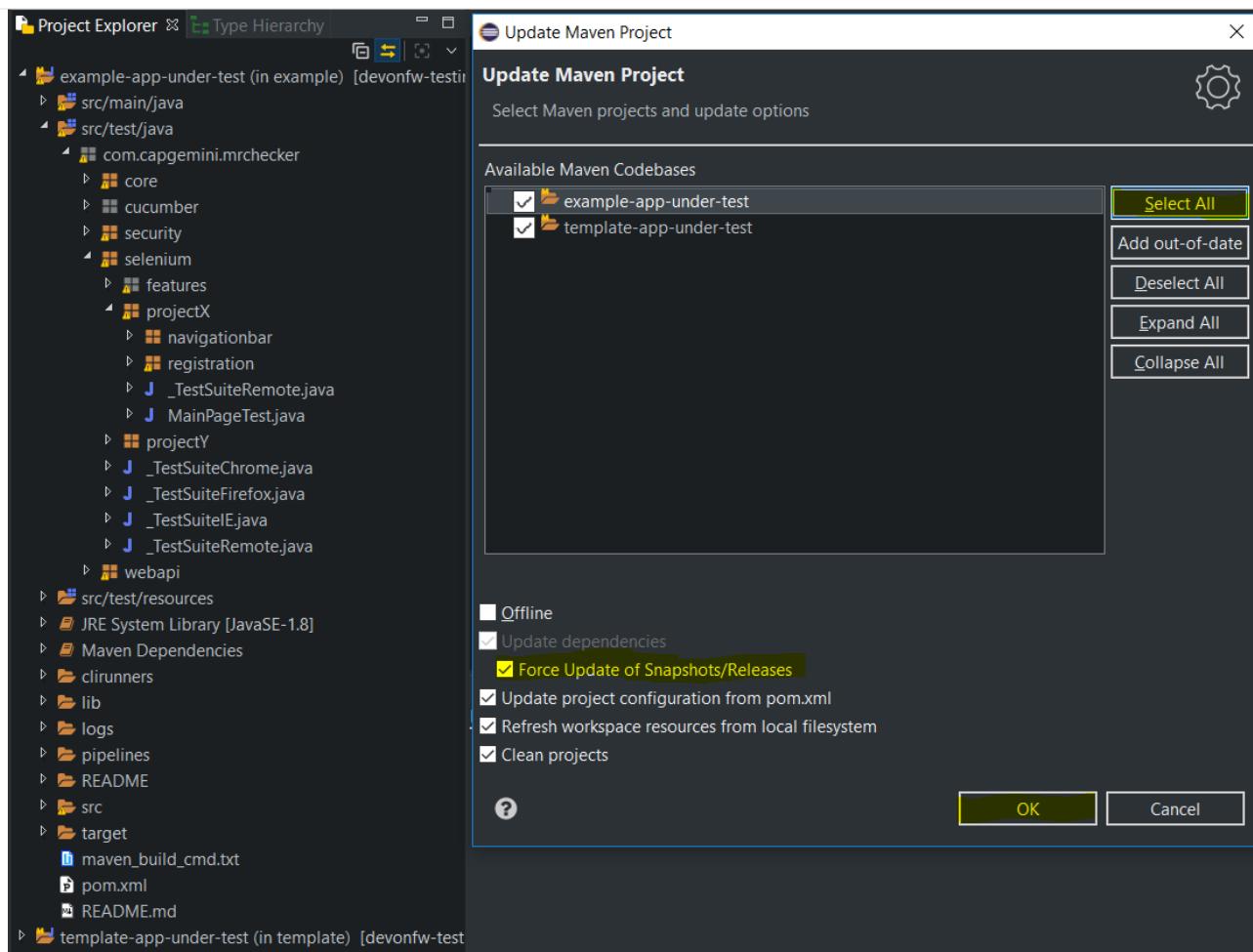
3. Go to folder C:\MrChecker_Test_Framework\ , into which Mr.Checker has been unzipped



4. In order to run the program, double click on *start-eclipse-with-java.bat*

(note that start-eclipse.bat won't detect Java)

5. Update project structure (*ALT + F5*)



77.1.2. Out of the box installation

1. Start from Easy out of the box installation
2. Open Eclipse
3. Manually Delete folders that appear in Eclipse
4. Click inside Eclipse with a right mouse button and open Import
5. Select Maven → existing Maven project
6. Select Mr Checker → workspace → devonfw-testing and click OK

All test folders should be imported into Eclipse and ready to use.

77.1.3. Advanced installation

Java installation

There is one important pre-requisite for Mr Checker installation - Java has to be installed on the computer and an environmental variable has to be set in order to obtain optimal functioning of the framework.

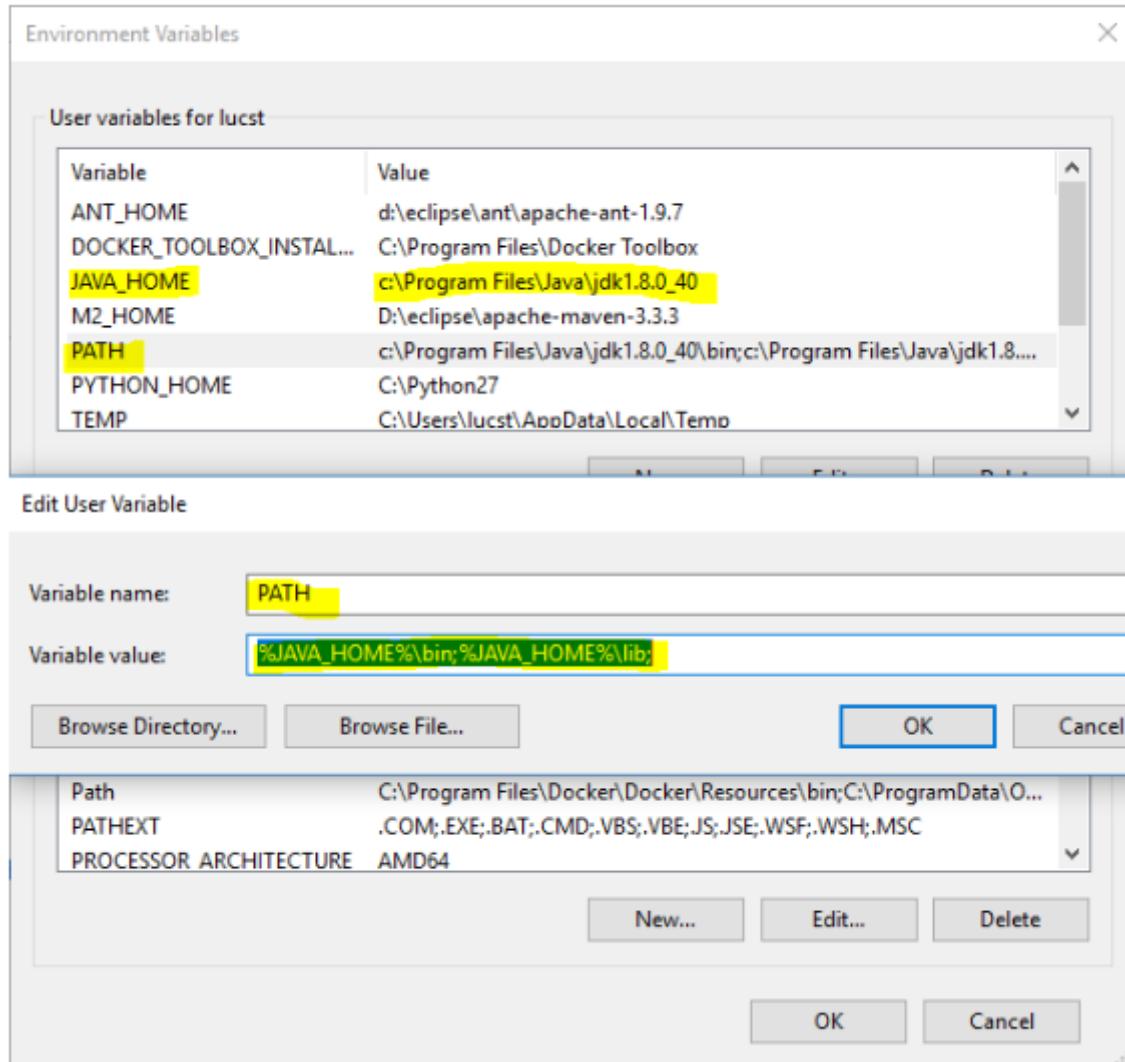
1. Install Java 1.8 JDK 64bit

Download and install [Java download link](#)

(To download JDK 8 from Oracle you have to have an account. It is recommended to get a JDK build based on OpenJDK from [AdoptOpenJDK](#))

2. Windows Local Environment - [How to set:](#)

- **Variable name:** JAVA_HOME | **Variable value:** C:\Where_You've_Installed_Java
- **Variable name:** PATH | **Variable value:** %JAVA_HOME%\bin;%JAVA_HOME%\lib



3. Next, verify it in the command line:

```
> java --version
```

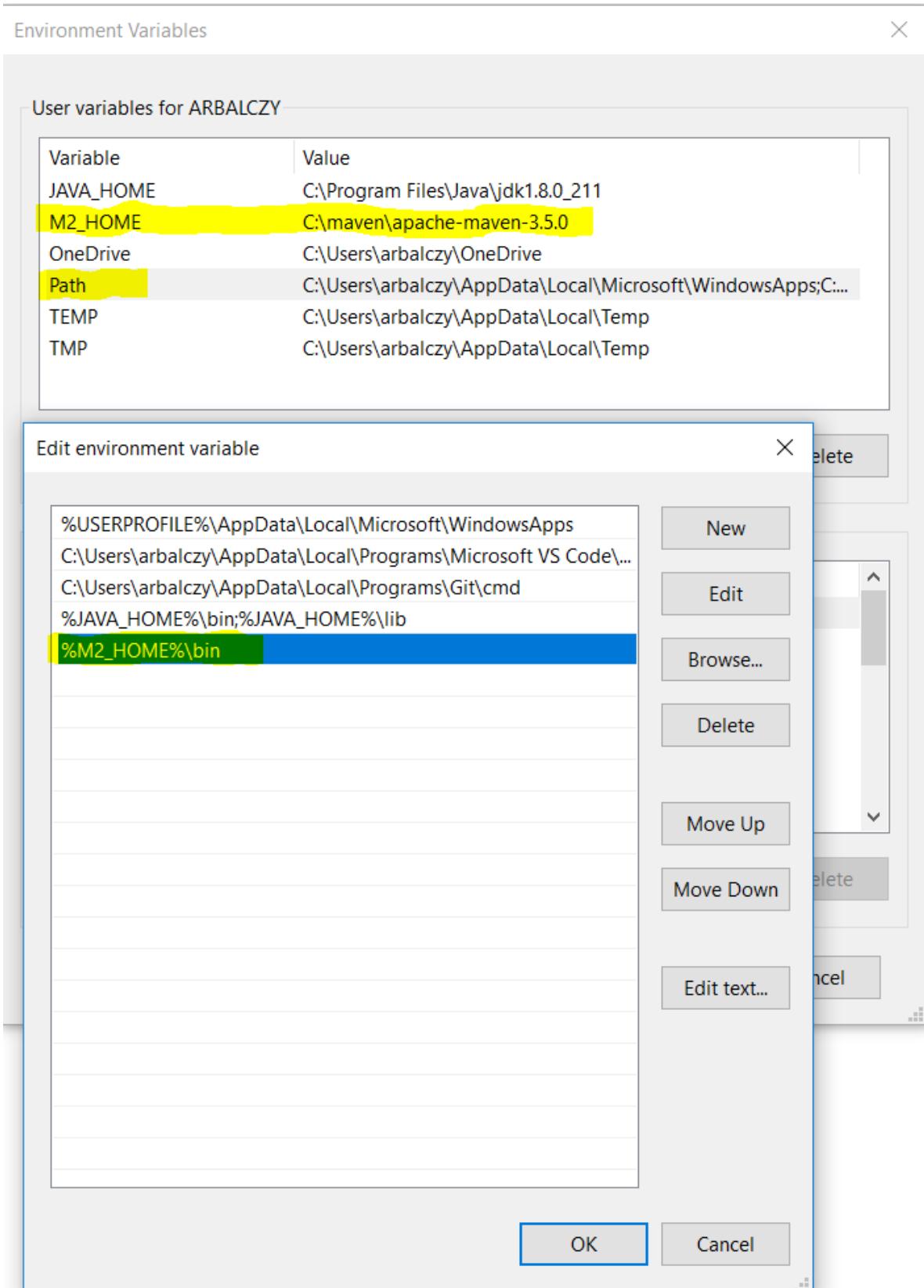
Other components installation

Install each component separately, or update the existing ones on your PC.

1. Maven 3.5

- [Download Maven](#)
- Unzip Maven in following location C:\maven
- Set Windows Local Environment

- **Variable name:** M2_HOME | **Variable value:** C:\maven\apache-maven-3.5.0
- **Variable name:** PATH | **Variable value:** %M2_HOME%\bin

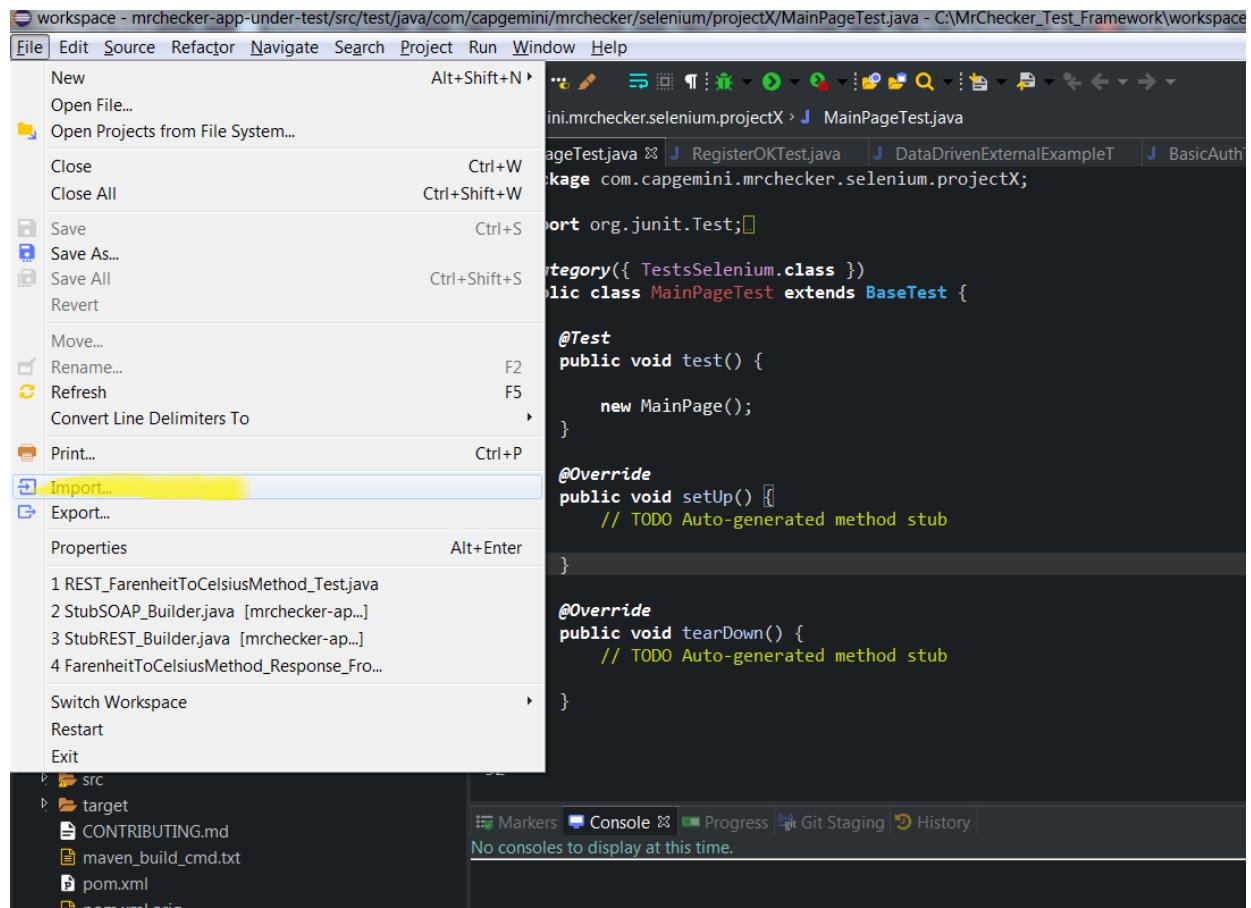


- Verify it in the command line:

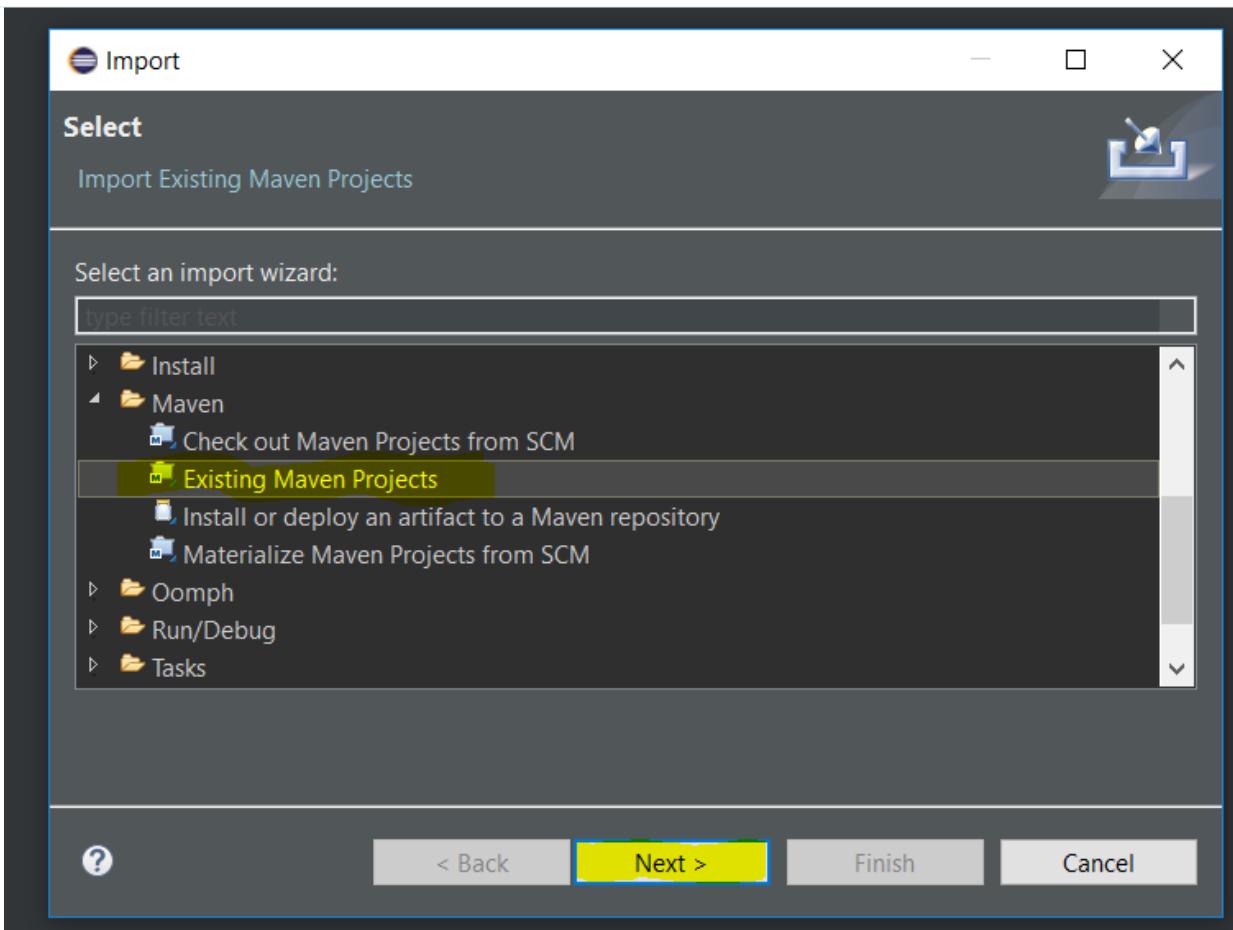
```
> mvn --version
```

2. Eclipse IDE

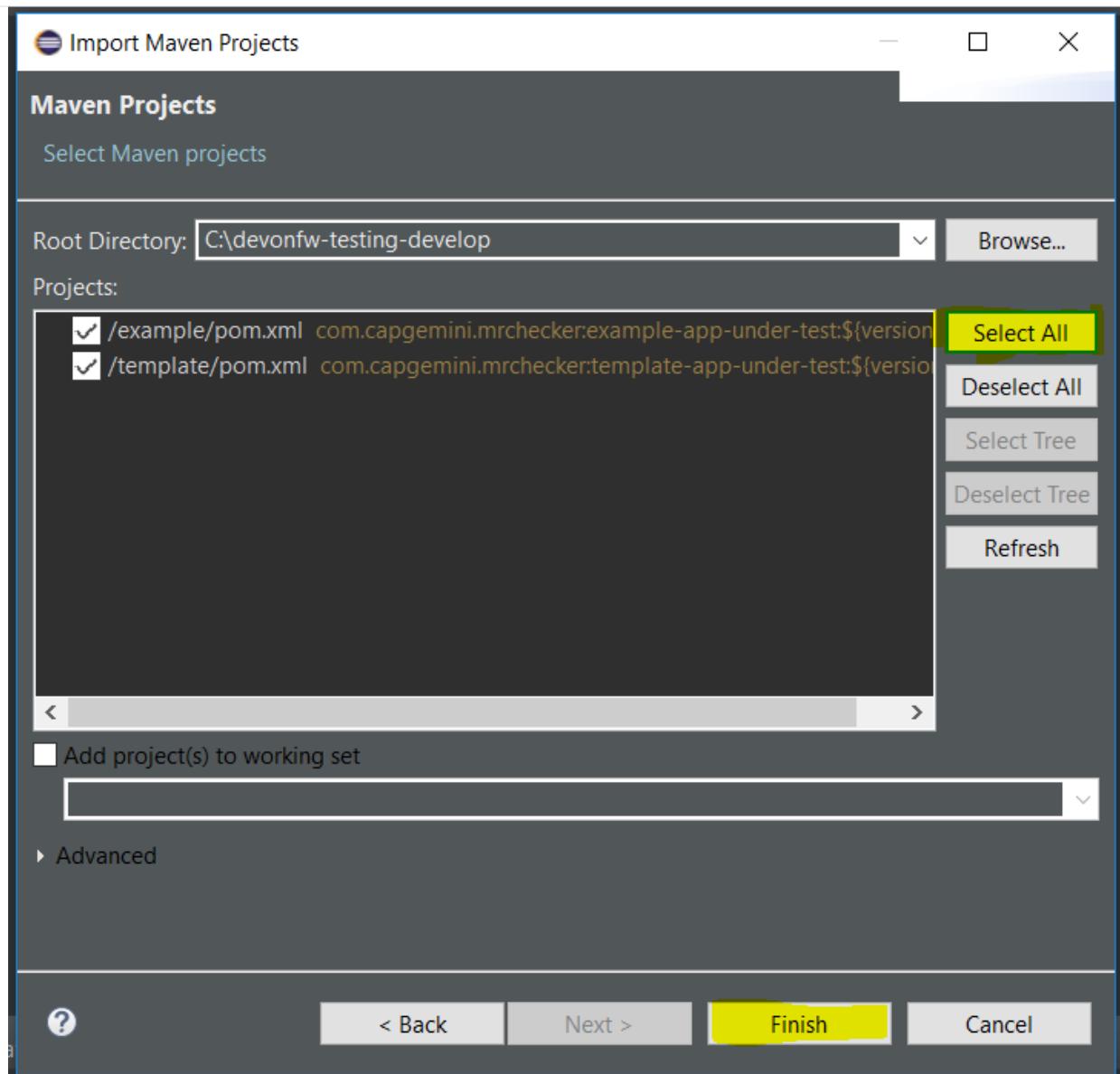
- Download an unzip [Eclipse](#)
- Download MrCheckerTestFramework [source code](#)
- Import:



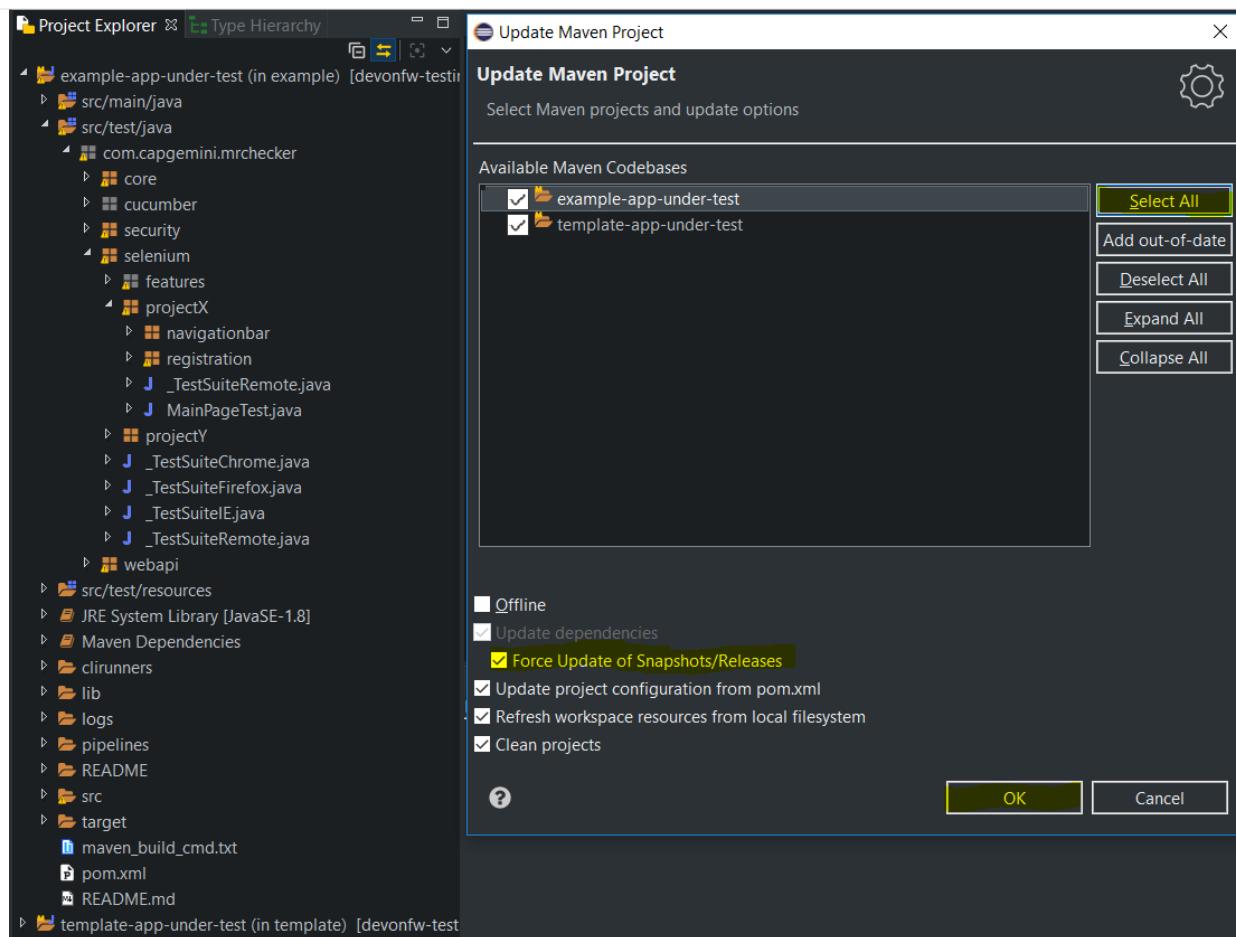
- Projects from folders



- Open already created projects:



- Update project structure - *ALT + F5*



77.2. Mac

77.2.1. MrChecker macOS installation

On this page, you can find all the details regarding MrChecker installation on your Mac.

Java installation

There is one important pre-requisite for Mr Checker installation - Java has to be installed on the computer and an environmental variable has to be set in order to obtain optimal functioning of the framework.

1. Install Java 1.8 JDK 64bit

Download and install [Java download link](#)

(To download JDK 8 from Oracle you have to have an account. It is recommended to get a JDK build based on OpenJDK from [AdoptOpenJDK](#))

2. Next, verify thx in the command line:

```
> java --version
```

Other components installation

Install each component separately, or update the existing ones on your Mac.

1. Maven 3.5

- [Download Maven](#)
- Unzip Maven in the following location /maven
- Add Maven to PATH

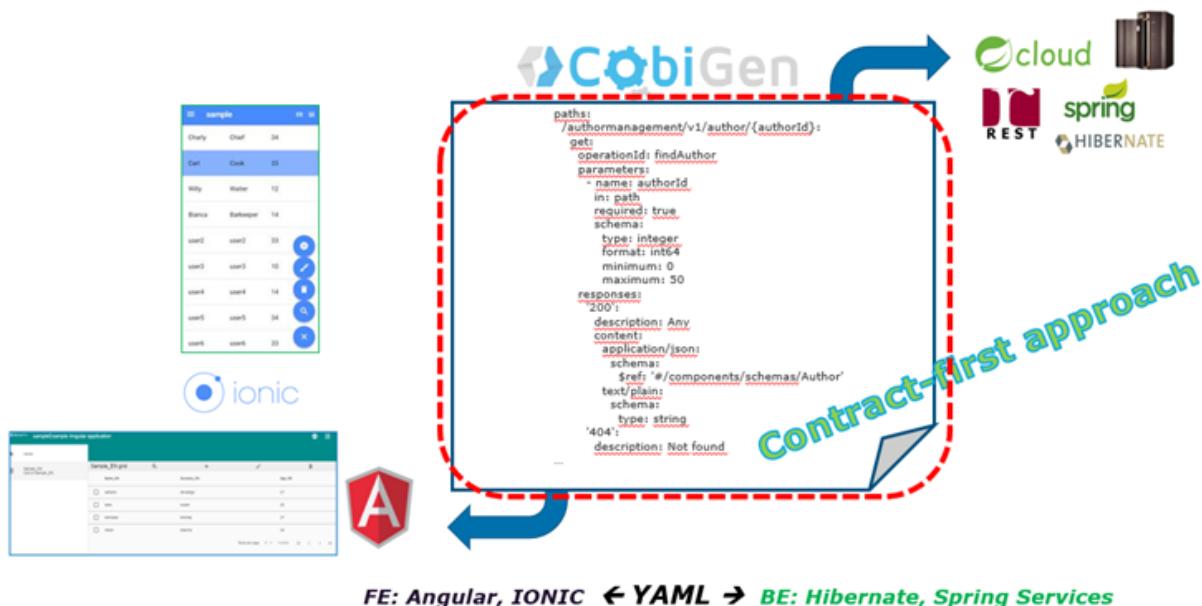
```
> $ export PATH=$PATH:/maven/apache-maven-3.5.0/bin/
```

- Verify in terminal:

```
> $ mvn -version
```

2. Eclipse IDE

- Download and unzip [Eclipse](#)
- Download MrCheckerTestFramework [source code](#)
- Import:



- Select Projects from folders:

```
C:\WINDOWS\system32\cmd.exe
IDE environment has been initialized.
Copied workspaces\main\development\settings\maven\settings.xml to conf\.m2\settings.xml
Copied workspaces\main\development\settings\version\settings.json to conf\settings.json
lug 16, 2018 4:12:14 PM io.oasp.ide.eclipse.configurator.core.Configurator logCall
INFO: io.oasp.ide.eclipse.configurator.core.Configurator -u
lug 16, 2018 4:12:14 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Updating workspace
lug 16, 2018 4:12:14 PM io.oasp.ide.eclipse.configurator.core.Configurator collectWorkspaceFiles
INFO: Collected 78 configuration files.
lug 16, 2018 4:12:15 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Completed
Eclipse preferences for workspace: "main" have been created/updated
Created eclipse-main.bat
Finished creating/updating workspace: "main"

Press any key to continue . . .
```

- Open already created projects:

```
C:\WINDOWS\system32\cmd.exe
IDE environment has been initialized.
IDE environment has been initialized.
Copied workspaces\main\development\settings\version\settings.json to conf\settings.json
lug 16, 2018 4:12:47 PM io.oasp.ide.eclipse.configurator.core.Configurator logCall
INFO: io.oasp.ide.eclipse.configurator.core.Configurator -u
lug 16, 2018 4:12:47 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Updating workspace
lug 16, 2018 4:12:47 PM io.oasp.ide.eclipse.configurator.core.Configurator collectWorkspaceFiles
INFO: Collected 78 configuration files.
lug 16, 2018 4:12:47 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Completed
Eclipse preferences for workspace: "examples" have been created/updated
Created eclipse-examples.bat
Finished creating/updating workspace: "examples"

IDE environment has been initialized.
Copied workspaces\main\development\settings\version\settings.json to conf\settings.json
lug 16, 2018 4:12:48 PM io.oasp.ide.eclipse.configurator.core.Configurator logCall
INFO: io.oasp.ide.eclipse.configurator.core.Configurator -u
lug 16, 2018 4:12:48 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Updating workspace
lug 16, 2018 4:12:48 PM io.oasp.ide.eclipse.configurator.core.Configurator collectWorkspaceFiles
INFO: Collected 78 configuration files.
lug 16, 2018 4:12:48 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Completed
Eclipse preferences for workspace: "main" have been created/updated
Finished creating/updating workspace: "main"

Finished updating all workspaces
Press any key to continue . . .
```

- Update project structure - *ALT + F5*

System (C:) > Devon-dist_2.4.0

Name

-  conf
-  doc
-  scripts
-  settings
-  software
-  system
-  workspaces
-  workspaces_vs
-  console.bat
-  create-or-update-workspace.bat
-  create-or-update-workspace-vs.bat
-  Devonfw2.4.0-Release-Notes.txt
-  EclipseConfigurator.log
-  eclipse-examples.bat
-  eclipse-itapoc.bat
-  eclipse-main.bat
-  ns-console.bat

78. Tutorials

In order to learn more about MrChecker structure, start from Project Organisation section and then check out our fantastic tutorials:

This tutorial will guide you through the series of test which perform basic actions on webpages using MrChecker.

Make sure you already have MrChecker Test Framework installed on your PC. How to install?

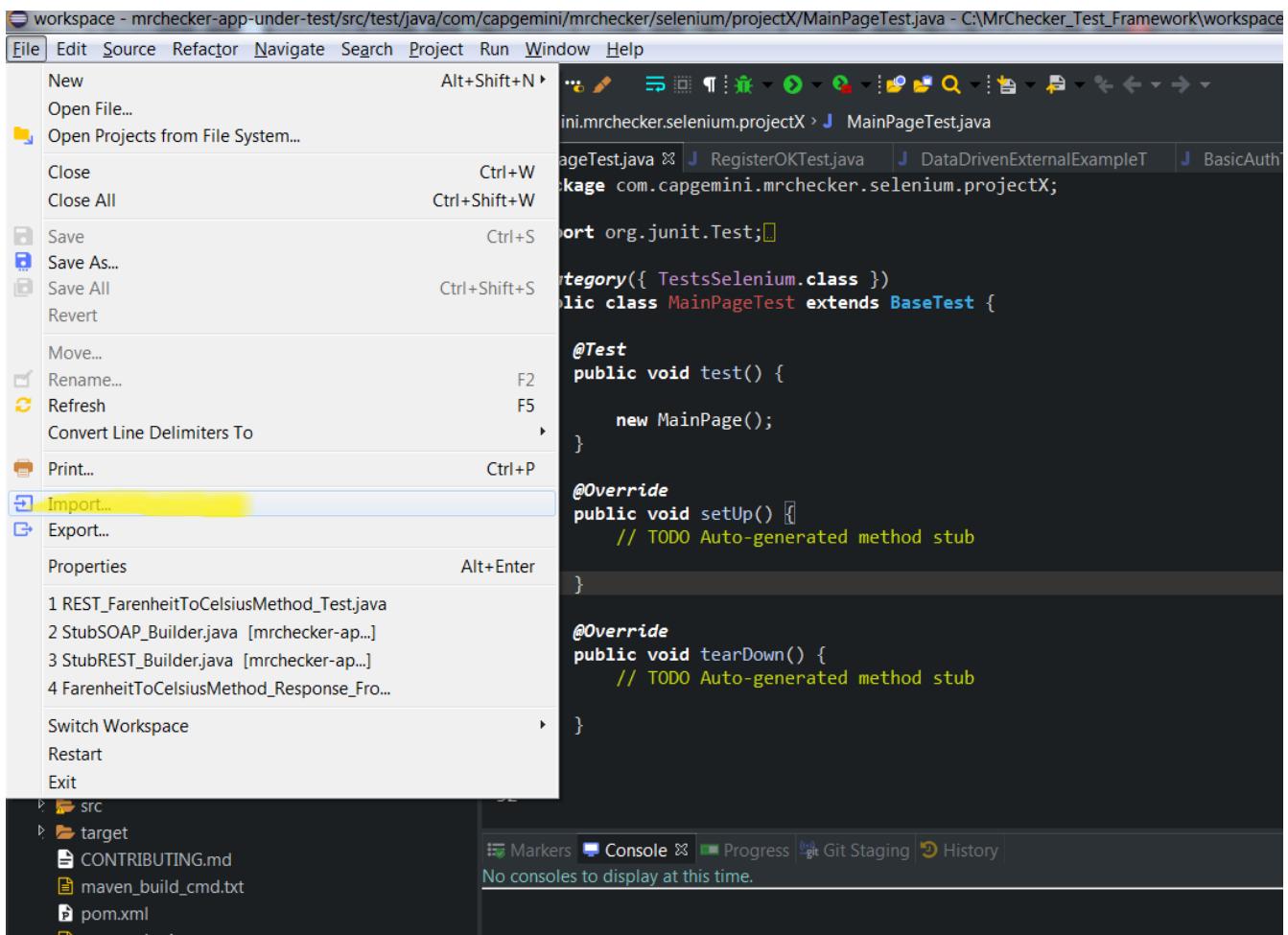
Your Product Under Test will be the following website: <http://the-internet.herokuapp.com/>

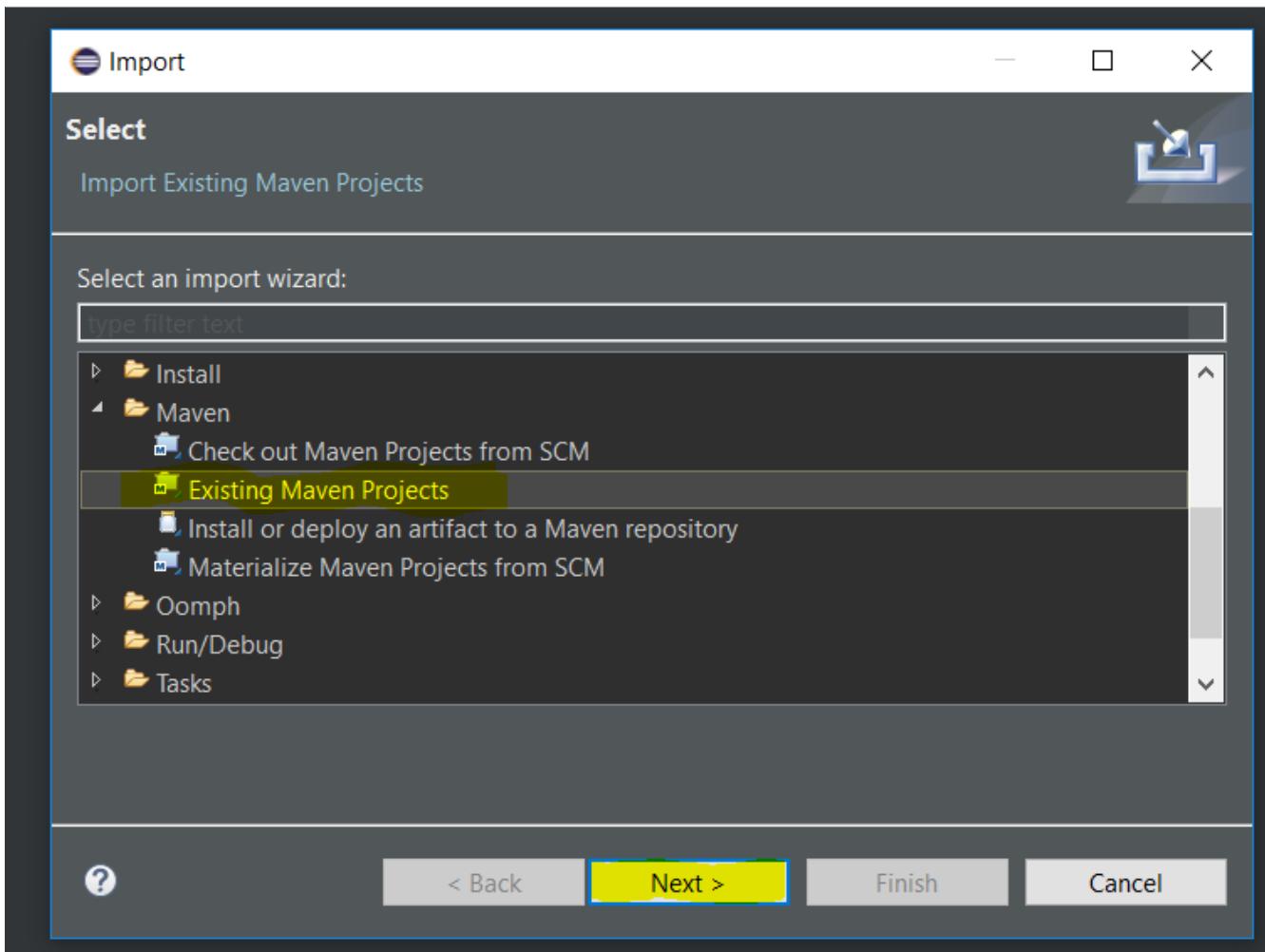
78.1. Project organization

78.1.1. Importing projects

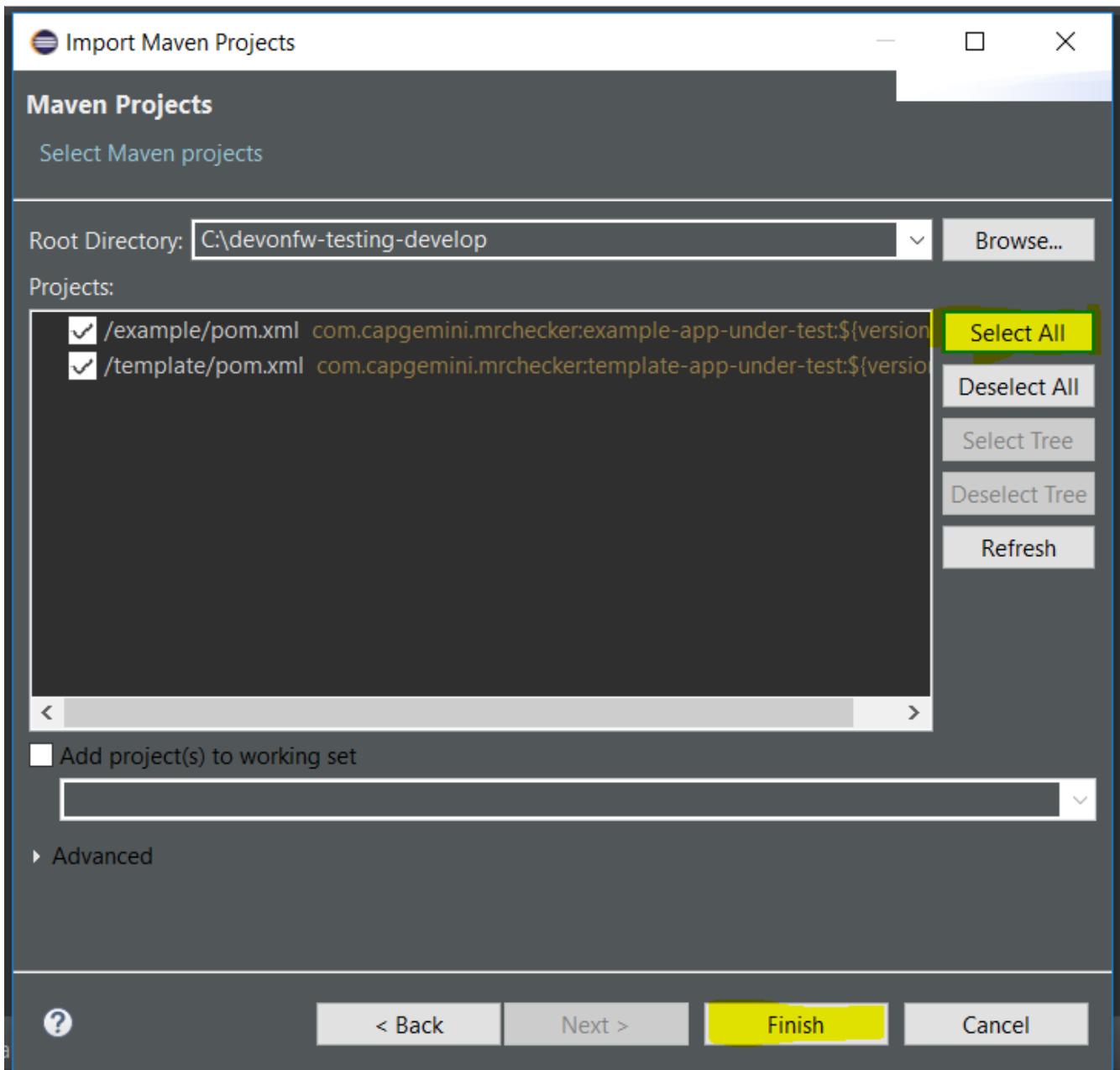
Every MrChecker project should be imported as a Maven Project.

Example from Eclipse IDE:

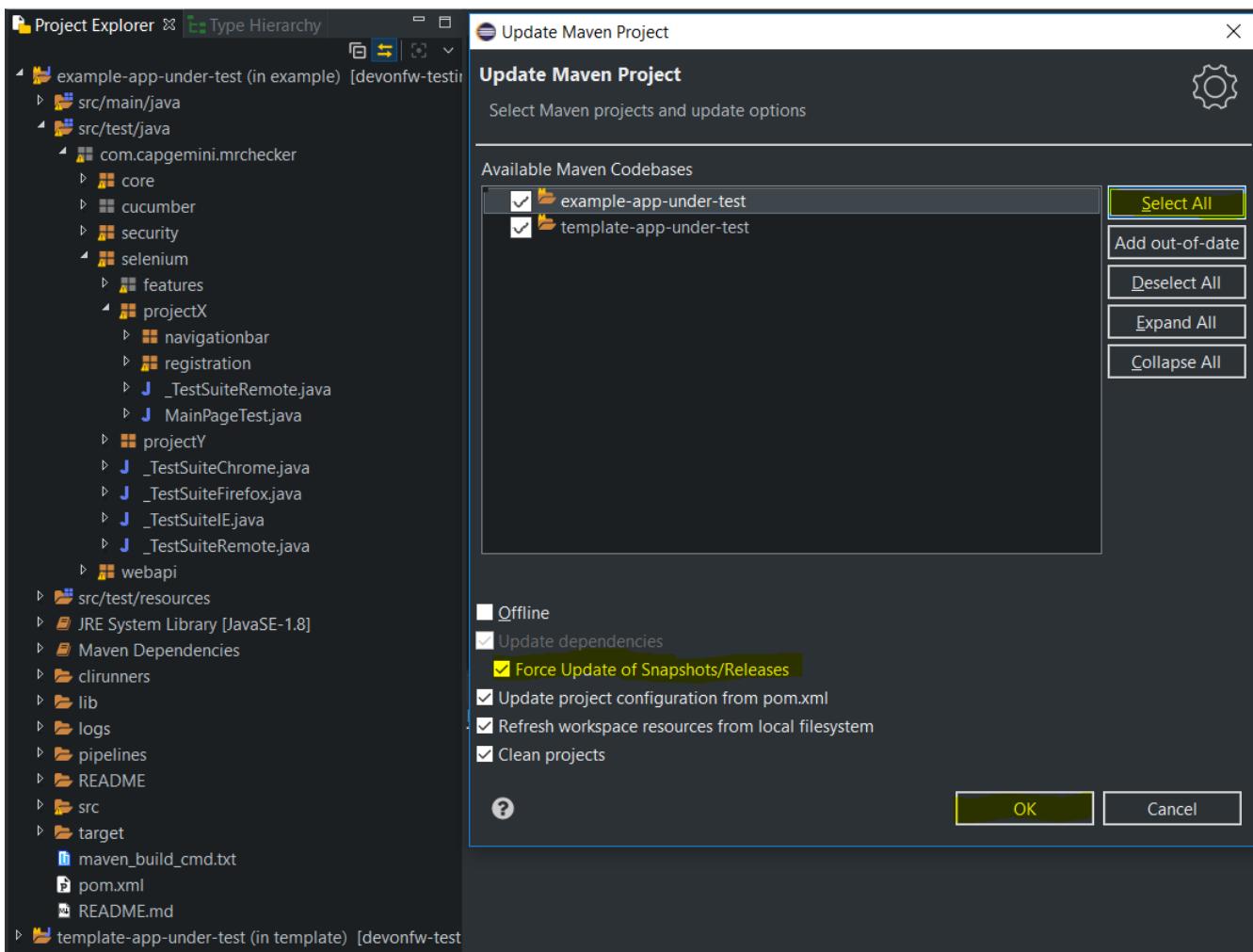




Enter the project path and select projects to import.



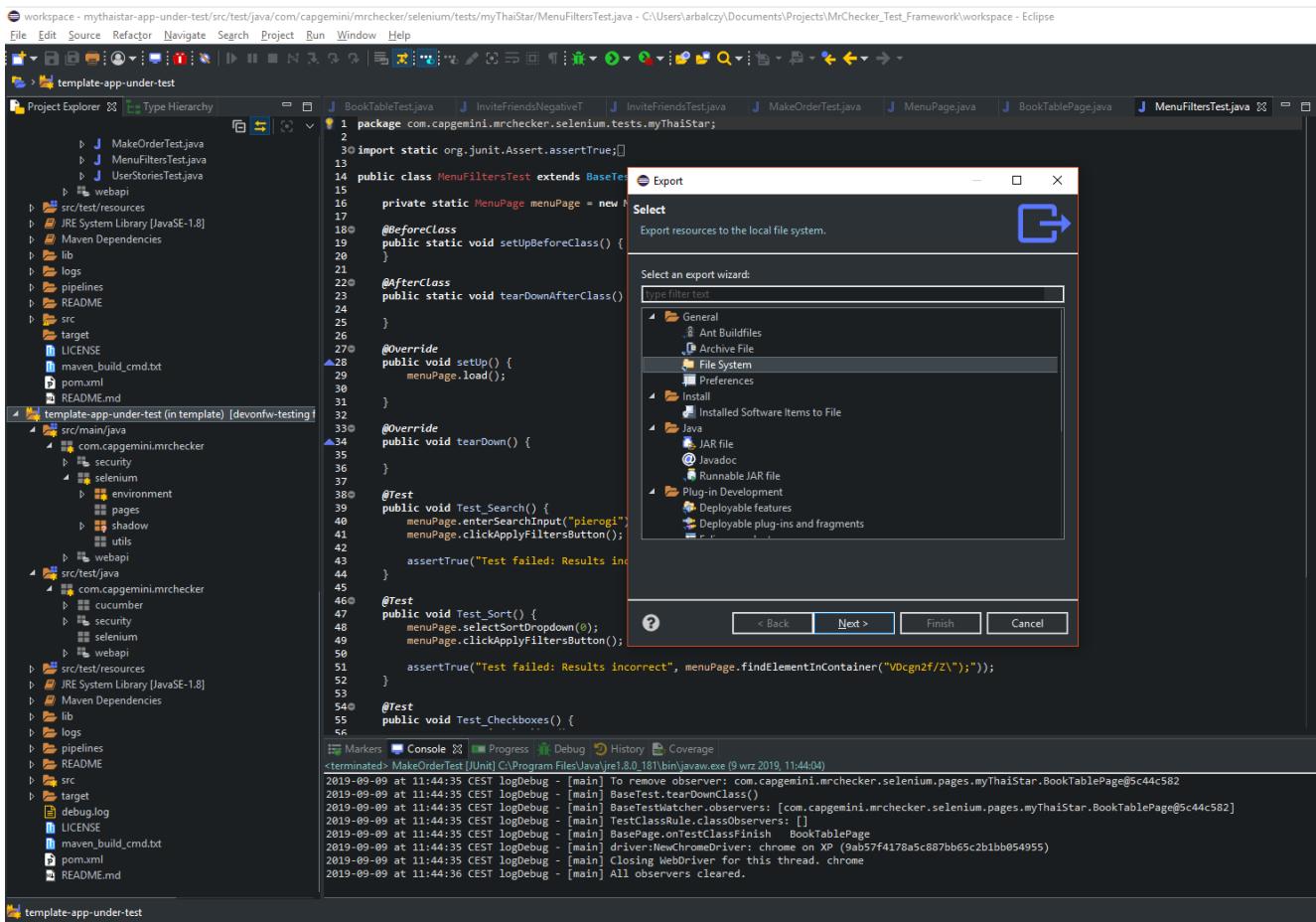
When the import is finished, update the project structure - ALT + F5



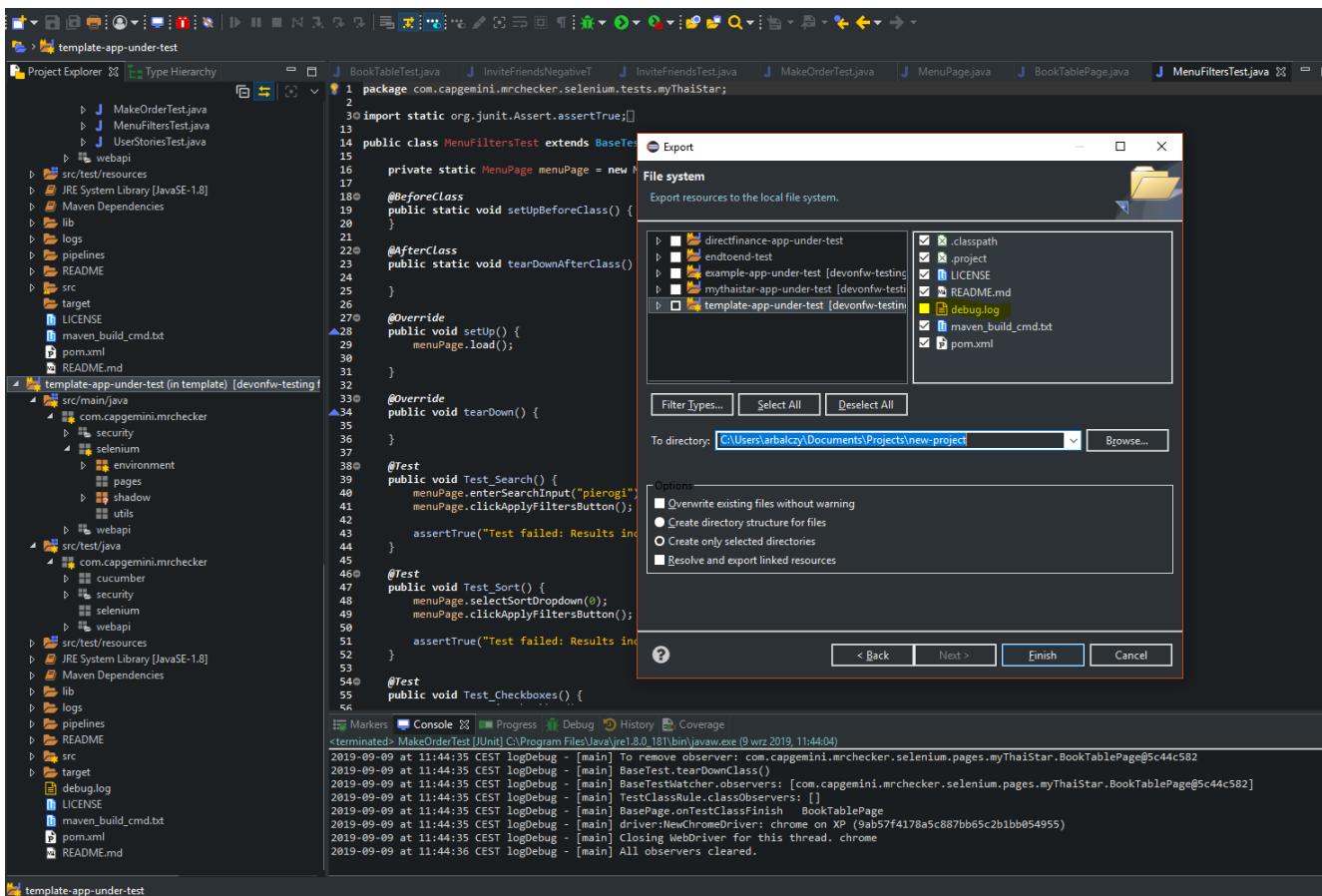
78.1.2. Exporting projects

In order to create a new standalone MrChecker project, you can use template-app-under-test and export it to the new folder:

[5] | *images/5.png*



Create a new folder for the project and enter its path. Select project and files to export:



Change project name and other properties, if necessary, in pom.xml file:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
5    4.0.0.xsd">
6    <artifactId>new-project</artifactId>
7    <groupId>com.capgemini.mrchecker</groupId>
8    <version>${version}</version>
9    <packaging>jar</packaging>
10
11   <name>MrCheckerTestFramework - List of Integration Test for Application Under Test</name>
12   <description>MrChecker Test Framework is an automated testing framework for functional a
13   <url>https://github.com/devonfw/devonfw-testing</url>
14
15
16
17   <organization>
18     <name>Capgemini - Nearshore Test Center - Poland</name>
19     <url>http://nsc.capgemini.com/nearshoretestcenter</url>
20   </organization>
21

```

Then you can import the project to the workspace and create new packages and classes.

78.1.3. Creating new packages

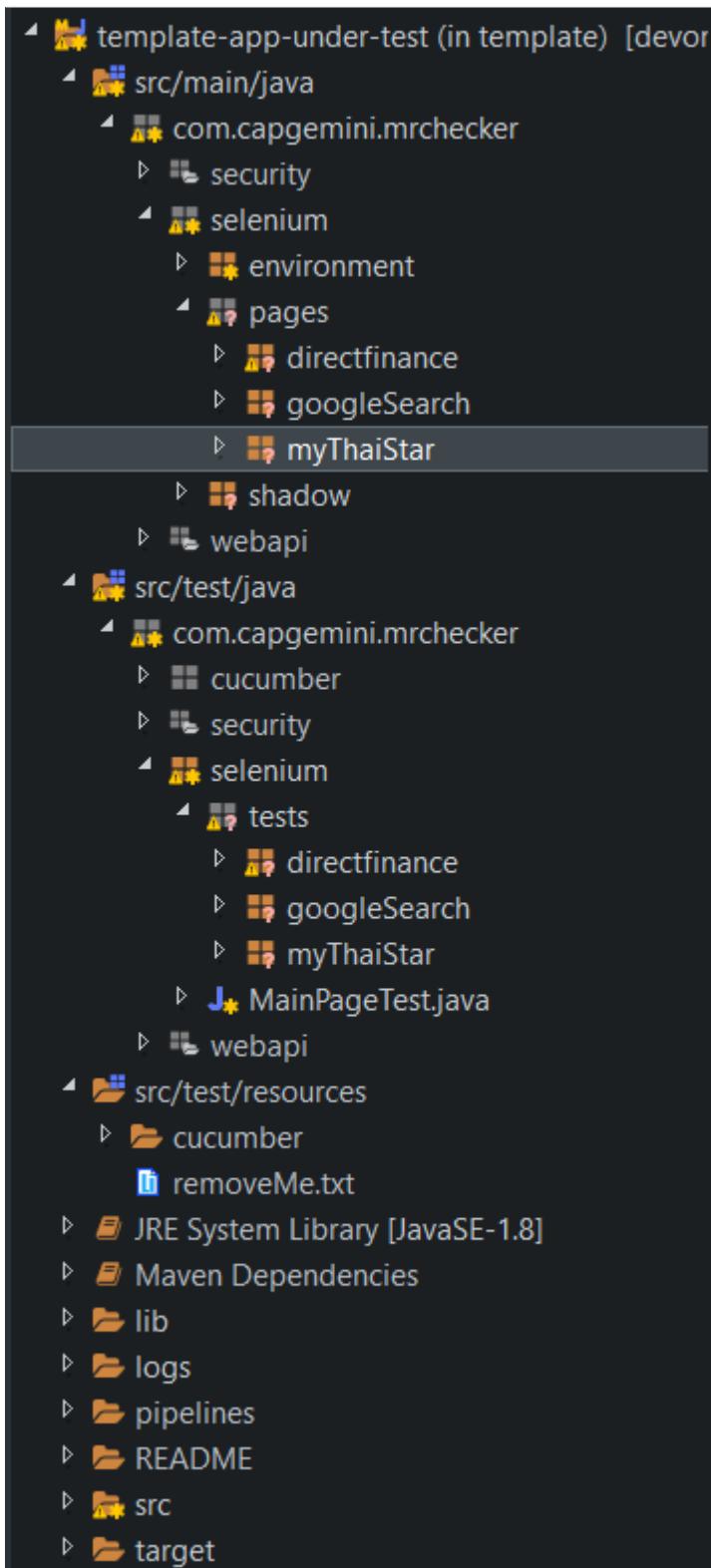
1. You will need two new packages: one for the new page classes, the other one for test classes:
 - Create a package for page classes

Open Eclipse
 Use the "Project Explorer" on the left
 Navigate to [your-project] → src/main/java → com.capgemini.mrchecker → selenium
 Right-click on "selenium"
 Click on "New" → New Package
 Name the new package "com.capgemini.mrchecker.selenium.pages.[your-product-name]"

- Create a package for test classes

Navigate to [your-project] → src/test/java → com.capgemini.mrchecker → selenium
 Right click on "selenium"
 Click on "New" → New Package
 Name the new package "com.capgemini.mrchecker.selenium.tests.[your-product-name]"

Example:



78.1.4. Creating new Page Classes

Navigate to: [your-project] → src/main/java → com.capgemini.mrchecker → selenium.pages.[your-product-name]
 Click on "New" → New Class
 Enter the name "YourPage"

Every Page Class should extend BasePage class. Import all necessary packages and override all

required methods:

- public boolean isLoaded() - returns true if the page is loaded and false if not
- public void load() - loads the page
- public String pageTitle() - returns page title

Example:

```
public class MainPage extends BasePage {

    @Override
    public boolean isLoaded() {
        return false;
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Main Page'");
    }

    @Override
    public String pageTitle() {
        return "Main Page Title";
    }
}
```

78.1.5. Creating new Test Classes

Navigate to [your-project] → src/test/java → com.cappgeminimrchecker → selenium.tests.[your-product-name]
 Click on "New" → New Class
 Enter the name "YourCaseTest"

Test classes should extend BaseTest class, import all necessary packages and override all required methods:

- public void setUp() - executes before each test
- public void tearDown() - executes after each test

Optionally, it is also possible to implement the following methods:

- @BeforeClass public static void setUpBeforeClass() - runs only once before all tests
- @AfterClass public static void tearDownAfterClass() - runs only once after performing all tests

Every test method has to be signed with "@Test" parameter.

```

public class YourCaseTest extends BaseTest {
    private static MainPage mainPage = new MainPage();

    @BeforeClass
    public static void setUpBeforeClass() {
        mainPage.load();
    }

    @AfterClass
    public static void tearDownAfterClass() {

    }

    @Override
    public void setUp() {
        if (!mainPage.isLoaded()) {
            mainPage.load();
        }
    }

    @Override
    public void tearDown() {

    }

    @Test
    public void shouldTestRunWithoutReturningError {
        ...
    }
}

```

78.1.6. Running Tests

Run the test by right-clicking on the test method → Run as → JUnit test.

78.2. Basic Tutorials

78.2.1. Basic Tests

A/B Test Control

Also known as split testing. This is a way in which businesses are able to simultaneously test and learn different versions of a page to see which text and/or functionality works best towards a desired outcome (e.g. a user action such as a click-through).

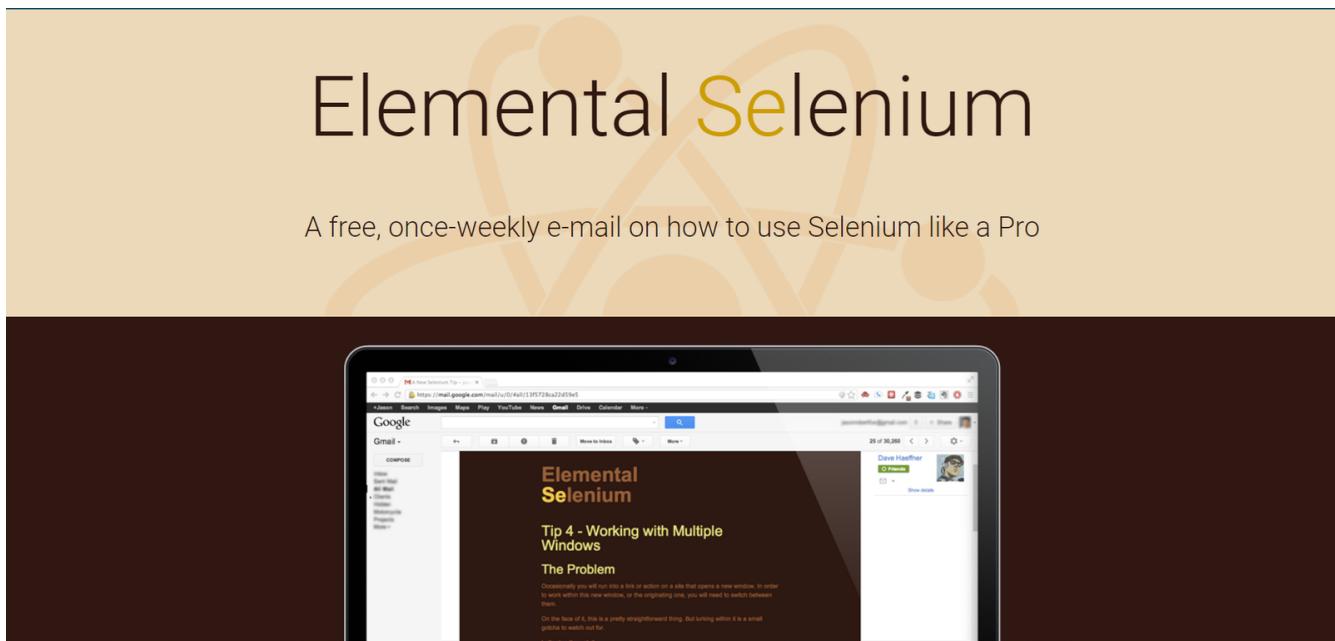


Powered by [Elemental Selenium](#)

The goal of this test is to open A/B Test subpage and redirect to another website.

Steps:

1. Open The Internet Main Page
2. Click A/B Testing link and go to A/B Test subpage
3. Click Elemental Selenium link and open it in new tab
4. Switch to Elemental Selenium page and check if it's loaded



Page Class

Create a Page class for AB Testing page. Override all the required methods:

```
public class ABtestPage extends BasePage {

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.ABTEST.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'A/B Test Control' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.ABTEST.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }
}
```

How to use Enum?

Similarly as in environmental variables case, create an enum for storing values of subURLs:

```
public enum PageSubURLsProjectYEnum {
    BASIC_AUTH("basic_auth"),
    NEW_WINDOW("windows/new"),
    WINDOW("windows"),
    CHECKBOX("checkboxes"),
    CONTEXT_MENU("context_menu"),
    KEY_PRESS("key_presses"),
    DYNAMIC_CONTENT("dynamic_content"),
    HOVERS("hovers"),
    SORTABLE_DATA_TABLES("tables"),
    REDIRECT("redirector"),
    JAVASCRIPT_ALERTS("javascript_alerts"),
    CHALLENGING_DOM("challenging_dom"),
    STATUS_CODES("status_codes"),
    LOGIN("login"),
    ABTEST("abtest"),
    BROKEN_IMAGES("broken_images"),
    DROPDOWN("dropdown"),
    HORIZONTAL_SLIDER("horizontal_slider"),
    DOWNLOAD("download"),
    FORGOT_PASSWORD("forgot_password"),
    FORGOT_PASSWORD_EMAIL_SENT("email_sent"),
    EXIT_INTENT("exit_intent"),
    DYNAMIC_LOADING("dynamic_loading"),
    DISAPPEARING_ELEMENTS("disappearing_elements"),
    DRAG_AND_DROP("drag_and_drop"),
    DYNAMIC_CONTROLS("dynamic_controls"),
    UPLOAD("upload"),
    FLOATING_MENU("floating_menu"),
    FRAMES("frames"),
    GEOLOCATION("geolocation"),
    INFINITE_SCROLL("infinite_scroll"),
    JQUERY_UI("jqueryui/menu"),
    JAVASCRIPT_ERROR("javascript_error"),
    LARGE_AND_DEEP_DOM("large"),
    NESTED_FRAMES("nested_frames"),
    NOTIFICATION_MESSAGE("notification_message"),
    DOWNLOAD_SECURE("download_secure"),
    SHIFTING_CONTENT("shifting_content"),
    SLOW_RESOURCES("slow"),
    TYPOS("typos"),
    WYSIWYGEDITOR("tinymce");

    /*
     * Sub URLs are used as real locations in the test environment
     */
}
```

```
private String subURL;

private PageSubURLsProjectYEnum(String subURL) {
    this.subURL = subURL;
}

;

private PageSubURLsProjectYEnum() {

}

@Override
public String toString() {
    return getValue();
}

public String getValue() {
    return subURL;
}

}
```

Instead of mapping data from an external file, you can store and access them directly from the enum class:

```
PageSubURLsProjectYEnum.ABTEST.getValue()
```

Selector

In this test case you need selector for only one page element:

```
private static final By elementalSeleniumLinkSelector = By.cssSelector("div > div > a");
```

Page methods

You need two methods for performing page actions:

```

/**
 * Clicks 'Elemental Selenium' link at the bottom of the page.
 *
 * @return ElementalSeleniumPage object.
 */
public ElementalSeleniumPage clickElementalSeleniumLink() {
    getDriver().findElementDynamic(elementalSeleniumLinkSelector)
        .click();
    getDriver().waitForPageLoaded();
    return new ElementalSeleniumPage();
}

/**
 * Switches window to the next one - different than the current.
 */
public void switchToNextTab() {
    ArrayList<String> tabsList = new
ArrayList<String>(getDriver().getWindowHandles());
    getDriver().switchTo()
        .window(tabsList.get(1));
}

```

Elemental Selenium Page Class

To return new Elemental Selenium Page object, implement its class. You only need to write basic methods to check if the page is loaded. There is no need to interact with objects on the site:

```

public class ElementalSeleniumPage extends BasePage {

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(GetEnvironmentParam.ELEMENTAL_SELENIUM_PAGE.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Elemental Selenium' page.");
        getDriver().get(GetEnvironmentParam.ELEMENTAL_SELENIUM_PAGE.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }
}

```

Test Class

Create a Test class and write a @Test method to execute the scenario:

```
@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class ABtestingTest extends TheInternetBaseTest {

    private static ABtestPage abTestPage;

    @Test
    public void shouldOpenElementalSeleniumPageWhenClickElementalSeleniumLink() {

        logStep("Click Elemental Selenium link");
        ElementalSeleniumPage elementalSeleniumPage =
abTestPage.clickElementalSeleniumLink();

        logStep("Switch browser's tab to newly opened one");
        abTestPage.switchToNextTab();

        logStep("Verify if Elemental Selenium Page is opened");
        assertTrue("Unable to open Elemental Selenium page",
elementalSeleniumPage.isLoaded());
    }

}
```

Assert

Asserts methods are used for creating test pass or fail conditions. The optional first parameter is a message which will be displayed in the test failure description.

- assertTrue(boolean condition) - test passes if condition returns true
- assertFalse(boolean condition) - test passes if condition returns false

Also, add the @BeforeClass method to open the tested page:

```
@BeforeClass
public static void setUpBeforeClass() {
    abTestPage = shouldTheInternetPageBeOpened().clickABtestingLink();
    logStep("Verify if ABTest page is opened");
    assertTrue("Unable to open ABTest page", abTestPage.isLoaded());
}
```

@BeforeClass method executes only once before all other @Test cases in the class. There is also a possibility to create a @AfterClass method which is performed also once after all @Test cases.

You don't need to implement @setUp and @tearDown methods because they're already in

The InternetBaseTest class which you extend.

Categories

You can group tests in categories. It's useful when running many tests at once. Use this parameter:

```
@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
```

Then create an interface representing each category. Example:

```
public interface TestsSelenium {  
    /* For test which are testing web pages considering UI (user interface) and using  
    selenium webdriver */  
}
```

To run a test from specified category create Test Suite class:

```
@RunWith(WildcardPatternSuite.class) //search for test files under /src/test/java  
@IncludeCategories({ TestsChrome.class }) // search all test files with category  
TestsChrome.class  
@ExcludeCategories({ TestsLocal.class, TestsNONParallel.class }) //exclude all test  
files with category TestsLocal.class and TestsNONParallel.class  
@SuiteClasses({ "../**/*Test.class" }) //search only test files, where file name ends  
with <anyChar/s>Test.class  
  
public class _TestSuiteChrome {  
  
}
```

You can run a Test Suite as a JUnit test.

Welcome to the Available Exam page.

Available Exam

- A/B Testing
- Add/Remove Elements
- Basic Auth (User and pass: admin)
- Broken Images
- Challenging DOM
- Checkboxes
- Context Menu
- Digest Authentication (user and pass: admin)
- Disappearing Elements
- Drag and Drop
- Dropdown
- Dynamic Content
- Dynamic Controls
- Dynamic Loading
- Entry Ad
- Exit Intent
- File Download
- File Upload
- Floating Menu
- Forgot Password
- Form Authentication

Zaloguj się
http://the-internet.herokuapp.com/
Twoje połączenie z tą stroną nie jest prywatne

Nazwa użytkownika:
Hasło:

Zaloguj się Anuluj

In this test case, the goal is to pass username and password authorization and login to the next page.

Steps:

1. Open The Internet Main Page
2. Click on Basic Auth link
3. Open pop-up login window
4. Enter valid username and password
5. Open next subpage and verify if the user logged in successfully.

Page Class

Create a page class which represents Basic Auth subpage after proper login.

[Fork me on GitHub](#)

Basic Auth

Congratulations! You must have the proper credentials.

Powered by [Elemental Selenium](#)

Override all the required methods:

```

public class BasicAuthPage extends BasePage {

    public BasicAuthPage() {
    }

    public BasicAuthPage(String login, String password) {
        this.enterLoginAndPasswordByUrl(login, password);
    }

    @Override
    public boolean isLoaded() {
        return true;
    }

    @Override
    public void load() {
        BFLogger.logDebug("load");
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }
}

```

In order to verify a login, create a selector to access the visible message.

```

private static final By selectorTextMessage = By.cssSelector("#content > div > p");
Then create a method to get message value:

```

```

/**
 *      Returns message displayed by system after user's log in.
 *      @return String object representing message displayed by system after user's log
in
 */
public String getMessageValue() {
    return getDriver().findElementDynamic(selectorTextMessage)
        .getText();
}

```

Also, create a method to access the pop-up login window and enter user credentials:

```

/**
 * Authenticates user using standard simple authentication popup.
 *
 * @param login    User's login
 * @param password User's password
 * @throws AWTException
 * @throws InterruptedException
 */
public void enterLoginAndPassword(String login, String password) throws
AWTException, InterruptedException {
    Robot rb = new Robot();

    Thread.sleep(2000);

    StringSelection username = new StringSelection(login);
    Toolkit.getDefaultToolkit()
        .getSystemClipboard()
        .setContents(username, null);
    rb.keyPress(KeyEvent.VK_CONTROL);
    rb.keyPress(KeyEvent.VK_V);
    rb.keyRelease(KeyEvent.VK_V);
    rb.keyRelease(KeyEvent.VK_CONTROL);

    rb.keyPress(KeyEvent.VK_TAB);
    rb.keyRelease(KeyEvent.VK_TAB);
    Thread.sleep(2000);

    StringSelection pwd = new StringSelection(password);
    Toolkit.getDefaultToolkit()
        .getSystemClipboard()
        .setContents(pwd, null);
    rb.keyPress(KeyEvent.VK_CONTROL);
    rb.keyPress(KeyEvent.VK_V);
    rb.keyRelease(KeyEvent.VK_V);
    rb.keyRelease(KeyEvent.VK_CONTROL);

    rb.keyPress(KeyEvent.VK_ENTER);
    rb.keyRelease(KeyEvent.VK_ENTER);
    Thread.sleep(2000);
}

```

Robot class

Creating a Robot object allows performing basic system actions such as pressing keys, moving the mouse or taking screenshots. In this case, it's used to paste login and password text from the clipboard using 'Ctrl + V' shortcut, go to the next field using 'Tab' key and submit by clicking 'Enter'.

Toolkit

Static class Toolkit can perform basic window actions such as scrolling to a specified position or moving context between components. In this case, it's used to set clipboard content to username and password value.

```
Thread.sleep(long millis)
```

Web drivers like Selenium perform actions much faster than the normal user. This may cause unexpected consequences e.g. some elements may not be loaded before the driver wants to access them. To avoid this problem you can use Thread.sleep(long millis) to wait given time and let browser load wanted component.

BEWARE: Using Thread.sleep(long millis) is not the recommended approach. Selenium driver gives methods to wait for a specified element to be enabled or visible with a timeout parameter. This is a more stable and effective way. Also, method waitForPageLoaded() will not solve that issue because it only waits for the ready state from the browser while some javascript actions might be performed after that.

Test Class

Create a Test class and write a @Test method to execute the scenario. Save parameters as class fields:

```

@Category({ TestsLocal.class, TestsNONParallel.class })
public class BasicAuthTest extends TheInternetBaseTest {

    private static BasicAuthPage basicAuthPage;

    private String login      = "admin";
    private String password = "admin";
    private String message   = "Congratulations! You must have the proper
credentials./";

    @Test
    public void shouldUserLogInWithValidCredentials() throws InterruptedException,
AWTException {
        basicAuthPage = shouldTheInternetPageBeOpened().clickBasicAuthLink();

        logStep("Enter login and password");
        basicAuthPage.enterLoginAndPassword(login, password);

        logStep("Verify if user logged in successfully");
        assertEquals("Unable to login user with valid credentials", message,
basicAuthPage.getMessageValue());
    }

    @Override
    public void tearDown() {
        logStep("Navigate back to The-Internet page");
        theInternetPage.load();
    }
}

```

`assertEquals(Object expected, Object actual)` - test passes if parameters are equal .

Alternative scenario:

There is also a possibility to log in with credentials as a part of URL: http://login:password@the-internet.herokuapp.com/basic_auth

Another page class method:

```
/**
 * Authenticates user passing credentials into URL.
 *
 * @param login User's login
 * @param password User's password
 */
private void enterLoginAndPasswordByUrl(String login, String password) {
    getDriver().get("http://" + login + ":" + password + "@" + "the-
internet.herokuapp.com/" +
        PageSubURLsProjectYEnum.BASIC_AUTH.getValue());
}
```

Another test class method:

```
@Test
public void shouldUserLogInWithValidCredentialsSetInURL() {
    logStep("Enter user's credentials into URL to log in");
    basicAuthPage = new BasicAuthPage(login, password);

    logStep("Verify if user logged in successfully");
    assertEquals("Unable to login user with valid credentials", message,
        basicAuthPage.getMessageValue());
}
```

After running test class as a JUnit test, both test cases will be performed.

This test goal is to check the dimensions of broken images on the subpage.



Steps:

1. Open The Internet Main Page
2. Click Broken Image link and go to Broken Image subpage
3. Get the 3 images' dimensions and compare them with expected values

Page Class

In this case, create an array of selectors to access images by index number:

```
public class BrokenImagePage extends BasePage {
```

```
private static final By[] selectorsImages = { By.cssSelector("div > img:nth-child(2)"),
                                             By.cssSelector("div > img:nth-child(3)"),
                                             By.cssSelector("div > img:nth-child(4)") };

@Override
public boolean isLoaded() {
    getDriver().waitForPageLoaded();
    return getDriver().getCurrentUrl()
        .contains(PageSubURLsProjectYEnum.BROKEN_IMAGES.getValue());
}

@Override
public void load() {
    BFLogger.logDebug("Load 'Broken Images' page.");
    getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
        PageSubURLsProjectYEnum.BROKEN_IMAGES.getValue());
    getDriver().waitForPageLoaded();
}

@Override
public String pageTitle() {
    return getActualPageTitle();
}

/**
 * Returns an image height in pixels.
 *
 * @param imageIndex An index of given image.
 * @return Height of an image in pixels.
 */
public int getImageHeight(int imageIndex) {
    return getImageDimension(imageIndex).getHeight();
}

/**
 * Returns an image width in pixels.
 *
 * @param imageIndex An index of given image.
 * @return Width of an image in pixels.
 */
public int getImageWidth(int imageIndex) {
    return getImageDimension(imageIndex).getWidth();
}

private Dimension getImageDimension(int imageIndex) {
    return getDriver().findElementDynamic(selectorsImages[imageIndex])
        .getSize();
}

}
```

Test Class

Create @Test and @BeforeClass methods. Save expected images' dimensions in class fields:

```

@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class BrokenImagesTest extends TheInternetBaseTest {

    private static BrokenImagePage brokenImagePage;

    private final int expectedHeight = 90;
    private final int expectedWidth = 120;

    @BeforeClass
    public static void setUpBeforeClass() {
        brokenImagePage = shouldTheInternetPageBeOpened().clickBrokenImageLink();

        logStep("Verify if Broken Image page is opened");
        assertTrue("Unable to open Broken Image page", brokenImagePage.isLoaded());
    }

    @Test
    public void shouldImageSizesBeEqualToExpected() {
        for (int i = 0; i < 3; i++) {
            logStep("Verify size of image with index: " + i);
            assertEquals("Height of image with index: " + i + " is incorrect",
expectedHeight,
                    brokenImagePage.getImageHeight(i));
            assertEquals("Width of image with index: " + i + " is incorrect",
expectedWidth,
                    brokenImagePage.getImageWidth(i));
        }
    }
}

```

The test will pass if every image had the correct width and height.

This case goal is to find out how to create stable selectors.

In the browser's developer mode, you can see how the page is built. Notice, that buttons' IDs change after click and values in the table haven't got unique attributes, which might be helpful in order to find them.

Challenging DOM

The hardest part in automated web testing is finding the best locators (e.g., ones that well named, unique, and unlikely to change). It's more often than not that the application you're testing was not built with this concept in mind. This example demonstrates that with unknowns (the table with no helpful locators) and a canvas element.

a93c03b160-9a55-0137-77c6-6ed6
d3cd3fb1.button 74.88 x 45

	Lorem	Ipsum	Dolor	Sit	Amet	Diceret	Action
luvaret0	Aperian0	Adipisci0	Definiebas0	Consequuntur0	Phaedrum0	edit delete	
luvaret1	Aperian1	Adipisci1	Definiebas1	Consequuntur1	Phaedrum1	edit delete	
luvaret2	Aperian2	Adipisci2	Definiebas2	Consequuntur2	Phaedrum2	edit delete	
luvaret3	Aperian3	Adipisci3	Definiebas3	Consequuntur3	Phaedrum3	edit delete	
luvaret4	Aperian4	Adipisci4	Definiebas4	Consequuntur4	Phaedrum4	edit delete	
luvaret5	Aperian5	Adipisci5	Definiebas5	Consequuntur5	Phaedrum5	edit delete	
luvaret6	Aperian6	Adipisci6	Definiebas6	Consequuntur6	Phaedrum6	edit delete	
luvaret7	Aperian7	Adipisci7	Definiebas7	Consequuntur7	Phaedrum7	edit delete	
luvaret8	Aperian8	Adipisci8	Definiebas8	Consequuntur8	Phaedrum8	edit delete	
luvaret9	Aperian9	Adipisci9	Definiebas9	Consequuntur9	Phaedrum9	edit delete	

Answer: 2225

Powered by Elemental Selenium

Fork me on GitHub

DOM - Document Object Model

HTML DOM is a model of the page created by the browser. The page could be represented as the tree of objects. Read more.

To create locators you can use element attributes such as id, class name etc.

In this case, since there are no unique attributes, the best approach is to use HTML document structure and identify page elements by their place in an object hierarchy.

Page Class

```
public class ChallengingDomPage extends BasePage {

    private final By selectorTableRows = By.cssSelector(".large-10 > table > tbody > tr");
    private final By selectorFirstButton = By.cssSelector(".large-2.columns > .button:nth-child(1)");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.CHALLENGING_DOM.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Challenging DOM' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.CHALLENGING_DOM.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Returns table text content as a list of String objects.
     *
     * @return A list of table values.
     */
    public List<String> getTableValues() {
        return JsoupHelper.findTexts(selectorTableRows);
    }

    /**
     * Clicks top button on the page from available button set.
     */
    public void clickFirstButton() {
        getDriver().elementButton(selectorFirstButton)
            .click();
        getDriver().waitForPageLoaded();
    }

}
```

Jsoup Helper

Jsoup Helper is the tool which helps to parse HTML document and get searched values. This is especially useful when values are organized in a generic structure such as a table.

`JsoupHelper.findTexts(By selector)` - this method returns text content of a table as a list of Strings

Test Class

Steps:

1. Open The Internet Main Page
2. Click Challenging DOM link and go to Challenging DOM subpage
3. Get and save table values
4. Click the first button
5. Get table values again
6. Compare table values before and after button click

```

@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class ChallengingDomTest extends TheInternetBaseTest {

    private static ChallengingDomPage challengingDomPage;

    @BeforeClass
    public static void setUpBeforeClass() {
        challengingDomPage =
shouldTheInternetPageBeOpened().clickChallengingDomLink();

        logStep("Verify if Challenging Dom page is opened");
        assertTrue("Unable to open Challenging Dom page",
challengingDomPage.isLoaded());
    }

    @Test
    public void shouldValuesInTableCellsStayUnchangedAfterClick() {

        logStep("Get table values (before click any button)");
        List<String> tableValuesBeforeClick = challengingDomPage.getTableValues();

        logStep("Click first button");
        challengingDomPage.clickFirstButton();

        logStep("Get table values (after click first button)");
        List<String> tableValuesAfterClick = challengingDomPage.getTableValues();

        logStep("Verify equality of table values before and after click");
        assertEquals("Values from table cells were changed after click",
tableValuesBeforeClick,
                tableValuesAfterClick);
    }

}

```

Because values in the table don't change, the test should pass if object locators are solid.

In this example, you will learn how to test checkboxes on the page.

Checkboxes

- checkbox 1
- checkbox 2

Powered by [Elemental Selenium](#)

Fork me on GitHub

A checkbox is a simple web element which can be selected or unselected by clicking on it.

Steps:

1. Open The Internet Main Page
2. Click Checkboxes link and go to Checkboxes page
3. Test if the first checkbox is unchecked
4. Select the first checkbox
5. Test if the first checkbox is checked
6. Test if the second checkbox is checked
7. Unselect second checkbox
8. Test if the second checkbox is unchecked

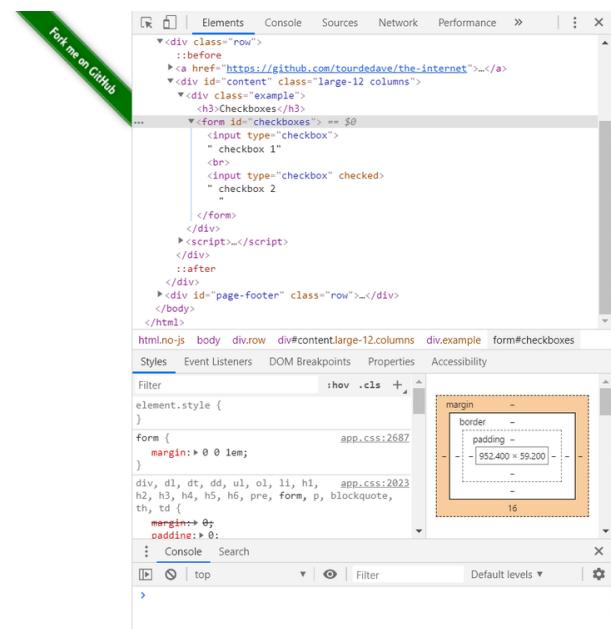
Page Class

Because both checkboxes are in one form, it's possible to locate them by one selector and then access each individual one by index.

Checkboxes

checkbox 1
 checkbox 2

Powered by Elemental Selenium



```
public class CheckboxesPage extends BasePage {

    private final static By checkboxesFormSelector = By.cssSelector("#checkboxes");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.CHECKBOX.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Checkboxes' page.");
    }
}
```

```

        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.CHECKBOX.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Verifies if checkbox form is visible on the page.
     *
     * @return true if checkboxes are present and displayed on the page
     */
    public boolean isElementCheckboxesFormVisible() {
        return getDriver().elementCheckbox(checkboxesFormSelector)
            .isDisplayed();
    }

    /**
     * Verifies if given checkbox is selected or not.
     *
     * @param index The index of given checkbox
     * @return true if given checkbox is selected
     */
    public boolean isCheckboxSelected(int index) {
        return getDriver().elementCheckbox(checkboxesFormSelector)
            .isCheckBoxSetByIndex(index);
    }

    /**
     * Selects given checkbox. Unselects, if it is already selected.
     *
     * @param index The index of given checkbox
     */
    public void selectCheckbox(int index) {
        CheckBox checkbox = getDriver().elementCheckbox(checkboxesFormSelector);
        if (isCheckboxSelected(index)) {
            checkbox.unsetCheckBoxByIndex(index);
        } else {
            checkbox.setCheckBoxByIndex(index);
        }
    }

}

```

CheckBox

CheckBox class contains a method to perform actions on checkboxes such as setting and unsetting

or verifying if the specified box is checked. Use method `elementCheckbox(By selector)` to create CheckBox Object.

Test Class

```
@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class CheckboxesTest extends TheInternetBaseTest {

    private static CheckboxesPage checkboxesPage;

    @Override
    public void setUp() {
        checkboxesPage = shouldTheInternetPageBeOpened().clickCheckboxesLink();

        logStep("Verify if Checkboxes page is opened");
        assertTrue("Unable to open Checkboxes page", checkboxesPage.isLoaded());
    }

    @Test
    public void shouldCheckboxBeSelectedAfterClick() {

        logStep("Verify if first checkbox is not selected");
        assertFalse("The checkbox is selected", checkboxesPage.isCheckboxSelected(0));

        logStep("Select first checkbox");
        checkboxesPage.selectCheckbox(0);

        logStep("Verify if first checkbox is selected");
        assertTrue("The checkbox is not selected",
        checkboxesPage.isCheckboxSelected(0));
    }

    @Test
    public void shouldCheckboxBeUnselectedAfterClick() {

        logStep("Verify if second checkbox is selected");
        assertTrue("The checkbox is not selected",
        checkboxesPage.isCheckboxSelected(1));

        logStep("Select second checkbox");
        checkboxesPage.selectCheckbox(1);

        logStep("Verify if second checkbox is not selected");
        assertFalse("The checkbox is selected", checkboxesPage.isCheckboxSelected(1));
    }
}
```

After running Test Class both `@Test` cases will be performed. Before each one, overrode `setUp`

method will be executed.

This case will show how to test changing website content.

Disappearing Elements

This example demonstrates when elements on a page change by disappearing/reappearing on each page load.

[Home](#) [About](#) [Contact Us](#) [Portfolio](#)

Powered by Elemental Selenium

After refreshing page (F5) a few times, a new element should appear:

Disappearing Elements

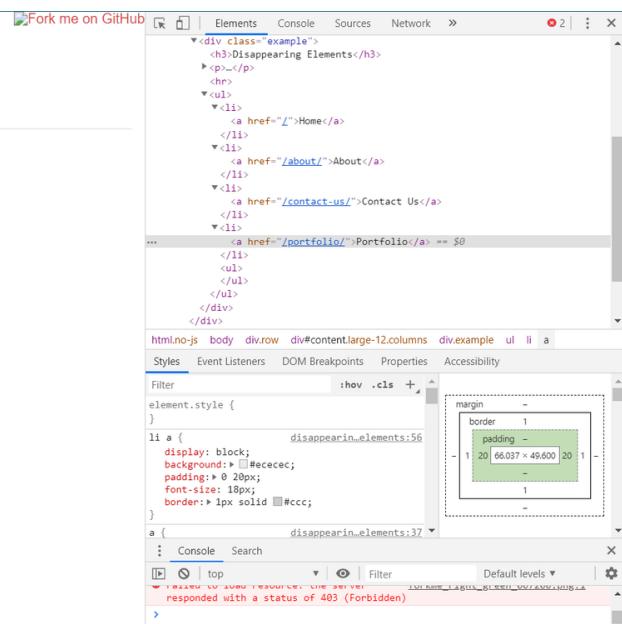
This example demonstrates when elements on a page change by disappearing/reappearing on each page load.

[Home](#) [About](#) [Contact Us](#) [Portfolio](#) [Gallery](#)

Powered by Elemental Selenium

Then, after another couple of refreshes, it should disappear.

You can check in developer mode that Gallery element does not exist in HTML document either, until appearing on the page. The element is created by Javascript.



Steps:

1. Load The Internet Main Page
2. Click Disappearing Elements link and go to that subpage
3. Check if Menu Buttons exist on the page
4. Refresh the page until a new element appears
5. Check if Gallery Button exists
6. Check if the number of buttons equals the expected value
7. Refresh the page until an element disappears
8. Check if Gallery Button does not exist
9. Check if the number of buttons is smaller than before

Page Class

```
public class DisappearingElementsPage extends BasePage {

    private static final By selectorGalleryMenuButton = By.cssSelector("li > a[href*=gallery]");
    private static final By selectorMenuButtons      = By.cssSelector("li");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.DISAPPEARING_ELEMENTS.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Disappearing Elements' page.");
    }
}
```

```

        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.DISAPPEARING_ELEMENTS.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Returns a number of WebElements representing menu buttons.
     *
     * @return A number of WebElements.
     */
    public int getNumberOfMenuButtons() {
        return getDriver().findElementDynamics(selectorMenuButtons)
            .size();
    }

    /**
     * Returns WebElement representing disappearing element of menu.
     *
     * @return Disappearing WebElement if visible, null otherwise.
     */
    public WebElement getGalleryMenuElement() {
        return getDriver().findElementQuietly(selectorGalleryMenuButton);
    }

    /**
     * Refreshes web page as many times as it is required to appear/disappear menu
     * button
     * WebElement.
     *
     * @param shouldAppear Determines if element should appear (true) or disappear
     * (false).
     */
    public void refreshPageUntilWebElementAppears(boolean shouldAppear) {
        int numberOfAttempts = 5;
        int counter = 0;
        while (!isVisibilityAsExpected(shouldAppear) ||
isMaxNumberOfAttemptsReached(counter++,
            numberOfAttempts)) {
            refreshPage();
        }
    }

    /**
     * Verify if visibility of Gallery button is the same as expected
     *
     * @param expected Determines if element should be visible (true) or not visible
    
```

```
(false).  
    */  
    private boolean isVisibilityAsExpected(boolean expected) {  
        boolean isVisibilityDifferentThanExpected = isGalleryMenuElementVisible() ^  
expected;  
        return !isVisibilityDifferentThanExpected;  
    }  
  
    private boolean isGalleryMenuElementVisible() {  
        boolean result = false;  
        WebElement gallery = getGalleryMenuElement();  
        if (gallery != null)  
            result = gallery.isDisplayed();  
        return result;  
    }  
  
    private boolean isMaxNumberOfAttemptsReached(int attemptNo, int  
maxNumberOfAttempts) {  
        return attemptNo == maxNumberOfAttempts;  
    }  
}
```

`findElementQuietly(By selector)` works similar as `findElementDynamics(By selector)` but won't throw an exception if an element wasn't found. In this case, the searched `WebElement` will have a `NULL` value.

Test Class

```

@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class DisappearingElementsTest extends TheInternetBaseTest {

    private static final int totalNumberOfMenuButtons = 5;
    private static DisappearingElementsPage disappearingElementsPage;
    private static int numberOfMenuButtons = 0;

    @BeforeClass
    public static void setUpBeforeClass() {
        disappearingElementsPage =
            shouldTheInternetPageBeOpened().clickDisappearingElementsLink();

        logStep("Verify if Disappearing Elements page is opened");
        assertTrue("Unable to open Disappearing Elements page",
                   disappearingElementsPage.isLoaded());

        logStep("Verify if menu button elements are visible");
        numberOfMenuButtons = disappearingElementsPage.getNumberOfMenuButtons();
        assertTrue("Unable to display menu", numberOfMenuButtons > 0);
    }

    @Test
    public void shouldMenuItemAppearAndDisappearAfterRefreshTest() {
        logStep("Click refresh button until menu button appears");
        disappearingElementsPage.refreshPageUntilWebElementAppears(true);

        logStep("Verify if menu button element appeared");
        assertNotNull("Unable to disappear menu button element",
                      disappearingElementsPage.getGalleryMenuElement());
        assertEquals("The number of button elements after refresh is incorrect",
                    totalNumberOfMenuButtons,
                    disappearingElementsPage.getNumberOfMenuButtons());

        logStep("Click refresh button until menu button disappears");
        disappearingElementsPage.refreshPageUntilWebElementAppears(false);

        logStep("Verify if menu button element disappeared");
        assertNull("Unable to appear menu button element",
                   disappearingElementsPage.getGalleryMenuElement());
        assertTrue("The number of button elements after refresh is incorrect",
                   totalNumberOfMenuButtons >
                    disappearingElementsPage.getNumberOfMenuButtons());
    }

}

```

`assertNull(Objetc object)` - test passes if Object returns NULL
`assertNotNull(Objetc object)` - test passes if Object does not return NULL

This case shows how to move draggables elements on the page. image::images/example13.png[]

Try to move A to B position and see what happens. Also, open browser developer mode and see how the DOM changes.

The page can easily be broken. You can try to do so and check how the page structure changed in browser developer mode.

Steps:

1. Open The Internet Main Page
2. Click Drag and Drop link and open subpage
3. Check if the Drag and Drop message is visible
4. Check if element A is in container A and B in container B
5. Move element A to position B

6. Check if element A is in container B and B in container A
7. Move element B to position A
8. Again check if element A is in container A and B in container B

Page Class

```
public class DragAndDropPage extends BasePage {

    private static final By selectorDragAndDropText = By.cssSelector("div#content h3");
    private static final By selectorAElementContainer = By.cssSelector("div#column-a");
    private static final By selectorBElementContainer = By.cssSelector("div#column-b");
    private static final By selectorDescriptionElement = By.cssSelector("header");

    private static final String dndHelperPath =
"src/test/resources/js/drag_and_drop_helper.js";

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.DRAG_AND_DROP.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Drag and Drop' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
PageSubURLsProjectYEnum.DRAG_AND_DROP.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Returns information if drag and drop message is visible or not.
     *
     * @return true if exit drag and drop message was found on web page.
     */
    public boolean isDragAndDropMessageVisible() {
        return getDriver().findElementDynamic(selectorDragAndDropText)
            .isDisplayed();
    }
}
```

```

/**
 * Verifies if specified element is placed in designated container.
 *
 * @param element WebElement to be verified.
 * @return true if element described as A exists in container A or element B
exists in container B, false otherwise.
 */
public boolean isElementPlacedInCorrectContainer(String element) {
    return getDescriptionElement(findElementByDescription(element)).getText()
        .equals(element);
}

private WebElement findElementByDescription(String element) {
    WebElement result;
    switch (element) {
        case "A":
            result = getContainerElement(selectorAElementContainer);
            break;
        case "B":
            result = getContainerElement(selectorBElementContainer);
            break;
        default:
            result = null;
            BFLogger.logDebug("Chosen element doesn't exist on web page");
    }
    return result;
}

private WebElement getContainerElement(By container) {
    return getDriver().findElementDynamic(container);
}

private WebElement getDescriptionElement(WebElement container) {
    return container.findElement(selectorDescriptionElement);
}

/**
 * Drags element to designated container and drops it.
 *
 * @param element      String describing WebElement expected to be dragged.
 * @param from         String describing WebElement representing container of
element expected to be dragged.
 * @param destinationDesc String describing WebElement representing destination
container where other element will be dragged.
 */
public void dragElementToPosition(String element, String from, String
destinationDesc) {
    WebElement source = findElementByDescription(from);
    WebElement description = getDescriptionElement(source);
    WebElement destination = findElementByDescription(destinationDesc);
    if (description.getText())

```

```

        .equals(element))
    dragElement(source, destination);
}

}

```

Since HTML5, normal Selenium drag-and-drop action stopped working, thus it's necessary to execute Javascript which performs the drag-and-drop. To do so, create a JavascriptExecutor object, then read the script from a file drag_and_drop_helper.js and execute it with additional arguments using method executeScript(String script).

An example drag-and-drop solution:

```

/**
 * Drags and drops given WebElement to its destination location.
 * <p>
 * Since HTML5 all Selenium Actions performing drag and drop operations stopped
working as expected, e.g.
 * original implementation, which was:
 * <code>
 * BasePage.getAction()
 * .clickAndHold(draggable)
 * .moveToElement(target)
 * .release()
 * .build()
 * .perform();
 * </code>
 * finishes with no effect. For this reason, there is javascript function used, to
make sure that
 * drag and drop operation will be successful.
 * JavaScript function is stored under the following path:
'src/test/resources/js/drag_and_drop_helper.js'.
 * Original source of the script:
 * <a href="https://gist.github.com/r correia/2362544">drag_and_drop_helper</a>
 * </p>
 *
 * @param draggable A WebElement to be dragged and dropped.
 * @param target A destination, where element will be dropped.
 * @see JavascriptExecutor
 * @see Actions
 */
private void dragElement(WebElement draggable, WebElement target) {
    JavascriptExecutor js;
    INewWebDriver driver = getDriver();
    List<String> fileContent;
    String draggableId = draggable.getAttribute("id");
    String targetId = target.getAttribute("id");
    String script = null;
    if (draggable.getAttribute("draggable")
        .contains("true")) {

```

```

        if (driver instanceof JavascriptExecutor) {
            js = (JavascriptExecutor) driver;
            Path path = Paths.get(dndHelperPath);
            try {
                fileContent = Files.readAllLines(path);
                script = fileContent.stream()
                    .collect(Collectors.joining());
            } catch (IOException e) {
                BFLogger.logDebug("Unable to read file content: " +
e.getMessage());
            }
            if (script != null && !script.isEmpty()) {
                String arguments = "${'#%s'}).simulateDragDrop({ dropTarget:
'#%s'});";
                js.executeScript(script + String.format(arguments, draggableId,
targetId));
            }
        }
    }
}

```

Drag and Drop helper file:

```

(function( $ ) {
    $.fn.simulateDragDrop = function(options) {
        return this.each(function() {
            new $.simulateDragDrop(this, options);
        });
    };
    $.simulateDragDrop = function(elem, options) {
        this.options = options;
        this.simulateEvent(elem, options);
    };
    $.extend($.simulateDragDrop.prototype, {
        simulateEvent: function(elem, options) {
            /*Simulating drag start*/
            var type = 'dragstart';
            var event = this.createEvent(type);
            this.dispatchEvent(elem, type, event);

            /*Simulating drop*/
            type = 'drop';
            var dropEvent = this.createEvent(type, {});
            dropEvent.dataTransfer = event.dataTransfer;
            this.dispatchEvent($(options.dropTarget)[0], type, dropEvent);

            /*Simulating drag end*/
            type = 'dragend';
            var dragEndEvent = this.createEvent(type, {});
            dragEndEvent.dataTransfer = event.dataTransfer;
        }
    });
})

```

```

        this.dispatchEvent(elem, type, dragEndEvent);
    },
    createEvent: function(type) {
        var event = document.createEvent("CustomEvent");
        event.initCustomEvent(type, true, true, null);
        event.dataTransfer = {
            data: {
            },
            setData: function(type, val){
                this.data[type] = val;
            },
            getData: function(type){
                return this.data[type];
            }
        };
        return event;
    },
    dispatchEvent: function(elem, type, event) {
        if(elem.dispatchEvent) {
            elem.dispatchEvent(event);
        }else if( elem.fireEvent ) {
            elem.fireEvent("on"+type, event);
        }
    }
});
})(jQuery);

```

Test Class

```

@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class
})
public class DragAndDropTest extends TheInternetBaseTest {

    private static final String ELEMENT_A    = "A";
    private static final String CONTAINER_A = "A";
    private static final String ELEMENT_B    = "B";
    private static final String CONTAINER_B = "B";

    private static DragAndDropPage dragAndDropPage;

    @BeforeClass
    public static void setUpBeforeClass() {
        dragAndDropPage = shouldTheInternetPageBeOpened().clickDragAndDropLink();

        logStep("Verify if Drag And Drop page is opened");
        assertTrue("Unable to open Drag And Drop page", dragAndDropPage.isLoaded());

        logStep("Verify if Drag And Drop message is visible");
        assertTrue("Drag And Drop message is not visible",
        dragAndDropPage.isDragAndDropMessageVisible());
    }
}

```

```

}

@Test
public void shouldDraggableElementBeMovedAndDropped() {
    logStep("Verify if elements are placed in proper containers");
    assertTrue("Element A doesn't exist in container A",
dragAndDropPage.isElementPlacedInCorrectContainer(ELEMENT_A));
    assertTrue("Element B doesn't exist in container B",
dragAndDropPage.isElementPlacedInCorrectContainer(ELEMENT_B));

    logStep("Step 7: Drag and drop element A into container B");
    dragAndDropPage.dragElementToPosition(ELEMENT_A, CONTAINER_A, CONTAINER_B);

    logStep("Step 8: Verify if elements are placed in improper containers");
    assertFalse("Element A doesn't exist in container B",
dragAndDropPage.isElementPlacedInCorrectContainer(ELEMENT_A));
    assertFalse("Element B doesn't exist in container A",
dragAndDropPage.isElementPlacedInCorrectContainer(ELEMENT_B));

    logStep("Drag and drop element B back into container B");
    dragAndDropPage.dragElementToPosition(ELEMENT_A, CONTAINER_B, CONTAINER_A);

    logStep("Verify if elements are placed in proper containers");
    assertTrue("Element A doesn't exist in container A",
dragAndDropPage.isElementPlacedInCorrectContainer(ELEMENT_A));
    assertTrue("Element B doesn't exist in container B",
dragAndDropPage.isElementPlacedInCorrectContainer(ELEMENT_B));
}

}

```

This example shows how to select an element from the dropdown list.



Check in the developer mode how a Dropdown List's content has been organized.

The screenshot shows a browser window with a dropdown list example. The DOM tree on the right highlights the `select#dropdown` element, which contains two options: `<option value="1" selected="selected">Option 1</option>` and `<option value="2">Option 2</option>`. The CSS panel on the right shows the style definition for the `option` element, including `font-weight: normal;` and `display: block;`. A detailed box model diagram is overlaid on the CSS panel, showing margins, borders, padding, and content dimensions.

Notice that the Dropdown Options have different attributes, such as "disabled" or "selected".

Steps:

1. Open The Internet Main Page
2. Click the Dropdown link and go to the subpage
3. Select first dropdown Option
4. Check if Option 1 is selected
5. Select second dropdown Option
6. Check if Option 2 is selected

Page Class

```

public class DropdownPage extends BasePage {

    private static final By dropdownListSelector = By.cssSelector("#dropdown");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.DROPDOWN.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Dropdown List' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.DROPDOWN.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Selects dropdown's value by given index.
     *
     * @param index Index of option to be selected
     */
    public void selectDropdownValueByIndex(int index) {
        getDriver().elementDropdownList(dropdownListSelector)
            .selectDropdownByIndex(index);
    }

    /**
     * Returns text value of first selected dropdown's option.
     *
     * @return String object representing value of dropdown's option
     */
    public String getSelectedDropdownValue() {
        return getDriver().elementDropdownList(dropdownListSelector)
            .getFirstSelectedOptionText();
    }
}

```

DropdownListElement class

DropdownListElement is MrChecker's class, which contains methods for performing the dropdown list of actions:

```
elementDropdownList() - returns DropdownListElement Object
```

Test Class

```
@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class DropdownTest extends TheInternetBaseTest {

    private static final String expectedFirstOptionValue = "Option 1";
    private static final String expectedSecondOptionValue = "Option 2";
    private static DropdownPage dropdownPage;

    @BeforeClass
    public static void setUpBeforeClass() {
        dropdownPage = shouldTheInternetPageBeOpened().clickDropdownLink();

        logStep("Verify if Dropdown page is opened");
        assertTrue("Unable to open Dropdown page", dropdownPage.isLoaded());
    }

    @Test
    public void shouldGetExpectedDropdownTextOptionAfterSelection() {

        logStep("Select first dropdown option");
        dropdownPage.selectDropdownValueByIndex(1);

        logStep("Verify if selected option text is equal to the expected one");
        assertEquals("Selected value is different than expected",
                    expectedFirstOptionValue,
                    dropdownPage.getSelectedDropdownValue());

        logStep("Select first dropdown option");
        dropdownPage.selectDropdownValueByIndex(2);

        logStep("Verify if selected option text is equal to the expected one");
        assertEquals("Selected value is different than expected",
                    expectedSecondOptionValue,
                    dropdownPage.getSelectedDropdownValue());
    }
}
```

This case shows how to compare dynamic content.

Dynamic Content

This example demonstrates the ever-evolving nature of content by loading new text and images on each page refresh.

To make some of the content static append `?with_content=static` or [click here](#).



Beatae praesentium sed possimus magni sed et iste non repudiandae sit ab voluptate deserunt a et blanditiis odit maiores nobis saepe id placeat eos eum dolores consecutu harum voluptatem quo dolore.



Labore non ut qui culpa corporis sed occaecati ut nesciunt vel natus laborum ut fuga repellendus illo modi cumque id veniam beatiae quae qui praesentium reiciendis dicta persiciatis ut maiores eaque hic harum quaerat reprehenderit.



Vel quas non amet consequatur aut alias deserunt eum in enim quam quis sunt iure eos
voluntatibus assumenda distinctio dolores ullam suscipit et recusandae dicta error
reprehendent modi sit eos doloribus neque quidem.

Powered by Elemental Selenium

Note that after site refresh, some of the content is different. You can see in the browser's developer mode how the text and image sources are being changed.

Dynamic Content

This example demonstrates the ever-evolving nature of content by loading new text and images on each page refresh.

To make some of the content static append `?with_content=static` or [click here](#)



Volutate tempore eos exercitationem dignissimos omnis est et eius consequatur sunt rerum quod nostrum reprehenderit odit molestiae magni et aut voluptates aspernatur temporibus his qui qui et repudiandae animi dolores velit.

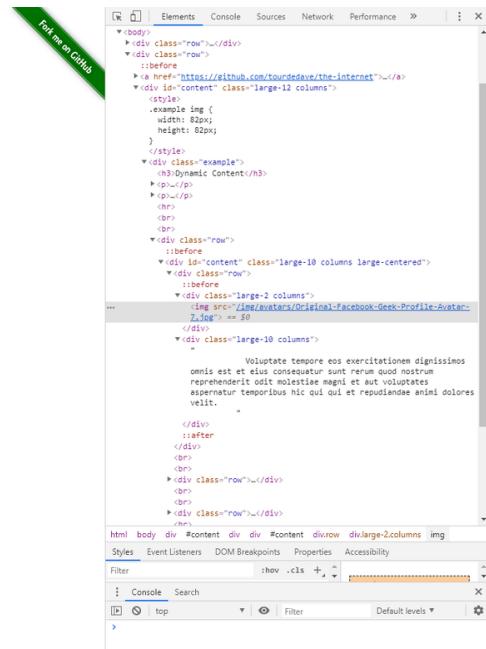


Magnam accusantium ex voluptatibus consequatur et eos eum quis cum suspicere quisquam liberò modi fugit eis vel vero voluptas deleniti reprehenderit illo qui laudantium nobis quo es voluptatem et aut quasi.



Harum error vitae consequatur enim illo in unde aut voluptas quo facilis dolorum explicabo sed dolore nulla qui sit atque beatiae sapiente omnis doloribus dolores quidem asperiores autem vero molestias.

Powered by Elemental Selenium



Steps:

1. Open The Internet Main Page
 2. Click Dynamic Content link and load subpage
 3. Save page images sources and descriptions before the refresh
 4. Refresh page
 5. Save page images sources and it's descriptions after refresh
 6. Compare page content before and after refresh and verify if it's different

Page Class

```
public class DynamicContentPage extends BasePage {

    private static final By imagesLinksSelector          = By.cssSelector("div#content > div.row img");
    private static final By imagesDescriptionsSelector = By.cssSelector("div#content > div.row div.large-10");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.DYNAMIC_CONTENT.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Dynamic Content' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.DYNAMIC_CONTENT.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Returns list of picture descriptions being present on the web page.
     *
     * @return List of String objects representing descriptions
     */
    public List<String> getDescriptions() {
        return new ListElements(imagesDescriptionsSelector).getTextList();
    }

    /**
     * Returns a list of image links being present on the web page.
     *
     * @return List of String objects representing paths to pictures
     */
    public List<String> getImageLinks() {
        return new ListElements(imagesLinksSelector)
            .getList()
            .stream()
            .map(element -> element.getAttribute("src"))
            .collect(Collectors.toList());
    }
}
```

ListElements

ListElements is MrChecker collection which can store WebElement Objects. Constructing ListElements with cssSelector allows you to store every element on the page which fits the selector. Example methods:

```
getList() - returns WebElements list,
getTextList() - returns list of contents of each Element,
getSize() - returns number of stored Elements
In getImageLinks() example it's shown how to get a list of specified Elements' attributes.
```

Test Class

```
@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class DynamicContentTest extends TheInternetBaseTest {

    private static DynamicContentPage dynamicContentPage;

    @BeforeClass
    public static void setUpBeforeClass() {
        dynamicContentPage =
shouldTheInternetPageBeOpened().clickDynamicContentLink();

        logStep("Verify if Dynamic Content page is opened");
        assertTrue("Unable to open Dynamic Content page",
dynamicContentPage.isLoaded());
    }

    @Test
    public void shouldImagesAndDescriptionsDifferAfterRefresh() {

        logStep("Read images and descriptions before refresh");
        List<String> descriptionsBeforeRefresh = dynamicContentPage.getDescriptions();
        List<String> imagesBeforeRefresh = dynamicContentPage.getImageLinks();

        logStep("Refres page");
        dynamicContentPage.refreshPage();
        assertTrue("The Dynamic Content page hasn't been refreshed",
dynamicContentPage.isLoaded());

        logStep("Read images and descriptions after refresh");
        List<String> descriptionsAfterRefresh = dynamicContentPage.getDescriptions();
        List<String> imagesAfterRefresh = dynamicContentPage.getImageLinks();

        logStep("Verify if descriptions are different after refresh");
        assertEquals("Different number of descriptions before and after refresh",
descriptionsAfterRefresh.size(), descriptionsBeforeRefresh.size());
    }
}
```

```

boolean diversity = false;
for (int i = 0; i < descriptionsAfterRefresh.size(); i++) {
    if (!descriptionsAfterRefresh.get(i)
        .equals(descriptionsBeforeRefresh.get(i))) {
        diversity = true;
        break;
    }
}
assertTrue("There are no differences between descriptions before and after
refresh",
    diversity);

logStep("Verify if images are different after refresh");
assertEquals("Different number of descriptions before and after refresh",
    imagesAfterRefresh.size(), imagesBeforeRefresh.size());

diversity = false;
for (int i = 0; i < imagesAfterRefresh.size(); i++) {
    if (!imagesAfterRefresh.get(i)
        .equals(imagesBeforeRefresh.get(i))) {
        diversity = true;
        break;
    }
}
assertTrue("There are no differences between images before and after refresh",
    diversity);
}
}

```

In the test method, during differences verification, the goal is to compare every element from the first and second list and find first diversity.

Unresolved directive in mrchecker.wiki/master-mrchecker.asciidoc - include::Who-Is-MrChecker/Tutorials/Basic-Tutorials/Basic-Tests/Example-10-Dynamically-loaded-elements.asciidoc[leveloffset=4]

Exit Intent

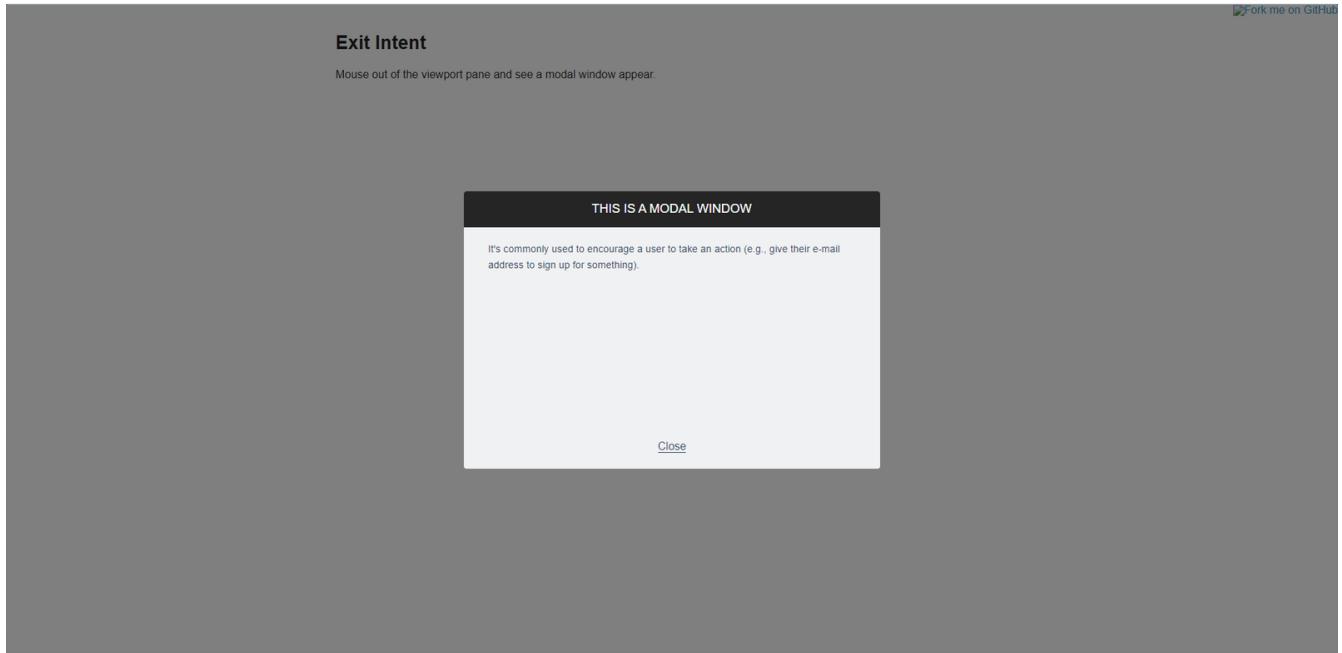
Mouse out of the viewport pane and see a modal window appear

Powered by Elemental Selenium

[Fork me on GitHub](#)

This case shows how to perform mouse actions and test modal windows.

After you move the mouse cursor out of the website, you should see a new window appearing:



Check in the browser's developer mode if this window exists in Page DOM

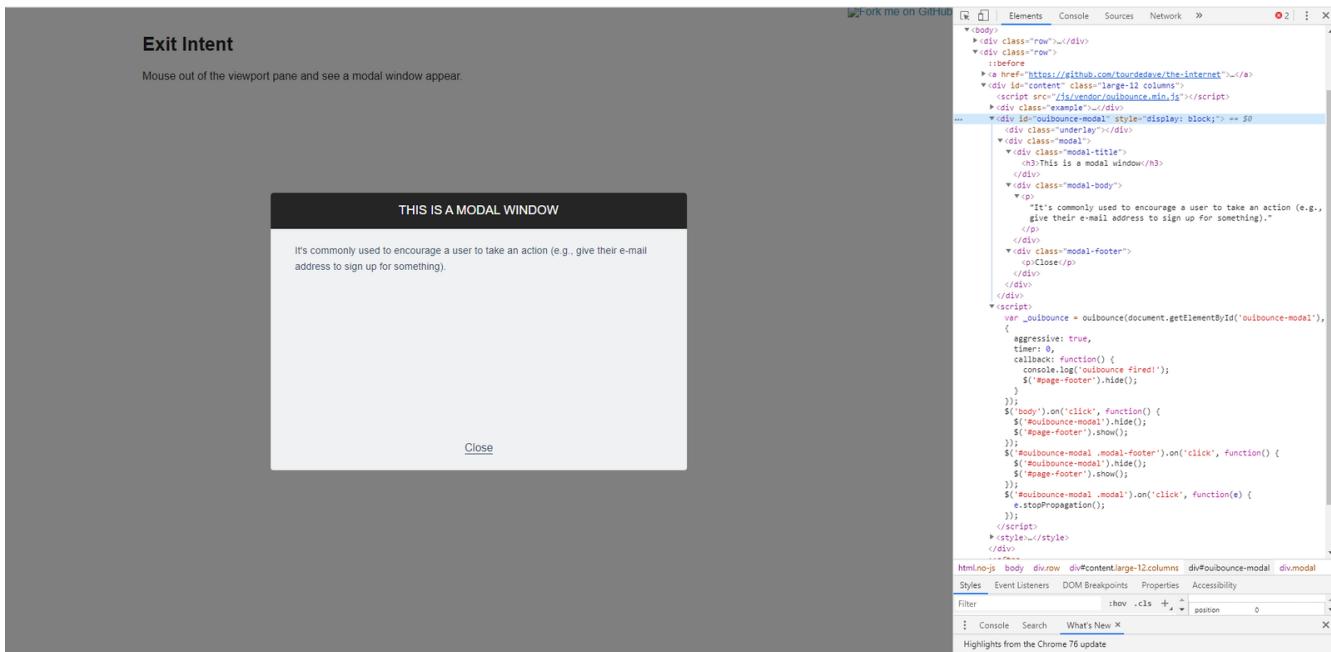
```

<div id="ouibounce-modal" style="display:none">
  <div class="modal__header">
    <h3>This is a modal window</h3>
  </div>
  <div class="modal__body">
    <p>This is a modal window</p>
  </div>
  <div class="modal__footer">
    <p>Close</p>
  </div>
</div>
<script>
  var _ouibounce = ouibounce(document.getElementById('ouibounce-modal'));
  {
    aggressive: true,
    timer: 0,
    callback: function() {
      console.log('ouibounce fired!');
      $('#page-footer').hide();
    }
  };
  $('body').on('click', function() {
    $('#ouibounce-modal').show();
    $('#page-footer').show();
  });
  $('#ouibounce-modal .modal__footer').on('click', function() {
    $('#ouibounce-modal').hide();
    $('#page-footer').show();
  });
  $('#ouibounce-modal .modal__body').on('click', function(e) {
    e.stopPropagation();
  });
</script>
<style>...</style>
</div>
</div>
</div>
</div>

```

Before you move the mouse out, the window exists, but it's not displayed.

When the mouse is moved, JavaScript changes display attribute. It also hides window after clicking "Close".



Page Class

```

public class ExitIntentPage extends BasePage {

    private static final String MODAL_WINDOW_HIDDEN           = "display: none;";
    private static final String MODAL_WINDOW DISPLAYED        = "display: block;";
    private static final String MODAL_WINDOW_STYLE_ATTRIBUTE = "style";

    private static final By selectorModalWindow               = By.cssSelector("div#ouibounce-modal");
    private static final By selectorExitIntentText           = By.cssSelector("div#content h3");
    private static final By selectorModalWindowTitle         = By.cssSelector("h3");
    private static final By selectorModalWindowCloseButton = By.cssSelector("div.modal-footer > p");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.EXIT_INTENT.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Exit Intent' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.EXIT_INTENT.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {

```

```

        return getActualPageTitle();
    }

    /**
     * Returns information if exit intent message is visible or not.
     *
     * @return true if exit intent message was found on web page.
     */
    public boolean isIntentMessageVisible() {
        return getDriver().findElementDynamic(selectorExitIntentText)
            .isDisplayed();
    }

    /**
     * Returns information if modal window is hidden.
     *
     * @return true if modal window is hidden.
     */
    public boolean isModalWindowHidden() {
        return getDriver().findElementDynamic(selectorModalWindow)
            .getAttribute(MODAL_WINDOW_STYLE_ATTRIBUTE)
            .equals(MODAL_WINDOW_HIDDEN);
    }

    /**
     * Returns information if modal window is showed on web page.
     *
     * @return true if modal window is displayed.
     */
    public boolean isModalWindowVisible() {
        return getDriver().findElementDynamic(selectorModalWindow)
            .getAttribute(MODAL_WINDOW_STYLE_ATTRIBUTE)
            .equals(MODAL_WINDOW_DISPLAYED);
    }

    /**
     * Returns information if modal window title is shown and correct.
     *
     * @param expectedValue String representing expected value of modal window's
     * title.
     * @return true if modal window's title is equal to expected value.
     */
    public boolean verifyModalWindowTitle(String expectedValue) {
        return getDriver().elementLabel(new ByChained(selectorModalWindow,
            selectorModalWindowTitle))
            .getText()
            .equals(expectedValue);
    }

    /**
     * Closes modal window by pressing 'close' button.
    
```

```

*/
public void closeModalWindow() {
    getDriver().elementButton(new ByChained(selectorModalWindow,
        selectorModalWindowCloseButton))
        .click();
}

/**
 * Moves mouse pointer to the top middle of screen, then to the centre of screen
and
 * again to the top.
 * <p>
 * This move simulates leaving the viewport and encourages the modal to show up.
There is
 * java.awt.Robot used
 * to move mouse pointer out of the viewport. There are timeouts used to let the
browser detect
 * mouse move.
 * </p>
 *
 * @see java.awt.Robot
 */
public void moveMouseOutOfViewport() {
    Robot robot;
    Dimension screenSize = getDriver().manage()
        .window()
        .getSize();
    int halfWidth = new BigDecimal(screenSize.getWidth() / 2).intValue();
    int halfHeight = new BigDecimal(screenSize.getHeight() / 2).intValue();

    try {
        robot = new Robot();
        robot.mouseMove(halfWidth, 1);
        getDriver().manage()
            .timeouts()
            .implicitlyWait(1, TimeUnit.SECONDS);
        robot.mouseMove(halfWidth, halfHeight);
        getDriver().manage()
            .timeouts()
            .implicitlyWait(1, TimeUnit.SECONDS);
        robot.mouseMove(halfWidth, 1);
    } catch (AWTException e) {
        BFLogger.LogError("Unable to connect with remote mouse");
        e.printStackTrace();
    }
}
}

```

Attributes

Elements on pages have attributes like "id", "class", "name", "style" etc. In order to check them, use method `getAttribute(String name)`. In this case attribute "style" determinates if the element is displayed.

Robot

Robot class can perform mouse movement. Method `mouseMove(int x, int y)` moves the remote mouse to given coordinates.

Manage Timeouts

`manage().timeouts()` methods allows you to change WebDriver timeouts values such as:

- `pageLoadTimeout(long time, TimeUnit unit)` - the amount of time to wait for a page to load before throwing an exception
- `setScriptTimeout(long time, TimeUnit unit)` - the amount of time to wait for finish execution of a script before throwing an exception
- `implicitlyWait(long time, TimeUnit unit)` - the amount of time the driver should wait when searching for an element if it is not immediately present. After that time, it throws an exception.

Changing timeouts can improve test stability but can also make them run slower.

Test Class

Steps:

1. Open The Internet Main Page
2. Click Exit Intent link and load subpage
3. Check if the page is loaded and "Exit Intent" message is visible
4. Verify if Modal Window is hidden
5. Move mouse out of the viewport
6. Check if Modal Window is visible
7. Verify if Modal Window title is correct
8. Click 'close' button
9. Again verify if Modal Window is hidden

```
@Category({ TestsLocal.class, TestsNONParallel.class })
public class ExitIntentTest extends TheInternetBaseTest {

    private static final String MODAL_WINDOW_TITLE = "This is a modal window";

    private static ExitIntentPage exitIntentPage;

    @BeforeClass
    public static void setUpBeforeClass() {
        exitIntentPage = shouldTheInternetPageBeOpened().clickExitIntentLink();

        logStep("Verify if Exit Intent page is opened");
        assertTrue("Unable to open Exit Intent page", exitIntentPage.isLoaded());

        logStep("Verify if exit intent message is visible");
        assertTrue("Exit intent message is not visible",
        exitIntentPage.isIntentMessageVisible());
    }

    @Test
    public void shouldModalWindowAppearWhenMouseMovedOutOfViewportTest() {

        logStep("Verify if modal window is hidden");
        assertTrue("Fail to hide modal window", exitIntentPage.isModalWindowHidden());

        logStep("Move mouse pointer out of viewport");
        exitIntentPage.moveMouseOutOfViewport();

        logStep("Verify if modal window showed up");
        assertTrue("Fail to show up modal window",
        exitIntentPage.isModalWindowVisible());

        logStep("Verify if modal window title displays properly");
        assertTrue("Fail to display modal window's title",
        exitIntentPage.verifyModalWindowTitle(MODAL_WINDOW_TITLE.toUpperCase()));

        logStep("Close modal window");
        exitIntentPage.closeModalWindow();

        logStep("Verify if modal window is hidden again");
        assertTrue("Fail to hide modal window", exitIntentPage.isModalWindowHidden());
    }
}
```

Remember not to move mouse manually during test execution.

File Downloader

```
ObjectivityTestAutomationCSharpFramework.txt
simple.txt
uploadTest.txt
some-file.txt
test.txt
Adobe.pdf
empty.txt
MyTest.txt
Screenshot 2019-08-06 at 10 13 36 AM.png
SeleniumWebdriverUploadFile.txt
text.txt
Capture.JPG
```

Powered by [Elemental Selenium](#)

This example shows how to check if file downloads properly.

After clicking on one of these links, a specific file should be downloaded to your computer.

Steps:

1. Open The Internet Main Page
2. Click on the File Download link and open subpage
3. Click on "some-file.txt" download link and download file
4. Check if the file exists in the appropriate folder
5. Delete the file
6. Check if the file doesn't exist in the folder

Page Class

```
public class FileDownloadPage extends BasePage {

    private static final By selectorSomeFileTxt = By.cssSelector("a[href*='some-file']");

    private final String DOWNLOAD_DIR = System.getProperty("java.io.tmpdir");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.DOWNLOAD.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'File Downloader' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.DOWNLOAD.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
```

```

}

/**
 * Verifies if the chosen file is already downloaded and if not, downloads it .
 * Throws RuntimeException otherwise.
 *
 * @return Downloaded file
 */
public File downloadTextFile() {
    String nameOfDownloadFile = getNameOfDownloadFile();
    File fileToDownload = new File(DOWNLOAD_DIR + nameOfDownloadFile);

    if (fileToDownload.exists()) {
        throw new RuntimeException("The file that you want to download already exists. "
            + "Please remove it manually. Path to the file: " +
        fileToDownload.getPath());
    }

    getDriver().elementButton(selectorSomeFileTxt)
        .click();

    waitForFileDownload(2000, fileToDownload);
    return fileToDownload;
}

private void waitForFileDownload(int totalTimeoutInMillis, File expectedFile) {
    FluentWait<WebDriver> wait = new FluentWait<WebDriver>(getDriver())
        .withTimeout(totalTimeoutInMillis, TimeUnit.MILLISECONDS)
        .pollingEvery(200, TimeUnit.MILLISECONDS);

    wait.until((WebDriver wd) -> expectedFile.exists());
}

private String getNameOfDownloadFile() {
    String urlToDownload = getDriver().findElementDynamic(selectorSomeFileTxt)
        .getAttribute("href");
    String[] urlHierarchy = urlToDownload.split("/");
    return urlHierarchy[urlHierarchy.length - 1];
}
}

```

Use FluentWait class and create an expected condition using a lambda expression to wait until the file downloads.

To perform operations on files, use java File class. To get a file name, find it in download URL.

Test Class

```

@Category({ TestsLocal.class, TestsNONParallel.class })
public class FileDownloadTest extends TheInternetBaseTest {

    private static FileDownloadPage fileDownloadPage;

    @BeforeClass
    public static void setUpBeforeClass() {
        fileDownloadPage = shouldTheInternetPageBeOpened().clickFileDownloadLink();

        logStep("Verify if File Download page is opened");
        assertTrue("Unable to open File Download page", fileDownloadPage.isLoaded());
    }

    @Test
    public void shouldfileBeDownloaded() {

        logStep("Download the some-file.txt");
        File downloadedFile = fileDownloadPage.downloadTextFile();

        logStep("Verify if downloaded file exists");
        assertTrue("Downloaded file does not exist", downloadedFile.exists());

        logStep("Remove downloaded file");
        downloadedFile.delete();

        logStep("Verify if downloaded file has been removed");
        assertFalse("Downloaded file still exists", downloadedFile.exists());
    }
}

```

[Fork me on GitHub](#)

Login Page

This is where you can log into the secure area. Enter `tomsmith` for the username and `SuperSecretPassword!` for the password. If the information is wrong you should see error messages.

Username	<input type="text"/>
Password	<input type="password"/>
<input type="button" value="Login"/>	

Powered by [Elemental Selenium](#)

This case shows how to pass through the standard authentication page.

When you enter the correct credentials, you should see the next page:



Secure Area

Welcome to the Secure Area. When you are done click logout below.

[Logout](#)

Powered by [Elemental Selenium](#)

If user data is wrong, an appropriate message appears:



Login Page

This is where you can log into the secure area. Enter `tomsmith` for the username and `SuperSecretPassword!` for the password. If the information is wrong you should see error messages.

Username

Password

[Login](#)

Powered by [Elemental Selenium](#)

Page Class

```
public class FormAuthenticationPage extends BasePage {

    private final static By selectorInputUsername      = By.cssSelector("#username");
    private final static By selectorInputUserPassword = By.cssSelector("#password");
    private final static By selectorLoginMessage     = By.cssSelector("#flash");
    private final static By selectorLoginButton       = By.cssSelector("#login > button > i");
    private final static By selectorLogoutButton      = By.cssSelector("#content > div > a ");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.LOGIN.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Login Page' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
PageSubURLsProjectYEnum.LOGIN.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }
}
```

```
}

/**
 * Sets user name to designated form's field.
 *
 * @param username String representing a user's name
 * @return FormAuthenticationPage object with user name set to the given one
 */
public FormAuthenticationPage setUsername(String username) {
    InputTextElement elementInputUsername = new
InputTextElement(selectorInputUsername);
    elementInputUsername.clearInputText();
    elementInputUsername.setInputText(username);
    return this;
}

/**
 * Sets user password to designated form's field.
 *
 * @param userPassword String representing a user's password
 * @return FormAuthenticationPage object with user's password set to the given one
 */
public FormAuthenticationPage setUserPassword(String userPassword) {
    InputTextElement elementInputPassword = new
InputTextElement(selectorInputUserPassword);
    elementInputPassword.clearInputText();
    elementInputPassword.setInputText(userPassword);
    return this;
}

/**
 * Returns login message.
 *
 * @return String object representing the message returned after login operation
is performed
 */
public String getLoginMessageText() {
    return new LabelElement(selectorLoginMessage).getText();
}

/**
 * Clicks 'Login' button.
 */
public void clickLoginButton() {
    new Button(selectorLoginButton).click();
}

/**
 * Clicks 'Logout' button.
 */
public void clickLogoutButton() {
```

```

        new Button(selectorLogoutButton).click();
    }
}

```

InputElement

Use methods from this class to perform actions on text fields:

- `clearInputText()` - remove all text from selected input field
- `setInputText(String text)` - enter given text

LabelElement

- String `getText()` method returns visible text from label

TestClass

Prepare six test cases:

1. Try to login with empty user data and check if the error message appears
2. Try to login with empty username and valid password and check if the error message appears
3. Try to login with a valid username and empty password and check if the error message appears
4. Try to login with invalid username and invalid password and check if the error message appears
5. Try to login with a valid username and valid password and check if success login message appears, then log out
6. Try to login with a valid username and valid password and check if success login message appears, then log out and check if success logout message is displayed

Before all tests: Open The Internet Main Page

Before each case: Click on the Form Authentication link and open login page

After each case: Go back to The Internet Main Page

```

@Category({ TestsLocal.class, TestsNONParallel.class })
public class FormAuthenticationTest extends TheInternetBaseTest {

    private static FormAuthenticationPage formAuthenticationPage;

    private String errorUsernameMessage = "Your username is invalid!\n" + "x";
    private String errorPasswordMessage = "Your password is invalid!\n" + "x";
    private String loginMessage         = "You logged into a secure area!\n" + "x";
    private String logoutMessage        = "You logged out of the secure area!\n" + "x";
    private String emptyUsername       = "";
    private String emptyUserPassword   = "";
    private String validUsername       = "tomsmith";
}

```

```

private String validPassword      = "SuperSecretPassword!";
private String randomUsername    = UUID.randomUUID()
    .toString();
private String randomUserPassword = UUID.randomUUID()
    .toString();

@BeforeClass
public static void setUpBeforeClass() {
    logStep("Open the Url http://the-internet.herokuapp.com/");
    theInternetPage = new TheInternetPage();
    theInternetPage.load();

    logStep("Verify if Url http://the-internet.herokuapp.com/ is opened");
    assertTrue("Unable to load The Internet Page", theInternetPage.isLoaded());
}

@Override
public void setUp() {
    logStep("Click subpage link");
    formAuthenticationPage = theInternetPage.clickFormAuthenticationLink();

    logStep("Verify if subpage is opened");
    assertTrue("The Internet subpage: FormAuthenticationPage was not open",
formAuthenticationPage.isLoaded());
}

@Test
public void shouldErrorMessageBeDisplayedWhenUserLogsWithEmptyData() {
    logStep("Log user with empty username and password");
    formAuthenticationPage.setUsername(emptyUsername)
        .setUserPassword(emptyUserPassword)
        .clickLoginButton();
    assertEquals("Unexpectedly user logged in with empty data",
errorUsernameMessage,
        formAuthenticationPage.getLoginMessageText());
}

@Test
public void
shouldErrorMessageBeDisplayedWhenUserLogsWithEmptyUsernameAndValidPassword() {
    logStep("Log user with empty username and valid password");
    formAuthenticationPage.setUsername(emptyUsername)
        .setUserPassword(validPassword)
        .clickLoginButton();
    assertEquals("Unexpectedly user logged in with empty username",
errorUsernameMessage,
        formAuthenticationPage.getLoginMessageText());
}

@Test
public void

```

```
shouldErrorMessageBeDisplayedWhenUserLogsWithValidUsernameAndEmptyPassword() {
    logStep("Log user with valid username and empty password");
    formAuthenticationPage.setUsername(validUsername)
        .setUserPassword(emptyUserPassword)
        .clickLoginButton();
    assertEquals("Unexpectedly user logged in with empty password",
errorPasswordMessage,
            formAuthenticationPage.getLoginMessageText());
}

@Test
public void
shouldErrorMessageBeDisplayedWhenUserLogsWithInvalidUsernameAndInvalidPassword() {
    logStep("Log user with invalid username and invalid password");
    formAuthenticationPage.setUsername(randomUsername)
        .setUserPassword(randomUserPassword)
        .clickLoginButton();
    assertEquals("Unexpectedly user logged in with random credentials",
errorUsernameMessage,
            formAuthenticationPage.getLoginMessageText());
}

@Test
public void shouldUserLogInWithValidCredentials() {
    logStep("Log user with valid username and valid password");
    formAuthenticationPage.setUsername(validUsername)
        .setUserPassword(validPassword)
        .clickLoginButton();
    assertEquals("Unable to login user with valid credentials", loginMessage,
            formAuthenticationPage.getLoginMessageText());
    logStep("Log out user");
    formAuthenticationPage.clickLogoutButton();
}

@Test
public void shouldUserLogOutAfterProperLogInAndClickLogoutButton() {
    logStep("Log user with valid username and valid password");
    formAuthenticationPage.setUsername(validUsername)
        .setUserPassword(validPassword)
        .clickLoginButton();
    assertEquals("Unable to login user with valid credentials", loginMessage,
            formAuthenticationPage.getLoginMessageText());
    logStep("Log out user");
    formAuthenticationPage.clickLogoutButton();
    assertEquals("User cannot log out after proper log in", logoutMessage,
            formAuthenticationPage.getLoginMessageText());
}

@Override
public void tearDown() {
    logStep("Navigate back to The-Internet page");
```

```
        theInternetPage.load();
    }
}
```

After running Test Class, cases might be performed in a different order.

[Fork me on GitHub](#)

Hovers

Hover over the image for additional information



Powered by Elemental Selenium

This example shows how to approach elements dynamically appearing after the user's action.

Move the mouse over an image to see the additional label.

[Fork me on GitHub](#)

Hovers

Hover over the image for additional information



name: user1
[View profile](#)

Powered by Elemental Selenium

Labels exist in page DOM all the time but their display attributes change. In this case, there is no JavaScript. Elements' visibility is managed by CSS.

[Fork me on GitHub](#)

Hovers

Hover over the image for additional information



Powered by Elemental Selenium

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>The Internet</title>
  </head>
  <body>
    <div id="content" class="large-12 columns">
      <div class="row">
        <div class="example">
          <h3>Hovers</h3>
          <p>Hover over the image for additional information</p>
          <div class="figure">
            
            <div class="caption" style="display: none; position: absolute; left: 0; top: 0; width: 100%; height: 100%; background-color: white; padding: 5px; border: 1px solid #ccc; border-radius: 5px; z-index: 1;>
              <strong>name:</strong> user1<br/>
              <a href="#">View profile</a>
            </div>
          </div>
        </div>
      </div>
    </div>
    <div id="page-footer" class="row">
      <div>The Internet</div>
    </div>
  </body>
</html>
```

Styles	Event Listeners	DOM Breakpoints	Properties	Accessibility
:hov .cls				

Page Class

```
public class HoversPage extends BasePage {

    private final static By selectorImages = By.cssSelector("div.figure > img");
    private final static By selectorNames = By.cssSelector("div.figcaption h5");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.HOVERS.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Hovers' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.HOVERS.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Moves mouse pointer over an image with given index.
     *
     * @param index An index of the picture, where mouse pointer should be moved
     */
    public void hoverOverAvatar(int index) {
        Actions action = new Actions(getDriver());
        WebElement avatarImage = getDriver().findElementDynamics(selectorImages)
            .get(index);
        action.moveToElement(avatarImage)
            .perform();
    }

    /**
     * Returns the information displayed under a picture with given index.
     *
     * @param index An index of the picture, where the information should be read
     * @return String object representing picture's information
     */
    public String getAvatarsInformation(int index) {
        return getDriver().findElementDynamics(selectorNames)
            .get(index)
            .getText();
    }
}
```

Actions

Actions class contains methods used to execute basic user actions such as mouse moving and clicking or keys sending. Action or actions series will be performed after calling `perform()` method.

Test Class

Steps:

1. Open The Internet Main Page
2. Go to Hovers page
3. Move mouse over random image
4. Check if displayed text is equal to expected.

```
@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class HoversTest extends TheInternetBaseTest {

    private static HoversPage      hoversPage;
    private final String          names[]     = { "name: user1", "name: user2", "name:
user3" };

    @BeforeClass
    public static void setUpBeforeClass() {
        hoversPage = shouldTheInternetPageBeOpened().clickHoversLink();

        logStep("Verify if Hovers page is opened");
        assertTrue("Unable to open Hovers page", hoversPage.isLoaded());
    }

    @Test
    public void
shouldProperInformationBeDisplayedWhenMousePointerHoveredOverRandomElement() {
        logStep("Hover mouse pointer over random element");
        int randomIndex = new Random().nextInt(names.length);
        hoversPage.hoverOverAvatar(randomIndex);
        assertEquals("Picture's information is different than expected",
names[randomIndex],
                hoversPage.getAvatarsInformation(randomIndex));
    }
}
```

Because in this case the tested content is being chosen randomly, each test run could check a different element.

JavaScript Alerts

Here are some examples of different JavaScript alerts which can be troublesome for automation

[Click for JS Alert](#)

[Click for JS Confirm](#)

[Click for JS Prompt](#)

Result:

Powered by [Elemental Selenium](#)

This case shows how to test pop-up JS alerts.

After clicking one of the buttons, an adequate alert should appear.

The screenshot shows a browser window with a JavaScript alert dialog. The dialog has a dark background with white text. It displays the message "I am a JS Confirm". At the bottom of the dialog are two buttons: "OK" and "Anuluj" (Cancel). Behind the dialog, the main content of the page is visible, including the title "Komunikat ze strony the-internet.herokuapp.com" and the three buttons mentioned in the text above.

Result:

Powered by [Elemental Selenium](#)

Performed action will be displayed under "Result" label.

In developer mode, you can view JavaScript which creates alerts.

The screenshot shows the developer tools of a browser with the "Elements" tab selected. The left sidebar shows the file structure and components of the page. The main area displays the raw HTML and JavaScript code. The code includes three functions: `jsAlert()`, `jsConfirm()`, and `jsPrompt()`. These functions are triggered by the buttons on the page. The developer tools also show the resulting DOM structure, including the `<div>` element where the result of the click is displayed. The status bar at the bottom indicates the result of the click: "You clicked: Cancel".

Page Class

```

public class JavaScriptAlertsPage extends BasePage {

    private static final By selectorAlertButton = By.cssSelector("button[onclick*=jsAlert]");
    private static final By selectorConfirmButton = By.cssSelector("button[onclick*=jsConfirm]");
    private static final By selectorPromptButton = By.cssSelector("button[onclick*=jsPrompt]");
    private static final By resultLabelSelector = By.cssSelector("p#result");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.JAVASCRIPT_ALERTS.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'JavaScript Alerts' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.JAVASCRIPT_ALERTS.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Clicks 'JS alert' button.
     */
    public void clickAlertButton() {
        new Button(selectorAlertButton).click();
        WebDriverWait wait = new WebDriverWait(getDriver(), 2);
        wait.until(ExpectedConditions.alertIsPresent());
    }

    /**
     * Clicks 'JS confirm' button.
     */
    public void clickConfirmButton() {
        new Button(selectorConfirmButton).click();
        WebDriverWait wait = new WebDriverWait(getDriver(), 2);
        wait.until(ExpectedConditions.alertIsPresent());
    }

    /**

```

```

    * Clicks 'JS prompt' button.
    */
public void clickPromptButton() {
    new Button(selectorPromptButton).click();
    WebDriverWait wait = new WebDriverWait(getDriver(), 2);
    wait.until(ExpectedConditions.alertIsPresent());
}

/**
 * Returns message displayed by popup.
 *
 * @return String object representing message displayed by popup
 */
public String readResultLabel() {
    return new LabelElement(resultLabelSelector).getText();
}

/**
 * Clicks alert's 'OK' button.
 */
public void clickAlertAccept() {
    getDriver().switchTo()
        .alert()
        .accept();
}

/**
 * Clicks alert's 'Cancel' button.
 */
public void clickAlertDismiss() {
    getDriver().switchTo()
        .alert()
        .dismiss();
}

/**
 * Types text into alert's text field.
 *
 * @param text String object sent into alert's text field
 */
public void writeTextInAlert(String text) {
    getDriver().switchTo()
        .alert()
        .sendKeys(text);
}
}

```

alert()

Using `switchTo()` method you can change processed content. `switchTo().alert()` allows performing

actions on appearing alerts such as accepting, dismissing or entering keys.

Test Class

Before all tests: Open The Internet Main Page and go to JavaScript Alert page

1. Click JS Alert button, accept alert and check if Result message returns performed an action
2. Click JS Confirm button, accept alert and check if Result message returns performed action
3. Click JS Confirm button, dismiss alert and check if Result message returns performed action
4. Click JS Prompt button, write random text, accept alert and check if Result message returns performed action with written text
5. Click JS Prompt button, dismiss the alert and check if Result message returns performed action

After each case: Refresh Page

After all tests: Navigate back to The Internet Main Page

```
@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class JavaScriptAlertsTest extends TheInternetBaseTest {

    private static JavaScriptAlertsPage javaScriptAlertsPage;

    private final String jsAlertCofirmMessage      = "You successfully clicked an alert";
    private final String jsConfirmConfirmMessage   = "You clicked: Ok";
    private final String jsConfirmCancelMessage   = "You clicked: Cancel";
    private final String jsPromptConfirmMessage   = "You entered: ";
    private final String jsPromptCancelMessage   = "You entered: null";
    private final String randomString           = "random";

    @BeforeClass
    public static void setUpBeforeClass() {
        javaScriptAlertsPage =
    shouldTheInternetPageBeOpened().clickJavaScriptAlertLink();

        logStep("Verify if JavaScript Alerts page is opened");
        assertTrue("Unable to open JavaScript Alerts page",
    javaScriptAlertsPage.isLoaded());
    }

    @AfterClass
    public static void tearDownAfterClass() {
        logStep("Navigate back to The-Internet page");
        BasePage.navigateBack();
    }

    @Test
    public void shouldJSAlertCloseWithProperMessageAfterPressOkButton() {
        logStep("Click Alert button");
    }
}
```

```
javaScriptAlertsPage.clickAlertButton();

logStep("Click 'OK' button on alert");
javaScriptAlertsPage.clickAlertAccept();

logStep("Verify returned message");
assertEquals("Incorrect message returned after click",
            jsAlertCofirmMessage, javaScriptAlertsPage.readResultLabel());
}

@Test
public void shouldJSConfirmCloseWithProperMessageAfterPressOkButton() {
    logStep("Click Confirm button");
    javaScriptAlertsPage.clickConfirmButton();

    logStep("Click 'OK' button on alert");
    javaScriptAlertsPage.clickAlertAccept();

    logStep("Verify returned message");
    assertEquals("Incorrect message returned after click",
                jsConfirmConfirmMessage, javaScriptAlertsPage.readResultLabel());
}

@Test
public void shouldJSConfirmCloseWithProperMessageAfterPressCancelButton() {
    logStep("Click Confirm button");
    javaScriptAlertsPage.clickConfirmButton();

    logStep("Click 'Cancel' button on alert");
    javaScriptAlertsPage.clickAlertDismiss();

    logStep("Verify returned message");
    assertEquals("Incorrect message returned after click",
                jsConfirmCancelMessage, javaScriptAlertsPage.readResultLabel());
}

@Test
public void shouldJSPromptCloseWithProperMessageAfterPressOKButton() {
    logStep("Click Prompt button");
    javaScriptAlertsPage.clickPromptButton();

    logStep("Insert text to alert: " + randomString);
    javaScriptAlertsPage.writeTextInAlert(randomString);

    logStep("Click 'OK' button on alert");
    javaScriptAlertsPage.clickAlertAccept();

    logStep("Verify returned message");
    assertEquals("Incorrect message returned after click",
                jsPromptConfirmMessage + randomString,
                javaScriptAlertsPage.readResultLabel());
}
```

```
}
```

```
@Test
public void shouldJSPromptCloseWithProperMessageAfterPressCancelButton() {
    logStep("Click Prompt button");
    javaScriptAlertsPage.clickPromptButton();

    logStep("Click 'Cancel' button on alert");
    javaScriptAlertsPage.clickAlertDismiss();

    logStep("Verify returned message");
    assertEquals("Incorrect message returned after click",
                jsPromptCancelMessage, javaScriptAlertsPage.readResultLabel());
}

@Override
public void tearDown() {
    logStep("Refresh JavaScriptAlersPage");
    javaScriptAlertsPage.refreshPage();
}

}
```

[Fork me on GitHub](#)

Key Presses

Key presses are often used to interact with a website (e.g., tab order, enter, escape, etc.). Press a key and see what you inputted.

You entered: G

Powered by [Elemental Selenium](#)

This simple case shows how to test key pressing

This site uses JavaScript to read the key pressed and display its value.

Key Presses

Key presses are often used to interact with a website (e.g., tab order, enter, escape, etc.). Press a key and see what you inputted.

You entered: J

Powered by Elemental Selenium

```

Fork me on GitHub
Elements Console Sources Network Performance > 02 | X
<div>
  <div class="row">
    <div id="content" class="large-12 columns">
      <script>
        $(document).keydown(function(event){
          output = keyboardMap[event.which];
          log('You entered: ' + output);
        });
        function log(msg) {
          var result = document.getElementById('result');
          result.innerHTML = msg;
        }
      // names of known key codes (0-255)
      var keyboardMap = {
        "0": "0", "1": "1", "2": "2", "3": "3", "4": "4", "5": "5", "6": "6", "7": "7", "8": "8", "9": "9", "A": "A", "B": "B", "C": "C", "D": "D", "E": "E", "F": "F", "G": "G", "H": "H", "I": "I", "J": "J", "K": "K", "L": "L", "M": "M", "N": "N", "O": "O", "P": "P", "Q": "Q", "R": "R", "S": "S", "T": "T", "U": "U", "V": "V", "Y": "Y", "X": "X", "Z": "Z", "F1": "F1", "F2": "F2", "F3": "F3", "F4": "F4", "F5": "F5", "F6": "F6", "F7": "F7", "F8": "F8", "F9": "F9", "F10": "F10", "F11": "F11", "F12": "F12", "F13": "F13", "F14": "F14", "F15": "F15", "F16": "F16", "F17": "F17", "F18": "F18", "F19": "F19", "F20": "F20", "F21": "F21", "F22": "F22", "F23": "F23", "F24": "F24", "F25": "F25", "F26": "F26", "F27": "F27", "F28": "F28", "F29": "F29", "F30": "F30", "F31": "F31", "F32": "F32", "F33": "F33", "F34": "F34", "F35": "F35", "F36": "F36", "F37": "F37", "F38": "F38", "F39": "F39", "F40": "F40", "F41": "F41", "F42": "F42", "F43": "F43", "F44": "F44", "F45": "F45", "F46": "F46", "F47": "F47", "F48": "F48", "F49": "F49", "F50": "F50", "F51": "F51", "F52": "F52", "F53": "F53", "F54": "F54", "F55": "F55", "F56": "F56", "F57": "F57", "F58": "F58", "F59": "F59", "F60": "F60", "F61": "F61", "F62": "F62", "F63": "F63", "F64": "F64", "F65": "F65", "F66": "F66", "F67": "F67", "F68": "F68", "F69": "F69", "F70": "F70", "F71": "F71", "F72": "F72", "F73": "F73", "F74": "F74", "F75": "F75", "F76": "F76", "F77": "F77", "F78": "F78", "F79": "F79", "F7A": "F7A", "F7B": "F7B", "F7C": "F7C", "F7D": "F7D", "F7E": "F7E", "F7F": "F7F", "F7G": "F7G", "F7H": "F7H", "F7I": "F7I", "F7J": "F7J", "F7K": "F7K", "F7L": "F7L", "F7M": "F7M", "F7N": "F7N", "F7O": "F7O", "F7P": "F7P", "F7Q": "F7Q", "F7R": "F7R", "F7S": "F7S", "F7T": "F7T", "F7U": "F7U", "F7V": "F7V", "F7Y": "F7Y", "F7X": "F7X", "F7Z": "F7Z", "F7F1": "F7F1", "F7F2": "F7F2", "F7F3": "F7F3", "F7F4": "F7F4", "F7F5": "F7F5", "F7F6": "F7F6", "F7F7": "F7F7", "F7F8": "F7F8", "F7F9": "F7F9", "F7F0": "F7F0", "F7F11": "F7F11", "F7F12": "F7F12", "F7F13": "F7F13", "F7F14": "F7F14", "F7F15": "F7F15", "F7F16": "F7F16", "F7F17": "F7F17", "F7F18": "F7F18", "F7F19": "F7F19", "F7F20": "F7F20", "F7F21": "F7F21", "F7F22": "F7F22", "F7F23": "F7F23", "F7F24": "F7F24", "F7F25": "F7F25", "F7F26": "F7F26", "F7F27": "F7F27", "F7F28": "F7F28", "F7F29": "F7F29", "F7F30": "F7F30", "F7F31": "F7F31", "F7F32": "F7F32", "F7F33": "F7F33", "F7F34": "F7F34", "F7F35": "F7F35", "F7F36": "F7F36", "F7F37": "F7F37", "F7F38": "F7F38", "F7F39": "F7F39", "F7F40": "F7F40", "F7F41": "F7F41", "F7F42": "F7F42", "F7F43": "F7F43", "F7F44": "F7F44", "F7F45": "F7F45", "F7F46": "F7F46", "F7F47": "F7F47", "F7F48": "F7F48", "F7F49": "F7F49", "F7F50": "F7F50", "F7F51": "F7F51", "F7F52": "F7F52", "F7F53": "F7F53", "F7F54": "F7F54", "F7F55": "F7F55", "F7F56": "F7F56", "F7F57": "F7F57", "F7F58": "F7F58", "F7F59": "F7F59", "F7F60": "F7F60", "F7F61": "F7F61", "F7F62": "F7F62", "F7F63": "F7F63", "F7F64": "F7F64", "F7F65": "F7F65", "F7F66": "F7F66", "F7F67": "F7F67", "F7F68": "F7F68", "F7F69": "F7F69", "F7F70": "F7F70", "F7F71": "F7F71", "F7F72": "F7F72", "F7F73": "F7F73", "F7F74": "F7F74", "F7F75": "F7F75", "F7F76": "F7F76", "F7F77": "F7F77", "F7F78": "F7F78", "F7F79": "F7F79", "F7F7A": "F7F7A", "F7F7B": "F7F7B", "F7F7C": "F7F7C", "F7F7D": "F7F7D", "F7F7E": "F7F7E", "F7F7F": "F7F7F", "F7F7G": "F7F7G", "F7F7H": "F7F7H", "F7F7I": "F7F7I", "F7F7J": "F7F7J", "F7F7K": "F7F7K", "F7F7L": "F7F7L", "F7F7M": "F7F7M", "F7F7N": "F7F7N", "F7F7O": "F7F7O", "F7F7P": "F7F7P", "F7F7Q": "F7F7Q", "F7F7R": "F7F7R", "F7F7S": "F7F7S", "F7F7T": "F7F7T", "F7F7U": "F7F7U", "F7F7V": "F7F7V", "F7F7Y": "F7F7Y", "F7F7X": "F7F7X", "F7F7Z": "F7F7Z", "F7F7F1": "F7F7F1", "F7F7F2": "F7F7F2", "F7F7F3": "F7F7F3", "F7F7F4": "F7F7F4", "F7F7F5": "F7F7F5", "F7F7F6": "F7F7F6", "F7F7F7": "F7F7F7", "F7F7F8": "F7F7F8", "F7F7F9": "F7F7F9", "F7F7F0": "F7F7F0", "F7F7F11": "F7F7F11", "F7F7F12": "F7F7F12", "F7F7F13": "F7F7F13", "F7F7F14": "F7F7F14", "F7F7F15": "F7F7F15", "F7F7F16": "F7F7F16", "F7F7F17": "F7F7F17", "F7F7F18": "F7F7F18", "F7F7F19": "F7F7F19", "F7F7F20": "F7F7F20", "F7F7F21": "F7F7F21", "F7F7F22": "F7F7F22", "F7F7F23": "F7F7F23", "F7F7F24": "F7F7F24", "F7F7F25": "F7F7F25", "F7F7F26": "F7F7F26", "F7F7F27": "F7F7F27", "F7F7F28": "F7F7F28", "F7F7F29": "F7F7F29", "F7F7F30": "F7F7F30", "F7F7F31": "F7F7F31", "F7F7F32": "F7F7F32", "F7F7F33": "F7F7F33", "F7F7F34": "F7F7F34", "F7F7F35": "F7F7F35", "F7F7F36": "F7F7F36", "F7F7F37": "F7F7F37", "F7F7F38": "F7F7F38", "F7F7F39": "F7F7F39", "F7F7F40": "F7F7F40", "F7F7F41": "F7F7F41", "F7F7F42": "F7F7F42", "F7F7F43": "F7F7F43", "F7F7F44": "F7F7F44", "F7F7F45": "F7F7F45", "F7F7F46": "F7F7F46", "F7F7F47": "F7F7F47", "F7F7F48": "F7F7F48", "F7F7F49": "F7F7F49", "F7F7F50": "F7F7F50", "F7F7F51": "F7F7F51", "F7F7F52": "F7F7F52", "F7F7F53": "F7F7F53", "F7F7F54": "F7F7F54", "F7F7F55": "F7F7F55", "F7F7F56": "F7F7F56", "F7F7F57": "F7F7F57", "F7F7F58": "F7F7F58", "F7F7F59": "F7F7F59", "F7F7F60": "F7F7F60", "F7F7F61": "F7F7F61", "F7F7F62": "F7F7F62", "F7F7F63": "F7F7F63", "F7F7F64": "F7F7F64", "F7F7F65": "F7F7F65", "F7F7F66": "F7F7F66", "F7F7F67": "F7F7F67", "F7F7F68": "F7F7F68", "F7F7F69": "F7F7F69", "F7F7F70": "F7F7F70", "F7F7F71": "F7F7F71", "F7F7F72": "F7F7F72", "F7F7F73": "F7F7F73", "F7F7F74": "F7F7F74", "F7F7F75": "F7F7F75", "F7F7F76": "F7F7F76", "F7F7F77": "F7F7F77", "F7F7F78": "F7F7F78", "F7F7F79": "F7F7F79", "F7F7F7A": "F7F7F7A", "F7F7F7B": "F7F7F7B", "F7F7F7C": "F7F7F7C", "F7F7F7D": "F7F7F7D", "F7F7F7E": "F7F7F7E", "F7F7F7F": "F7F7F7F", "F7F7F7G": "F7F7F7G", "F7F7F7H": "F7F7F7H", "F7F7F7I": "F7F7F7I", "F7F7F7J": "F7F7F7J", "F7F7F7K": "F7F7F7K", "F7F7F7L": "F7F7F7L", "F7F7F7M": "F7F7F7M", "F7F7F7N": "F7F7F7N", "F7F7F7O": "F7F7F7O", "F7F7F7P": "F7F7F7P", "F7F7F7Q": "F7F7F7Q", "F7F7F7R": "F7F7F7R", "F7F7F7S": "F7F7F7S", "F7F7F7T": "F7F7F7T", "F7F7F7U": "F7F7F7U", "F7F7F7V": "F7F7F7V", "F7F7F7Y": "F7F7F7Y", "F7F7F7X": "F7F7F7X", "F7F7F7Z": "F7F7F7Z", "F7F7F7F1": "F7F7F7F1", "F7F7F7F2": "F7F7F7F2", "F7F7F7F3": "F7F7F7F3", "F7F7F7F4": "F7F7F7F4", "F7F7F7F5": "F7F7F7F5", "F7F7F7F6": "F7F7F7F6", "F7F7F7F7": "F7F7F7F7", "F7F7F7F8": "F7F7F7F8", "F7F7F7F9": "F7F7F7F9", "F7F7F7F0": "F7F7F7F0", "F7F7F7F11": "F7F7F7F11", "F7F7F7F12": "F7F7F7F12", "F7F7F7F13": "F7F7F7F13", "F7F7F7F14": "F7F7F7F14", "F7F7F7F15": "F7F7F7F15", "F7F7F7F16": "F7F7F7F16", "F7F7F7F17": "F7F7F7F17", "F7F7F7F18": "F7F7F7F18", "F7F7F7F19": "F7F7F7F19", "F7F7F7F20": "F7F7F7F20", "F7F7F7F21": "F7F7F7F21", "F7F7F7F22": "F7F7F7F22", "F7F7F7F23": "F7F7F7F23", "F7F7F7F24": "F7F7F7F24", "F7F7F7F25": "F7F7F7F25", "F7F7F7F26": "F7F7F7F26", "F7F7F7F27": "F7F7F7F27", "F7F7F7F28": "F7F7F7F28", "F7F7F7F29": "F7F7F7F29", "F7F7F7F30": "F7F7F7F30", "F7F7F7F31": "F7F7F7F31", "F7F7F7F32": "F7F7F7F32", "F7F7F7F33": "F7F7F7F33", "F7F7F7F34": "F7F7F7F34", "F7F7F7F35": "F7F7F7F35", "F7F7F7F36": "F7F7F7F36", "F7F7F7F37": "F7F7F7F37", "F7F7F7F38": "F7F7F7F38", "F7F7F7F39": "F7F7F7F39", "F7F7F7F40": "F7F7F7F40", "F7F7F7F41": "F7F7F7F41", "F7F7F7F42": "F7F7F7F42", "F7F7F7F43": "F7F7F7F43", "F7F7F7F44": "F7F7F7F44", "F7F7F7F45": "F7F7F7F45", "F7F7F7F46": "F7F7F7F46", "F7F7F7F47": "F7F7F7F47", "F7F7F7F48": "F7F7F7F48", "F7F7F7F49": "F7F7F7F49", "F7F7F7F50": "F7F7F7F50", "F7F7F7F51": "F7F7F7F51", "F7F7F7F52": "F7F7F7F52", "F7F7F7F53": "F7F7F7F53", "F7F7F7F54": "F7F7F7F54", "F7F7F7F55": "F7F7F7F55", "F7F7F7F56": "F7F7F7F56", "F7F7F7F57": "F7F7F7F57", "F7F7F7F58": "F7F7F7F58", "F7F7F7F59": "F7F7F7F59", "F7F7F7F60": "F7F7F7F60", "F7F7F7F61": "F7F7F7F61", "F7F7F7F62": "F7F7F7F62", "F7F7F7F63": "F7F7F7F63", "F7F7F7F64": "F7F7F7F64", "F7F7F7F65": "F7F7F7F65", "F7F7F7F66": "F7F7F7F66", "F7F7F7F67": "F7F7F7F67", "F7F7F7F68": "F7F7F7F68", "F7F7F7F69": "F7F7F7F69", "F7F7F7F70": "F7F7F7F70", "F7F7F7F71": "F7F7F7F71", "F7F7F7F72": "F7F7F7F72", "F7F7F7F73": "F7F7F7F73", "F7F7F7F74": "F7F7F7F74", "F7F7F7F75": "F7F7F7F75", "F7F7F7F76": "F7F7F7F76", "F7F7F7F77": "F7F7F7F77", "F7F7F7F78": "F7F7F7F78", "F7F7F7F79": "F7F7F7F79", "F7F7F7F7A": "F7F7F7F7A", "F7F7F7F7B": "F7F7F7F7B", "F7F7F7F7C": "F7F7F7F7C", "F7F7F7F7D": "F7F7F7F7D", "F7F7F7F7E": "F7F7F7F7E", "F7F7F7F7F": "F7F7F7F7F", "F7F7F7F7G": "F7F7F7F7G", "F7F7F7F7H": "F7F7F7F7H", "F7F7F7F7I": "F7F7F7F7I", "F7F7F7F7J": "F7F7F7F7J", "F7F7F7F7K": "F7F7F7F7K", "F7F7F7F7L": "F7F7F7F7L", "F7F7F7F7M": "F7F7F7F7M", "F7F7F7F7N": "F7F7F7F7N", "F7F7F7F7O": "F7F7F7F7O", "F7F7F7F7P": "F7F7F7F7P", "F7F7F7F7Q": "F7F7F7F7Q", "F7F7F7F7R": "F7F7F7F7R", "F7F7F7F7S": "F7F7F7F7S", "F7F7F7F7T": "F7F7F7F7T", "F7F7F7F7U": "F7F7F7F7U", "F7F7F7F7V": "F7F7F7F7V", "F7F7F7F7Y": "F7F7F7F7Y", "F7F7F7F7X": "F7F7F7F7X", "F7F7F7F7Z": "F7F7F7F7Z", "F7F7F7F7F1": "F7F7F7F7F1", "F7F7F7F7F2": "F7F7F7F7F2", "F7F7F7F7F3": "F7F7F7F7F3", "F7F7F7F7F4": "F7F7F7F7F4", "F7F7F7F7F5": "F7F7F7F7F5", "F7F7F7F7F6": "F7F7F7F7F6", "F7F7F7F7F7": "F7F7F7F7F7", "F7F7F7F7F8": "F7F7F7F7F8", "F7F7F7F7F9": "F7F7F7F7F9", "F7F7F7F7F0": "F7F7F7F7F0", "F7F7F7F7F11": "F7F7F7F7F11", "F7F7F7F7F12": "F7F7F7F7F12", "F7F7F7F7F13": "F7F7F7F7F13", "F7F7F7F7F14": "F7F7F7F7F14", "F7F7F7F7F15": "F7F7F7F7F15", "F7F7F7F7F16": "F7F7F7F7F16", "F7F7F7F7F17": "F7F7F7F7F17", "F7F7F7F7F18": "F7F7F7F7F18", "F7F7F7F7F19": "F7F7F7F7F19", "F7F7F7F7F20": "F7F7F7F7F20", "F7F7F7F7F21": "F7F7F7F7F21", "F7F7F7F7F22": "F7F7F7F7F22", "F7F7F7F7F23": "F7F7F7F7F23", "F7F7F7F7F24": "F7F7F7F7F24", "F7F7F7F7F25": "F7F7F7F7F25", "F7F7F7F7F26": "F7F7F7F7F26", "F7F7F7F7F27": "F7F7F7F7F27", "F7F7F7F7F28": "F7F7F7F7F28", "F7F7F7F7F29": "F7F7F7F7F29", "F7F7F7F7F30": "F7F7F7F7F30", "F7F7F7F7F31": "F7F7F7F7F31", "F7F7F7F7F32": "F7F7F7F7F32", "F7F7F7F7F33": "F7F7F7F7F33", "F7F7F7F7F34": "F7F7F7F7F34", "F7F7F7F7F35": "F7F7F7F7F35", "F7F7F7F7F36": "F7F7F7F7F36", "F7F7F7F7F37": "F7F7F7F7F37", "F7F7F7F7F38": "F7F7F7F7F38", "F7F7F7F7F39": "F7F7F7F7F39", "F7F7F7F7F40": "F7F7F7F7F40", "F7F7F7F7F41": "F7F7F7F7F41", "F7F7F7F7F42": "F7F7F7F7F42", "F7F7F7F7F43": "F7F7F7F7F43", "F7F7F7F7F44": "F7F7F7F7F44", "F7F7F7F7F45": "F7F7F7F7F45", "F7F7F7F7F46": "F7F7F7F7F46", "F7F7F7F7F47": "F7F7F7F7F47", "F7F7F7F7F48": "F7F7F7F7F48", "F7F7F7F7F49": "F7F7F7F7F49", "F7F7F7F7F50": "F7F7F7F7F50", "F7F7F7F7F51": "F7F7F7F7F51", "F7F7F7F7F52": "F7F7F7F7F52", "F7F7F7F7F53": "F7F7F7F7F53", "F7F7F7F7F54": "F7F7F7F7F54", "F7F7F7F7F55": "F7F7F7F7F55", "F7F7F7F7F56": "F7F7F7F7F56", "F7F7F7F7F57": "F7F7F7F7F57", "F7F7F7F7F58": "F7F7F7F7F58", "F7F7F7F7F59": "F7F7F7F7F59", "F7F7F7F7F60": "F7F7F7F7F60", "F7F7F7F7F61": "F7F7F7F7F61", "F7F7F7F7F62": "F7F7F7F7F62", "F7F7F7F7F63": "F7F7F7F7F63", "F7F7F7F7F64": "F7F7F7F7F64", "F7F7F7F7F65": "F7F7F7F7F65", "F7F7F7F7F66": "F7F7F7F7F66", "F7F7F7F7F67": "F7F7F7F7F67", "F7F7F7F7F68": "F7F7F7F7F68", "F7F7F7F7F69": "F7F7F7F7F69", "F7F7F7F7F70": "F7F7F7F7F70", "F7F7F7F7F71": "F7F7F7F7F71", "F7F7F7F7F72": "F7F7F7F7F72", "F7F7F7F7F73": "F7F7F7F7F73", "F7F7F7F7F74": "F7F7F7F7F74", "F7F7F7F7F75": "F7F7F7F7F75", "F7F7F7F7F76": "F7F7F7F7F76", "F7F7F7F7F77": "F7F7F7F7F77", "F7F7F7F7F78": "F7F7F7F7F78", "F7F7F7F7F79": "F7F7F7F7F79", "F7F7F7F7F7A": "F7F7F7F7F7A", "F7F7F7F7F7B": "F7F7F7F7F7B", "F7F7F7F7F7C": "F7F7F7F7F7C", "F7F7F7F7F7D": "F7F7F7F7F7D", "F7F7F7F7F7E": "F7F7F7F7F7E", "F7F7F7F7F7F": "F7F7F7F7F7F", "F7F7F7F7F7G": "F7F7F7F7F7G", "F7F7F7F7F7H": "F7F7F7F7F7H", "F7F7F7F7F7I": "F7F7F7F7F7I", "F7F7F7F7F7J": "F7F7F7F7F7J", "F7F7F7F7F7K": "F7F7F7F7F7K", "F7F7F7F7F7L": "F7F7F7F7F7L", "F7F7F7F7F7M": "F7F7F7F7F7M", "F7F7F7F7F7N": "F7F7F7F7F7N", "F7F7F7F7F7O": "F7F7F7F7F7O", "F7F7F7F7F7P": "F7F7F7F7F7P", "F7F7F7F7F7Q": "F7F7F7F7F7Q", "F7F7F7F7F7R": "F7F7F7F7F7R", "F7F7F7F7F7S": "F7F7F7F7F7S", "F7F7F7F7F7T": "F7F7F7F7F7T", "F7F7F7F7F7U": "F7F7F7F7F7U", "F7F7F7F7F7V": "F7F7F7F7F7V", "F7F7F7F7F7Y": "F7F7F7F7F7Y", "F7F7F7F7F7X": "F7F7F7F7F7X", "F7F7F7F7F7Z": "F7F7F7F7F7Z", "F7F7F7F7F7F1": "F7F7F7F7F7F1", "F7F7F7F7F7F2": "F7F7F7F7F7F2", "F7F7F7F7F7F3": "F7F7F7F7F7F3", "F7F7F7F7F7F4": "F7F7F7F7F7F4", "F7F7F7F7F7F5": "F7F7F7F7F7F5", "F7F7F7F7F7F6": "F7F7F7F7F7F6", "F7F7F7F7F7F7": "F7F7F7F7F7F7", "F7F7F7F7F7F8": "F7F7F7F7F7F8", "F7F7F7F7F7F9": "F7F7F7F7F7F9", "F7F7F7F7F7F0": "F7F7F7F7F7F0", "F7F7F7F7F7F11": "F7F7F7F7F7F11", "F7F7F7F7F7F12": "F7F7F7F7F7F12", "F7F7F7F7F7F13": "F7F7F7F7F7F13", "F7F7F7F7F7F14": "F7F7F7F7F7F14", "F7F7F7F7F7F15": "F7F7F7F7F7F15", "F7F7F7F7F7F16": "F7F7F7F7F7F16", "F7F7F7F7F7F17": "F7F7F7F7F7F17", "F7F7F7F7F7F18": "F7F7F7F7F7F18", "F7F7F7F7F7F19": "F7F7F7F7F7F19", "F7F7F7F7F7F20": "F7F7F7F7F7F20", "F7F7F7F7F7F21": "F7F7F7F7F7F21", "F7F7F7F7F7F22": "F7F7F7F7F7F22", "F7F7F7F7F7F23": "F7F7F7F7F7F23", "F7F7F7F7F7F24": "F7F7F7F7F7F24", "F7F7F7F7F7F25": "F7F7F7F7F7F25", "F7F7F7F7F7F26": "F7F7F7F7F7F26", "F7F7F7F7F7F27": "F7F7F7F7F7F27", "F7F7F7F7F7F28": "F7F7F7F7F7F28", "F7F7F7F7F7F29": "F7F7F7F7F7F29", "F7F7F7F7F7F30": "F7F7F7F7F7F30", "F7F7F7F7F7F31": "F7F7F7F7F7F31", "F7F7F7F7F7F32": "F7F7F7F7F7F32", "F7F7F7F7F7F33": "F7F7F7F7F7F33", "F7F7F7F7F7F34": "F7F7F7F7F7F34", "F7F7F7F7F7F35": "F7F7F7F7F7F35", "F7F7F7F7F7F36": "F7F7F7F7F7F36", "F7F7F7F7F7F37": "F7F7F7F7F7F37", "F7F7F7F7F7F38": "F7F7F7F7F7F38", "F7F7F7F7F7F39": "F7F7F7F7F7F39", "F7F7F7F7F7F40": "F7F7F7F7F7F40", "F7F7F7F7F7F41": "F7F7F7F7F7F41", "F7F7F7F7F7F42": "F7F7F7F7F7F42", "F7F7F7F7F7F43": "F7F7F7F7F7F43", "F7F7F7F7F7F44": "F7F7F7F7F7F44", "F7F7F7F7F7F45": "F7F7F7F7F7F45", "F7F7F7F7F7F46": "F7F7F7F7F7F46", "F7F7F7F7F7F47": "F7F7F7F7F7F47", "F7F7F7F7F7F48": "F7F7F7F7F7F48", "F7F
```

```

public class KeyPressesPage extends BasePage {

    private static final By selectorResult = By.cssSelector("#result");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.KEY_PRESS.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Key Presses' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.KEY_PRESS.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Presses given keyboard key.
     *
     * @param keyToPress Key to be pressed on keyboard
     */
    public void pressKey(String keyToPress) {
        getAction().sendKeys(keyToPress)
            .perform();
    }

    /**
     * Returns information from web page about pressed keyboard key.
     *
     * @return Information from web page about pressed key
     */
    public String getPressedKeyInformation() {
        return getDriver().findElementDynamic(selectorResult)
            .getText();
    }
}

```

Test Class

Steps:

1. Open The Internet Main Page

2. Go to Key Presses site
3. Press a key
4. Check if a displayed message contains the pressed key

```
@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class KeyPressesTest extends TheInternetBaseTest {

    private static KeyPressesPage keyPressesPage;

    private final String keyToBePressed = "Q";
    private final String expectedMessage = "You entered: Q";

    @BeforeClass
    public static void setUpBeforeClass() {
        keyPressesPage = shouldTheInternetPageBeOpened().clickKeyPressesLink();

        logStep("Verify if Key Presses page is opened");
        assertTrue("Unable to open Key Presses page", keyPressesPage.isLoaded());
    }

    @Test
    public void shouldWebsiteReturnInformationAboutPressedKey() {
        logStep("Press a keyboard key");
        keyPressesPage.pressKey(keyToBePressed);

        logStep("Verify if website give valid information about pressed keyboard
key");
        assertEquals("Information about the pressed key is invalid", expectedMessage,
                     keyPressesPage.getPressedKeyInformation());
    }
}
```

Opening a new window

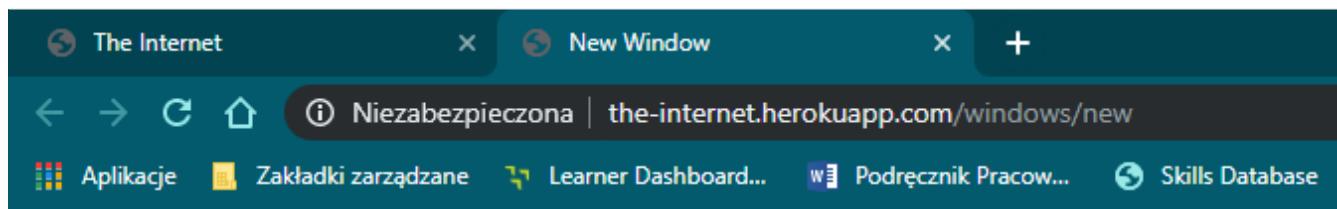
[Click Here](#)

Fork me on GitHub

Powered by Elemental Selenium

This simple example shows how operate on many browser tabs

When you click the link, a new website will be opened in the second tab.



New Window

Page Class

```
public class MultipleWindowsPage extends BasePage {

    private final static By selectorLink = By.cssSelector("#content > div > a");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.WINDOW.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Opening a new window' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.WINDOW.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Clicks 'click here' link.
     *
     * @return NewWindowPage object
     */
    public NewWindowPage clickHereLink() {
        getDriver().findElementDynamic(selectorLink)
            .click();
        getDriver().waitForPageLoaded();
        return new NewWindowPage();
    }
}
```

You also need a second page class for New Window Page. Implement only the required methods.

```
public class NewWindowPage extends BasePage {

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.NEW_WINDOW.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'New window' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.NEW_WINDOW.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }
}
```

Test Class

Steps:

1. Open The Internet Main Page
2. Go to Multiple Windows Page
3. Click the link
4. Check if a new page is opened in the second tab

```

@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class MultipleWindowsTest extends TheInternetBaseTest {

    private static MultipleWindowsPage    multipleWindowsPage;
    private static NewTabPage             newTabPage;

    @BeforeClass
    public static void setUpBeforeClass() {
        multipleWindowsPage =
shouldTheInternetPageBeOpened().clickmultipleWindowsLink();

        logStep("Verify if Multiple Windows page is opened");
        assertTrue("Unable to open Multiple Windows page",
multipleWindowsPage.isLoaded());
    }

    @Test
    public void verifyIfNewBrowserWindowOpen() {
        logStep("Click 'Click here' link");
        newTabPage = multipleWindowsPage.clickHereLink();

        logStep("Verify if 'New window page' is opened");
        assertTrue("Unable to open a new browser window", newTabPage.isLoaded());
    }
}

```

[Fork me on GitHub](#)

Redirection

This is separate from directly returning a redirection status code, in that some browsers cannot handle a raw redirect status code without a destination page as part of the HTTP response.

[Click here](#) to trigger a redirect (and be taken to the status codes page).

Powered by [Elemental Selenium](#)

This simple case shows how to approach redirecting links.

After clicking on the link, you will be redirected to Status Codes Page.

[Fork me on GitHub](#)

Status Codes

HTTP status codes are a standard set of numbers used to communicate from a web server to your browser to indicate the outcome of the request being made (e.g. Success, Redirection, Client Error, Server Error). For a complete list of status codes, go [here](#).

Some standard status codes you will run into include but are not limited to:

- 200
- 301
- 404
- 500

Powered by [Elemental Selenium](#)

Page Class

Redirect Link Page

```
public class RedirectLinkPage extends BasePage {

    private static final By selectorRedirectHere = By.cssSelector("a#redirect");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.REDIRECT.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Redirection' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.REDIRECT.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Clicks 'Redirect here' link.
     *
     * @return StatusCodesHomePage object
     */
    public StatusCodesHomePage clickRedirectHereLink() {
        new Button(selectorRedirectHere).click();
        return new StatusCodesHomePage();
    }
}
```

Status Codes Page

```
public class StatusCodesHomePage extends BasePage {

    private static final By selectorLink200Code = By.linkText("200");
    private static final By selectorLink301Code = By.linkText("301");
    private static final By selectorLink404Code = By.linkText("404");
    private static final By selectorLink500Code = By.linkText("500");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.STATUS_CODES.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Status Codes' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.STATUS_CODES.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }
}
```

Test Class

Steps:

1. Open The Internet Main Page
2. Go to Redirection Page
3. Click the link
4. Check if Status Codes Page is loaded

```

@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class
})
public class RedirectLinkTest extends TheInternetBaseTest {

    private static RedirectLinkPage    redirectLinkPage;
    private static StatusCodesHomePage statusCodesHomePage;

    @BeforeClass
    public static void setUpBeforeClass() {
        redirectLinkPage = shouldTheInternetPageBeOpened().clickRedirectLink();

        logStep("Verify if Redirect Link page is opened");
        assertTrue("Unable to open Redirect Link page", redirectLinkPage.isLoaded());
    }

    @Test
    public void shouldUserBeRedirectedToStatusCodePage() {
        logStep("Click 'Redirect here' link");
        statusCodesHomePage = redirectLinkPage.clickRedirectHereLink();

        logStep("Verify redirection to Status Code page");
        assertTrue("User hasn't been redirected to the expected website",
                   statusCodesHomePage.isLoaded());
    }
}

```

[Fork me on GitHub](#)

Horizontal Slider

Set the focus on the slider (by clicking on it) and use the arrow keys to move it right and left. Or click and drag the slider with your mouse. It will indicate the value of the slider to the right.



Powered by [Elemental Selenium](#)

This case shows how to move horizontal slider.

You can move the slider by dragging it with a mouse or using arrow keys. The page uses a simple script to get slider position and display set value.

Horizontal Slider

Set the focus on the slider (by clicking on it) and use the arrow keys to move it right and left. Or click and drag the slider with your mouse. It will indicate the value of the slider to the right.



Powered by Elemental Selenium

The screenshot shows the browser's developer tools with the 'Elements' tab selected. The DOM tree is displayed, showing the structure of the HTML page. The slider element is identified by the class 'sliderContainer'. The input field within the slider container has an ID of 'range' and a value of '2.5'. A script block is also visible, containing a function named 'showValue' which updates the value of the input field.

Page Class

```
public class HorizontalSliderPage extends BasePage {

    private static final By selectorHorizontalSlider =
        By.cssSelector("div.sliderContainer");
    private static final By sliderSelector = By.cssSelector("input");
    private static final By valueSelector = By.cssSelector("#range");

    private HorizontalSliderElement horizontalSlider;

    public HorizontalSliderPage() {
        horizontalSlider =
            getDriver().elementHorizontalSlider(selectorHorizontalSlider,
                sliderSelector, valueSelector, BigDecimal.ZERO, new BigDecimal(5),
                new BigDecimal(0.5));
    }

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.HORIZONTAL_SLIDER.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Horizontal Slider' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.HORIZONTAL_SLIDER.getValue());
        getDriver().waitForPageLoaded();
    }
}
```

```
@Override
public String pageTitle() {
    return getActualPageTitle();
}

/**
 * Validates if WebElement representing horizontal slider is visible on the page.
 *
 * @return true if horizontal slider is visible, false otherwise.
 */
public boolean isElementHorizontalSliderVisible() {
    return getDriver().elementHorizontalSlider(selectorHorizontalSlider)
        .isDisplayed();
}

/**
 * Returns the value of slider's start position.
 *
 * @return BigDecimal representing the lowest possible value of slider.
 */
public BigDecimal getStartPosition() {
    return horizontalSlider.getMinRange();
}

/**
 * Returns the value of slider's middle position.
 *
 * @return BigDecimal representing the average value between start and end
position.
 */
public BigDecimal getMiddlePosition() {
    return horizontalSlider.getMaxRange()
        .subtract(horizontalSlider.getMinRange())
        .divide(new BigDecimal(2));
}

/**
 * Returns the value of slider's end position.
 *
 * @return BigDecimal representing the highest possible value of slider.
 */
public BigDecimal getEndPosition() {
    return horizontalSlider.getMaxRange();
}

/**
 * Returns current value of slider's position.
 *
 * @return BigDecimal representing current value of slider.
 */
public BigDecimal getCurrentPosition() {
```

```

        return horizontalSlider.getCurrentSliderValue();
    }

    /**
     * Sets horizontal slider to a given position using one of the available methods:
     * using keyboard
     * or using mouse move.
     *
     * @param position
     * @param method
     */
    public void setSliderPositionTo(BigDecimal position, int method) {
        horizontalSlider.setSliderPositionTo(position, method);
    }

    /**
     * Verifies the correctness of the given position value and rounds it when
     * necessary.
     *
     * @param position
     * @return Correct value of horizontal slider's position.
     */
    public BigDecimal verifyAndCorrectPositionValue(BigDecimal position) {
        return horizontalSlider.verifyAndCorrectPositionValue(position);
    }
}

```

Horizontal Slider Element

This class implements methods which can perform actions on slider:

Create Slider Object using method:

- `getDriver().elementHorizontalSlider(By sliderContainerSelector, By sliderSelector, By valueSelector, BigDecimal minRange, BigDecimal maxRange, BigDecimal step)`

And use:

- `BigDecimal getMaxRange()`
- `BigDecimal getMinRange()`
- `BigDecimal getCurrentSliderValue()`
- `setSliderPositionTo(BigDecimal position, int method)` - moves slider to a given position. If the position is not valid, it changes it to the nearest proper value. Second parameter determines movement method: 0 - Keyboard, 1 - Mouse
- `BigDecimal verifyAndCorrectPositionValue(BigDecimal position)` - returns nearest correct position

Test Class

Before all tests: Open The Internet Main Page

Before each case:

1. Go to Horizontal Slider Page
2. Check if the slider is visible
3. Save start, middle and end position

Case 1 - Moving with the keyboard:

1. Move slider to start position, and check if the current position equals the beginning value
2. Move the slider to middle position, and check if the current position equals the middle value
3. Move slider to end position, and check if the current position equals the end value
4. Try to move slider before start position, and check if the current position equals the beginning value
5. Try to move slider after end position, and check if the current position equals the end value
6. Try to move the slider to an improperly defined position between start and middle, and check if the current position equals the corrected value
7. Try to move the slider to an improperly defined random position, and check if the current position equals the corrected value
8. Move the slider back to start position, and check if the current position equals the beginning value

Case 2 - Moving with a mouse: Repeat each Case 1 step using a mouse instead of keyboard

```
@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class SliderTest extends TheInternetBaseTest {

    private static HorizontalSliderPage horizontalSliderPage;

    BigDecimal startPosition;
    BigDecimal middlePosition;
    BigDecimal endPosition;

    @BeforeClass
    public static void setUpBeforeClass() {
        logStep("Open the Url http://the-internet.herokuapp.com/");
        theInternetPage = new TheInternetPage();
        theInternetPage.load();

        logStep("Verify if Url http://the-internet.herokuapp.com/ is opened");
        assertTrue("Unable to load The Internet Page", theInternetPage.isLoaded());
    }

    @Override
    public void setUp() {
        logStep("Click Horizontal Slider link");
    }
}
```

```
horizontalSliderPage = theInternetPage.clickHorizontalSliderLink();

logStep("Verify if Horizontal Slider page is opened");
assertTrue("Unable to load Horizontal Slider page",
horizontalSliderPage.isLoaded());

logStep("Verify if horizontal slider element is visible");
assertTrue("Horizontal slider is not visible",
    horizontalSliderPage.isElementHorizontalSliderVisible());

startPosition = horizontalSliderPage.getStartPosition();
middlePosition = horizontalSliderPage.getMiddlePosition();
endPosition = horizontalSliderPage.getEndPosition();
}

@Test
public void shouldHorizontalSliderMoveWhenKeyboardArrowButtonsArePressed() {
    BigDecimal position;
    logStep("Move slider to start position: " + startPosition);
    horizontalSliderPage.setSliderPositionTo(startPosition,
HorizontalSliderElement.KEYBOARD);
    assertEquals("Fail to set horizontal sliders position", startPosition,
        horizontalSliderPage.getCurrentPosition());

    logStep("Move slider to middle position: " + middlePosition);
    horizontalSliderPage.setSliderPositionTo(middlePosition,
HorizontalSliderElement.KEYBOARD);
    assertEquals("Fail to set horizontal sliders position",
        horizontalSliderPage.verifyAndCorrectPositionValue(middlePosition),
        horizontalSliderPage.getCurrentPosition());

    logStep("Move slider to end position: " + endPosition);
    horizontalSliderPage.setSliderPositionTo(endPosition,
HorizontalSliderElement.KEYBOARD);
    assertEquals("Fail to set horizontal sliders position", endPosition,
        horizontalSliderPage.getCurrentPosition());

    position = startPosition.subtract(BigDecimal.ONE);
    logStep("Move slider to position before start position: " + position);
    horizontalSliderPage.setSliderPositionTo(position,
HorizontalSliderElement.KEYBOARD);
    assertEquals("Fail to set horizontal sliders position", startPosition,
        horizontalSliderPage.getCurrentPosition());

    position = endPosition.add(BigDecimal.ONE);
    logStep("Move slider to position after end position: " + position);
    horizontalSliderPage.setSliderPositionTo(position,
HorizontalSliderElement.KEYBOARD);
    assertEquals("Fail to set horizontal sliders position", endPosition,
        horizontalSliderPage.getCurrentPosition());
}
```

```

position = middlePosition.divide(new BigDecimal(2));
logStep("Move slider to improperly defined position: " + position);
horizontalSliderPage.setSliderPositionTo(position,
HorizontalSliderElement.KEYBOARD);
assertEquals("Fail to set horizontal sliders position",
            horizontalSliderPage.verifyAndCorrectPositionValue(position),
            horizontalSliderPage.getCurrentPosition());

position = new BigDecimal(new BigInteger("233234"), 5);
logStep("Move slider to improperly defined random position: " + position);
horizontalSliderPage.setSliderPositionTo(position,
HorizontalSliderElement.KEYBOARD);
assertEquals("Fail to set horizontal sliders position",
            horizontalSliderPage.verifyAndCorrectPositionValue(position),
            horizontalSliderPage.getCurrentPosition());

logStep("Move slider back to start position: " + startPosition);
horizontalSliderPage.setSliderPositionTo(startPosition,
HorizontalSliderElement.KEYBOARD);
assertEquals("Fail to set horizontal sliders position", startPosition,
            horizontalSliderPage.getCurrentPosition());
}

@Test
public void shouldHorizontalSliderMoveWhenMouseButtonIsPressedAndMouseIsMoving() {
    BigDecimal position;
    logStep("Move slider to start position: " + startPosition);
    horizontalSliderPage.setSliderPositionTo(startPosition,
HorizontalSliderElement.MOUSE);
    assertEquals("Fail to set horizontal sliders position", startPosition,
            horizontalSliderPage.getCurrentPosition());

    logStep("Move slider to middle position: " + middlePosition);
    horizontalSliderPage.setSliderPositionTo(middlePosition,
HorizontalSliderElement.MOUSE);
    assertEquals("Fail to set horizontal sliders position",
            horizontalSliderPage.verifyAndCorrectPositionValue(middlePosition),
            horizontalSliderPage.getCurrentPosition());

    logStep("Move slider to end position: " + endPosition);
    horizontalSliderPage.setSliderPositionTo(endPosition,
HorizontalSliderElement.MOUSE);
    assertEquals("Fail to set horizontal sliders position", endPosition,
            horizontalSliderPage.getCurrentPosition());

    position = startPosition.subtract(BigDecimal.ONE);
    logStep("Move slider to position before start position: " + position);
    horizontalSliderPage.setSliderPositionTo(position,
HorizontalSliderElement.MOUSE);
    assertEquals("Fail to set horizontal sliders position", startPosition,
            horizontalSliderPage.getCurrentPosition());
}

```

```

        position = endPosition.add(BigDecimal.ONE);
        logStep("Move slider to position after end position: " + position);
        horizontalSliderPage.setSliderPositionTo(position,
HorizontalSliderElement.MOUSE);
        assertEquals("Fail to set horizontal sliders position", endPosition,
            horizontalSliderPage.getCurrentPosition());

        position = middlePosition.divide(new BigDecimal(2));
        logStep("Move slider to improperly defined position: " + position);
        horizontalSliderPage.setSliderPositionTo(position,
HorizontalSliderElement.MOUSE);
        assertEquals("Fail to set horizontal sliders position",
            horizontalSliderPage.verifyAndCorrectPositionValue(position),
            horizontalSliderPage.getCurrentPosition());

        position = new BigDecimal(new BigInteger("212348"), 5);
        logStep("Move slider to improperly defined random position: " + position);
        horizontalSliderPage.setSliderPositionTo(position,
HorizontalSliderElement.MOUSE);
        assertEquals("Fail to set horizontal sliders position",
            horizontalSliderPage.verifyAndCorrectPositionValue(position),
            horizontalSliderPage.getCurrentPosition());

        logStep("Move slider back to start position: " + startPosition);
        horizontalSliderPage.setSliderPositionTo(startPosition,
HorizontalSliderElement.MOUSE);
        assertEquals("Fail to set horizontal sliders position", startPosition,
            horizontalSliderPage.getCurrentPosition());
    }
}

```

[Fork me on GitHub](#)

Data Tables

Often times when you see a table it contains data which is sortable – sometimes with actions that can be taken within each row (e.g. edit, delete). And it can be challenging to automate interaction with sets of data in a table depending on how it is constructed.

Example 1

No Class or ID attributes to signify groupings of rows and columns

Last Name	First Name	Email	Due	Web Site	Action
Smith	John	jsmith@gmail.com	\$50.00	http://www.jsmith.com	edit delete
Bach	Frank	fbach@yahoo.com	\$51.00	http://www.frank.com	edit delete
Doe	Jason	jdoe@hotmail.com	\$100.00	http://www.jdoe.com	edit delete
Conway	Tim	tconway@earthlink.net	\$50.00	http://www.timconway.com	edit delete

Example 2

Class and ID attributes to signify groupings of rows and columns

Last Name	First Name	Email	Due	Web Site	Action
Smith	John	jsmith@gmail.com	\$50.00	http://www.jsmith.com	edit delete
Bach	Frank	fbach@yahoo.com	\$51.00	http://www.frank.com	edit delete
Doe	Jason	jdoe@hotmail.com	\$100.00	http://www.jdoe.com	edit delete
Conway	Tim	tconway@earthlink.net	\$50.00	http://www.timconway.com	edit delete

Powered by [Elemental Selenium](#)

This example shows how to sort and read data from tables.

After clicking on a column header, the data will be sorted descending and after another click sorted ascending by selected attribute. Watch how both tables' content changes on page DOM. Sorting is performed by JavaScript functions.

Last Name	First Name	Email	Due	Web Site	Action
Smith	John	jsmith@gmail.com	\$50.00	http://www.jsmith.com	edit delete
Doe	Jason	jdoe@hotmail.com	\$100.00	http://www.jdoe.com	edit delete
Conway	Tim	tconway@earthlink.net	\$50.00	http://www.timconway.com	edit delete
Bach	Frank	fbach@yahoo.com	\$51.00	http://www.frank.com	edit delete

Last Name	First Name	Email	Due	Web Site	Action
Smith	John	jsmith@gmail.com	\$50.00	http://www.jsmith.com	edit delete
Bach	Frank	fbach@yahoo.com	\$51.00	http://www.frank.com	edit delete
Doe	Jason	jdoe@hotmail.com	\$100.00	http://www.jdoe.com	edit delete
Conway	Tim	tconway@earthlink.net	\$50.00	http://www.timconway.com	edit delete

Powered by [Elemental Selenium](#)

Page Class

```
public class SortableDataTablesPage extends BasePage {

    private static final By selectorTable = By.cssSelector("table.tablesorter");
    private static final By selectorHeader = By.cssSelector("th");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.SORTABLE_DATA_TABLES.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Data Tables' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.SORTABLE_DATA_TABLES.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Sorts data in given column using ascending order.
    }
```

```

/*
 * @param columnNumber The number of column where data should be sorted
 * @param tableNumber The number of table where data should be sorted
 */
public void sortColumnAscending(int columnNumber, int tableNumber) {
    WebElement header = this.getTableHeaders(columnNumber, tableNumber);
    String className = header.getAttribute("class");
    if (className.contains("headerSortUp") || !className.contains("headerSortDown")) {
        header.click();
    }
}

/**
 * Sorts data in given column using descending order.
 *
 * @param columnNumber The number of the column where data should be sorted
 * @param tableNumber The number of the table where data should be sorted
 */
public void sortColumnDescending(int columnNumber, int tableNumber) {
    WebElement header = this.getTableHeaders(columnNumber, tableNumber);
    String className = header.getAttribute("class");
    if (!className.contains("headerSortUp")) {
        header.click();
        if (!className.contains("headerSortDown")) {
            header.click();
        }
    }
}

/**
 * Return given column values from chosen table.
 *
 * @param columnNumber The number of the column the data should be retrieved from
 * @param tableNumber The number of the table the data should be retrieved from
 * @return list of values from given column
 */
public List<String> getColumnValues(int columnNumber, int tableNumber) {
    WebElement table = getTable(tableNumber);
    return JsoupHelper.findTexts(table, By.cssSelector("tr > td:nth-child(" +
(columnNumber + 1)
        + ")"));
}

/**
 * Returns column's class name.
 *
 * @param columnNumber The number of the column to get class number from
 * @param tableNumber The number of the table to get column class name from
 * @return String object representing column's class name
 */

```

```

public String readColumnClass(int columnNumber, int tableNumber) {
    return this.getTableHeaders(columnNumber, tableNumber)
        .getAttribute("class");
}

private WebElement getTable(int tableNumber) {
    return new ListElements(selectorTable).getList()
        .get(tableNumber);
}

private WebElement getTableHeaders(int columnNumber, int tableNumber) {
    return getTable(tableNumber).findElements(selectorHeader)
        .get(columnNumber);
}
}

```

Finding values

Using proper selectors, save elements such as tables and their columns' headers as Web Element Lists. Afterwards, you can get the desired element finding it by index (e. g. table or column number). To get column values, use [JsoupHelper](#) and to check if the column is sorted get its class attribute.

Test Class

Before all tests: Open The Internet Main Page

Before each case: Go to Sortable Data Tables Page

Case 1:

1. Choose a random table
2. Sort first column "Last Name" in ascending order
3. Check if column header class contains "headerSortDown"
4. Save column content to the List
5. Create List copy and sort it
6. Compare sorted values and values from the table

Case 2:

1. Choose a random table
2. Sort second column "First Name" in descending order
3. Check if column header class contains "headerSortUp"
4. Save column content to the List
5. Create List copy and sort it then reverse it
6. Compare reversed sorted values and values from the table

```
@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class SortableDataTablesTest extends TheInternetBaseTest {

    private static SortableDataTablesPage sortableDataTablesPage;

    private List<String> actualValues;
    private List<String> expectedValues;

    @BeforeClass
    public static void setUpBeforeClass() {
        logStep("Open the Url http://the-internet.herokuapp.com/");
        theInternetPage = new TheInternetPage();
        theInternetPage.load();

        logStep("Verify if Url http://the-internet.herokuapp.com/ is opened");
        assertTrue("Unable to load The Internet Page", theInternetPage.isLoaded());
    }

    @Override
    public void setUp() {
        logStep("Click subpage link");
        sortableDataTablesPage = theInternetPage.clickSortableDataTablesLink();

        logStep("Verify if subpage is opened");
        assertTrue("Unable to open Sortable Data Tables page",
        sortableDataTablesPage.isLoaded());
    }

    @Test
    public void shouldLastNameColumnBeOrderedAscendingAfterSort() {
        int columnNumber = 0;
        int tableNumber = new Random().nextInt(2);

        logStep("Sort 'Last Name' column");
        sortableDataTablesPage.sortColumnAscending(columnNumber, tableNumber);
        assertTrue("Unable to set ascending order for 'Last Name' column",
            sortableDataTablesPage.readColumnClass(columnNumber, tableNumber)
                .contains("headerSortDown"));

        logStep("Verify data order for 'Last Name' column");
        actualValues = sortableDataTablesPage.getColumnValues(columnNumber,
        tableNumber);
        expectedValues = new ArrayList<String>(actualValues);
        Collections.sort(expectedValues);
        assertEquals("'Last Name' column is not sorted in ascending order",
            expectedValues, actualValues);
    }

    @Test
```

```

public void shouldFirstNameColumnBeOrderedDescendingAfterSort() {
    int columnNumber = 1;
    int tableNumber = new Random().nextInt(2);

    logStep("Sort 'First Name' column");
    sortableDataTablesPage.sortColumnDescending(columnNumber, tableNumber);
    assertTrue("Unable to set descending order for 'First Name' column",
               sortableDataTablesPage.readColumnClass(columnNumber, tableNumber)
               .contains("headerSortUp"));

    logStep("Verify data order for 'First Name' column");
    actualValues = sortableDataTablesPage.getColumnValues(columnNumber,
    tableNumber);
    expectedValues = new ArrayList<String>(actualValues);
    Collections.sort(expectedValues);
    Collections.reverse(expectedValues);
    assertEquals("'First Name' column is not sorted in descending order",
               expectedValues, actualValues);
}
}

```

[Fork me on GitHub](#)

Status Codes

HTTP status codes are a standard set of numbers used to communicate from a web server to your browser to indicate the outcome of the request being made (e.g. Success, Redirection, Client Error, Server Error). For a complete list of status codes, go [here](#).

Some standard status codes you will run into include but are not limited to:

- [200](#)
- [301](#)
- [404](#)
- [500](#)

Powered by [Elemental Selenium](#)

This example shows how to process HTTP status codes returned by page

When you click status code link, you will be redirected to the subpage which returns the proper HTTP status code. In order to check what code was returned:

1. Open developer tools
2. Go to Network tab
3. Click request name
4. Find a code number in Headers section

The screenshot shows the Network tab of a browser's developer tools. A single request is listed for the URL `http://the-internet.herokuapp.com/status_codes/500`. The request took 200ms and the response took 800ms. The response headers include the status code `500 Internal Server Error`. The response body contains the text "Internal Server Error" and some log entries from the server.

Page Class

Add new methods to existing Status Codes Home Page Class

```
public class StatusCodesHomePage extends BasePage {

    private static final By selectorLink200Code = By.linkText("200");
    private static final By selectorLink301Code = By.linkText("301");
    private static final By selectorLink404Code = By.linkText("404");
    private static final By selectorLink500Code = By.linkText("500");

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.STATUS_CODES.getValue());
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Status Codes' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.STATUS_CODES.getValue());
        getDriver().waitForPageLoaded();
    }

    @Override
    public String pageTitle() {
        return getActualPageTitle();
    }

    /**
     * Verifies if given link is displayed.
    }
```

```
/*
 * @param selector Selector of the given link
 * @return true if link is displayed
 */
public boolean isLinkCodeDisplayed(By selector) {
    return getDriver().findElementDynamic(selector)
        .isDisplayed();

}

/***
 * Clicks '200' link.
 *
 * @return StatusCodesCodePage object
 */
public StatusCodesCodePage clickCode200Link() {
    return clickCodeLink(selectorLink200Code);
}

/***
 * Clicks '301' link.
 *
 * @return StatusCodesCodePage object
 */
public StatusCodesCodePage clickCode301Link() {
    return clickCodeLink(selectorLink301Code);
}

/***
 * Clicks '404' link.
 *
 * @return StatusCodesCodePage object
 */
public StatusCodesCodePage clickCode404Link() {
    return clickCodeLink(selectorLink404Code);
}

/***
 * Clicks '500' link.
 *
 * @return StatusCodesCodePage object
 */
public StatusCodesCodePage clickCode500Link() {
    return clickCodeLink(selectorLink500Code);
}

/***
 * Clicks code link according to given code number.
 *
 * @param code Given code
 * @return StatusCodesCodePage object
 */
```

```

*/
public StatusCodesCodePage clickCodeLink(String code) {
    return clickCodeLink(By.linkText(code));
}

private StatusCodesCodePage clickCodeLink(By selector) {
    String codeNumber = getCodeNumberToCheck(selector);
    getDriver().findElementDynamic(selector)
        .click();
    return new StatusCodesCodePage(codeNumber);
}

private String getCodeNumberToCheck(By selector) {
    return getDriver().findElementDynamic(selector)
        .getText();
}
}

```

Create a page class for status codes subpages as well. In the class constructor specify which code number should be returned.

```

public class StatusCodesCodePage extends BasePage {

    private static final By selectorDisplayedText = By.cssSelector("#content > div > p");
    private static final By selectorLinkToCodesPage = By.cssSelector("#content > div > p > a");

    private String codeNumber;

    public StatusCodesCodePage(String codeNumber) {
        this.codeNumber = codeNumber;
    }

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded();
        return getDriver().getCurrentUrl()
            .contains(PageSubURLsProjectYEnum.STATUS_CODES.getValue() + '/');
    }

    @Override
    public void load() {
        BFLogger.logDebug("Load 'Status Codes' page.");
        getDriver().get(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
            PageSubURLsProjectYEnum.STATUS_CODES.getValue() + '/' + codeNumber);
        getDriver().waitForPageLoaded();
    }

    @Override

```

```

public String pageTitle() {
    return getActualPageTitle();
}

public String getCodeNumber() {
    return codeNumber;
}

/**
 * Verifies if page is loaded with given code number.
 *
 * @param codeNumber Expected code number
 * @return true if expected code number is loaded with web page
 */
public boolean isLoadedWithStatusCode(String codeNumber) {
    return getDriver().getCurrentUrl()
        .equals(GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue() +
        PageSubURLsProjectYEnum.STATUS_CODES.getValue() + "/" + codeNumber);
}

/**
 * Returns displayed code number.
 * <p>
 * Code number is retrieved from following text displayed on the page:<b>
 * 'This page returned a *** status code.', where *** represent the code number to
be
 * returned.
 * </p>
 *
 * @return String object representing the displayed code number retrieved from
specific sentence.
 */
public String getDisplayedCodeNumber() {
    return getDriver().findElementDynamic(selectorDisplayedText)
        .getText()
        .substring(21, 24);
}

/**
 * Clicks link to return to 'Code Page'.
 *
 * @return StatusCodesHomePage object
 */
public StatusCodesHomePage clickLinkToCodePage() {
    getDriver().findElementDynamic(selectorLinkToCodesPage)
        .click();
    return new StatusCodesHomePage();
}
}

```

Test Class

Before all tests: Open The Internet Main Page, go to Status Codes page

Steps:

For each status code

1. Click code link
2. Check if the page is loaded with an expected code number
3. Check if the displayed code number equals the expected number
4. Go back to Status Codes Home Page

```

@Category({ TestsSelenium.class, TestsChrome.class, TestsFirefox.class, TestsIE.class })
public class StatusCodeTest extends TheInternetBaseTest {

    private static StatusCodesHomePage statusCodesHomePage;
    private StatusCodesCodePage statusCodesCodePage;

    private String[] codes = { "200", "301", "404", "500" };

    @BeforeClass
    public static void setUpBeforeClass() {
        statusCodesHomePage = shouldTheInternetPageBeOpened().clickStatusCodesLink();

        logStep("Verify if Status Codes Home page is opened");
        assertTrue("Unable to open Status Codes Home page",
        statusCodesHomePage.isLoaded());
    }

    @Test
    public void shouldProperCodeBeDisplayedAfterClickCodeLink() {

        for (String code : codes) {
            logStep("Click link to " + code + " code");
            statusCodesCodePage = statusCodesHomePage.clickCodeLink(code);

            logStep("Verify if proper web page corresponding to the code is opened");
            assertTrue("Unable to open proper web page",
            statusCodesCodePage.isLoadedWithStatusCode(code));

            logStep("Verify if the displayed code is equal to the expected one");
            assertEquals(code, statusCodesCodePage.getDisplayedCodeNumber());

            logStep("Click link to come back to 'Status Codes' page");
            statusCodesCodePage.clickLinkToCodePage();
        }
    }
}

```

78.2.2. First Steps

Unresolved directive in mrchecker.wiki/master-mrchecker.asciidoc - include::Who-Is-MrChecker/Tutorials/Basic-Tutorials/First-Steps/Page-objects.asciidoc[leveloffset=4]

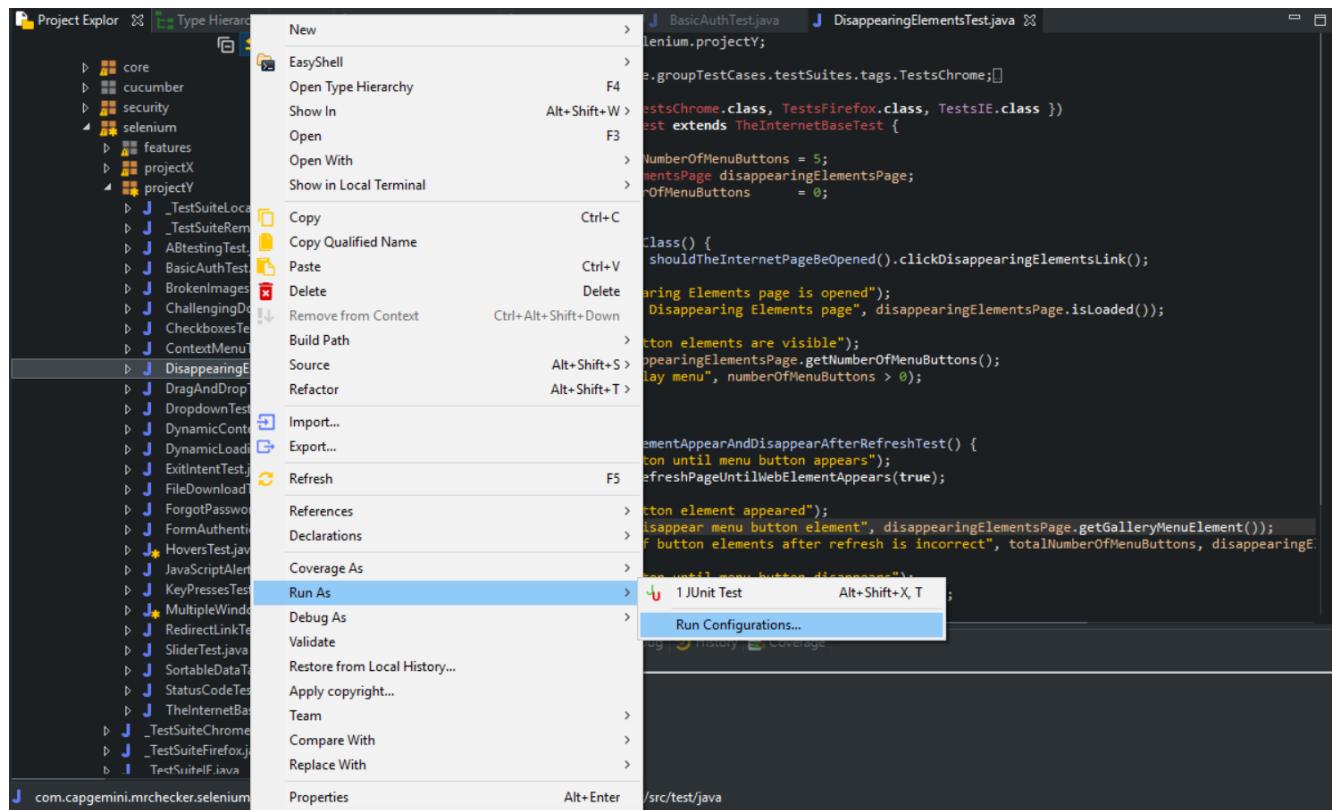
Environment Variables

In Page classes, when you load/start web, it is uncommon to save fixed main URL.

Instead of *hardcoded* main URL variable, you build your Page class with a dynamic variable.

How to create / update system environment

Dynamic variable values are stored under path `|src|resources|environments|environments.csv`.



By default, the environment takes value from DEV column.

Access to the external file variables

Create a class `GetEnvironmentParam` to map values from an external file with Page class:

```

public enum GetEnvironmentParam {

    // Name if enum must be in line with cell name in
    // /src/resources/environments/environment.csv
    WWW_FONT_URL,
    TOOLS_QA,
    WEB_SERVICE,
    THE_INTERNET_MAIN_PAGE,
    ELEMENTAL_SELENIUM_PAGE;

    public String getValue() {

        if (null == BaseTest.getEnvironmentService()) {
            throw new BFInputDataException("Environment Parameters class wasn't
initialized properly");
        }

        return BaseTest.getEnvironmentService()
            .getValue(this.name());
    }

    @Override
    public String toString() {

        return this.getValue();
    }
}

```

When you add a new row to *environments.csv* also add a new variable to *GetEnvironmentParam* class.

In Page class access environmental variable using this method:

```
GetEnvironmentParam.THE_INTERNET_MAIN_PAGE.getValue();
```

Selectors

Create selectors

Create a selector for every interactable element on a webpage using By type. Find elements and it's attributes using browser developer mode (F12).



```

private static final By abTestLinkSelector          = By.cssSelector("li >
    a[href*='abtest']");
private static final By basicAuthLinkSelector       = By.cssSelector("li >
    a[href*='basic_auth']");
private static final By brokenImageLinkSelector     = By.cssSelector("li >
    a[href*='broken_images']");
private static final By challengingDomLinkSelector = By.cssSelector("li >
    a[href*='challenging_dom']");
private static final By checkboxesLinkSelector      = By.cssSelector("li >
    a[href*='checkboxes']");
private static final By contextMenuLinkSelector     = By.cssSelector("li >
    a[href*='context_menu']");
private static final By disappearingElementsLinkSelector = By.cssSelector("li >
    a[href*='disappearing_elements']");
private static final By dragAndDropLinkSelector     = By.cssSelector("li >
    a[href*='drag_and_drop']");
private static final By dropdownLinkSelector        = By.cssSelector("li >
    a[href*='dropdown']");
private static final By dynamicContentLinkSelector = By.cssSelector("li >
    a[href*='dynamic_content']");
private static final By dynamicControlsLinkSelector= By.cssSelector("li >
    a[href*='dynamic_controls']");
private static final By dynamicLoadingLinkSelector  = By.cssSelector("li >
    a[href*='dynamic_loading']");
private static final By exitIntentLinkSelector       = By.cssSelector("li >
    a[href*='exit_intent']");
private static final By fileDownloadLinkSelector    = By.cssSelector("li >
    a[href$='download']");
private static final By fileUploadLinkSelector      = By.cssSelector("li >
    a[href*='upload']");
private static final By floatingMenuLinkSelector   = By.cssSelector("li >
    a[href*='floating_menu']");
private static final By forgotPasswordLinkSelector = By.cssSelector("li >
    a[href*='forgot_password']");
private static final By formAuthenticationLinkSelector= By.cssSelector("li >
    a[href*='login']");
private static final By framesLinkSelector          = By.cssSelector("li >
    a[href*='frames']");
private static final By geolocationLinkSelector    = By.cssSelector("li >
    a[href*='geolocation']");
private static final By horizontalSliderLinkSelector= By.cssSelector("li >
    a[href*='horizontal_slider']");
  
```

```

private static final By hoversLinkSelector          = By.cssSelector("li >
    a[href*='hovers']");
private static final By infiniteScrollLinkSelector = By.cssSelector("li >
    a[href*='infinite_scroll']");
private static final By javaScriptAlertLinkSelector = By.cssSelector("li >
    a[href*='javascript_alerts']");
private static final By javaScriptErrorLinkSelector = By.cssSelector("li >
    a[href*='javascript_error']");
private static final By jQueryUIMenuLinkSelector    = By.cssSelector("li >
    a[href*='jqueryui/menu']");
private static final By keyPressesLinkSelector      = By.cssSelector("li >
    a[href*='key_presses']");
private static final By largeAndDeepDOMLinkSelector = By.cssSelector("li >
    a[href*='large']");
private static final By multipleWindowsLinkSelector = By.cssSelector("li >
    a[href*='windows']");
private static final By nestedFramesLinkSelector    = By.cssSelector("li >
    a[href*='nested_frames']");
private static final By notificationMessagesLinkSelector = By.cssSelector("li >
    a[href*='notification_message']");
private static final By redirectLinkSelector         = By.cssSelector("li >
    a[href*='redirection']");
private static final By secureFileDownloadLinkSelector = By.cssSelector("li >
    a[href*='download_secure']");
private static final By shiftingContentLinkSelector = By.cssSelector("li >
    a[href*='shifting_content']");
private static final By slowResourcesLinkSelector   = By.cssSelector("li >
    a[href*='slow']");
private static final By sortableDataTablesLinkSelector = By.cssSelector("li >
    a[href*='tables']");
private static final By statusCodesLinkSelector     = By.cssSelector("li >
    a[href*='status_codes']");
private static final By typosLinkSelector           = By.cssSelector("li >
    a[href*='typos']");
private static final By wYSIWYGEditorLinkSelector  = By.cssSelector("li >
    a[href*='tinymce']");

```

Implement methods

Then use these selectors to create Objects and perform actions on page elements:

```

public ABtestPage clickABtestingLink() {
    new Button(abTestLinkSelector).click();
    return new ABtestPage();
}

public BasicAuthPage clickBasicAuthLink() {
    getDriver().waitForPageLoaded();
    WebElement link = getDriver().findElementDynamic(basicAuthLinkSelector);
    JavascriptExecutor executor = (JavascriptExecutor) getDriver();
}

```

```
executor.executeScript("var elem=arguments[0]; setTimeout(function()\n{elem.click();}, 100)",\n    link);\n    return new BasicAuthPage();\n}\n\npublic BrokenImagePage clickBrokenImageLink() {\n    new Button(brokenImageLinkSelector).click();\n    return new BrokenImagePage();\n}\n\npublic ChallengingDomPage clickChallengingDomLink() {\n    new Button(challengingDomLinkSelector).click();\n    return new ChallengingDomPage();\n}\n\npublic CheckboxesPage clickCheckboxesLink() {\n    new Button(checkboxesLinkSelector).click();\n    return new CheckboxesPage();\n}\n\npublic ContextMenuPage clickContextMenuLink() {\n    new Button(contextMenuLinkSelector).click();\n    return new ContextMenuPage();\n}\n\npublic DisappearingElementsPage clickDisappearingElementsLink() {\n    new Button(disappearingElementsLinkSelector).click();\n    return new DisappearingElementsPage();\n}\n\npublic DragAndDropPage clickDragAndDropLink() {\n    new Button(dragAndDropLinkSelector).click();\n    return new DragAndDropPage();\n}\n\npublic DropdownPage clickDropdownLink() {\n    new Button(dropDownListSelector).click();\n    return new DropdownPage();\n}\n\npublic DynamicContentPage clickDynamicContentLink() {\n    new Button(dynamicContentLinkSelector).click();\n    return new DynamicContentPage();\n}\n\npublic DynamicControlsPage clickDynamicControlsLink() {\n    new Button(dynamicControlsLinkSelector).click();\n    return new DynamicControlsPage();\n}
```

```
public DynamicLoadingPage clickDynamicLoadingLink() {
    new Button(dynamicLoadingLinkSelector).click();
    return new DynamicLoadingPage();
}

public ExitIntentPage clickExitIntentLink() {
    new Button(exitIntentLinkSelector).click();
    return new ExitIntentPage();
}

public FileDownloadPage clickFileDownloadLink() {
    new Button(fileDownloadLinkSelector).click();
    return new FileDownloadPage();
}

public FileUploadPage clickFileUploadLink() {
    new Button(fileUploadLinkSelector).click();
    return new FileUploadPage();
}

public FloatingMenuPage clickFloatingMenuLink() {
    new Button(floatingMenuLinkSelector).click();
    return new FloatingMenuPage();
}

public ForgotPasswordPage clickForgotPasswordLink() {
    new Button(forgotPasswordLinkSelector).click();
    return new ForgotPasswordPage();
}

public FormAuthenticationPage clickFormAuthenticationLink() {
    new Button(formAuthenticationLinkSelector).click();
    return new FormAuthenticationPage();
}

public FramesPage clickFramesLink() {
    new Button(framesLinkSelector).click();
    return new FramesPage();
}

public GeolocationPage clickGeolocationLink() {
    new Button(geolocationLinkSelector).click();
    return new GeolocationPage();
}

public HorizontalSliderPage clickHorizontalSliderLink() {
    new Button(horizontalSliderLinkSelector).click();
    return new HorizontalSliderPage();
}

public HoversPage clickHoversLink() {
```

```
        new Button(hoversLinkSelector).click();
        return new HoversPage();
    }

    public InfiniteScrollPage clickInfiniteScrollLink() {
        new Button(infiniteScrollLinkSelector).click();
        return new InfiniteScrollPage();
    }

    public JavaScriptAlertsPage clickJavaScriptAlertLink() {
        new Button(javaScriptAlertLinkSelector).click();
        return new JavaScriptAlertsPage();
    }

    public JavaScriptErrorPage clickJavaScriptErrorLink() {
        new Button(javaScriptErrorLinkSelector).click();
        return new JavaScriptErrorPage();
    }

    public JQueryUIMenuPage clickJQueryUIMenuLink() {
        new Button(jQueryUIMenuLinkSelector).click();
        return new JQueryUIMenuPage();
    }

    public KeyPressesPage clickKeyPressesLink() {
        new Button(keyPressesLinkSelector).click();
        return new KeyPressesPage();
    }

    public LargeAndDeepDOMPage clickLargeAndDeepDOMLink() {
        new Button(largeAndDeepDOMLinkSelector).click();
        return new LargeAndDeepDOMPage();
    }

    public MultipleWindowsPage clickmultipleWindowsLink() {
        new Button(multipleWindowsLinkSelector).click();
        return new MultipleWindowsPage();
    }

    public NestedFramesPage clickNestedFramesLink() {
        new Button(nestedFramesLinkSelector).click();
        return new NestedFramesPage();
    }

    public NotificationMessagesPage clickNotificationMessagesLink() {
        new Button(notificationMessagesLinkSelector).click();
        return new NotificationMessagesPage();
    }

    public RedirectLinkPage clickRedirectLink() {
        new Button(redirectLinkSelector).click();
```

```

        return new RedirectLinkPage();
    }

    public SecureFileDownloadPage clickSecureFileDownloadLink() {
        new Button(secureFileDownloadLinkSelector).click();
        return new SecureFileDownloadPage();
    }

    public ShiftingContentPage clickShiftingContentLink() {
        new Button(shiftingContentLinkSelector).click();
        return new ShiftingContentPage();
    }

    public SlowResourcesPage clickSlowResourcesLink() {
        new Button(slowResourcesLinkSelector).click();
        return new SlowResourcesPage();
    }

    public SortableDataTablesPage clickSortableDataTablesLink() {
        new Button(sortableDataTablesLinkSelector).click();
        return new SortableDataTablesPage();
    }

    public StatusCodesHomePage clickStatusCodesLink() {
        new Button(statusCodesLinkSelector).click();
        return new StatusCodesHomePage();
    }

    public TyposPage clickTyposLink() {
        new Button(typosLinkSelector).click();
        return new TyposPage();
    }

    public WYSIWYGEditorPage clickWYSIWYGEditorLink() {
        new Button(wYSIWYGEditorLinkSelector).click();
        return new WYSIWYGEditorPage();
    }
}

```

These methods create a Button object for every link on The Internet Page and click it to redirect on a different subpage.

Elements types

MrChecker includes Object types for various elements existing on webpages such as Button, TextBox etc. There is also WebElement class and `getDriver().findElementDynamic(By selector)` method for creating webpage objects dynamically and performing basic actions:

Instead of using static types you can use:

```
public TyposPage clickTyposLink() {
    WebElement checkboxesLink =
getDriver().findElementDynamic(checkboxesLinkSelector);
    checkboxesLink.click();
    return new TyposPage();
}
```

Or perform actions without creating a variable:

```
public TyposPage clickTyposLink() {
    getDriver().findElementDynamic(checkboxesLinkSelector).click();
    return new TyposPage();
}
```

The Internet Base Test

Test Class

Create Test class and override methods:

- `public void setUp()` - executes before each test
- `public void tearDown()` - executes after each test

```
public class TheInternetBaseTest extends BaseTest {
    @Override
    public void setUp() {

    }

    @Override
    public void tearDown() {
        logStep("Navigate back to The-Internet page");
        BasePage.navigateBack();
    }
}
```

`logStep(String message)` method doesn't exist yet so you should create it:

```

protected static int step = 0;

/**
 * Logs test step including step number calculated individually for each test.
 *
 * @param message Text message representing step description.
 */
public static void logStep(String message) {
    BFLogger.logInfo("Step " + ++step + ": " + message);
}

```

Write a method for loading The Internet Page and checking if it is properly opened:

```

protected static TheInternetPage theInternetPage;

/**
 * Performs operations required for verifying if The Internet Page is properly
opened.
 *
 * @return TheInternetPage
 */
public static TheInternetPage shouldTheInternetPageBeOpened() {

    logStep("Open the Url http://the-internet.herokuapp.com/");
    theInternetPage = new TheInternetPage();
    theInternetPage.load();

    logStep("Verify if Url http://the-internet.herokuapp.com/ is opened");
    assertTrue("Unable to load The Internet Page", theInternetPage.isLoaded());

    return theInternetPage;
}

```

This Test class can't be launched because it doesn't contain any @Test methods. It's been created only for supporting other Test classes.

BFLogger

BFLogger is a default MrChecker logging tool. Use it to communicate important information from test execution. There are three basic logging methods:

- **logInfo(String message)** - used for test steps
- **logDebug(String message)** - used for non-official information, either during the test build process or in Page Object files
- **logError(String message)** - used to emphasize critical information

Logs will be visible in the console and in the log file under path:
MrChecker_Test_Framework\workspace\project-folder\logs

78.3. E2E Tutorials

78.3.1. MrChecker E2E tutorials

In order to learn more about MrChecker structure, start from Project Organisation section and then check out our fantastic tutorials:

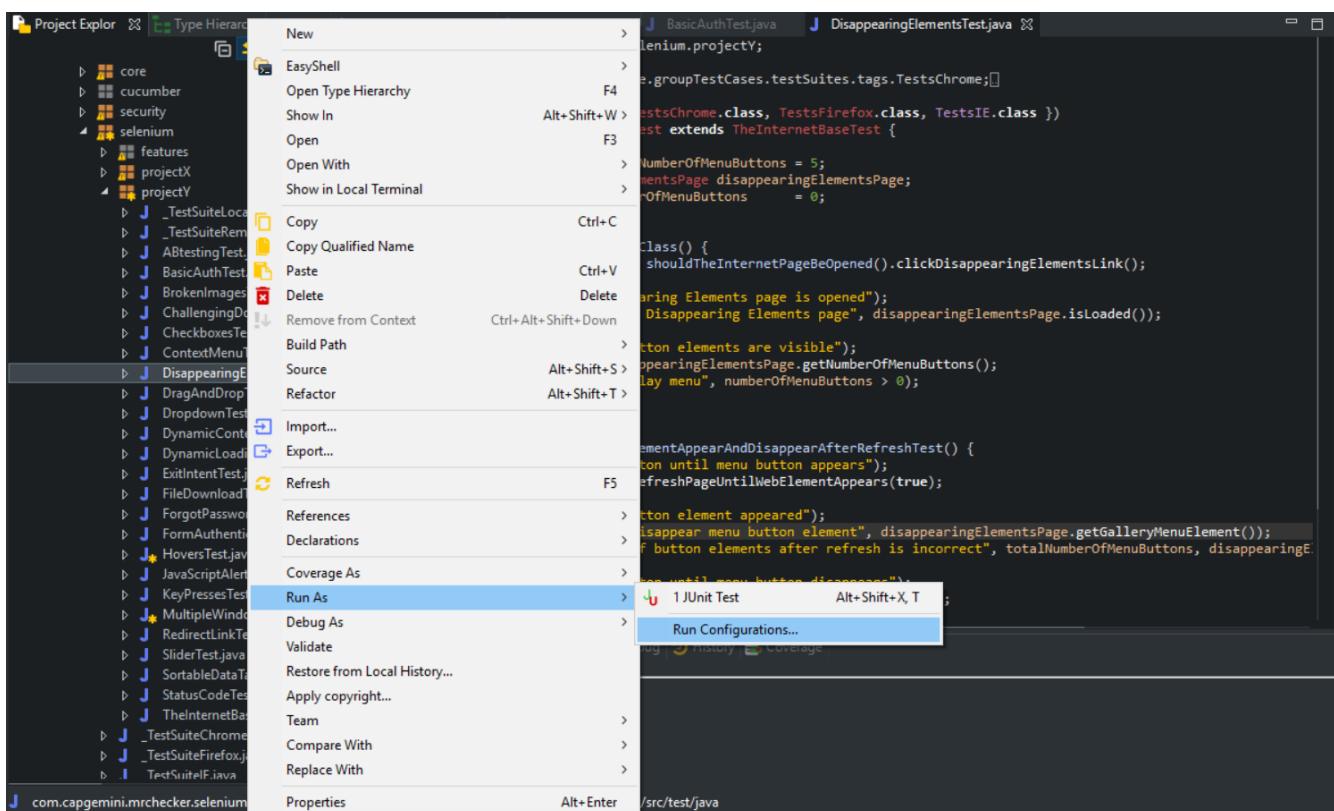
How to create a basic test in MrChecker

Example: Booking a table

As an example to test we will use [MyThaiStar](#) booking page.

In order to book a table, do the following steps:

1. Open MyThaiStar Book Table Page
2. Enter booking data: Date and time, Name, Email and number of Table guests
3. Click Accept terms
4. Click Book table
5. Display confirmation box and send booking
6. Check if the booking was successful.





You can go through these steps manually and doublecheck the result.

How to prepare a test

1. Create BookTablePage class

You will need a class which will represent MyThaiStart booking page.

Fill the required methods with the following code:

```
public class BookTablePage extends BasePage {

    @Override
    public boolean isLoaded() {
        getDriver().waitForPageLoaded(); //waits until the page is loaded
        return getDriver().getCurrentUrl()
            .equals("http://de-mucdevondepl01:8090/bookTable"); //checks if
        current page address equals MyThaiStar booking page address
    }

    @Override
    public void load() {
        getDriver().get("http://de-mucdevondepl01:8090/bookTable"); //loads page under
        specified address
        getDriver().waitForPageLoaded(); //waits until the page is loaded
    }

    @Override
    public String pageTitle() {
        return "My Thai Star"; //returns page title
    }
}
```

`getDriver()` method allows accessing Selenium Web Driver which performs actions on the webpage.

As this page class represents the MyThaiStar booking page, you have to set up selectors for web elements required in the test case. In the example you have to create selectors for elements we'll interact with:

- Date and time input field
- Name input field

- Email input field
- Table guests input field
- Accept terms checkbox
- Book table button

Selectors will be implemented as fields.

Example of the selector for Date and time input field:

```
/** Date field search criteria */
private static final By dateSearch =
By.cssSelector("input[formcontrolname='bookingDate']);
```

The input field's name "bookingDate" was found by using the developer console in Google Chrome.
[How to prepare an everlasting selector?](#)



This selector can be used to create a WebElement object of the said input field. Therefore, you will create a new method and call it "enterTimeAndDate".

```
public void enterTimeAndDate(String date) {
    WebElement dateInput = getDriver().findElementDynamic(dateSearch); //creates a new
    WebElement to access Date and time input field
    dateInput.sendKeys(date); //enters date value
}
```

Now you can create other selectors and objects and methods for every element on the webpage:

```
/** Name input field search criteria */
private static final By nameSearch = By.cssSelector("input[formcontrolname='name']");

/** Email input field search criteria */
private static final By emailSearch =
By.cssSelector("input[formcontrolname='email']");

/** Number of guests search criteria */
private static final By guestsSearch =
By.cssSelector("input[formcontrolname='assistants']");

/** Check box search criteria */
private static final By checkboxSearch = By.cssSelector("mat-checkbox[data-
name='bookTableTerms']");

/** Book table button search criteria */
private static final By bookTableSearch = By.name("bookTableSubmit");
```

```
public void enterName(String name) {
    WebElement nameInput = getDriver().findElementDynamic(nameSearch); //creates a new
WebElement to access name input field
    nameInput.sendKeys(name); //enters name value
}

public void enterEmail(String email) {
    WebElement emailInput = getDriver().findElementDynamic(emailSearch); //creates a
new WebElement to access email input field
    emailInput.sendKeys(email); //enters email value
}

public void enterGuests(int amountOfGuests) {
    WebElement guestsInput = getDriver().findElementDynamic(guestsSearch); //creates a
new WebElement to access amount of guests input field
    guestsInput.sendKeys(Integer.toString(amountOfGuests)); //enters the number of
guests value converted from integer to string
}

public void acceptTerms() {
    WebElement checkbox = getDriver().findElementDynamic(checkboxSearch); //creates aa
new WebElement to access accept terms checkbox
    WebElement square = checkbox.findElement(By.className("mat-checkbox-inner-
container")); //creates a new WebElement to access inner square
    JavascriptExecutor js = (JavascriptExecutor) getDriver(); //creates a Javascript
executor object
    js.executeScript("arguments[0].click()", square); //executes a script which clicks
the square
}

public void clickBookTable() {
    WebElement buttonbutton = getDriver().findElementDynamic(bookTableSearch);
//creates a new WebElement to access book table button
    getDriver().waitUntilElementIsClickable(bookTableSearch); //waits until a button
might be clicked
    buttonbutton.click(); //clicks the button
}
```

You can use those methods in order to create a new method to go through the whole booking process:

```
public ConfirmBookPage enterBookingData(String date, String name, String email, int  
guests) {  
    enterTimeAndDate(date);  
    enterName(name);  
    enterEmail(email);  
    enterGuests(guests);  
    acceptTerms();  
  
    clickBookTable();  
  
    return new ConfirmBookPage();  
}
```

2. Create ConfirmBookPage class

As you can see, this method returns another page object that has not yet been created. This step is required, as the booking information that you would like to check is on another webpage. This means that you will have to create another page class and call it ConfirmBookPage:

```
public class ConfirmBookPage extends BasePage {

    /** Confirmation dialog search criteria */
    private static final By confirmationDialogSearch = By.className("mat-dialog-container");

    /** Send confirmation button search criteria */
    private static final By sendButtonSearch = By.name("bookTableConfirm");

    /** Cancel confirmation button search criteria */
    private static final By cancelButtonSearch = By.name("bookTableCancel");

    @Override
    public boolean isLoaded() {
        //creates a new WebElement to access confirmation dialog box
        WebElement confirmationDialog =
getDriver().findElementDynamic(confirmationDialogSearch);

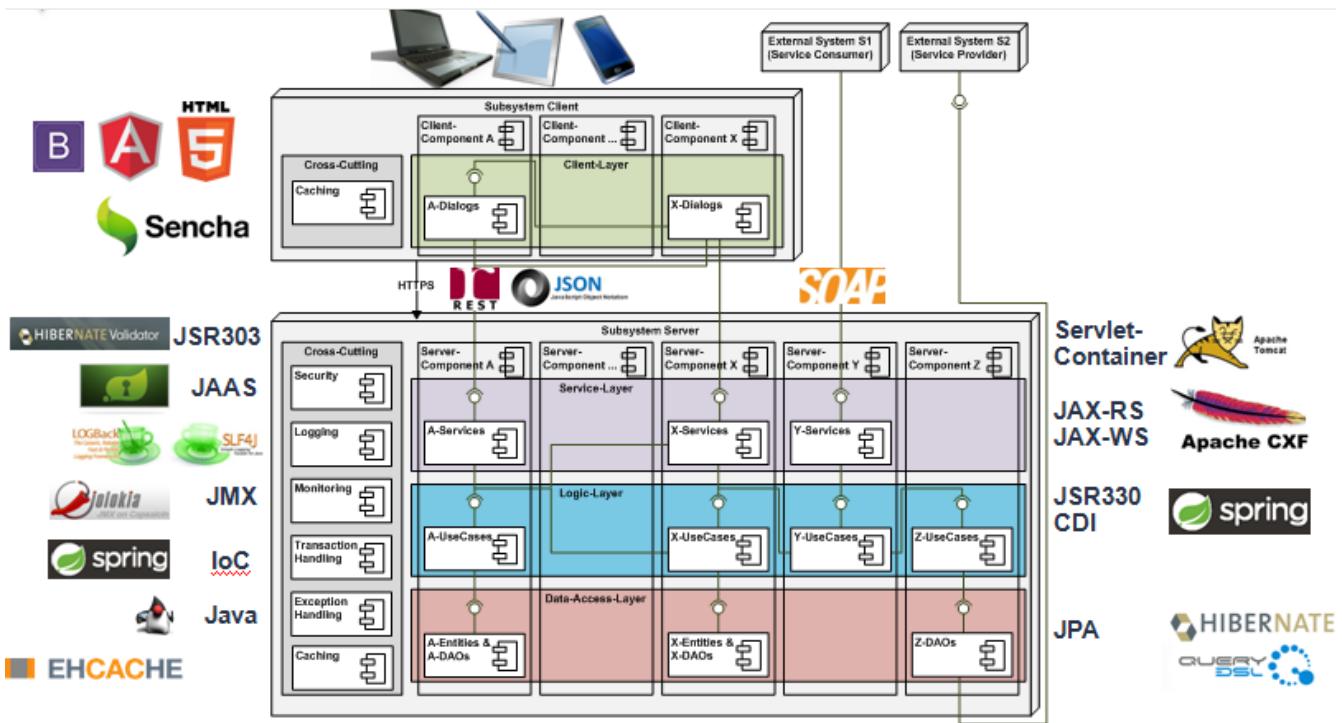
        return confirmationDialog.isDisplayed(); //checks if the box is displayed
    }

    //this method won't be called because the page is loaded only after clicking book
    table button
    @Override
    public void load() {
        BFLogger.LogError("MyThaiStar booking confirmation page was not loaded.");
//logs error
    }

    @Override
    public String pageTitle() {
        return "My Thai Star";
    }

    public void confirmBookingData() {
        WebElement sendButton = getDriver().findElementDynamic(sendButtonSearch);
//creates a new WebElement to access confirmation button
        sendButton.click(); //clicks the send button
    }

    public void cancelBookingData() {
        WebElement cancelButton = getDriver().findElementDynamic(cancelButtonSearch);
//creates a new WebElement to access resignation button
        cancelButton.click(); //clicks the cancel button
    }
}
```

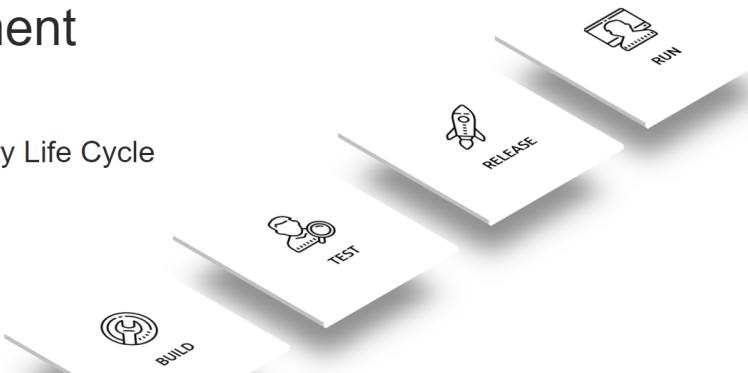


After the click on Send button - the green confirmation dialogue appears with the message "Table successfully booked":



The standard open source software development platform

Serving the full Software Delivery Life Cycle



To be able to check if the booking was successful, you should go back to the `BookTablePage` class and add one more method in order to check if the green box was displayed:

```
/** Dialog search criteria */
private static final By dialogSearch = By.className("bgc-green-600");

public boolean checkConfirmationDialog() {
    WebElement greenConfirmationDialog = getDriver().findElementDynamic(dialogSearch);
    //creates a new WebElement to access confirmation dialog

    return greenConfirmationDialog.isDisplayed(); //checks if the dialog is displayed
}
```

3. Create BookTableTest class

At this point you can start creating a test class:

```
import static org.junit.Assert.assertTrue;

public class BookTableTest extends BaseTest {
    private static BookTablePage bookTablePage = new BookTablePage(); //the field
contains book table page object

    @BeforeClass
    public static void setUpBeforeClass() {
        bookTablePage.load(); //loads book table page
    }

    @AfterClass
    public static void tearDownAfterClass() {

    }

    @Override
    public void setUp() {
        if (!bookTablePage.isLoaded()) {
            bookTablePage.load(); //if the page is not loaded, loads it
        }
    }

    @Override
    public void tearDown() {

    }
}
```

4. Write the first test

You can prepare our first test method using the methods from page classes

```

@Test
public void Test_BookTableAndCheckConfirmation() {
    String date = "07/23/2019 1:00 PM"; //replace with tomorrow's date in format
    "MM/dd/yyyy hh:mm a"
    String name = "Smith"; //name field
    String email = "smith@somesmail.com"; //email field
    int guests = 3; //number of guests

    //enters booking data and returns a new confirmation page
    ConfirmBookPage confirmBookPage = bookTablePage.enterBookingData(date, name,
    email, guests);
    confirmBookPage.confirmBookingData(); //confirms booking

    //checks if the green dialog box appears, if it does, test is passed, if not, the
    test failed and displays message given in the first argument
    assertTrue("Test failed: Table not booked",
    bookTablePage.checkConfirmationDialog()); //returns true if dialog box appears and
    false if not
}

```

5. Run the test

Run the test by right-clicking on the test method → Run as → JUnit test.

79. Migration from JUnit4 to JUnit5

79.1. Migration guide

79.1.1. Junit4 to Junit5 migration guide

mrchecker-core-module version 5.6.2.1 features the upgrade of Junit4 to Junit5. Consequently, the Junit4 features are now obsolete and current test projects require migration in order to use the latest revision of MrChecker. This site provides guidance on the migration.

References: Junit5 User Guide - <https://junit.org/junit5/docs/current/user-guide/#overview>

POM

The project `pom.xml` file needs to be adjusted in the first place. An exemplary POM file for download can be found here: <https://github.com/devonfw/mrchecker/blob/develop/template/pom.xml>

Test Annotations

Junit5 redefines annotations defining a test flow. The annotations need to be adjusted as per the following table.

Test Annotations

Junit4	Junit5	Needed code changes
<code>org.junit.Test</code>	<code>org.junit.jupiter.api.Test</code>	- import statement
<code>org.junit.BeforeClass</code>	<code>org.junit.jupiter.api.BeforeAll</code>	- import statement - method annotations
<code>org.junit.Before</code>	<code>org.junit.jupiter.api.BeforeEach</code>	- import statement - method annotations
<code>org.junit.After</code>	<code>org.junit.jupiter.api.AfterEach</code>	- import statement - method annotations
<code>org.junit.AfterClass</code>	<code>org.junit.jupiter.api.AfterAll</code>	- import statement - method annotations
<code>org.junit.Ignore</code>	<code>org.junit.jupiter.api.Disabled</code>	- import statement - method annotations - additionally a description can be added
<code>org.junit.experimental.categories.Category</code>	<code>org.junit.jupiter.api.Tag</code>	- import statement - method annotations - description - OR introduce alternative annotations: <pre>@Target(ElementType.TYPE) @Retention(RetentionPolicy.RUNTIME) @Tag("TestsChrome") public @interface TestsChrome { }</pre>

Rule, ClassRule, TestRule and TestMethod

Junit4 `@Rule` and `@ClassRule` annotations as well as `TestRule` and `TestMethod` interfaces have been replaced with the Junit5 extension mechanism (<https://junit.org/junit5/docs/current/user-guide/#extensions>). During the migration to Junit5, all the instances of the mentioned types need to be rewritten according to the Junit5 User Guide. The extension mechanism is far more flexible than the Junit4 functionality based on rules.

Note: as per Junit5 API spec: `ExpectedExceptionSupport`, `ExternalResourceSupport`, `VerifierSupport` provide native support of the corresponding Junit4 rules.

Extension registration example:

```
public abstract class BaseTest implements IBaseTest {

    private final static PropertiesCoreTest setPropertiesSettings;

    @Rule
    public TestWatcher testWatcher = getBaseTestWatcher();

    @ClassRule
    public static TestClassRule classRule = new TestClassRule();
}
```



```
public abstract class BaseTest {
    private static PropertiesCoreTest propertiesCoreTest;

    @RegisterExtension
    public static final Extension TEST_EXECUTION_OBSERVER = TestExecutionObserver.getInstance();
}
```

TestRule (`TestWatcher` and `ExternalResource`) to Extension (`TestWatcher` and `AfterAllCallback`) example:

```
public class BaseTestWatcher extends TestWatcher {

    public static class TestClassRule extends ExternalResource {

        static final ThreadLocal<List<ITestObserver>> classObservers = initialValue() -> {
            return new ArrayList<ITestObserver>();
        };

        @Override
        protected void after() {
            classObservers.get()
                .clear();
        }
    }
}
```



```

public class TestExecutionObserver implements TestWatcher, AfterAllCallback {

    @Override
    public void afterAll(ExtensionContext extensionContext) {
        BFLogger.logDebug(getClass().getName() + ".observers: " + observers.get()
            .toString());

        BFLogger.logDebug(getClass().getName() + ".classObservers: " + classObservers.get()
            .toString());

        observers.get()
            .forEach(ITestObserver::onTestClassFinish);
        classObservers.get()
            .forEach(ITestObserver::onTestClassFinish);

        observers.get()
            .clear();
        classObservers.get()
            .clear();

        BFLogger.logDebug("All observers cleared.");
    }
}

```

Page, BasePageAutoRegistration and PageFactory classes

Page class is a new MrChecker class. It was introduced to provide common implementation for its subpages in specific MrChecker modules. In order to receive test lifecycle notifications, particular **Pages** need to be registered by calling `addToTestExecutionObserver()` method. To facilitate this process, **PageFactory** class was designed and it's usage is a recommended way of creating **Page** objects for tests. Although in MrChecker based on Junit4, the registration process was done in a specific **BasePage** constructor, it's been considered error prone and reimplemented. Furthermore, to reduce migration cost **BasePageAutoRegistration** classes are available in MrChceker modules. They use the old way of registration. Given that three ways of migration are possible.

Migration with **PageFactory** class example (**RECOMMENDED**):

```

public static TheInternetPage shouldTheInternetPageBeOpened() {

    logStep("Open the Url http://the-internet.herokuapp.com/");
    theInternetPage = new TheInternetPage();
    theInternetPage.load();

    logStep("Verify if Url http://the-internet.herokuapp.com/ is opened");
    assertTrue(message: "Unable to load The Internet Page", theInternetPage.isLoaded());

    return theInternetPage;
}

```



```
public static TheInternetPage shouldTheInternetPageBeOpened() {

    logStep("Open the Url http://the-internet.herokuapp.com/");
    theInternetPage = PageFactory.getPageInstance(TheInternetPage.class);
    theInternetPage.load();

    logStep("Verify if Url http://the-internet.herokuapp.com/ is opened");
    assertTrue( message: "Unable to load The Internet Page", theInternetPage.isLoaded());

    return theInternetPage;
}
```



Migration with calling `addToTestExecutionObserver()` method example:

```
public static TheInternetPage shouldTheInternetPageBeOpened() {

    logStep("Open the Url http://the-internet.herokuapp.com/");
    theInternetPage = new TheInternetPage();
    theInternetPage.load();

    logStep("Verify if Url http://the-internet.herokuapp.com/ is opened");
    assertTrue( message: "Unable to load The Internet Page", theInternetPage.isLoaded());

    return theInternetPage;
}
```



```
public static TheInternetPage shouldTheInternetPageBeOpened() {

    logStep("Open the Url http://the-internet.herokuapp.com/");
    theInternetPage = new TheInternetPage();
    theInternetPage.addToTestExecutionObserver();
    theInternetPage.load();

    logStep("Verify if Url http://the-internet.herokuapp.com/ is opened");
    assertTrue( message: "Unable to load The Internet Page", theInternetPage.isLoaded());

    return theInternetPage;
}
```

Migration with `BasePageAutoRegistration` class example:

```
public class TheInternetPage extends BasePage {
```



```
public class TheInternetPage extends BasePageAutoRegistration {
```

Test suites

Test suite migration example:

```
@RunWith(WildcardPatternSuite.class)
@IncludeCategories({ TestsChrome.class })
@ExcludeCategories({ TestsLocal.class, TestsNONParallel.class })
@SuiteClasses({ "../**/*Test.class" })
public class _TestSuiteChrome {
```



```
@RunWith(JUnitPlatform.class)
@IncludeTags({ "TestsChrome" })
@ExcludeTags({ "TestsLocal", "TestsNONParallel" })
@SelectPackages("com.cappgemini.mrchecker.selenium")
public class _TestSuiteChrome {
```

Running tests from Maven:

```
>mvn verify -Dtest=_TestSuiteChrome
```



```
>mvn verify -Dgroups=TestsChrome -DexcludedGroups=TestsLocal,TestsNONParallel
```

Concurrency

Junit5 provides native thread count and parallel execution control in contrast to Junit4 where it was controlled by Maven Surefire plugin. To enable concurrent test execution, `junit-platform.properties` file needs to placed in the `test/resources` directory of a project.

Exemplary file contents:

```
junit.jupiter.extensions.autodetection.enabled=true
junit.jupiter.execution.parallel.enabled=true
#junit.jupiter.execution.parallel.mode.default=concurrent
junit.jupiter.execution.parallel.mode.default=same_thread
junit.jupiter.execution.parallel.mode.classes.default=concurrent
#junit.jupiter.execution.parallel.mode.classes.default=same_thread
junit.jupiter.execution.parallel.config.strategy=dynamic
junit.jupiter.execution.parallel.config.dynamic.factor=1
```

A ready-to-use file can be found [here](#).

MrChecker supports only concurrent test class execution. `@ResourceLock` can be used to synchronize between classes if needed:

```
@UnitTest
@ResourceLock(value = "SingleThread")
public class DataEncryptionModuleTest {
```

Cucumber

If Cucumber is used in a project, it is neccessary to change a hook class. An exemplary hook source file for download can be found [here](#).

Data driven tests

Junit5 implements new approach to data driven tests by various data resolution mechanisms.

An example of method source parameters migration version one:

```
@Test
@Parameters(method = "addParameters")
public void addProducesCorrectValue_usingNamedMethodParameters(final int a, final int b, final int expectedResult) {
    assertEquals(expectedResult, testSubject.add(a, b));
}

private Object[] addParameters() {
    return new Object[] {
        new Object[] { 1, 2, 3 },
        new Object[] { 3, 4, 7 },
        new Object[] { 5, 6, 11 },
        new Object[] { 7, 8, 15 }
    };
}
```



```
@ParameterizedTest
@MethodSource("argumentsStream")
public void addProducesCorrectValue_usingMethodSourceArgumentsStream(final int a, final int b, final int expectedResult) {
    assertEquals(expectedResult, testSubject.add(a, b));
}

static Stream<Arguments> argumentsStream() {
    return Stream.of(
        Arguments.of(1, 2, 3),
        Arguments.of(3, 4, 7),
        Arguments.of(5, 6, 11),
        Arguments.of(7, 8, 15));
}
```

An example of method source parameters migration version two:

```
@Test
@Parameters(method = "addParameters")
public void addProducesCorrectValue_usingNamedMethodParameters(final int a, final int b, final int expectedResult) {
    assertEquals(expectedResult, testSubject.add(a, b));
}

private Object[] addParameters() {
    return new Object[] {
        new Object[] { 1, 2, 3 },
        new Object[] { 3, 4, 7 },
        new Object[] { 5, 6, 11 },
        new Object[] { 7, 8, 15 }
    };
}
```



```
@ParameterizedTest
@MethodSource("arrayStream")
public void addProducesCorrectValue_usingMethodSourceArrayStream(final int[] values) {
    assertEquals(values[2], testSubject.add(values[0], values[1]));
}

static Stream<int[]> arrayStream() {
    return Stream.of(new int[] { 1, 2, 3 }, new int[] { 3, 4, 7 }, new int[] { 5, 6, 11 }, new int[] { 7, 8, 15 });
}
```

An example of method source in another class parameters migration:

```

@在这
@Parameters(source = MyContainsTestProvider.class)
public void testContains_usingSeparateClass(final List<String> list, final String searchString, final boolean expectedResult) {
    assertEquals(expectedResult, testSubject.contains(list, searchString));
}

public static class MyContainsTestProvider {
    public static Object[] provideContainsTrueParameters() {
        return new Object[] {
            new Object[] { Arrays.asList("a", "b", "c", "d", "e"), "c", true },
            new Object[] { Arrays.asList("a", "b", "c", "d", "e"), "e", true },
            new Object[] { Arrays.asList("a", "b"), "b", true },
            new Object[] { Arrays.asList("a"), "a", true }
        };
    }

    public static Object[] provideContainsFalseParameters() {
        return new Object[] {
            new Object[] { Arrays.asList("a", "b", "c", "d", "e"), "f", false },
            new Object[] { Arrays.asList("a", "b", "c", "d", "e"), "z", false },
            new Object[] { Arrays.asList("a", "b"), "e", false },
            new Object[] { Arrays.asList(), "e", false }
        };
    }
}

```



```

@ParameterizedTest
@MethodSource("com.cappemini.mrchecker.core.datadriven.MyContainsTestProvider#provideContainsTrueParameters")
public void testContains_usingSeparateClass(final List<String> list, final String searchString, final boolean expectedResult) {
    assertEquals(expectedResult, testSubject.contains(list, searchString));
}

//unused/
class MyContainsTestProvider {
    public static Stream<Arguments> provideContainsTrueParameters() {
        return Stream.of(
            Arguments.of(Arrays.asList("a", "b", "c", "d", "e"), "c", true),
            Arguments.of(Arrays.asList("a", "b", "c", "d", "e"), "e", true),
            Arguments.of(Arrays.asList("a", "b"), "b", true),
            Arguments.of(Collections.singletonList("a"), "a", true));
    }

    public static Stream<Arguments> provideContainsFalseParameters() {
        return Stream.of(
            Arguments.of(Arrays.asList("a", "b", "c", "d", "e"), "f", false),
            Arguments.of(Arrays.asList("a", "b", "c", "d", "e"), "z", false),
            Arguments.of(Arrays.asList("a", "b"), "e", false),
            Arguments.of(Collections.emptyList(), "e", false));
    }
}

```

Providing parameters directly in annotations has no analogy in JUnit5 and needs to be replaced with e.g. method source:

```

@在这
@Parameters({ "1, 2, 3",
    "3, 4, 7",
    "5, 6, 11",
    "7, 8, 15" })
public void addProducesCorrectValue_usingAnnotatedParameters(final int a, final int b, final int expectedResult) {
    assertEquals(expectedResult, testSubject.add(a, b));
}

```



```

@ParameterizedTest
@MethodSource("argumentsStream")
public void addProducesCorrectValue_usingMethodSourceArgumentsStream(final int a, final int b, final int expectedResult) {
    assertEquals(expectedResult, testSubject.add(a, b));
}

static Stream<Arguments> argumentsStream() {
    return Stream.of(
        Arguments.of(1, 2, 3),
        Arguments.of(3, 4, 7),
        Arguments.of(5, 6, 11),
        Arguments.of(7, 8, 15));
}

```

An example of csv parameters source with no header line migration:

```

@Test
@FileParameters("src/test/resources/datadriven/test.csv")
public void loadParamsFromCsv(String age, String name) {
    BFLogger.logDebug("DataDrivenExampleTest.loadParamsFromCsv()");
    BFLogger.logDebug("\t" + "Name=" + name + " " + "Age=" + age);
}

```



```

@ParameterizedTest
@CsvFileSource(resources = "/datadriven/test.csv")
public void loadParamsFromCsv(String age, String name) {
    BFLogger.logDebug("DataDrivenExampleTest.loadParamsFromCsv()");
    BFLogger.logDebug("\t" + "Name=" + name + " " + "Age=" + age);
}

```

An example of csv parameters source with the header line migration:

```

@Test
@FileParameters(value = "src/test/resources/datadriven/with_header.csv", mapper = CsvWithHeaderMapper.class)
public void loadParamsFromCsvWithHeader(String age, String name) {
    BFLogger.logDebug("DataDrivenExampleTest.loadParamsFromCsvWithHeader()");
    BFLogger.logDebug("\t" + "Name=" + name + " " + "Age=" + age);
}

```



```

@ParameterizedTest
@CsvFileSource(resources = "/datadriven/with_header.csv", numLinesToSkip = 1)
public void loadParamsFromCsvWithHeader(String age, String name) {
    BFLogger.logDebug("DataDrivenExampleTest.loadParamsFromCsvWithHeader()");
    BFLogger.logDebug("\t" + "Name=" + name + " " + "Age=" + age);
}

```

An example of csv parameters source with object mapping migration step1:

```

@Test
@FileParameters(value = "src/test/resources/datadriven/test.csv", mapper = PersonMapper.class)
public void loadParamsFromAnyFile(Person person) {
    BFLogger.logDebug("DataDrivenExampleTest.loadParamsFromAnyFile()");
    BFLogger.logDebug("\t" + "Name=" + person.getName() + " " + "Age=" + person.getAge());
}

```



```

@ParameterizedTest
@CsvFileSource(resources = "/datadriven/test.csv")
public void loadParamsFromAnyFile(@AggregateWith(PersonAggregator.class) PersonAggregator.Person person) {
    BFLogger.logDebug("DataDrivenExampleTest.loadParamsFromAnyFile()");
    BFLogger.logDebug("\t" + "Name=" + person.getName() + " " + "Age=" + person.getAge());
}

```

An example of csv parameters source with object mapping migration step 2:

```

public class PersonMapper extends CsvWithHeaderMapper {
    @Override
    public Object[] map(Reader reader) {
        Object[] map = super.map(reader);
        List<Object[]> result = new LinkedList<~>();
        for (Object lineObj : map) {
            String line = (String) lineObj;

            // Splitted with ","
            Object[] lineSplitted = line.split( regex: "[,]"); // Example line { 21,John }

            // Order of arguments must be inline with Person class constructor argument list
            result.add(lineSplitted);
        }
        return result.toArray();
    }

    public static class Person {
        private String name;
        private Integer age;

        // Arguments order depends on data in CSV line
        public Person(String age, String name) {
            this.name = name;
            setAge(age);
        }
    }
}

```



```

public class PersonAggregator implements ArgumentsAggregator {

    @Override
    public Object aggregateArguments(ArgumentsAccessor argumentsAccessor, ParameterContext parameterContext) throws ArgumentsAggregationException {
        return new Person(argumentsAccessor.getString( 0),
                          argumentsAccessor.getString( 1));
    }

    public static class Person {
        private String name;
        private Integer age;

        // Arguments order depends on data in CSV line
        public Person(String age, String name) {
            this.name = name;
            setAge(age);
        }
    }
}

```

setUp() and tearDown()

`BaseTest.setUp()` and `BaseTest.tearDown()` methods are now not abstract and need no implementation in subclasses. `@Override` when a custom implemenatation is needed.

80. FAQ

Here you can find the most frequently asked questions regarding working with MrChecker and installation problems.

80.1. Common problems

80.1.1. I can't find the boilerplate module. Has it been removed?

The *boilerplate* module has been removed from the GitHub project on purpose.

There were problems with naming and communication, not everybody was aware of the meaning of the word *boilerplate*.

The name of the folder has been changed to *template*. It can be found in the GitHub project.

80.1.2. Is it possible to use Docker with MrChecker?

MrChecker works seamlessly with Docker.

A direct example of how to use it can be found in following JenkinsFile https://github.com/devonfw/devonfw-testing/blob/develop/mrchecker-app-under-test/pipelines/CI/Jenkinsfile_node.groovy

Note that the structure of the folders can be changed. If that happens - search in repo for */pipeline/CI/Jenkinsfile_node.groovy*

80.1.3. Tests are not stable

Selenium tests perform actions much faster than a normal user would. Because pages can contain dynamically changing content, some web elements can still not be loaded when Selenium driver tries to access them.

`getDriver().waitForPageLoaded()` method checks ready state in the browser, that's why stability problems may happen in advanced frontend projects.

To improve test stability you can:

- add waiting methods before dynamically loading elements e.g. `getDriver().waitForElement(By selector)`
- add timeout parameter in `method getDriver().findElementDynamic(By selector, int timeOut)`
- change global waiting timeout value using method `getDriver().manage().timeouts().implicitlyWait(long time, TimeUnit unit)`

Furthermore, if the page displays visible loading bars or spinners, create FluentWait method to wait until they disappear.

Notice that by increasing timeouts you may improve stability but too long waiting time makes tests run slower.

Learn more from tutorials:

[Example 10: Dynamically loaded Elements](#)

[Example 11: Exit intent](#)

[Example 12: File download test](#)

80.2. How to

80.2.1. How to: Change timeouts?

If you would like to change timeouts - you don't have to change them globally. It is possible to add waiting time parameter to searching methods, such as:

```
getDriver().findElementDynamic(By selector, int timeOut)  
timeout - in seconds
```

It is recommended to use methods that significantly level up the repetitiveness of the code:

```
getDriver().waitForElement(By selector);  
  
getDriver().waitForElementVisible(By selector);  
  
getDriver().waitForPageLoaded();  
  
getDriver().waitUntilElementIsClickable(By selector);
```

Or Fluent Wait methods with changed timeout and interval:

```
FluentWait<WebDriver> wait = new FluentWait<WebDriver>(getDriver())  
    .withTimeout(long duration, TimeUnit unit)  
    .pollingEvery(long duration, TimeUnit unit);  
wait.until((WebDriver wd) -> expectedCondition.isTrue());  
getWebDriverWait().withTimeout(millis, TimeUnit.MILLISECONDS)  
    .withTimeout(long duration, TimeUnit unit)  
    .pollingEvery(long duration, TimeUnit unit)  
    .until((WebDriver wd) -> expectedCondition.isTrue());
```

These methods allow You to change WebDriver timeouts values such as:

```
getDriver().manage().timeouts().pageLoadTimeout(long time, TimeUnit unit)
```

the amount of time to wait for a page to load before throwing an exception. This is the default timeout for method `getDriver().waitForPageLoaded()`

```
getDriver().manage().timeouts().setScriptTimeout(long time, TimeUnit unit)
```

the amount of time to wait for execution of script to finish before throwing an exception

```
getDriver().manage().timeouts().implicitlyWait(long time, TimeUnit unit)
```

the amount of time the driver should wait when searching for an element if it is not immediately present. After that time, it throws an exception. This is the default timeout for methods such as

`getDriver().findElementDynamic(By selector) or getDriver().waitForElement(By selector)`

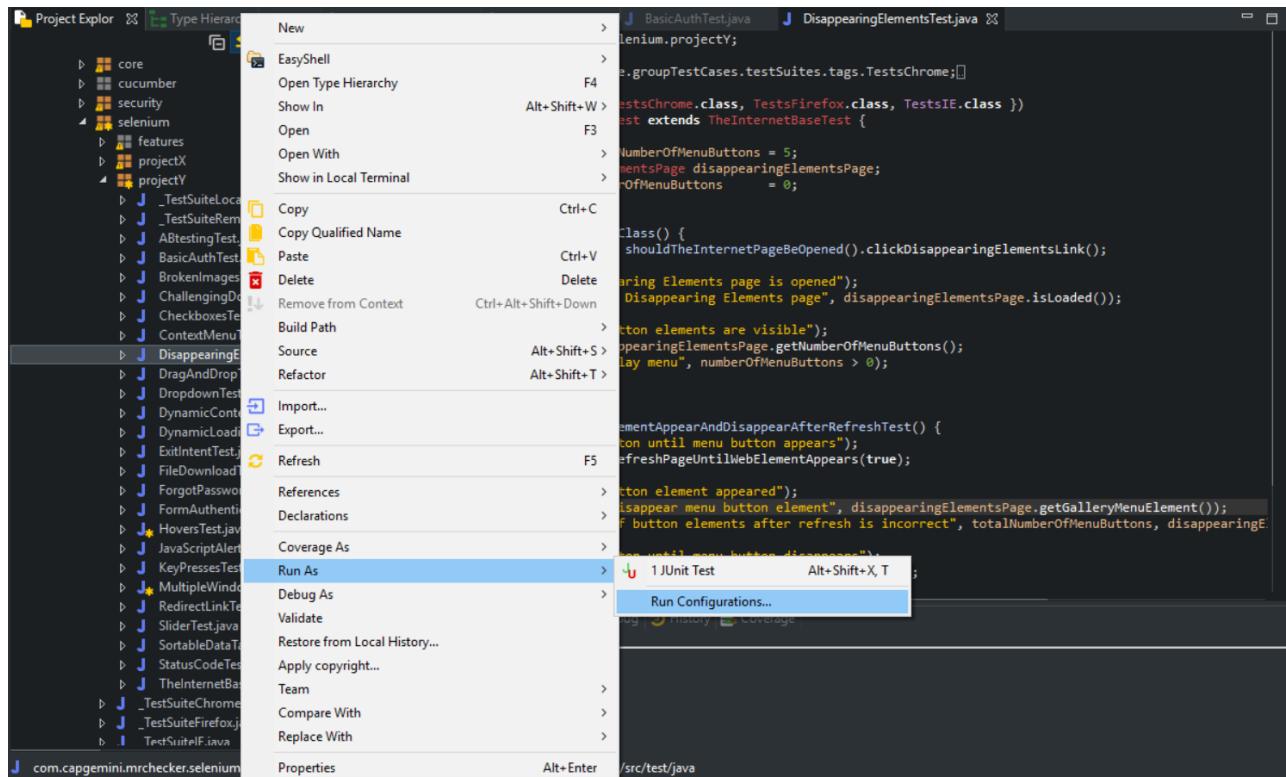
Changing timeouts can improve test stability but can also make test run time longer.

80.2.2. How to: Start a browser in Incognito/Private mode?

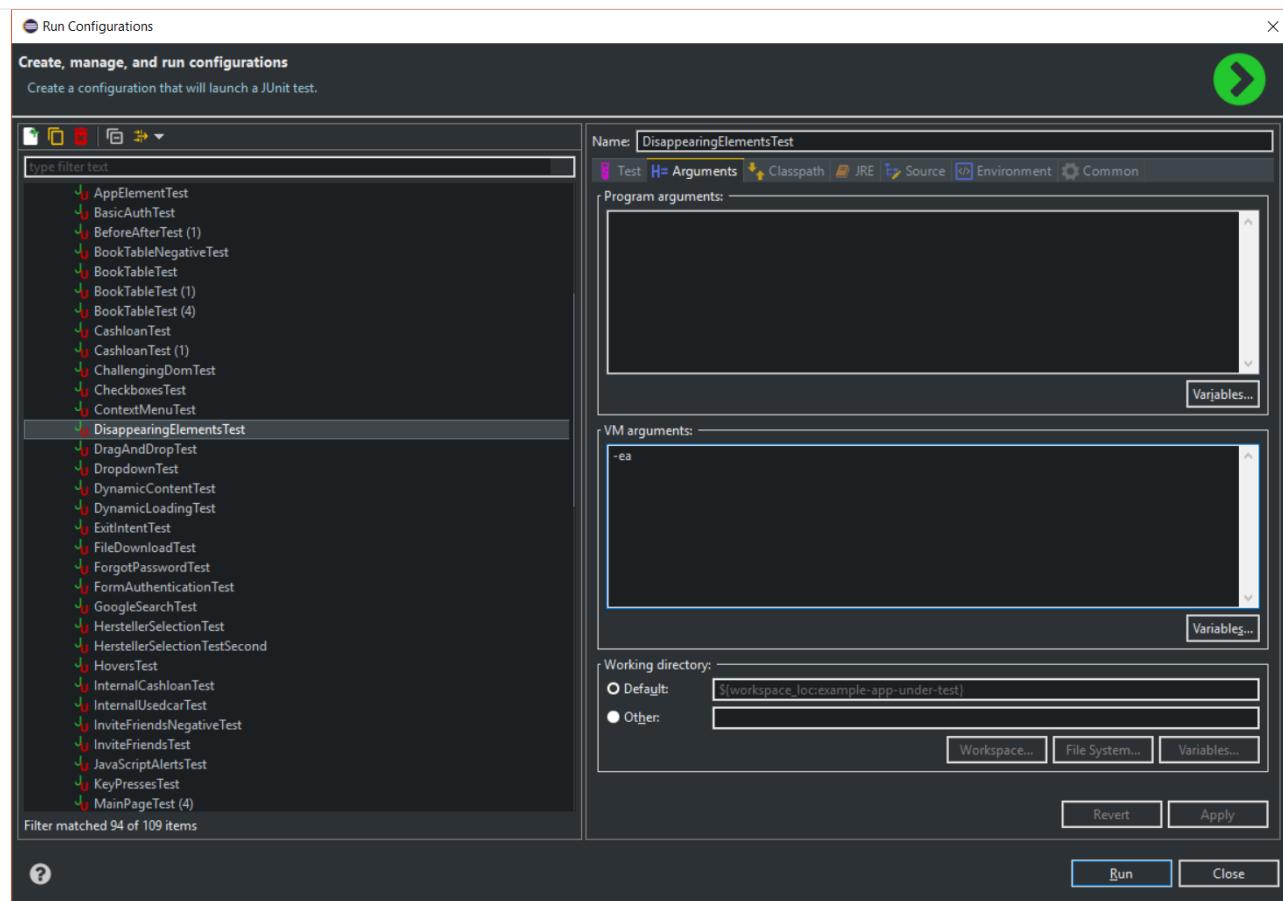
In MrChecker there is a possibility of changing browser options during runtime execution.

To run the browser in incognito mode:

1. In Eclipse - open Run Configurations window:

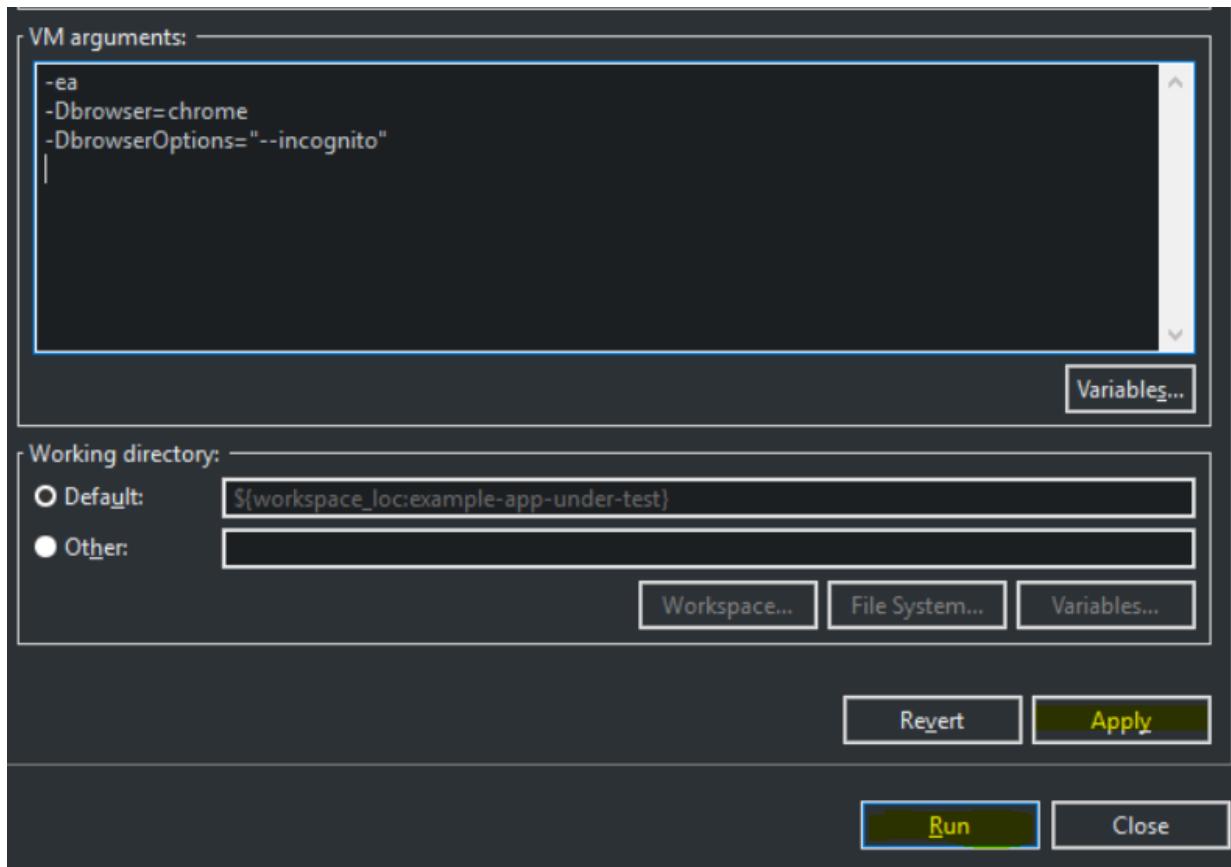


2. Select a test which you want to run and switch to arguments tab:



3. Add VM argument:

- for the incognito mode in chrome:



Read more about browsers' options: <https://github.com/devonfw/devonfw-testing/wiki/Run-with->

80.3. Installation problems

80.3.1. Chromedriver version is not compatible with Chrome browser

Problem:

During the tests your web browser window opens and immediately closes, all your tests are broken.

Following error message is visible in the test description:

```
session not created: This version of ChromeDriver only supports Chrome version 76
Build info: version: '<build_version>', revision: '<build_revision>', time: '<time>'
System info: host: '<your_computer_name>', ip: '<your_ip_address>', os.name: '<your_os_name>', os.arch: '<your_os_architecture>', os.version: '<your_os_version>', java.version: '<java_version_installed>'
Driver info: driver.version: NewChromeDriver
```

Solution:

1. Make a change in the following files:

- `MrChecker_Test_Framework\workspace\devonfw-testing\src\resources\settings.properties`
- For project template-app-under-test: `MrChecker_Test_Framework\workspace\devonfw-testing\template\src\resources\settings.properties`
- For project example-app-under-test: `MrChecker_Test_Framework\workspace\devonfw-testing\example\src\resources\settings.properties`

Change the value of `selenium.driverAutoUpdate` field from `true` to `false`

2. Replace the following file with a version compatible with your browser: `MrChecker_Test_Framework\workspace\devonfw-testing\example\lib\webdrivers\chrome\chromedriver.exe`.

80.3.2. My browser opens up in German by default

Problem:

I would like my browser to use the English language, but the default language for the browser is German. How can I change the settings?

Solution:

There is a Properties file installed together with MrCheker installation. It is possible to set the language in which a browser could be opened for testing purposes in *Properties > Selenium configuration*,.

Part XIII: MyThaiStar

81. 1.My Thai Star – Agile Framework

81.1. 1.1 Team Setup

The team working on the development of the My Thai Star app and the documentation beside the technical development works distributed in various locations across Germany, the Netherlands, Spain and Poland. For the communication part the team uses the two channels Skype and Mail and for the documentation part the team makes usage mainly of GitHub and Jira.

81.2. 1.2 Scrum events

81.2.1. Sprint Planning

Within the My Thai Star project we decided on having one hour Sprint Planning meetings for a four-week Sprints. This decision is based on the fact that this project is not the main project of the team members. As the backlog refinement is done during the Sprint Planning we make usage of the planningpoker.com tool for the estimation of the tasks.

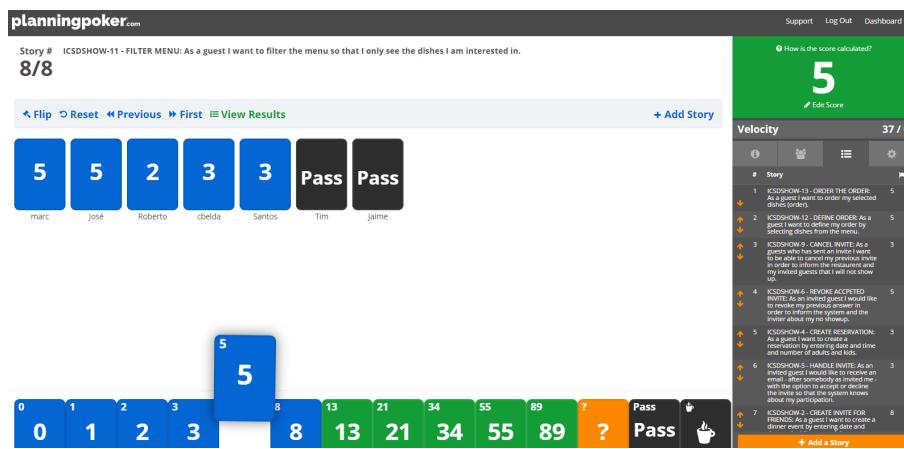


Figure 86. Screenshot of planningpoker.com during Sprint 1 Planning

During the Sprint Planning meeting the team receives support from Devon colleagues outside the development. This feedback helps the team to focus on important functionalities and task by keeping the eyes on the overall aim which is to have a working application by the end of June 2017.

81.2.2. Sprint Review

The Sprint Review meetings are time boxed to one hour for the four week Sprint. Within the Sprint Review meeting the team plans to do a retrospective of the finished Sprint. As well as it is done during the Sprint Planning the team receives support from Devon colleagues.

81.2.3. Sprint Retrospective

For this project the team aligned on not having a specific Sprint Retrospective meeting. The team is going to have a retrospective of a finished Sprint during the Sprint Review.

81.2.4. Daily Standups

The team aligned on having two weekly Standup meetings instead of a Daily Standup meeting. In comparison with the time boxed length of 15mins described in the CAF for this project the team extended the Standup meeting to 30mins. The content of the meetings remains the same.

81.2.5. Backlog refinement

The team decided that the backlog refinement meeting is part of the Sprint Planning meeting.

81.3. 1.3 Establish Product Backlog

For the My Thai Stair project the team decided on using the Jira agile documentation which is one of the widely used agile tools. Jira is equipped with several of useful tools regarding the agile software development (e.g. Scrum-Board). One of the big advantages of Jira are the extensive configuration and personalization possibilities. With having a list of the Epics and User Stories for the My Thai Star development in GitHub, the team transferred the User Stories into the Jira backlog as it is shown in the screenshot below. All User Stories are labeled colorfully with the related Epic which shapes the backlog in clearly manner.

Issue	Epic	Count
ICSDSHOW-2 Create invite for friends	Invite friends	8
ICSDSHOW-4 Create reservation	Invite friends	3
ICSDSHOW-11 Filter menu	Digital Menu	5
ICSDSHOW-12 Define order	Digital Menu	5
ICSDSHOW-5 Handle invite	Invite friends	3
ICSDSHOW-6 Revoke accepted invite	Invite friends	5
ICSDSHOW-9 Cancel invite	Invite friends	3
ICSDSHOW-13 Order the order	Digital Menu	5
ICSDSHOW-7 Calculate best table	Invite friends	8
ICSDSHOW-22 See all orders/reservations	Waiter cockpit	5
ICSDSHOW-8 Find table by reservation info	Invite friends	3
ICSDSHOW-14 Cancel order	Digital Menu	3
ICSDSHOW-17 User Profile and Twitter integration	User profile	13
ICSDSHOW-19 Receive mail to rate your dish	Rate by twitter	5
ICSDSHOW-20 Rate your dish	Rate by twitter	5
ICSDSHOW-15 Read twitter rating for dishes	Digital Menu	5

Figure 87. Screenshot of the Jira backlog during Sprint 2

We decided on working with Subtask as a single user story comprised a number of single and separated tasks. Another benefit of working with subtask is that every single subtask can be assigned to a single team member whereas a user story can only be assigned to one team member. By picking single subtask the whole process of a user story is better organized.

Active sprints: All sprints -

QUICK FILTERS: Only My Issues Recently Updated

To Do In Progress Done

The screenshot shows a Jira board with three columns: To Do, In Progress, and Done. The To Do column contains two items: 'ICSDSHOW-2' and 'ICSDSHOW-4'. The In Progress column contains two items: 'ICSDSHOW-38' and 'ICSDSHOW-40'. The Done column contains five items: 'ICSDSHOW-23', 'ICSDSHOW-24', 'ICSDSHOW-25', 'ICSDSHOW-39', and 'ICSDSHOW-73'. Each item has a small icon, a title, a description, and a user profile picture.

To Do	In Progress	Done
ICSDSHOW-2 GraphQL service methods	ICSDSHOW-38 Xamarin view components development	ICSDSHOW-23 Angular view components development
ICSDSHOW-4 GraphQL service methods	ICSDSHOW-40 Xamarin view components development	ICSDSHOW-24 Angular services development
		ICSDSHOW-25 OASAPI server side methods
		ICSDSHOW-39 OASNET server side methods
		ICSDSHOW-73 OASAPIFN server side methods

Figure 88. Screenshots of Subtasks during Sprint 2

82. 2.My Thai Star – Agile Diary

In parallel to the Diary Ideation we use this Agile Diary to document our Scrum events. The target of this diary is to describe the differences to the Scrum methodology as well as specific characteristics of the project. We also document the process on how we approach the Scrum methodology over the length of the project.

82.1. 24.03.2017 Sprint 1 Planning

Within the Sprint 1 Planning we used planning poker.com for the estimation of the user stories. The estimation process usually is part of the backlog refinement meeting. Regarding the project circumstances we decided to estimate the user stories during the Sprint Planning. Starting the estimation process we noticed that we had to align our interpretation of the estimation effort as these story points are not equivalent to a certain time interval. The story points are relative values to compare the effort of the user stories. With this in mind we proceeded with the estimation of the user stories. We decided to start Sprint 1 with the following user stories and the total amount of 37 story points:

- ICSDSHOW-2 Create invite for friends (8 Story Points)
- ICSDSHOW-4 Create reservation (3)
- ICSDSHOW-5 Handle invite (3)
- ICSDSHOW-6 Revoke accepted invite (5)
- ICSDSHOW-9 Cancel invite (3)
- ICSDSHOW-11 Filter menu (5)
- ICSDSHOW-12 Define order (5)
- ICSDSHOW-13 Order the order (5)

As the Sprint Planning is time boxed to one hour we managed to hold this meeting within this time window.

82.2. 27.04.2017 Sprint 1 Review

During the Sprint 1 Review we had a discussion about the data model proposal. For the discussion we extended this particular Review meeting to 90min. As this discussion took almost 2/3 of the Review meeting we only had a short time left for our review of Sprint 1. For the following scrum events we decided to focus on the primary target of these events and have discussions needed for alignments in separate meetings. Regarding the topic of splitting user stories we had the example of a certain user story which included a functionality of a twitter integration (ICSDSHOW-17 User Profile and Twitter integration). As the twitter functionality could not have been implemented at this early point of time we thought about cutting the user story into two user stories. We aligned on mocking the twitter functionality until the dependencies are developed in order to test the components. As this user story is estimated with 13 story points it is a good example for the question whether to cut a user story into multiple user stories or not. Unfortunately not all user stories of Sprint 1 could have been completed. Due this situation we discussed on whether pushing all unfinished user stories into the status done or moving them to Sprint 2. We aligned on transferring the unfinished user stories into the next Sprint. During the Sprint 1 the team underestimated that a lot of holidays crossed the Sprint 1 goals. As taking holidays and absences of team members into consideration is part of a Sprint Planning we have a learning effect on setting a Sprint Scope.

82.3. 03.05.2017 Sprint 2 Planning

As we aligned during the Sprint 1 Review on transferring unfinished user stories into Sprint 2 the focus for Sprint 2 was on finishing these transferred user stories. During our discussion on how

many user stories we could work on in Sprint 2 we needed to remind ourselves that the overall target is to develop an example application for the DevonFW. Considering this we aligned on a clear target for Sprint 2: To focus on finishing User Stories as we need to aim for a practicable and realizable solution. Everybody aligned on the aim of having a working application at the end of Sprint 2. For the estimation process of user stories we make again usage of planningpoker.com as the team prefers this “easy-to-use” tool. During our second estimation process we had the situation in which the estimated story points differs strongly from one team member to another. In this case the team members shortly explains how the understood and interpreted the user story. It turned out that team members misinterpreted the user stories. With having this discussion all team members got the same understanding of the specific functionality and scope of a user story. After the alignment the team members adjusted their estimations. Beside this need for discussion the team estimated most of the user stories with very similar story points. This fact shows the increase within the effort estimation for each team member in comparison to Sprint 1 planning. Over the short time of two Sprint planning the team received a better understanding and feeling for the estimation with story points.

82.4. 01.06.2017 Sprint 2 Review

As our Sprint 1 Review four weeks ago was not completely structured like a Sprint Review meeting we focused on the actual intention of a Sprint Review meeting during Sprint 2 Review. This means we demonstrated the completed and implemented functionalities with screen sharing and the product owner accepted the completed tasks. Within the User Story ICSDSHOW-22 “See all orders/reservations” the functionality “filtering the list by date” could have not been implemented during Sprint 2. The team was unsure on how to proceed with this task. One team member added that especially in regards of having a coherent release, implementing less but working functionalities is much better than implementing more but not working functionalities. For this the team reminded itself focusing on completing functionalities and not working straight to a working application.

83. User Stories

The list of user stories, exported from JIRA, can be downloaded from [here](#).

83.1. Epic: Invite friends

83.1.1. US: create invite for friends

Epic: Invite friends

As a guest I want to create an dinner event by entering date and time and adding potential guests by their emails so that each potential guests receives a email in order to confirm or decline my invite.

Acceptance criteria

1. only date and time in future possible and both required
2. only valid email addresses: text@text.xx, one entered email-address is required
3. if AGB are not checked, an error message is shown
4. after the invite is done
 - a. I see the confirmation screen of my invite (see wireframe)
 - b. I receive a confirmation email about my invite containing date, time and invited guests
 - c. all guests receive a mail with my invite

83.1.2. US: create reservation

Epic: Invite friends

As a guest I want to create a reservation by entering date and time and number of adults and kids

Acceptance criteria

1. only date and time in future possible and both required
2. only valid email addresses: text@text.xx, one entered email-address is required
3. if AGB are not checked, an error message is shown
4. after the reservation is done
 - a. I see a confirmation screen of my reservation with datetime, number of persons and kids
 - b. I receive a confirmation email about my reservation

Wireframes

see realtimeboard

83.1.3. US: handle invite

As an invited guest I would like to receive an email - after somebody has invited me - with the option to accept or decline the invite so that the system knows about my participation

AC:

1. the mail contains the following information about the invite
 - a. who has invited
 - b. who else is invited
 - c. date and time of the invite
 - d. button to accept or decline
 - e. after pressing the buttons the system will store the status (yes/no) of my invite

83.1.4. US: revoke accepted invite

As an invited guest I would like to revoke my previous answer in order to inform the system and the inviter about my no showup

AC:

1. the inviter and myself receives an email about my cancellation
2. the system sets my status of my invite to no
3. in case I have placed an order, the order is also removed from the system.
4. the cancellation is only possible 10 minutes before the event takes place. The system shows a message that cancellation is not possible anymore.

83.1.5. US: calculate best table

As a guest I would like the system to check (1 hour before my invite) all my invites and to reserve a table fitting the number of accepted users

Details

Pseudo-algorithm for reservation: Find table for given date and time where seats of guests \geq Count of invited guests plus one. In case no results, decline request and show error message to user. In case of any result, make a reservation for table.... For each decline of a guest remove guest and search with reduced number for new table. In case table is found, reserve it and remove reservation from previous table. In case not, do not change reservations.

83.1.6. US: find table by reservation info

As a waiter I would like to search by reference number or email address for the reserved table in order to know the table for my visit. (when arriving at the restaurant)

AC:

1. After entering the email the system shows the number of the table. In case no reservation found, a message is shown.
2. Entered email address could be email of inviter or any invited guest.

83.1.7. US: cancel invite

Epic: Invite friends

As a guest who has sent an invite I want to be able to cancel my previous invite in order to inform the restaurant and my invited guests that I will not show up

AC:

1. the option to cancel the invite is available in the confirmation-mail about my invite
2. after my cancellation all invited guests receive a mail about the cancellation
3. I see a confirmation that my invite was cancelled successfully
4. after my cancellation my invite and reservation and all orders related to it are deleted from the system and no one can accept or decline any invite for it
5. the cancellation is only possible one hour before the invite takes place. After that I am not allowed to cancel it any more.

83.2. Epic: Digital Menu

83.2.1. US: filter menu

As a guest I want to filter the menu so that I only see the dishes I am interested in

AC:

1. the guest can filter by
 - a. type: starter | main dish | dessert; XOR; if nothing is selected all are shown (default value)
 - b. veggy (yes | no | does not matter (default))
 - c. vegan (yes | no | does not matter (default))
 - d. rice (yes | no | does not matter (default))
 - e. curry (yes | no | does not matter (default))
 - f. noodle (yes | no | does not matter (default))
 - g. price (range)
 - h. ratings (range)
 - i. my favorite (yes | no | does not matter (default)) — free text (search in title and description)
2. the guest can sort by price asc, rating asc
3. after setting the filter only dishes are shown which fulfill those criteria

-
4. by pressing the button reset filter all filter are reset to the initial value
 5. by pressing the filter button the filter is applied [or is it triggered after each change?]

83.2.2. US: Define order

As a guest I want to define my order by selecting dishes from the menu

AC:

- The guest can add each dish to the order
- In case the guest adds the same dish multiple times, a counter in the order for this dish is increased for this dish
- The guest can remove the dish from the order
- The guest can add for each main dish the type of meat (pork, chicken, tofu)
- The guest can add for each dish a free-text-comment
- After adding/removing any dish the price is calculated including VAT

83.2.3. US: Order the order

As a guest I want to order my selected dishes (order)

AC:

1. I receive a mail containing my order with all dishes and the final price
2. precondition for ordering:
 - a. Each order must be associated with a reservation / invite. Without any reference no order could be placed. The reference could be obtained from a previous reservation/invite (created during same session) or by the previous accepted invite (link in email) or by entering the reference id when asked by the system.
 - i. In case precondition is not fulfilled, the guest is asked
 - A. whether he/she would like to create a reservation/invite and is forwarded to US Invite Friends. Only after finalizing the reservation the order is accepted.
 - B. or he/she would enter previous created reservation-id he/she knows in order to associate his/her order with this reservation

83.2.4. US: Cancel order

As a guest I want to cancel my order.

AC:

1. in my received confirmation mail I have the option to cancel my order
2. the cancellation is only possible one hour before my reservation takes place
3. my order is deleted from the system

Remark: Changing the order is not possible. For that the order must be canceled and created from scratch again

83.2.5. US: Read twitter rating for dishes

As a guest I want to read for all dishes the rating done by twitter because I would like to know the opinion of others

AC:

1. For each dish I see the latest 3 comments done by twitter for this vote (text, username, avatar)
2. For each dish I see the number of likes done by twitter

83.3. Epic: User Profile

83.3.1. US: User Profile

As a guest I want to have a user profile to associate it with my twitter account to be able to like/rate dishes

AC:

1. Username of my profile is my email address
2. My profile is protected by password
3. I can log in and log out to my profile
4. I can reset my password by triggering the reset by mail
5. I can associate my profile with my twitter account in order to rate dishes and store my favorites by liking posts associated to dishes

83.4. Epic: Rate by twitter

83.4.1. US: Receive mail to rate your dish

As a guest I want to receive a mail by the system in order to rate my dish

83.4.2. US: Rate your dish

As a guest I want to add a comment or a like via my twitter account for a dish

AC:

1. Before I write my rate I would like to be able to read all tweets of other users for this dish
2. I would like to see the number of likes for a dish

83.5. Epic: Waiter Cockpit

83.5.1. US: See all orders/reservations

As a waiter I want to see all orders/reservation in order to know what is going on in my restaurant

AC:

1. all orders/reservations are shown in a list view (read-only). Those list can be filtered and sorted (similar to excel-data-filters)
2. orders/reservations are shown in separate lists.
3. for each order the dish, meat, comment, item, reservation-id, reservation datetime, creation-datetime is shown
4. for each reservation the inviters email, the guests-emails, the number of accepts and declines, calculated table number, the reservation-id, reservation date-time and creation-datetime are shown
5. the default filter for all lists is the todays date for reservation datetime. this filter can be deleted.
6. only reservations and orders with reservation date in the future shall be available in this view. All other orders and reservation shall not be deleted; for data analytics those orders and reservation shall still exist in the system.

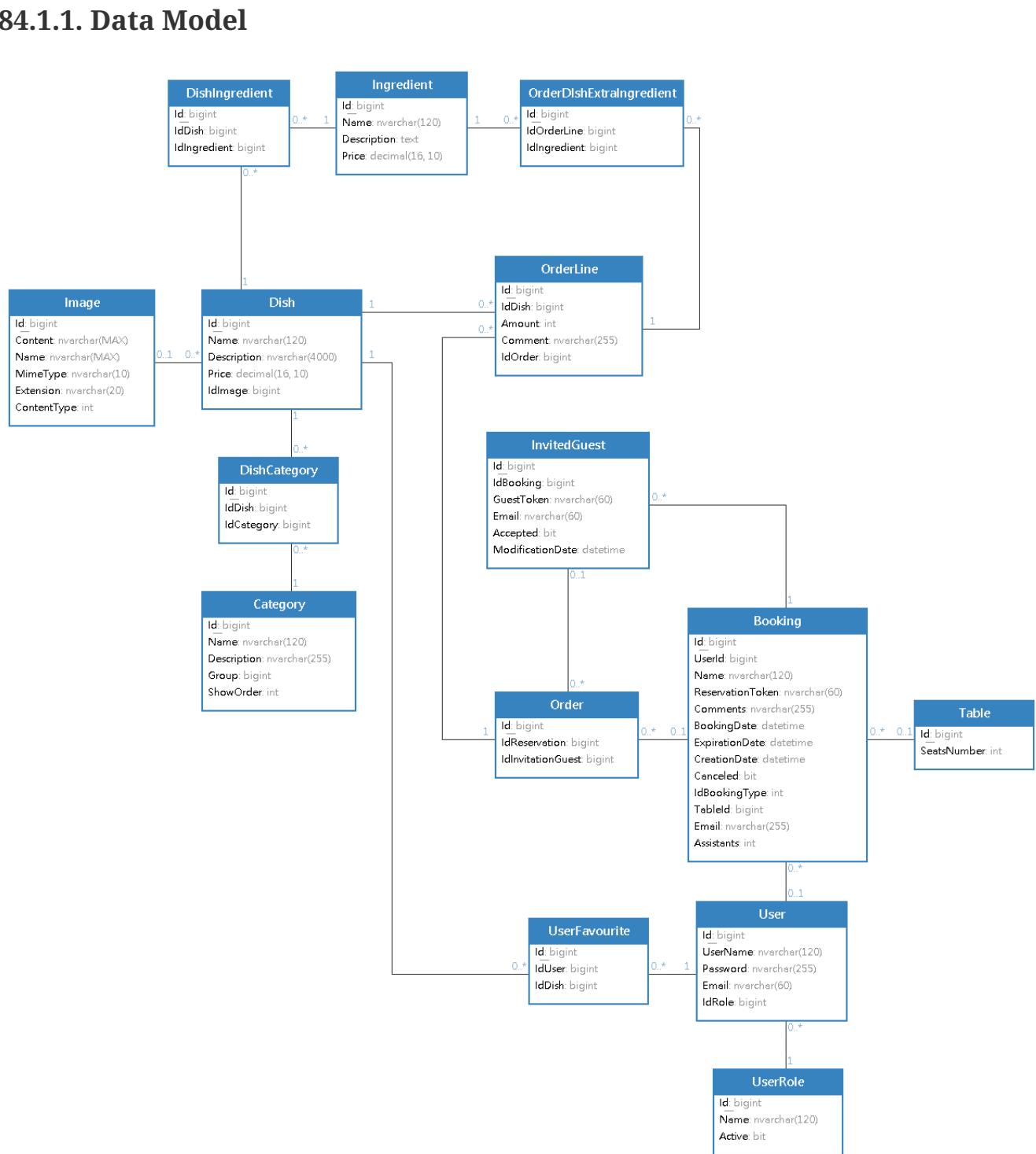
checklist:

talk about:

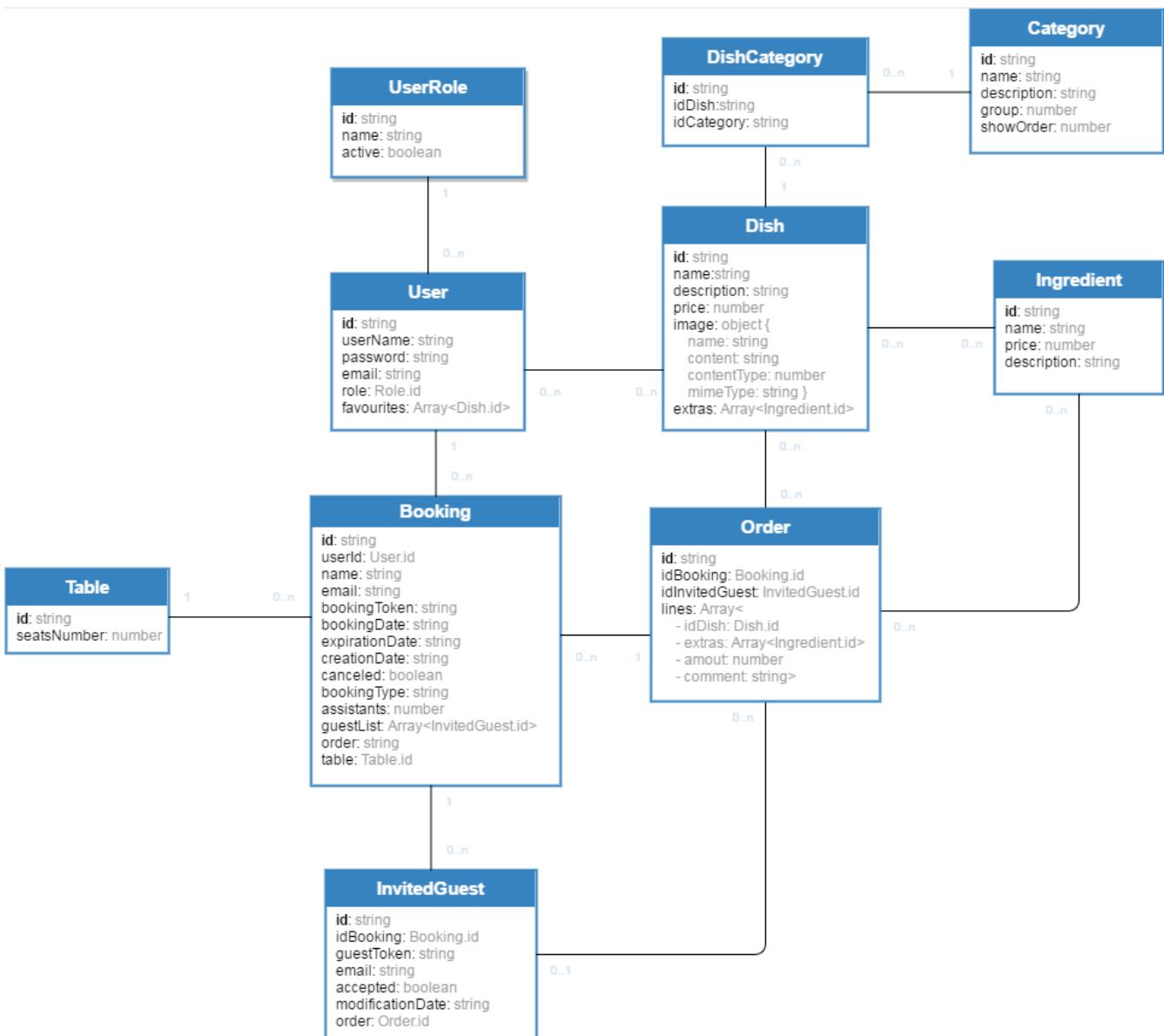
- who?
- what?
- why (purpose)
- why (objective)
- what happens outside the software
- what might go wrong
- any question or assumptions (write them down) , DoR should check that those sections are empty.
- is there any better solution?
- how (technical perspective)
- do a rough estimate
- check INVEST

84. Technical design

84.1. Data Model



84.1.2. NoSQL Data Model



84.2. Server Side

84.2.1. Java design

Introduction

The Java backend for My Thai Star application is going to be based on:

- **DEVON4J** as the Java framework
- **Devonfw** as the Development environment
- **Cobigen** as code generation tool

To know more details about the above technologies please visit the following documentation:

- [DEVON4J](#)
- [Devonfw](#)
- [Cobigen](#)

Basic architecture details

Following the DEVON4J conventions the Java My Thai Star backend is going to be developed dividing the application in *Components* and using a three layers architecture.

Project modules

Using the DEVON4J approach for the Java backend project we will have a structure of a *Maven* project formed by three projects

mtsj

- > api
- > core
- > server
- pom.xml

mtsj-api

- > src/main/java
- > JRE System Library [JavaSE-1.8]
- > Maven Dependencies
- > src
- > target
- pom.xml

mtsj-core

- > src/main/java
- > src/main/resources
- > src/test/java
- > src/test/resources
- > JRE System Library [JavaSE-1.8]
- > Maven Dependencies
- > src
- > target
- pom.xml

mtsj-server

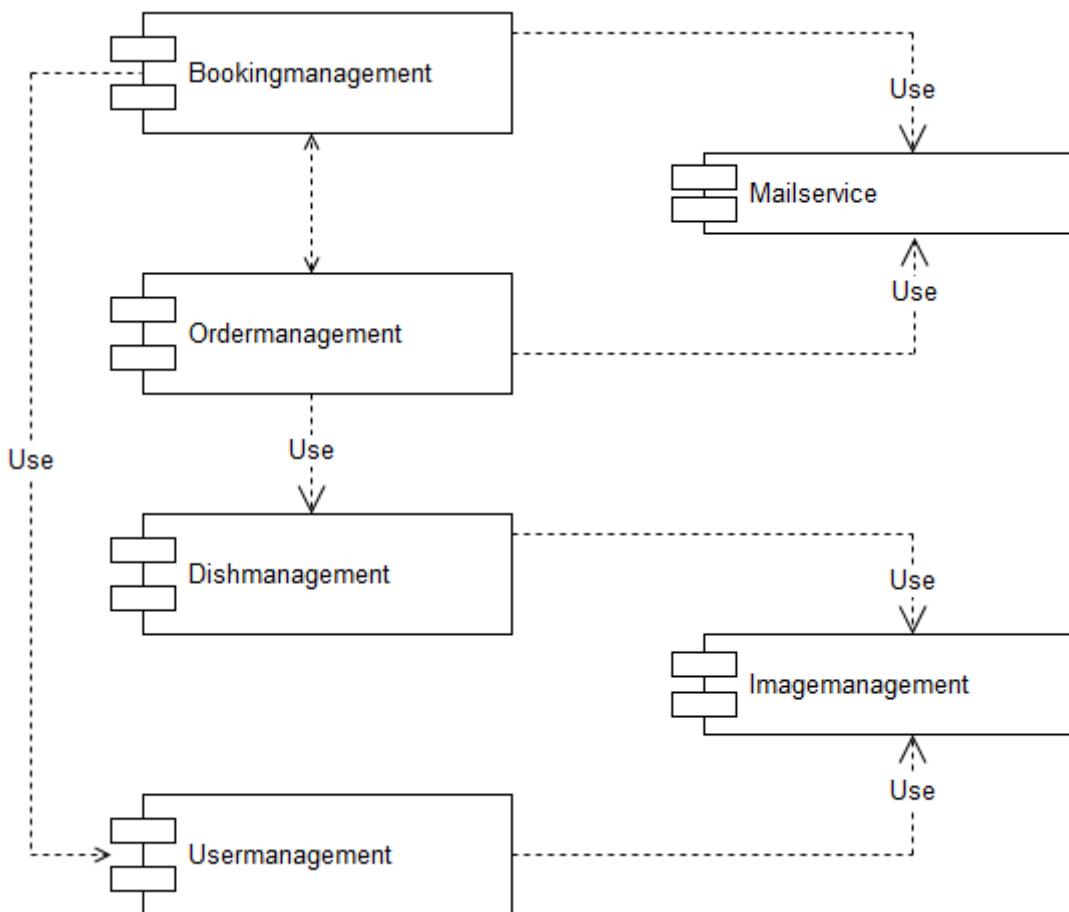
- > Deployment Descriptor: mtsj-server
- > JAX-WS Web Services
- > Java Resources
- > JavaScript Resources
- > Deployed Resources
- > src
- > target

- *api*: Stores all the REST interfaces and corresponding Request/Response objects.
- *core*: Stores all the logic and functionality of the application.
- *server*: Configures the packaging of the application.

We can automatically generate this project structure [using the DEVON4J Maven archetype](#)

Components

The application is going to be divided in different components to encapsulate the different domains of the application functionalities.



As *main components* we will find:

- *Bookingmanagement*: Manages the bookings part of the application. With this component the users (anonymous/logged in) can create new bookings or cancel an existing booking. The users with waiter role can see all scheduled bookings.
- *Ordermanagement*: This component handles the process to order dishes (related to bookings). A user (as a host or as a guest) can create orders (that contain dishes) or cancel an existing one. The users with waiter role can see all ordered orders.
- *Dishmanagement*: This component groups the logic related to the menu (dishes) view. Its main feature is to provide the client with the data of the available dishes but also can be used by other components (Ordermanagement) as a data provider in some processes.
- *Usermanagement*: Takes care of the User Profile management, allowing to create and update the data profiles.

As *common components* (that don't exactly represent an application's area but provide functionalities that can be used by the *main components*):

- *Imagenagement*: Manages the images of the application. In a first approach the *Dishmanagement* component and the *Usermanagement* component will have an image as part of its data. The *Imagenagement* component will expose the functionality to store and retrieve this kind of data.

- **Mailservice:** with this service we will provide the functionality for sending email notifications. This is a shared service between different app components such as *bookingmanagement* or *ordercomponent*.

Other components:

- Security (will manage the access to the *private* part of the application using a [jwt](#) implementation).
- Twitter integration: planned as a *Microservice* will provide the twitter integration needed for some specific functionalities of the application.

Layers

- [Service Layer](#): this layer will expose the REST api to exchange information with the client applications.
- [Logic Layer](#): the layer in charge of hosting the business logic of the application.
- [Data Access Layer](#): the layer to communicate with the data base.

This architecture is going to be reflected dividing each component of the application in different packages to match those three layers.

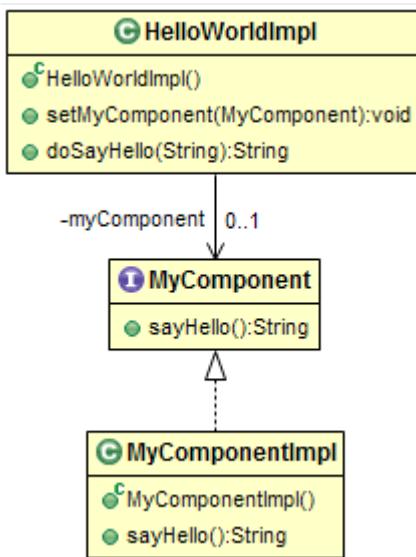
Component structure

Each one of the components defined previously are going to be structured using the *three-layers* architecture. In each case we will have a *service* package, a *logic* package and a *dataaccess* package to fit the layers definition.

```
▲ 📂 src/main/java
  ▲ 📂 io.oasp.application.mtsj
    ▲ 📂 dishmanagement
      ▷ 📂 common.api
      ▷ 📂 dataaccess
      ▷ 📂 logic
      ▷ 📂 service
```

Dependency injection

As it is explained in the [devonfw documentation](#) we are going to implement the *dependency injection* pattern basing our solution on *Spring* and the Java standards: *java.inject* (JSR330) combined with JSR250.



- Separation of API and implementation: Inside each layer we will separate the elements in different packages: *api* and *impl*. The *api* will store the *interface* with the methods definition and inside the *impl* we will store the class that implements the *interface*.

```

    ▲ 📁 dishmanagement
      ▷ 📁 common.api
      ▷ 📁 dataaccess
      ▷ 📁 logic
    ▲ 📁 service
      ▷ 📁 api.rest
        ▷ 📜 DishmanagementRestService.java
      ▷ 📁 impl.rest
        ▷ 📜 DishmanagementRestServiceImpl.java
  
```

- Usage of JSR330: The Java standard set of annotations for *dependency injection* (`@Named`, `@Inject`, `@PostConstruct`, `@PreDestroy`, etc.) provides us with all the needed annotations to define our beans and inject them.

```

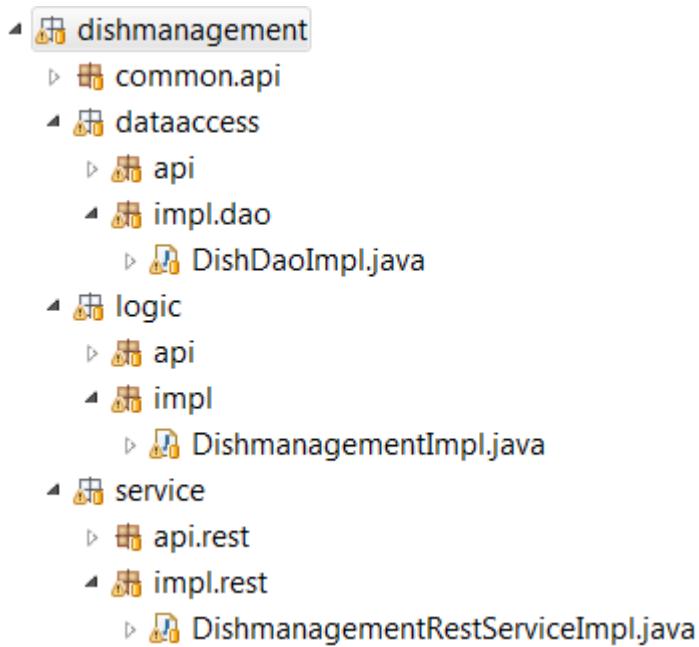
@Named
public class MyBeanImpl implements MyBean {
    @Inject
    private MyOtherBean myOtherBean;

    @PostConstruct
    public void init() {
        // initialization if required (otherwise omit this method)
    }

    @PreDestroy
    public void dispose() {
        // shutdown bean, free resources if required (otherwise omit this method)
    }
}
  
```

Layers communication

The connection between layers, to access to the functionalities of each one, will be solved using the *dependency injection* and the JSR330 annotations.



Connection Service - Logic

```

@Named("DishmanagementRestService")
public class DishmanagementRestServiceImpl implements DishmanagementRestService {

    @Inject
    private Dishmanagement dishmanagement;

    // use the 'this.dishmanagement' object to access to the functionalities of the
    // logic layer of the component

    ...
}
  
```

Connection Logic - Data Access

```

@Named
public class DishmanagementImpl extends AbstractComponentFacade implements
Dishmanagement {

    @Inject
    private DishDao dishDao;

    // use the 'this.dishDao' to access to the functionalities of the data access layer
    of the component
    ...

}

```

Service layer

The services layer will be solved using REST services with the [JAX-RS implementation](#).

To give service to the defined *User Stories* we will need to implement the following services:

- provide all available dishes.
- save a booking.
- save an order.
- provide a list of bookings (only for waiters) and allow filtering.
- provide a list of orders (only for waiters) and allow filtering.
- login service (see the *Security* section).
- provide the *current user* data (see the *Security* section)

Following the [naming conventions](#) proposed for *Devon4j* applications we will define the following *end points* for the listed services.

- (POST) `/mythaistar/services/rest/dishmanagement/v1/dish/search`.
- (POST) `/mythaistar/services/rest/bookingmanagement/v1/booking`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order`.
- (POST) `/mythaistar/services/rest/bookingmanagement/v1/booking/search`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order/search`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order/filter` (to filter with fields that does not belong to the Order entity).
- (POST) `/mythaistar/login`.
- (GET) `/mythaistar/services/rest/security/v1/currentuser/`.

You can find all the details for the services implementation in the [Swagger definition](#) included in the My Thai Star project on Github.

Service api

The *api.rest* package in the *service* layer of a *component* will store the definition of the service by a *Java interface*. In this definition of the service we will set-up the *endpoints* of the service, the type of data expected and returned, the *HTTP* method for each endpoint of the service and other configurations if needed.

```
@Path("/dishmanagement/v1")
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
public interface DishmanagementRestService {

    @GET
    @Path("/dish/{id}/")
    public DishCto getDish(@PathParam("id") long id);

    ...
}
```

Service impl

Once the service *api* is defined we need to implement it using the *Java interface* as reference. We will add the *service implementation* class to the *impl.rest* package and implement the *RestService interface*.

```
@Named("DishmanagementRestService")
public class DishmanagementRestServiceImpl implements DishmanagementRestService {

    @Inject
    private Dishmanagement dishmanagement;

    @Override
    public DishCto getDish(long id) {
        return this.dishmanagement.findDish(id);
    }

    ...
}
```



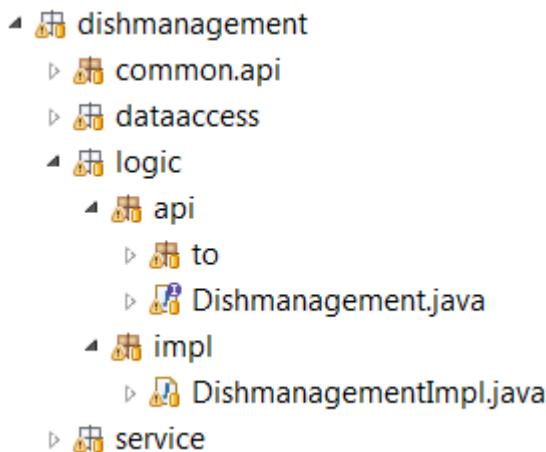
You can see the Devon4j conventions for REST services [here](#). And the My Thai Star services definition [here](#) as part of the [My Thai Star project](#).

Logic layer

In the *logic* layer we will locate all the *business logic* of the application. We will keep the same schema as we have done for the *service* layer, having an *api* package with the definition of the

methods and a *impl* package for the implementation.

Also, inside the *api* package, a *to* package will be the place to store the *transfer objects* needed to pass data through the layers of the component.



The logic *api* definition:

```

public interface Dishmanagement {
    DishCto findDish(Long id);
    ...
}
  
```

The logic *impl* class:

```

@Named
public class DishmanagementImpl extends AbstractComponentFacade implements
Dishmanagement {

    @Inject
    private DishDao dishDao;

    @Override
    public DishCto findDish(Long id) {

        return getBeanMapper().map(this.dishDao.findOne(id), DishCto.class);
    }

    ...
}
  
```

The *BeanMapper* will provide the needed transformations between *entity* and *transfer objects*.

Also, the *logic* layer is the place to add validation for *Authorization* based on *roles* as we will see later.

Data Access layer

The data-access layer is responsible for managing the connections to access and process data. The mapping between java objects to a relational database is done in *Devon4j* with the [spring-data-jpa](#).

As in the previous layers, the *data-access* layer will have both *api* and *impl* packages. However, in this case, the implementation will be slightly different. The *api* package will store the *component* main *entities* and, *inside the _api package*, another *api.repo* package will store the Repositories. The *repository* interface will extend `DefaultRepository` interface (located in `com.devonfw.module.jpa.dataaccess.api.data` package of [devon4j-starter-spring-data-jpa](#)).

For queries we will differentiate between *static queries* (that will be located in a mapped file) and *dynamic queries* (implemented with [QueryDsl](#)). You can find all the details about how to manage queries with *Devon4j* [here](#).

The default data base included in the project will be the [H2](#) instance included with the *Devon4j* projects.

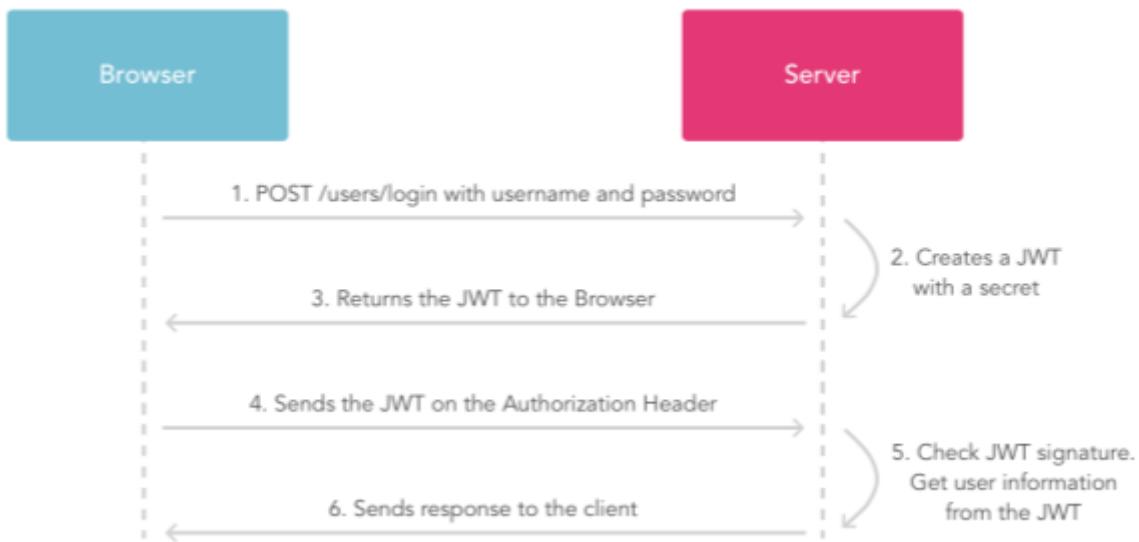
To get more details about *pagination*, *data base security*, *_concurrency control*, *inheritance* or how to solve the different *relationships* between entities visit the official [devon4j dataaccess documentation](#).

Security with Json Web Token

For the *Authentication* and *Authorization* the app will implement the [json web token](#) protocol.

Jwt basics

- A user will provide a username / password combination to our auth server.
- The auth server will try to identify the user and, if the credentials match, will issue a token.
- The user will send the token as the *Authorization* header to access resources on server protected by JWT Authentication.



Jwt implementation details

The *Json Web Token* pattern will be implemented based on the [Spring Security](#) framework that is provided by default in the *Devon4j* projects.

Authentication

Based on the *Spring Security* approach, we will implement a class extending *WebSecurityConfigurerAdapter* (*Devon4j* already provides the *BaseWebSecurityConfig* class) to define the security *entry point* and filters. Also, as *My Thai Star* is a mainly *public* application, we will define here the resources that won't be secured.

List of *unsecured resources*:

- */services/rest/dishmanagement/***: to allow anonymous users to see the dishes info in the *menu* section.
- */services/rest/ordermanagement/v1/order*: to allow anonymous users to save an order. They will need a *booking token* but they won't be authenticated to do this task.
- */services/rest/bookingmanagement/v1/booking*: to allow anonymous users to create a booking. Only a *booking token* is necessary to accomplish this task.
- */services/rest/bookingmanagement/v1/booking/cancel/***: to allow cancelling a booking from an email. Only the *booking token* is needed.
- */services/rest/bookingmanagement/v1/invitedguest/accept/***: to allow guests to accept an invite. Only a *guest token* is needed.
- */services/rest/bookingmanagement/v1/invitedguest/decline/***: to allow guests to reject an invite. Only a *guest token* is needed.

To configure the *login* we will set up the *HttpSecurity* object in the *configure* method of the class. We will define a *JWTLoginFilter* class that will handle the requests to the */login endpoint*.

```
http.[...].antMatchers(HttpMethod.POST, "/login").permitAll().[...].addFilterBefore
(new JWTLoginFilter("/login", authenticationManager()),
UsernamePasswordAuthenticationFilter.class);
```

In the same *HttpSecurity* object we will set up the filter for the rest of the requests, to check the presence of the JWT token in the header. First we will need to create a *JWTAuthenticationFilter* class extending the *GenericFilterBean* class. Then we can add the filter to the *HttpSecurity* object

```
http.[...].addFilterBefore(new JWTAuthenticationFilter(),
UsernamePasswordAuthenticationFilter.class);
```

Finally, as default users to start using the *My Thai Star* app we are going to define two profiles using the *inMemoryAuthentication* of the *Spring Security* framework. In the *configure(AuthenticationManagerBuilder auth)* method we will create:

- user: *waiter*
- password: *waiter*
- role: *Waiter*
- user: *user0*
- password: *password*
- role: *Customer*

```
auth.inMemoryAuthentication().withUser("waiter").password("waiter").roles("Waiter").an
d().withUser("user0").password("password").roles("Customer");
```

Token set up

Following the [official documentation](#) the implementation details for the MyThaiStar's jwt will be:

- *Secret*: Used as part of the signature of the token, acting as a private key. For the showcase purposes we will use simply "ThisIsASecret".
- *Token Prefix* schema: Bearer. The token will look like `Bearer <token>`
- *Header*: Authorization. The response header where the token will be included. Also, in the requests, when checking the token it will be expected to be in the same header.
- The *Authorization* header should be part of the `Access-Control-Expose-Headers` header to allow clients access to the *Authorization* header content (the token);
- The *claims* are the content of the *payload* of the token. The *claims* are statements about the user, so we will include the user info in this section.
 - *subject*: "sub". The username.
 - *issuer*: "iss". Who creates the token. We could use the *url* of our service but, as this is a showcase app, we simply will use "MyThaiStarApp"

- *expiration date*: "exp". Defines when the token expires.
- *creation date*: "iat". Defines when the token has been created.
- *scope*: "scope". Array of strings to store the user roles.
- Signature Algorithm: To encrypt the token we will use the default algorithm HS512.

An example of a token claims before encryption would be:

```
{sub=waiter, scope=[ROLE_Waiter], iss=MyThaiStarApp, exp=1496920280, iat=1496916680}
```

Current User request

To provide to the client with the current user data our application should expose a service to return the user details. In *Devon4j* applications the `/general/service/impl/rest/SecurityRestServiceImpl.java` class is ready to do that.

```
@Path("/security/v1")
@Named("SecurityRestService")
public class SecurityRestServiceImpl {

    @Produces(MediaType.APPLICATION_JSON)
    @GET
    @Path("/currentuser/")
    public UserDetailsClientTo getCurrentUserDetails(@Context HttpServletRequest
request) {

    }
}
```

we only will need to implement the `getCurrentUserDetails` method.

Authorization

We need to secure three services, that only should be accessible for users with role *Waiter*:

- (POST) `/mythaistar/services/rest/bookingmanagement/v1/booking/search`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order/search`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order/filter`.

As part of the token we are providing the user *Role*. So, when validating the token, we can obtain that same information and build a `UsernamePasswordAuthenticationToken` with username and the roles as collection of *Granted Authorities*.

Doing so, afterwards, in the implementation class of the *logic* layer we can set up the related methods with the *java security* '@RolesAllowed' annotation to block the access to the resource to users that does not match the expected roles.

```

@RolesAllowed(Roles.WAITER)
public PaginatedListTo<BookingEto> findBookings(BookingSearchCriteriaTo criteria) {
    return findBookings(criteria);
}

```

84.2.2. .NET design

TODO

84.2.3. Node.js design (deprecated)

Introduction

The Node.js backend for My Thai Star application is going to be based on:

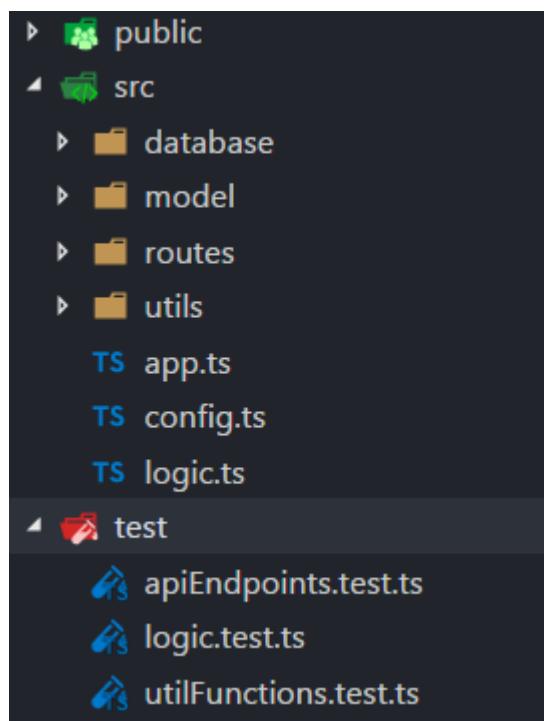
- **Express.js** as the web application framework
- **OASP4Fn** as data access layer framework
- **DynamoDB** as NoSQL Database

To know more details about the above technologies please visit the following documentation:

- [Express.js](#)
- [OASP4Fn](#)
- [DynamoDB](#)

Basic architecture details

This structure can be shown in the following example image:



- public - All files which be exposed on the server directly
- src
 - database folder - Folder with scripts to create/delete/seed the database
 - model - Folder with all data model
 - routes - Folder with all Express.js routers
 - utils - Folder with all utils like classes and functions
 - *app.ts* - File with Express.js declaration
 - *config.ts* - File with server configs
 - *logic.ts* - File with the business logic
- test - Folder with all tests

Layers

- Service Layer: this layer will expose the REST api to exchange information with the client applications.
- Logic Layer: the layer in charge of hosting the business logic of the application.
- Data Access Layer: the layer to communicate with the data base.

Service layer

The services layer will be solved using REST services with [Express.js](#)

To give service to the defined *User Stories* we will need to implement the following services:

- provide all available dishes.
- save a booking.
- save an order.
- provide a list of bookings (only for waiters) and allow filtering.
- provide a list of orders (only for waiters) and allow filtering.
- login service (see the *Security* section).
- provide the *current user* data (see the *Security* section)

In order to be compatible with the other backend implementations, we must follow the [naming conventions](#) proposed for *Devon4j* applications. We will define the following *end points* for the listed services.

- (POST) `/mythaistar/services/rest/dishmanagement/v1/dish/search`.
- (POST) `/mythaistar/services/rest/bookingmanagement/v1/booking`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order`.
- (POST) `/mythaistar/services/rest/bookingmanagement/v1/booking/search`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order/search`.

- (POST) `/mythaistar/services/rest/ordermanagement/v1/order/filter` (to filter with fields that does not belong to the Order entity).
- (POST) `/mythaistar/login`.
- (GET) `/mythaistar/services/rest/security/v1/currentuser/`.

You can find all the details for the services implementation in the [Swagger definition](#) included in the My Thai Star project on Github.

To treat these services separately, the following routers were created:

- bookingmanagement: will answer all requests with the prefix `/mythaistar/services/rest/bookingmanagement/v1`
- dishmanagement: will answer all requests with the prefix `/mythaistar/services/rest/dishmanagement/v1`
- ordermanagement: will answer all requests with the prefix `/mythaistar/services/rest/ordermanagement/v1`

These routers will define the behavior for each service and use the logical layer.

An example of service definition:

```
router.post('/booking/search', (req: types.CustomRequest, res: Response) => {
  try {
    // body content must be SearchCriteria
    if (!types.isSearchCriteria(req.body)) {
      throw {code: 400, message: 'No booking token given'};
    }

    // use the searchBooking method defined at business logic
    business.searchBooking(req.body, (err: types.Error | null, bookingEntity: types.PaginatedList) => {
      if (err) {
        res.status(err.code || 500).json(err.message);
      } else {
        res.json(bookingEntity);
      }
    });
  } catch (err) {
    res.status(err.code || 500).json({ message: err.message });
  }
});
```

Logic layer and Data access layer

In the *logic* layer we will locate all the *business logic* of the application. It will be located in the file `logic.ts`. If in this layer we need to get access to the data, we make use of data access layer directly, in this case using OASP4fn with the DynamoDB adapter.

Example:

```

export async function cancelOrder(orderId: string, callback: (err: types.Error | null) => void) {
    let order: dbtypes.Order;

    try {
        // Data access
        order = await oasp4fn.table('Order', orderId).promise() as dbtypes.Order;
    }
    [...]
}

```

We could define the data access layer separately, but oasp4fn allows us to do this in a simple and clear way. So, we decided to not separate the access layer to the logic business.

Security with Json Web Token

For the *Authentication* and *Authorization* the app will implement the [json web token](#) protocol.

Jwt basics

Refer to [Jwt basics](#) for more information.

Jwt implementation details

The *Json Web Token* pattern will be implemented based on the [JSON web token](#) library available on npm.

Authentication

Based on the *JSON web token* approach, we will implement a class *Authentication* to define the security *entry point* and filters. Also, as *My Thai Star* is a mainly *public* application, we will define here the resources that won't be secured.

List of *unsecured resources*:

- */services/rest/dishmanagement/***: to allow anonymous users to see the dishes info in the *menu* section.
- */services/rest/ordermanagement/v1/order*: to allow anonymous users to save an order. They will need a *booking token* but they won't be authenticated to do this task.
- */services/rest/bookingmanagement/v1/booking*: to allow anonymous users to create a booking. Only a *booking token* is necessary to accomplish this task.
- */services/rest/bookingmanagement/v1/booking/cancel/***: to allow cancelling a booking from an email. Only the *booking token* is needed.
- */services/rest/bookingmanagement/v1/invitedguest/accept/***: to allow guests to accept an invite. Only a *guest token* is needed.
- */services/rest/bookingmanagement/v1/invitedguest/decline/***: to allow guests to reject an invite. Only a *guest token* is needed.

To configure the *login* we will create an instance of *Authentication* in the app file and then we will use the method *auth* for handle the requests to the /login endpoint.

```
app.post('/mythaistar/login', auth.auth);
```

To verify the presence of the *Authorization token* in the headers, we will register in the express the *Authentication.registerAuthentication* middleware. This middleware will check if the token is correct, if so, it will place the user in the request and continue to process it. If the token is not correct it will continue processing the request normally.

```
app.use(auth.registerAuthentication);
```

Finally, we have two default users created in the database:

- user: *waiter*
- password: *waiter*
- role: *WAITER*
- user: *user0*
- password: *password*
- role: *CUSTOMER*

Token set up

Following the [official documentation](#) the implementation details for the MyThaiStar's jwt will be:

- *Secret*: Used as part of the signature of the token, acting as a private key. It can be modified at config.ts file.
- *Token Prefix* schema: Bearer. The token will look like `Bearer <token>`
- *Header*: Authorization. The response header where the token will be included. Also, in the requests, when checking the token it will be expected to be in the same header.
- The *Authorization* header should be part of the `Access-Control-Expose-Headers` header to allow clients access to the *Authorization* header content (the token);
- Signature Algorithm: To encrypt the token we will use the default algorithm HS512.

Current User request

To provide to the client with the current user data our application should expose a service to return the user details. In this case the *Authentication* has a method called *getCurrentUser* which will return the user data. We only need register it at express.

```
app.get('/mythaistar/services/rest/security/v1/currentuser', auth.getCurrentUser);
```

Authorization

We need to secure three services, that only should be accessible for users with role *Waiter*:

- (POST) `/mythaistar/services/rest/ordermanagement/v1/order/filter`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order/search`.
- (POST) `/mythaistar/services/rest/bookingmanagement/v1/booking/search`.

To ensure this, the *Authorization* class has the *securizedEndpoint* method that guarantees access based on the role. This method can be used as middleware in secure services. As the role is included in the token, once validated we will have this information in the request and the middleware can guarantee access or return a 403 error.

```
app.use('/mythaistar/services/rest/ordermanagement/v1/order/filter', auth
    .securizedEndpoint('WAITER'));
app.use('/mythaistar/services/rest/ordermanagement/v1/order/search', auth
    .securizedEndpoint('WAITER'));
app.use('/mythaistar/services/rest/bookingmanagement/v1/booking/search', auth
    .securizedEndpoint('WAITER'));
```

84.2.4. Serverless design (deprecated)

Introduction

The Node.js backend for My Thai Star application is going to be based on:

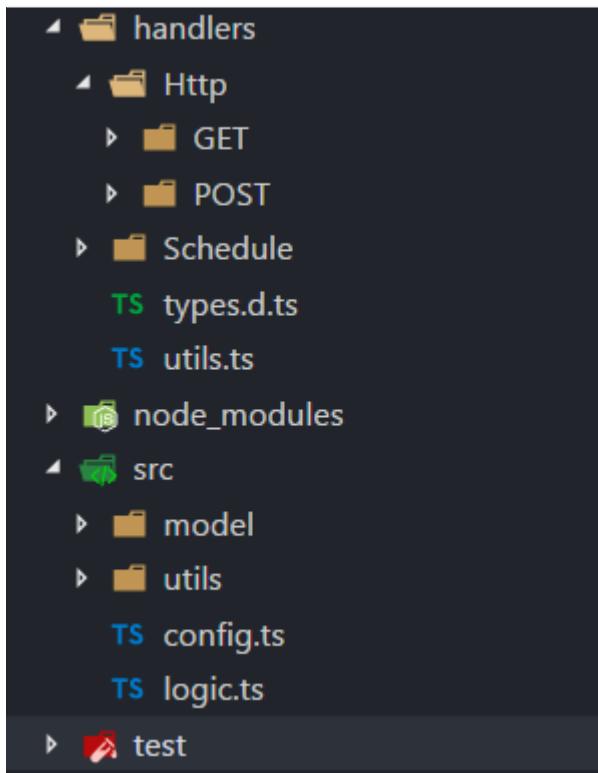
- **Serverless** as serverless framework
- **OASP4Fn** as data access layer framework
- **DynamoDB** as NoSQL Database

To know more details about the above technologies please visit the following documentation:

- [Serverless](#)
- [OASP4Fn](#)
- [DynamoDB](#)

Basic architecture details

This structure can be shown in the following example image:



- handlers - All function handlers following oasp4fn structure
- src
 - model - Folder with all data model
 - utils - Folder with all utils like classes and functions
 - config.ts - File with server configs
 - logic.ts - File with the business logic
- test - Folder with all tests

Layers

- Service Layer: this layer will expose the REST api to exchange information with the client applications.
- Logic Layer: the layer in charge of hosting the business logic of the application.
- Data Access Layer: the layer to communicate with the data base.

Service layer

The services layer will be solved using REST services with [Serverless](#)

To give service to the defined *User Stories* we will need to implement the following services:

- provide all available dishes.
- save a booking.
- save an order.
- provide a list of bookings (only for waiters) and allow filtering.

- provide a list of orders (only for waiters) and allow filtering.
- login service (see the *Security* section).
- provide the *current user* data (see the *Security* section)

In order to be compatible with the other backend implementations, we must follow the [naming conventions](#) proposed for *Devon4j* applications. We will define the following *end points* for the listed services.

- (POST) `/mythaistar/services/rest/dishmanagement/v1/dish/search`.
- (POST) `/mythaistar/services/rest/bookingmanagement/v1/booking`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order`.
- (POST) `/mythaistar/services/rest/bookingmanagement/v1/booking/search`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order/search`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order/filter` (to filter with fields that does not belong to the Order entity).
- (POST) `/mythaistar/login`.
- (GET) `/mythaistar/services/rest/security/v1/currentuser/`.

You can find all the details for the services implementation in the [Swagger definition](#) included in the My Thai Star project on Github.

To treat these http services, we must define the handlers following the [oasp4fn](#) convention:

- (handlers/Http/POST/dish-search-handler)
`/mythaistar/services/rest/dishmanagement/v1/dish/search`.
- (handlers/Http/POST/booking-handler)
`/mythaistar/services/rest/bookingmanagement/v1/booking`.
- (handlers/Http/POST/order-handler) `/mythaistar/services/rest/ordermanagement/v1/order`.
- (handlers/Http/POST/booking-search-handler)
`/mythaistar/services/rest/bookingmanagement/v1/booking/search`.
- (handlers/Http/POST/order-search-handler)
`/mythaistar/services/rest/ordermanagement/v1/order/search`.
- (handlers/Http/POST/order-filter-handler)
`/mythaistar/services/rest/ordermanagement/v1/order/filter` (to filter with fields that does not belong to the Order entity).
- (handlers/Http/POST/login-handler) `/mythaistar/login`.
- (handlers/Http/GET/current-user-handler) `/mythaistar/services/rest/security/v1/currentuser/`.

These handlers will define the behavior for each service and use the logical layer.

An example of handler definition:

```

oasp4fn.config({ path: '/mythaistar/services/rest/bookingmanagement/v1/booking/search'
});
export async function bookingSearch(event: HttpEvent, context: Context, callback: Function) {
    try {
        const search = <types.SearchCriteria>event.body;
        const authToken = event.headers.Authorization;
        // falta lo que viene siendo comprobar el token y eso

        auth.decode(authToken, (err, decoded) => {
            if (err || decoded.role !== 'WAITER') {
                throw { code: 403, message: 'Forbidden' };
            }

            // body content must be SearchCriteria
            if (!types.isSearchCriteria(search)) {
                throw { code: 400, message: 'No booking token given' };
            }

            business.searchBooking(search, (err: types.Error | null, bookingEntity: types.PaginatedList) => {
                if (err) {
                    callback(new Error(`[${err.code} || 500] ${err.message}`));
                } else {
                    callback(null, bookingEntity);
                }
            });
        });
    } catch (err) {
        callback(new Error(`[${err.code} || 500] ${err.message}`));
    }
}

```

The default integration for a handler is *lambda*. See [oasp documentation](#) for more information about default values and how to change it.



If you change the integration to lambda-proxy, you must take care that in this case the data will not be parsed. You must do JSON.parse explicitly

After defining all the handlers, we must execute the *fun* command, which will generate the files `serverless.yml` and `webpack.config.js`.

Logic layer and Data access layer

[See in nodejs section](#)

Security with Json Web Token

For the *Authentication* and *Authorization* the app will implement the [json web token](#) protocol.

Jwt basics

Refer to [Jwt basics](#) for more information.

Jwt implementation details

The *Json Web Token* pattern will be implemented based on the [JSON web token](#) library available on npm.

Authentication

Based on the *JSON web token* approach, we will implement two methods in order to verify and user + generate the token and decode the token + return the user data. Also, as *My Thai Star* is a mainly *public* application, we will define here the resources that won't be secured.

List of *unsecured* resources:

- */services/rest/dishmanagement/***: to allow anonymous users to see the dishes info in the *menu* section.
- */services/rest/ordermanagement/v1/order*: to allow anonymous users to save an order. They will need a *booking token* but they won't be authenticated to do this task.
- */services/rest/bookingmanagement/v1/booking*: to allow anonymous users to create a booking. Only a *booking token* is necessary to accomplish this task.
- */services/rest/bookingmanagement/v1/booking/cancel/***: to allow cancelling a booking from an email. Only the *booking token* is needed.
- */services/rest/bookingmanagement/v1/invitedguest/accept/***: to allow guests to accept an invite. Only a *guest token* is needed.
- */services/rest/bookingmanagement/v1/invitedguest/decline/***: to allow guests to reject an invite. Only a *guest token* is needed.

To configure the *login* we will create a handler called *login* and then we will use the method *code* to verify the user and generate the token.

```
app.post(oasp4fn.config({ integration: 'lambda-proxy', path: '/mythaistar/login' });
export async function login(event: HttpEvent, context: Context, callback: Function) {
  .
  .
  .
}
```

We have two default users created in the database:

- user: *waiter*
- password: *waiter*
- role: *WAITER*

- user: *user0*
- password: *password*
- role: *CUSTOMER*

Token set up

[See in nodejs section](#)

Current User request

To provide the client with the current user data our application should expose a service to return the user details. In order to do this, we must define a handler called current-user-handler. This handler must decode the *Authorization token* and return the user data.

```
oasp4fn.config({
  path: '/mythaistar/services/rest/security/v1/currentuser',
});
export async function currentUser(event: HttpEvent, context: Context, callback: Function) {
  let authToken = event.headers.Authorization;
  try {
    auth.decode(authToken, (err: any, decoded?: any) => {
      if (err) {
        callback(new Error(`[403] Forbidden`));
      } else {
        callback(null, decoded);
      }
    });
  } catch (err) {
    callback(new Error(`[${err.code} || 500] ${err.message}`));
  }
}
```

Authorization

We need to secure three services, that only should be accessible for users with role *Waiter*:

- (POST) */mythaistar/services/rest/bookingmanagement/v1/booking/search*.
- (POST) */mythaistar/services/rest/ordermanagement/v1/order/search*.
- (POST) */mythaistar/services/rest/ordermanagement/v1/order/filter*.

To ensure this, we must decode the *Authorization token* and check the result. As the role is included in the token, once validated we will have this information and can guarantee access or return a 403 error.

```

oasp4fn.config({ path: '/mythaistar/services/rest/bookingmanagement/v1/booking/search'
});
export async function bookingSearch(event: HttpEvent, context: Context, callback: Function) {
    const authToken = event.headers.Authorization;
    auth.decode(authToken, (err, decoded) => {
        try {
            if (err || decoded.role !== 'WAITER') {
                throw { code: 403, message: 'Forbidden' };
            }
        }
        [...]
    } catch (err) {
        callback(new Error(`[${err.code} || 500] ${err.message}`));
    }
});
}

```

84.2.5. GraphQL design

TODO

84.3. Client Side

84.3.1. Angular design

Introduction

MyThaiStar client side has been built using latest frameworks, component libraries and designs:

Angular 4 as main front-end Framework. <https://angular.io/>

Angular/CLI 1.0.5 as Angular tool helper. <https://github.com/angular/angular-cli>

Covalent Teradata 1.0.0-beta4 as Angular native component library based on Material Design.
<https://teradata.github.io/covalent/#/>

Angular/Material2 1.0.0-beta5 used by Covalent Teradata. <https://github.com/angular/material2>

Note: this dependencies are evolving at this moment and if it is possible, we are updating it on the project.

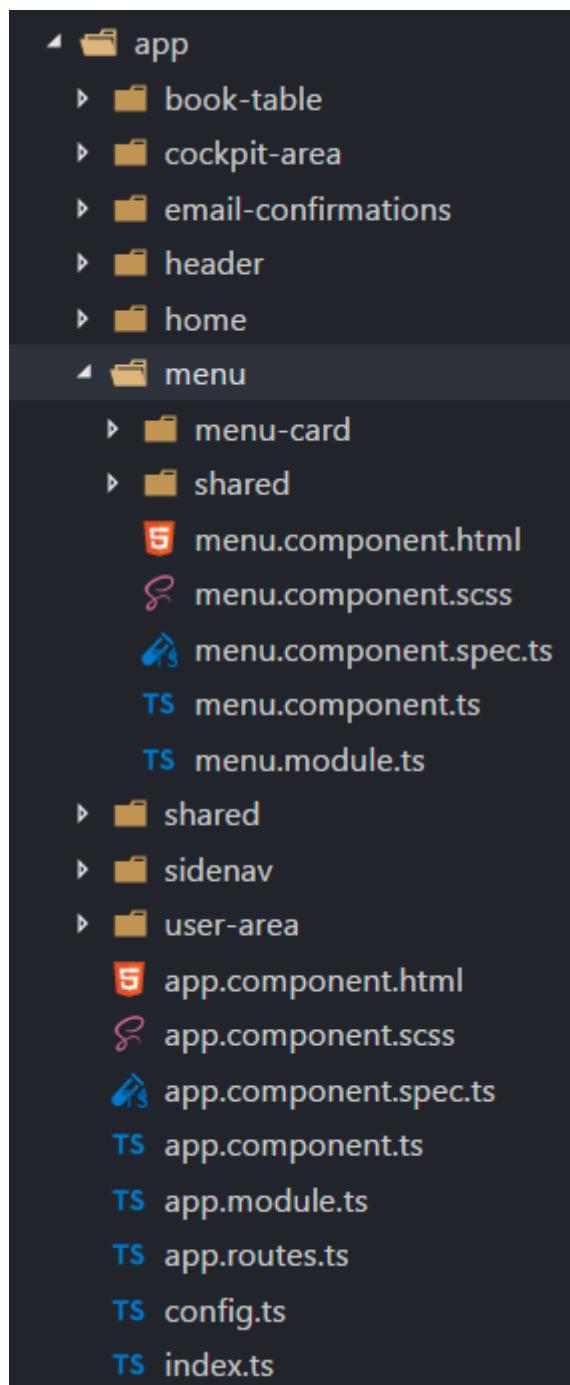
Basic project structure

The project is using the basic project seed that Angular/CLI provides with “ng new <project name>”. Then the app folder has been organized as Angular recommends and goes as follows:

- app

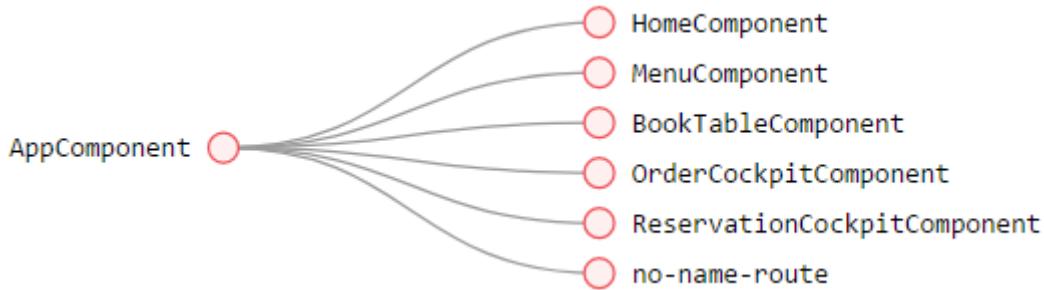
- components
 - sub-components
 - shared
 - component files
- main app component
- assets folder
- environments folder
- rest of angular files

This structure can be shown in the following example image:



Main Views and components

List of components that serve as a main view to navigate or components developed to make atomically a group of functionalities which given their nature, can be highly reusable through the app.



Note: no-name-route corresponds to whatever URL the user introduced and does not exist, it redirects to HomeComponent.

Public area

AppComponent

Contains the components that are on top of all views, including:

Order sidenav

Sidenav where selected orders are displayed with their total price and some comments.

Navigation sidenav (only for mobile)

This sidenav proposal is to let user navigate through the app when the screen is too small to show the navigation buttons on the header.

Header

It contains the title, and some other basic functions regarding open and close sidenavs.

Footer (only for desktop)

At the end of the page that shows only when open on desktop.

HomeComponent

Main view that shows up when the app initializes.

MenuComponent

View where the users can view, filter and select the dishes (with their extras) they want to order it contains a component to each menu entry:

Menu-card

This component composes all the data of a dish in a card. Component made to display indeterminate number of dishes easily.

BookTableComponent

View to make book a table in a given data with a given number of assistants or create a reservation with a number of invitations via email.

Book-table-dialog

Dialog which opens as a result of fulfilling the booking form, it displays all the data of the booking attempt, if everything is correct, the user can send the information or cancel if something is wrong.

Invitation-dialog

Dialog which opens as a result of fulfilling the invitation form, it displays all the data of the booking with friends attempt, if everything is correct, the user can send the information or cancel if something is wrong.

UserArea

Group of dialogs with the proposal of giving some functionalities to the user, as login, register, change password or connect with Twitter.

Login-dialog

Dialog with a tab to navigate between login and register.

Password-dialog

Functionality reserved to already logged users, in this dialog the user can change freely their password.

Twitter-dialog

Dialog designed specifically to connect your user account with Twitter.

Waiter cockpit area

Restricted area to workers of the restaurant, here we can see all information about booked tables with the selected orders and the reservations with all the guests and their acceptance or decline of the event.

OrderCockpitComponent

Data table with all the booked tables and a filter to search them, to show more info about that table you can click on it and open a dialog.

Order-dialog

Complete display of data regarding the selected table and its orders.

ReservationCockpitComponent

Data table with all the reservations and a filter to search them, to show more info about that table you can click on it and open a dialog.

Reservation-dialog

Complete display of data regarding the selected table and its guests.

Email Management

As the application send emails to both guests and hosts, we choose an approach based on URL's where the email contain a button with an URL to a service in the app and a token, front-end read that token and depending on the URL, will redirect to one service or another. For example:

```
http://localhost:4200/booking/cancel/CB_20170605_8fb5bc4c84a1c5049da1f6beb1968afc
```

This URL will tell the app that is a cancellation of a booking with the token `CB_20170605_8fb5bc4c84a1c5049da1f6beb1968afc`. The app will process this information, send it to back-end with the correct headers, show the confirmation of the event and redirect to home page.

The main cases at the moment are:

Accept Invite

A guest accept an invitation sent by a host. It will receive another email to decline if it change its mind later on.

Reject Invite

A guest decline the invitation.

Cancel Reservation

A host cancel the reservation, everybody that has accepted or not already answered will receive an email notifying this event is canceled. Also all the orders related to this reservations will be removed.

Cancel Orders

When you have a reservation, you will be assigned to a token, with that token you can save your order in the restaurant. When sent, you will receive an email confirming the order and the possibility to remove it.

Services and directives

Services are where all the main logic between components of that view should be. This includes calling a remote server, composing objects, calculate prices, etc.

Directives are a single functionality that are related to a component.

As it can be seen in the basic structure, every view that has a minimum of logic or need to call a server has its own service located in the shared folder.

Also, services and directives can be created to compose a reusable piece of code that will be reused in some parts of the code:

Price-calculator-service

This service located in the shared folder of sidenav contains the basic logic to calculate the price of a single order (with all the possibilities) and to calculate the price of a full list of orders for a table. As this is used in the sidenav and in the waiter cockpit, it has been exported as a service to be imported where needed and easily testable.

Authentication

Authentication services serves as a validator of roles and login and, at the same time, stores the basic data regarding security and authentication.

Main task of this services is to provide visibility at app level of the current user information:

- Check if the user is logged or not.
- Check the permissions of the current user.
- Store the username and the JWT token.

SnackService

Service created to serve as a factory of Angular Material Snackbars, which are used commonly through the app. This service accepts some parameters to customize the snackBar and opens it with this parameters.

WindowService

For responsiveness reasons, the dialogs have to accept a width parameter to adjust to screen width and this information is given by Window object, as it is a good practice to have it in an isolated service, which also calculates the width percentage to apply on the dialogs.

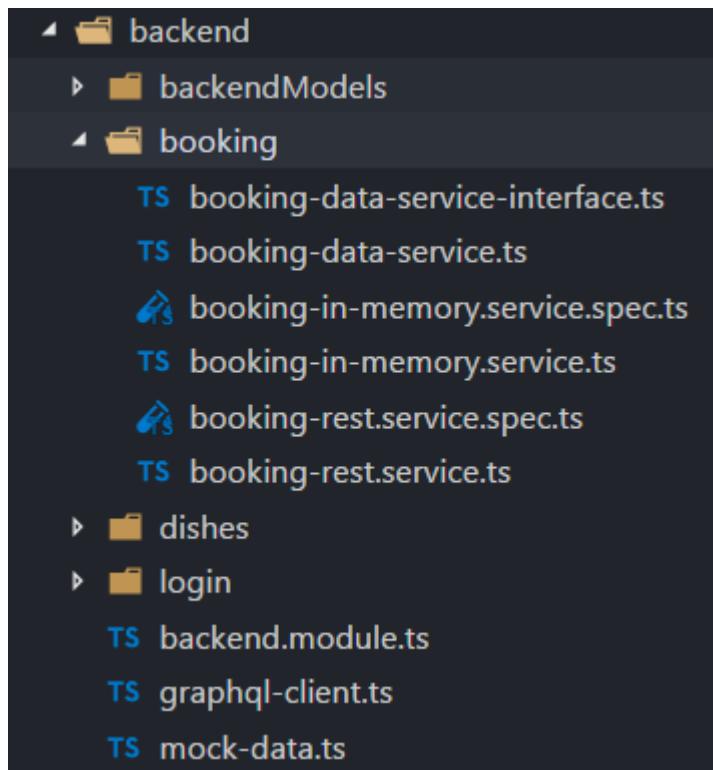
Equal-validator-directive

This directive located in the shared folder of userArea is used in 2 fields to make sure they have the same value. This directive is used in confirm password fields in register and change password.

Mock Backend

To develop meanwhile a real back-end is being developed let us to make a more realistic

application and to make easier the adaptation when the backend is able to be connected and called. Its structure is as following:



Contains the three main groups of functionalities in the application. Every group is composed by:

- An **interface** with all the methods to implement.
- A **service** that implements that interface, the main task of this service is to choose between real backend and mock backend depending on an environment variable.
- **Mock backend service** which implements all the methods declared in the interface using mock data stored in a local file and mainly uses Lodash to operate the arrays.
- **Real backend service** works as Mock backend but in this case the methods call for server rest services through http.

Booking

The booking group of functionalities manages the calls to reserve a table with a given time and assistants or with guests, get reservations filtered, accept or decline invitations or cancel the reservation.

Orders

Management of the orders, including saving, filtering and cancel an order.

Dishes

The dishes group of functionalities manages the calls to get and filter dishes.

Login

Login manages the userArea logic: login, register and change password.

Security

My Thai Star security is composed by two main security services:

Auth-guard

Front-end security approach, this service implements an interface called CanActivate that comes from angular/router module. CanActivate interface forces you to implement a canActivate() function which returns a Boolean. This service checks with the AuthService stored data if the user is logged and if he has enough permission to access the waiter cockpit. This prevents that a forbidden user could access to waiter cockpit just by editing the URL in the browser.

JWT

JSON Web Token consists of a token that is generated by the server when the user logs in. Once provided, the token has to be included in an Authentication header on every Http call to the rest service, otherwise the call will be forbidden. JWT also has an expiration date and a role checking, so if a user has not enough permissions or keeps logged for a long certain amount of time that exceeds this expiration date, the next time he calls for a service call, the server will return an error and forbid the call. You can log again to restore the token.

HttpClient

To implement this Authorization header management, an HttpClient service has been implemented. This service works as an envelope of Http, providing some more functionalities, like a header management and an automatically management of a server token error in case the JWT has expired, corrupted or not permitted.

84.3.2. Xamarin design

TODO

85. Security

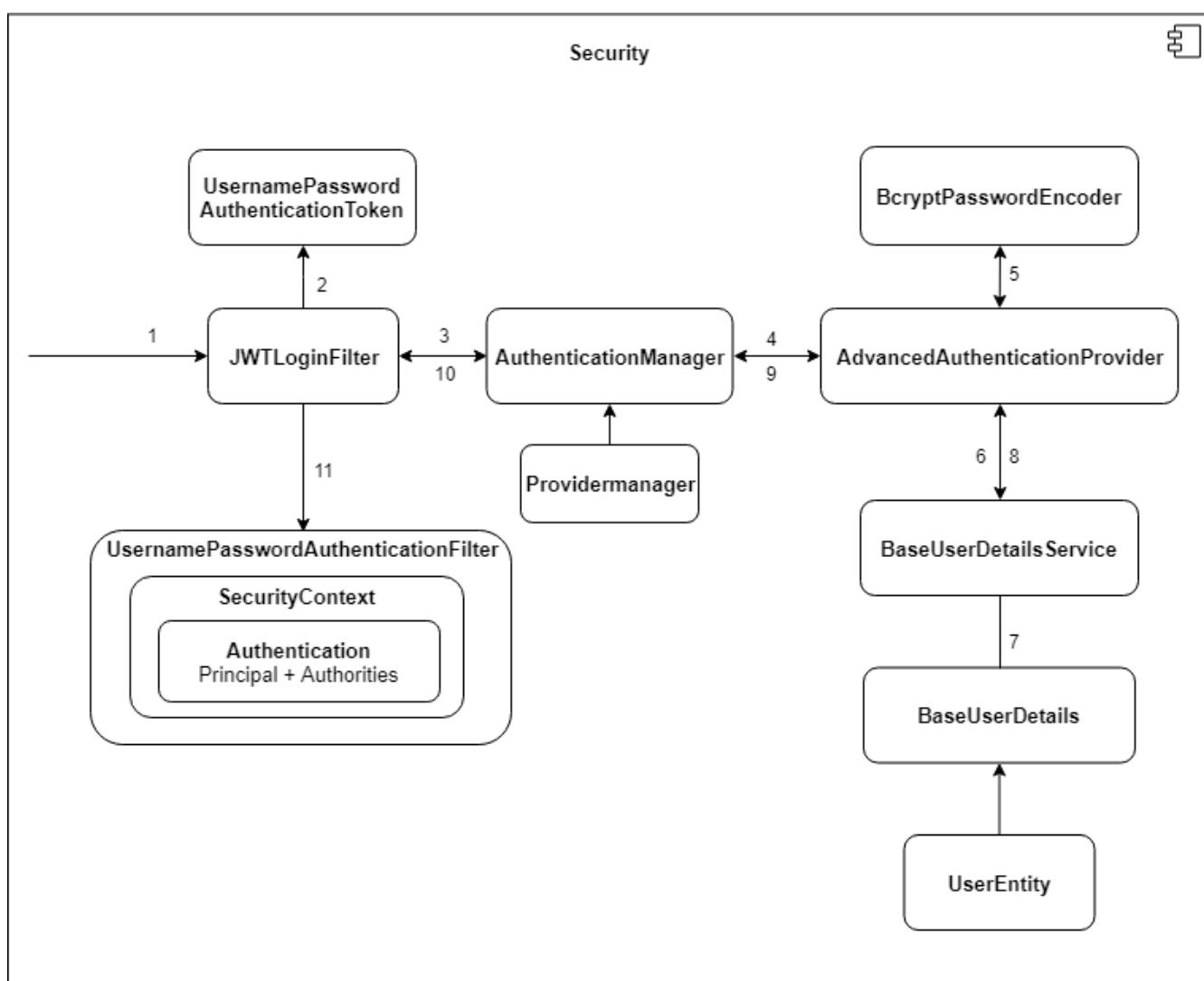
85.1. Two-Factor Authentication

Two-factor Authentication (2FA) provides an additional level of security to your account. Once enabled, in addition to supplying your username and password to login, you'll be prompted for a code generated by your Google authenticator. For example, a password manager on one of your devices.

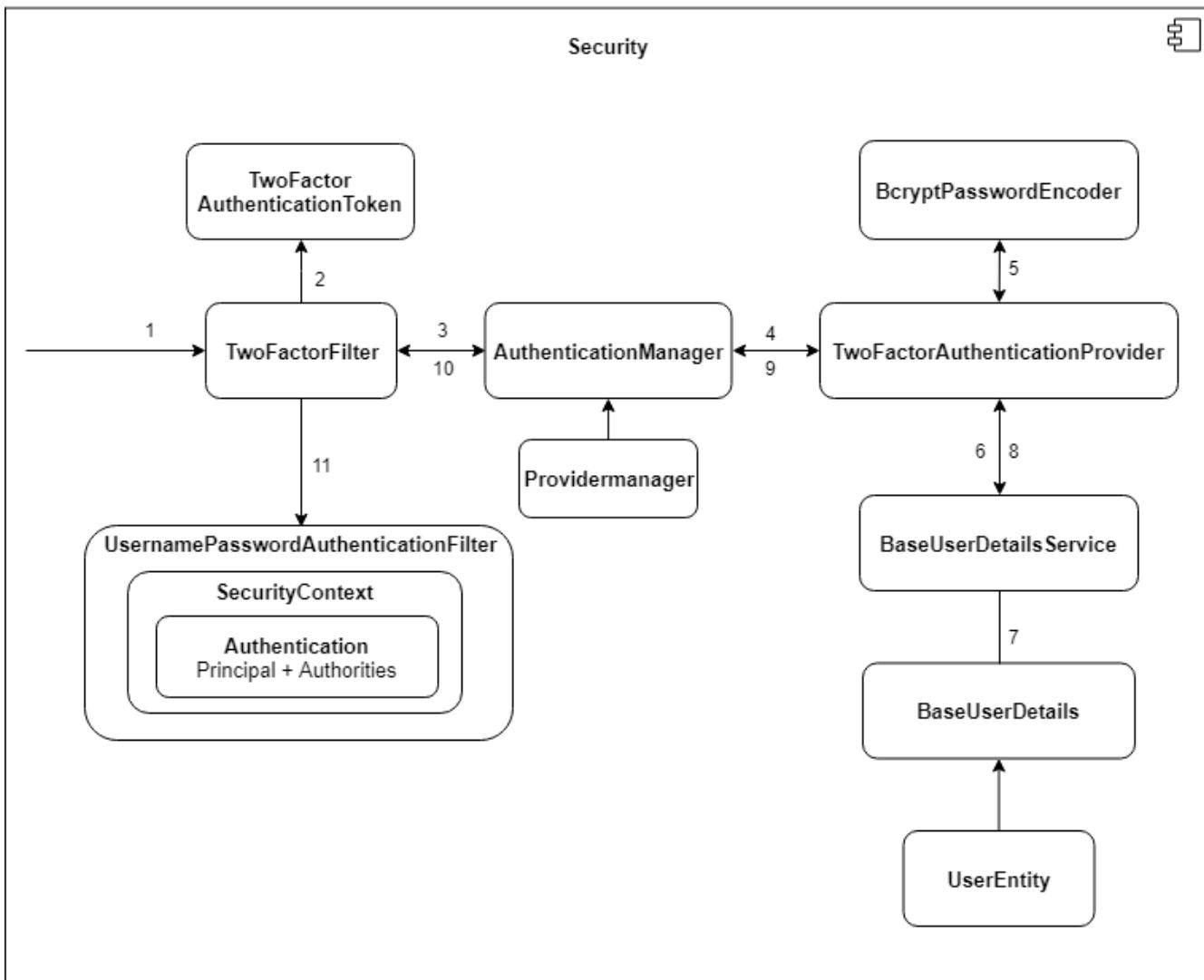
By enabling 2FA, to log into your account an additional one-time password is required what requires access to your paired device. This massively increases the barrier for an attacker to break into your account.

Backend mechanism

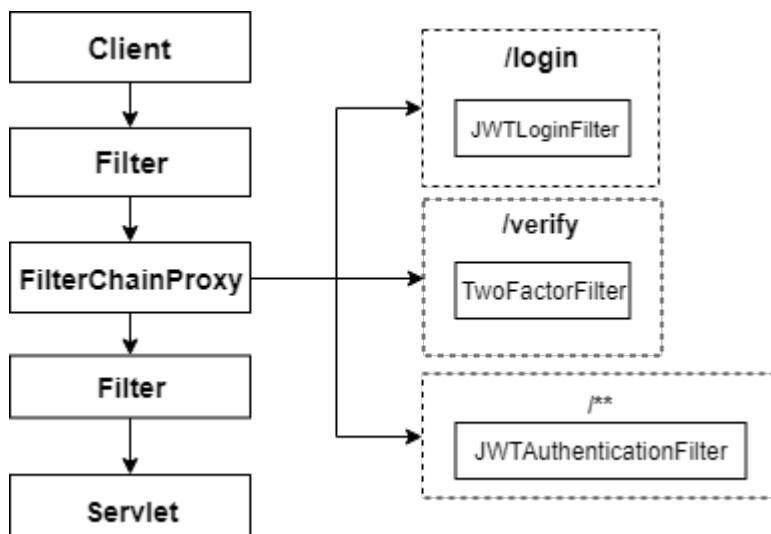
In the backend, we utilize Spring Security for any authentication. Following the arrows, one can see all processes regarding authentication. The main idea is to check all credentials depending on their 2FA status and then either grant access to the specific user or deny access. This picture illustrates a normal authentication with username and password.



When dealing with 2FA, another provider and filter is handling the request from **/verify**



Here you can observe which filter will be used. **JWTAuthenticationFilter** does intercept any request, which enforces being authenticated via JWT



Whenever the secret or qr code gets transferred between two parties, one must enforce [SSL/TLS](#) or [IPsec](#) to be comply with [RFC 6238](#).

Activating Two-Factor Authentication

In the current state, **TOTP** will be used for OTP generation. For this purpose we recommend the **Google Authenticator** or any TOTP generator out there.

- Login with your account
- Open the 2FA settings
- Activate the 2FA Status
- Initialize your device with either a **QR-Code** or a **secret**

Frontend

These are the two main options, which you can obtain my toggling between **QR-Code** and **secret**.

The screenshot shows a web application interface for 'My Thai Star'. At the top, there's a navigation bar with links for ORDERS, RESERVATIONS, waiter, and a language switcher. Below the navigation is a table titled 'ORDERS' with columns for Reservation Date, Email, and Reference Number. A modal window is overlaid on the page, titled 'Set up Two Factor Authentication'. It contains a QR code for scanning with a Google Authenticator app, and two toggle buttons: 'Two Factor Authentication Status' (which is selected) and 'QR Code'. A 'Close' button is at the bottom right of the modal. The background table has several rows of data, and at the bottom right of the table area, there are pagination controls for 'Rows per page: 8' and '1-8 of 8'.

★ My Thai Star

RESERVATIONS

waiter

Filter

ORDERS

Reservation Date	Email	Reference Number
Jun 30, 2019 3:12 PM	user0@mail.com	CB_20170509_123502655Z
Jun 30, 2019 3:12 PM	host1@mail.com	CB_20170510_123502655Z
Jun 30, 2019 3:12 PM	host1@mail.com	CB_20170510_123502655Z
Jun 30, 2019 3:12 PM	host1@mail.com	CB_20170510_123502655Z
Jun 30, 2019 3:12 PM	host1@mail.com	CB_20170510_123502655Z
Jun 30, 2019 3:12 PM	host1@mail.com	CB_20170510_123502655Z
Jun 30, 2019 3:12 PM	host1@mail.com	CB_20170510_123502655Z
Jun 30, 2019 3:12 PM	host1@mail.com	CB_20170510_123503600Z
Jun 30, 2019 3:12 PM	host1@mail.com	CB_20170510_123503600Z

Set up Two Factor Authentication

V4GVY475PAOBSZJE

Please scan the QR code with the Google Authenticator or use the secret key

Two Factor Authentication Status

Secret Key

Rows per page: 8 ▾ 1-8 of 8 |◀ ▶▶|

After an activation and logout. This prompt will ask you to enter the OTP given from your device.

The image shows the homepage of the My Thai Star website. The background is a dark wood grain texture. In the center, there is a decorative logo with a crown-like shape containing the text "MY THAI STAR" in large letters, with the smaller text "More than just delicious food" underneath. Below the logo is a white rectangular modal box. Inside the modal, the text "One Time Password *" is displayed above a horizontal input field containing the number "123456". At the bottom of the modal are two buttons: "CANCEL" in grey and "APPLY" in green. At the top of the page, there is a dark navigation bar with the text "My Thai Star" on the left, and "HOME", "MENU", "BOOK TABLE", and icons for user profile and shopping cart on the right. At the bottom of the page, there are two smaller images: one showing the interior of the restaurant with hanging lamps, and another showing a dish of food.

86. Testing

86.1. Server Side

86.1.1. Java testing

Component testing

We are going to test our components as a unit using *Spring Test* and *Devon4j-test* modules.

In order to test a basic component of the app first we will create a test class in the `src/test/java` folder and inside the main package of the test module. We will name the class following the convention.

[Component]Test

Then, in the declaration of the test class, we will use the `@SpringBootTest` annotation to run the application context. In addition, we will extend the `ComponentTest` from *Devon4j-test* module to have access to the main functionalities of the module, [see more details here](#).

Spring Test allows us to use *Dependency Injection* so we can inject our component directly using the `@Inject` annotation.

Each test will be represented by a method annotated with `@Test`. Inside the method we will test one functionality, evaluating the result thanks to the *asserts* provided by the `ComponentTest` class that we are extending.

A simple test example

```
@SpringBootTest(classes = SpringBootApp.class)
public class DishmanagementTest extends ComponentTest {

    @Inject
    private Dishmanagement dishmanagement;

    @Test
    public void findAllDishes() {

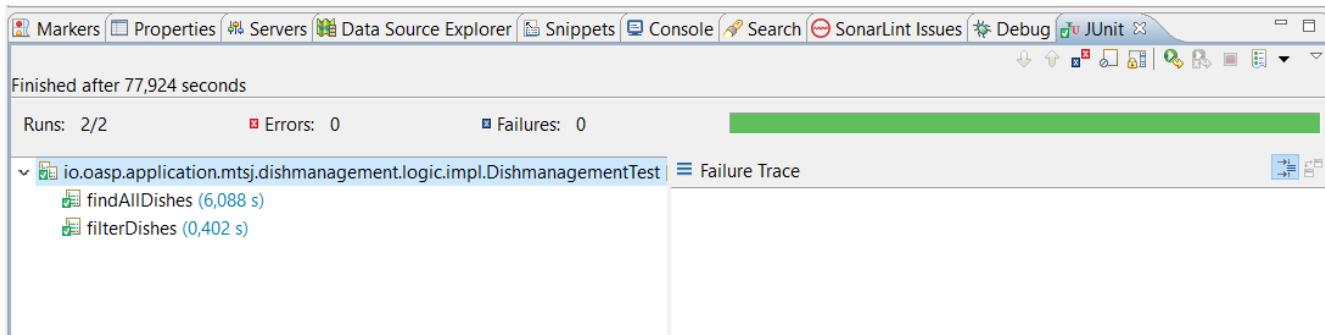
        PaginatedListTo<DishCto> result = this.dishmanagement.findDishes();
        assertThat(result).isNotNull();
    }

    ...
}
```

Running the tests

From Eclipse

We can run the test from within *Eclipse* with the contextual menu *Run As > JUnit Test*. This functionality can be launched from method level, class level or even package level. The results will be shown in the *JUnit* tab.



From command line using Maven

We can also run tests using *Maven* and the command line, using the command *mvn test* (or *mvn clean test*).

```
C:\MyThaiStar>mvn clean test
```

Doing this we will run all the tests of the project (recognized by the *Test* word at the end of the classes) and the results will be shown by sub-project.

...

```
[D: 2017-07-17 09:30:08,457] [P: INFO ] [C: ] [T: Thread-5] [L:  
org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean] - [M: Closing JPA  
EntityManagerFactory for persistence unit 'default']
```

Results :

```
Tests run: 11, Failures: 0, Errors: 0, Skipped: 1
```

...

```
[INFO]  
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ mtsj-server  
---  
[INFO] No sources to compile  
[INFO]  
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ mtsj-server ---  
[INFO] No tests to run.  
[INFO] -----  
[INFO] Reactor Summary:  
[INFO]  
[INFO] mtsj ..... SUCCESS [ 0.902 s]  
[INFO] mtsj-core ..... SUCCESS [02:30 min]  
[INFO] mtsj-server ..... SUCCESS [ 1.123 s]  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 02:35 min  
[INFO] Finished at: 20XX-07-17T09:30:13+02:00  
[INFO] Final Memory: 39M/193M  
[INFO] -----
```

86.1.2. .NET testing

TODO

86.1.3. Node.js testing

TODO

86.1.4. GraphQL testing

TODO

86.2. Client Side

86.2.1. Angular testing

Karma v1.4.1 - connected

DEBUG

Chrome 58.0.3029 (Windows 7 0.0.0) is idle

Jasmine 2.5.2

finished in 20.533s

45 specs, 0 failures

raise exceptions

MyThaiStar testing is made using Angular default testing environment and syntax language: [Karma](#) and [Jasmine](#)

To test an element of the application, you indicate that tests are a special type of files with the extension `.spec.ts`, then, in MyThaiStar angular/CLI config you can notice that there is an array with only one entry, Karma, with at the same time has one entry to Karma.config.js.

In the configuration of Karma we indicate which syntax language we are going to use (currently Jasmine as said before) between some other configurations, it is remarkable the last one: `browsers`. By default, the only available browser is chrome, that is because Karma works opening a chrome view to run all the tests, in that same window, Karma shows the result or errors of the test run. But we can add some other browser to adjust to our necessities, for example, in some automatic processes that run from console, it is not an option to open a chrome window, in that case, MyThaiStar used PhantomJS and ChromeHeadless.

Taking all of this into account, to run the test in MyThaiStar we need to move to project root folder and run this command : `ng test --browser <browser>`



If you run just `ng test` it will run the three browser options **simultaneously**, giving as a result three test runs and outputs, it can cause timeouts and unwanted behaviors, if you want a shortcut to run the test with chrome window you can just run `yarn test` so we really encourage to **not use** just `ng test`.

Here we are going to see how Client side testing of MyThaiStar has been done.

Testing Components

Angular components were created using angular/CLI `ng create component` so they already come with an spec file to test them. The only thing left to do is to add the providers and imports needed in the component to work as the component itself, once this is done, the most basic test is to be sure that all the dependencies and the component itself can be correctly created.

As an example, this is the `spec.ts` of the menu view component:

all the imports...

```
describe('MenuComponent', () => {
  let component: MenuComponent;
  let fixture: ComponentFixture<MenuComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ MenuComponent, MenuCardComponent ],
      providers: [SidenavService, MenuService, SnackBarService],
      imports: [
        BrowserAnimationsModule,
        BackendModule.forRoot({environmentType: 0, restServiceRoot: 'v1'}),
        CovalentModule,
      ],
    })
    .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(MenuComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

First we declare the component to be tested and a Fixture object, then, we configure the testingModule right in the same way we could configure the MenuModule with the difference here that tests always have to use the mockBackend because we do not want to really depend on a server to test our components.

Once configured the test module, we have to prepare the context of the test, in this case we create the component, that is exactly what is going on in the `beforeEach()` function.

Finally, we are ready to use the component and its fixture to check if the component has been correctly created.

At this moment this is the case for most of the components, in the future, some work would be applied on this matter to have a full testing experience in MyThaiStar components.

Dialog components

Dialog components are in a special category because they can not be tested normally. In the way Material implements the opening of dialogs, you have to create a component that will load into a dialog, to tell the module to load this component when needed, they have to be added into a special array category: *EntryComponents*. So, to test them, we need to import them in the test file as

well.

Also, the testing code to open the component is a bit different too:

```
...
beforeEach(() => {
  dialog = TestBed.get(MdDialog);
  component = dialog.open(CommentDialogComponent).componentInstance;
});
...
...
```

That is right, the `beforeEach()` function is slightly different from the example above, in this case we have to force the test to know that the component is only displayed in a dialog, so we have to open a dialog with this component in order to access it.

Testing Services

As well as components, services can be tested too, actually, they are even more necessary to be tested because they have inside more complex logic and data management.

As an example of testing services i am going to use a well done services, with a specific purpose and with its logic completely tested, the price-calculator service:

...

```

describe('PriceCalculatorService', () => {

  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [PriceCalculatorService],
    });
  });

  it('should be properly injected', inject([PriceCalculatorService], (service: PriceCalculatorService) => {
    expect(service).toBeTruthy();
  }));

  describe('check getPrice method', () => {

    it('should calculate price for single order without extras', inject([PriceCalculatorService], (service: PriceCalculatorService) => {
      const order: OrderView = {
        dish: {
          id: 0,
          price: 12.50,
          name: 'Order without extras',
        },
        orderLine: {
          comment: '',
          amount: 1,
        },
        extras: [],
      };

      expect(service.getPrice(order)).toEqual(order.dish.price);
    }));
  });
}

```

In services test, we have to inject the service in order to use it, then we can define some initializing contexts to test if the functions of the services returns the expected values, in the example we can see how an imaginary order is created and expected the function `getPrice()` to correctly calculate the price of that order.

In this same test file you can find some more test regarding all the possibilities of use in that services: orders with and without extras, single order, multiple orders and so on.

Some services as well as the components have only tested that they are correctly created and their dependencies properly injected, in the future, will be full covering regarding this services test coverage.

Testing in a CI environment

86.2.2. Xamarin testing

TODO

86.3. End to end

86.3.1. MrChecker E2E Testing

Introduction

MrChecker is a testing framework included in devonfw with several useful modules, from which we will focus on the Selenium Module, a module designed to make end-to-end testing easier to implement.

How to use it

First of all download the repository.

You must run My Thai Star frontend and backend application and modify your url to the front in `mrchecker/endtoend-test/src/resources/settings.properties`

Now you can run end to end test to **check** if the application works properly.

To run the e2e test you have two options:

The first option is using the command line in devonfw distribution

```
cd mrchecker/endtoend-test/  
mvn test -Dtest=MyThaiStarTest -Dbrowser=Chrome
```

optionally you can use it with a headless version or using another navigator:

```
// chrome headless (without visual component)  
mvn test -Dtest=MyThaiStarTest -Dbrowser=ChromeHeadless  
// use firefox navigator  
mvn test -Dtest=MyThaiStarTest -Dbrowser=FireFox
```

The second is importing the project in devonfw Eclipse and running *MyThaiStarTest.java* as JUnit (right click, run as JUnit)

They can be executed one by one or all in one go, comment or uncomment `@Test` before those tests to enable or disable them.

For more information about how to use MrChecker and build your own end to end test read:
* MrChecker documentation * MrChecker tutorial for My Thai Star

End to end tests in My Thai Star

We have included a test suite with four tests to run in My Thai Star to verify everything works properly.

The included tests do the following:

- *Test_loginAndLogOut*: Log in and log out.
- *Test_loginFake*: Attempt to log in with a fake user.
- *Test_bookTable*: Log in and book a table, then login with a waiter and check if the table was successfully booked.
- *Test_orderMenu*: Log in and order food for a certain booked table.

These four tests can be found inside **MyThaiStarTest.java** located [here](#).

87. UI design

87.1. Style guide

The screenshot displays the 'MY THAI STAR STYLE GUIDE' with the following sections:

- Colors**: Main color (BR: #46362C), Secondary colors (GR1: #339966, GR2: #006633, GR3: #D7E8DF, RE: #FF6666, YE: #FFCC66, BL: #0099CC). Gray scale (G1: #333333, G2: #666666, G3: #999999, G4: #CCCCCC, G5: #F2F2F2, WH: #FFFFFF).
- Fonts**: Roboto Regular, Roboto Bold. Examples of text sizes and descriptions:
 - H1: Super size title, 30px, Apparently we had reached ...
 - H2: Title, 18px, Apparently we had reached a great height in the...
 - H3: Subtitle, 16px, Apparently we had reached a great height in the ...
 - H4: Paragraph, 14px, Apparently we had reached a great height in the atmosphere
 - H5: Labels, messages..., 12px, Apparently we had reached a great height in the atmosphere
- Buttons**: Main buttons (NORMAL, HOVER, ACTIVE / PRESSED) and Secondary buttons (NORMAL, HOVER, ACTIVE / PRESSED). Descriptions for each state.
- Form components**: Field empty (Label: Text color: G3, Text size: H4 Regular; Field border: Border width: 2px, Border color: G4), Field content (Label: Text color: G3, Text size: H5 Regular; Content: Text color: G1, Text size: H4 Regular; Field border: Border width: 2px, Border color: G4), Field focused (Label: Text color: GR1, Text size: H5 Regular; Content: Text color: G1, Text size: H4 Regular; Field border: Border width: 2px, Border color: GR1), Checkbox unselected (Label: Unselected, Text color: G1, Text size: H4 Regular; Box: Border width: 1px, Border color: G4, Background color: WH), Checkbox selected (Selected, Label: Text color: G1, Text size: H4 Regular; Box: Border width: 1px, Border color: GR1, Background color: GR1), Slider (Slider: Text color: G3, Text size: H5 Regular; Label: Text color: G3, Text size: H5 Regular; Border: Border color: G4, Border color active: GR1; Circle: Background color: WH, Border color: GR1), Google Material Design Icons (User, List, Arrow, Calendar, Star, Heart icons).

87.2. Low and high fidelity wireframes

History of mockup designs for My Thai Star.

- [MTS Wireframes Low Fidelity](#)
- [MTS Wireframes High Fidelity \(Sprint 1\)](#)
- [MTS Wireframes High Fidelity \(Sprint 1\) - Copy](#)
- [MTS Wireframes High Fidelity \(Sprint 1\) - Mobile](#)
- [MTS Wireframes High Fidelity \(Sprint 2\)](#)
- [MTS Wireframes High Fidelity \(Sprint 2\) - Modifications](#)

88. CI/CD

88.1. My Thai Star in Production Line

What is PL?

The Production Line Project is a set of server-side collaboration tools for Capgemini engagements. It has been developed for supporting project engagements with individual tools like issue tracking, continuous integration, continuous deployment, documentation, binary storage and much more!



Introduction

Although the PL Project is a wide set of tools, only 3 are going to be mainly used for My Thai Star projects to build a Continuous Integration and Continuos Delivery environment. All three are available in the [PL instance](#) used for this project.

1. Jenkins

This is going to be the "main tool". Jenkins helps to automate the non-human part of the development with Continuos Integration and is going to host all Pipelines (and, obviously, execute them).

2. Nexus

Nexus manages software "artifacts" required for development. It is possible to both download dependencies from Nexus and publish artifacts as well. It allows to share resources within an organization.

3. SonarQube

It is a platform for continuous inspection of the code. It is going to be used for the Java back-end.

88.1.3. Where can I find all My Thai Star Pipelines?

They are located under the **MTS** folder of the PL instance:



Jobs generated by the MyThaiStar Template.

S	W	Nombre ↓
		MyThaiStar_FRONTEND_BUILD
		MyThaiStar_FRONTEND_DEPLOY
		MyThaiStar_REVERSE-PROXY_DEPLOY
		MyThaiStar_SERVER_BUILD
		MyThaiStar_SERVER_DEPLOY

Icono: [S](#) [M](#) [L](#)

Those Jenkins Pipelines will not have any code to execute. They're just pointing to all **Jenkinsfiles** under the `/jenkins` folder of the repository. They can be found [here](#).

88.1.4. CI in My Thai Star stack

- [Angular CI](#)
- [Java CI](#)

88.1.5. How to configure everything out of the box

Production Line currently has a template to integrate My Thai Star. All information can be found at [devonfw production line repository](#)

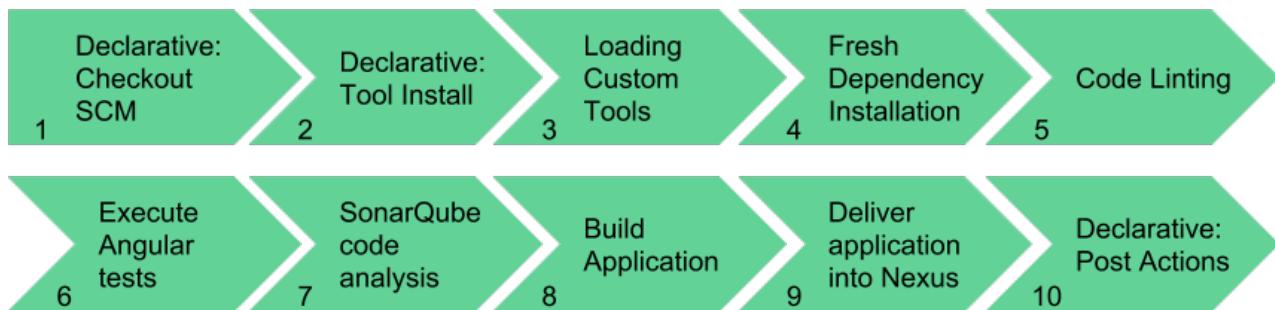
88.1.6. Angular CI

The Angular client-side of My Thai Star is going to have some specific needs for the CI-CD Pipeline to perform mandatory operations.

Pipeline

The Pipeline for the Angular client-side is going to be called **MyThaiStar_FRONTEND_BUILD**. It is

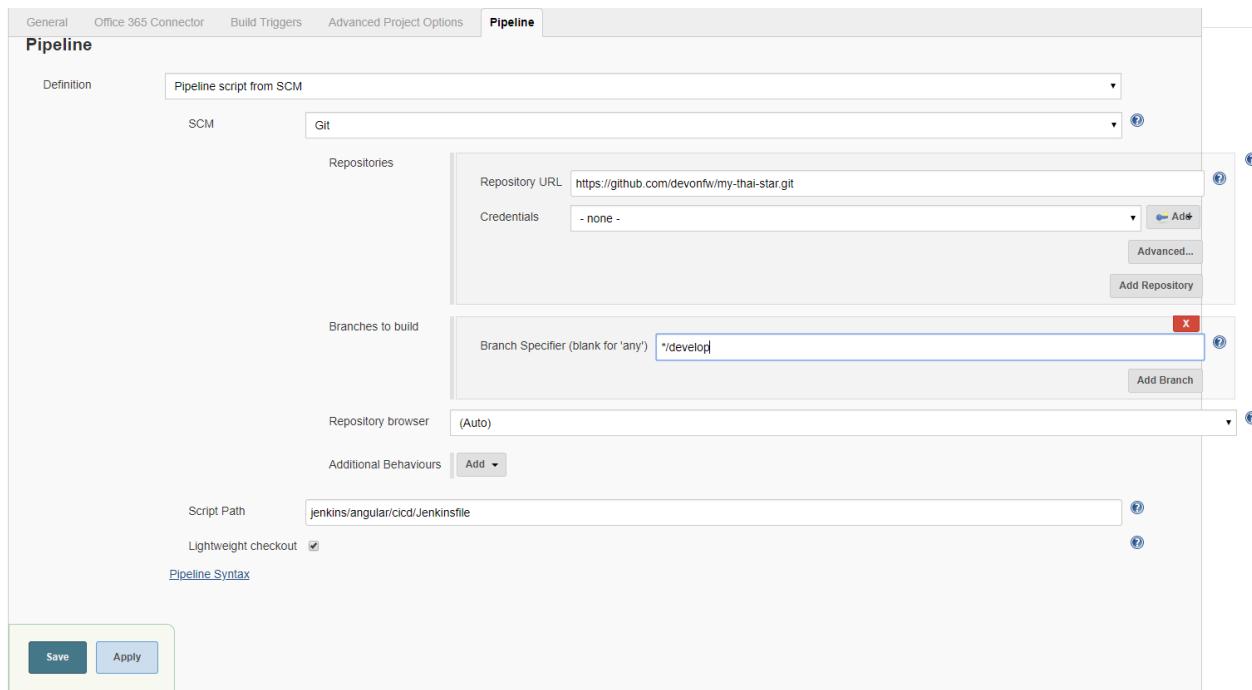
located in the PL instance, under the [MTS folder](#) (as previously explained). It is going to follow a process flow like this one:



Each of those steps are called *stages* in the Jenkins context. Let's see what those steps mean in the context of the Angular application:

1. Declarative: Checkout SCM

Retrieves the project from the GitHub repository which it's located. This step is not defined directly in our pipeline, but as it is loaded from the repository this step should always be done at the beginning.



2. Declarative: Tool Install

The Pipeline needs some Tools to perform some operations with the Angular project. These tool is a correct version of **NodeJS** (10.17.0 LTS) with **Yarn** installed as global package.

```

tools {
    nodejs "NodeJS 10.14.0"
}

```

3. Loading Custom Tools

The Pipeline also needs a browser in order to execute the tests, so in this step the chrome-stable will be loaded. We will use it in a headless mode.

```
tool chrome
```

4. Fresh Dependency Installation

The script `$ yarn` does a package installation. As we always clean the workspace after the pipeline, all packages must be installed in every execution.

5. Code Linting

This script executes a linting process of TypeScript. Rules can be defined in the `tslint.json` file of the project. It throws an exception whenever a file contains a non-compliant piece of code.

6. Execute Angular tests

The CI testing of the Angular client is different than the standard local testing (adapted to CI environments, as specified in the **Adaptation** section of document). This script just executes the following commands:

```
ng test --browsers ChromeHeadless --watch=false
```

7. Check dependencies

Before continue, we print the result of `yarn audit`. It shows the vulnerabilities in the dependencies. It do not process the response. The purpose is only to track the result of the command.

```
yarn audit
```

8. SonarQube code analysis

The script load and execute the tool `sonar-scanner`. This tool is loaded here because it's not used in any other part of the pipeline. The `sonar-scanner` will take all code, upload it to sonarQube and wait until sonarQube send us a response with the quality of our code. If the code do not pass the quality gate, the pipeline will stop at this point.

9. Build Application

The building process of the Angular client would result in a folder called `/dist` in the main Angular's directory. That folder is the one that is going to be served afterwards as an artifact. This process has also been adapted to some Deployment needs. This building script executes the following:

```
ng build --configuration=docker
```

10. Deliver application into Nexus

Once the scripts produce the Angular artifact ([/dist](#) folder), it's time to package it and store into nexus.

11. Declarative: Post Actions

At the end, this step is always executed, even if a previous stage fail. We use this step to clean up the workspace for future executions

```
post {  
    always {  
        cleanWs()  
    }  
}
```

Adjustments

The Angular project Pipeline needed some "extra" features to complete all planned processes. Those features resulted in some additions to the project.

Pipeline Environment

In order to easily reuse the pipeline in other angular projects, all variables have been defined in the block environment. All variables have the default values that Production Line uses, so if you're going to work in production line you won't have to change anything. Example:

```

environment {
    // Script for build the application. Defined at package.json
    buildScript = 'build --configuration=docker'
    // Script for lint the application. Defined at package.json
    lintScript = 'lint'
    // Script for test the application. Defined at package.json
    testScript = 'test:ci'
    // Angular directory
    angularDir = 'angular'
    // SRC folder. It will be angularDir/srcDir
    srcDir = 'src'
    // Name of the custom tool for chrome stable
    chrome = 'Chrome-stable'

    // sonarQube
    // Name of the sonarQube tool
    sonarTool = 'SonarQube'
    // Name of the sonarQube environment
    sonarEnv = "SonarQube"

    // Nexus
    // Artifact groupId
    groupId = 'com.devonfw.mythaistar'
    // Nexus repository ID
    repositoryId = 'pl-nexus'
    // Nexus internal URL
    repositoryUrl = 'http://nexus3-core:8081/nexus3/repository/maven-snapshots'
    // Maven global settings configuration ID
    globalSettingsId = 'MavenSettings'
    // Maven tool id
    mavenInstallation = 'Maven3'
}

```

Description

- **buildScript**: script for build the application. It must be defined at package.json.

Example (package.json):

```
{
  "name": "mythaistar-restaurant",
  ...
  "scripts": {
    ...
    "build": "ng build",
    ...
  }
  ...
}
```

This will be used as follows:

```
sh """yarn ${buildScript}"""
```

- **lintScript**: Script for lint the application. Defined at package.json

Example (package.json):

```
{
  "name": "mythaistar-restaurant",
  ...
  "scripts": {
    ...
    "lint": "ng lint",
    ...
  }
  ...
}
```

This will be used as follows:

```
sh """yarn ${lintScript}"""
```

- **testScript**: Script for test the application. Defined at package.json

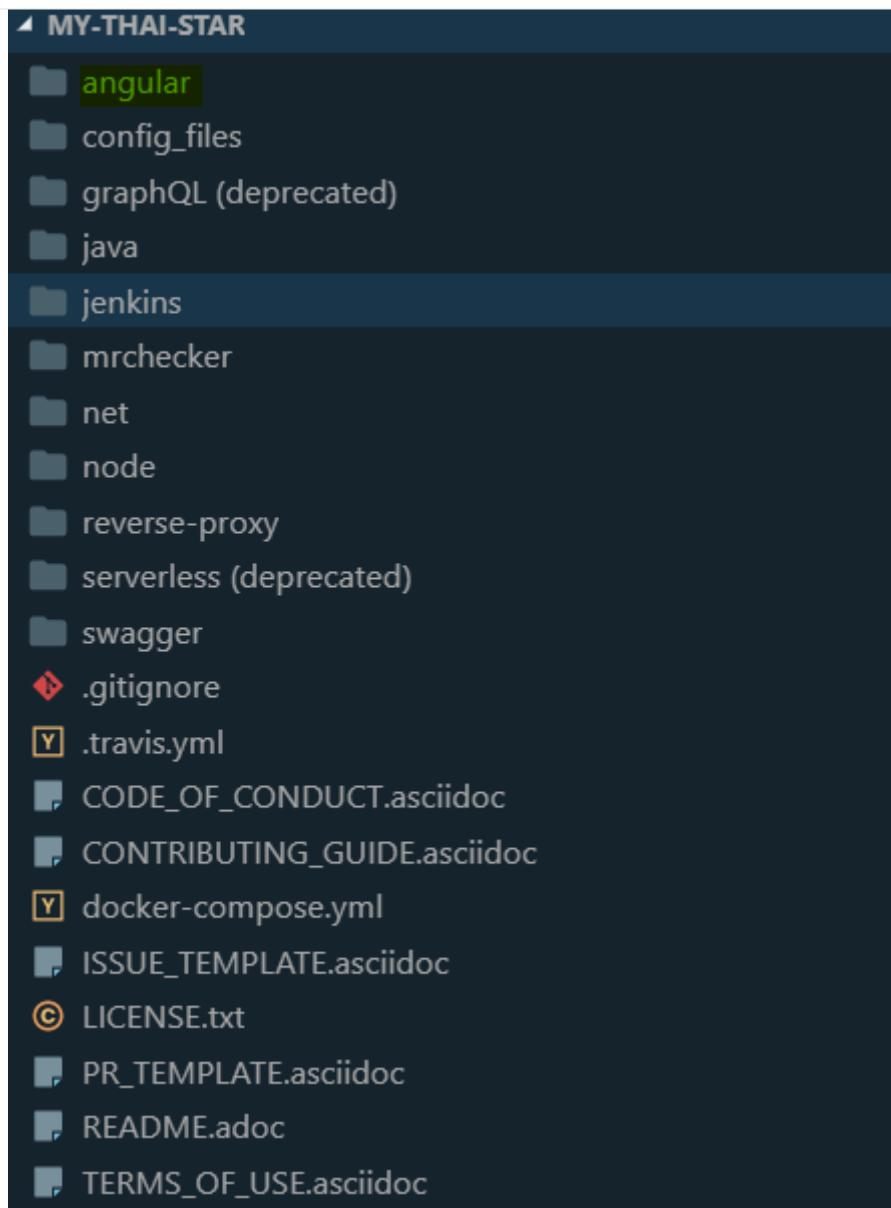
Example (package.json):

```
{  
  "name": "mythaistar-restaurant",  
  ...  
  "scripts": {  
    ...  
    "test:ci": "npm run postinstall:web && ng test --browsers ChromeHeadless  
--watch=false",  
    ...  
  }  
  ...  
}
```

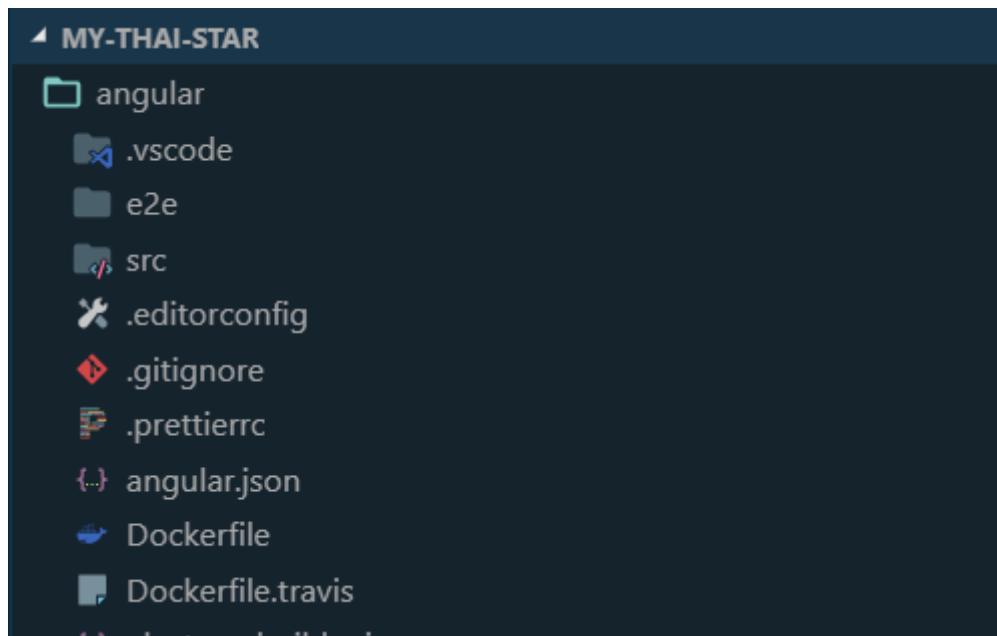
This will be used as follows:

```
sh """yarn ${testScript}"""
```

- **angularDir**: Relative route to angular application. In My Thai Star this is the angular folder. The actual directory (.) is also allowed.



- **srcDir:** Directory where you store the source code. For angular applications the default value is `src`



- **chrome:** Since you need a browser to run your tests, we must provide one. This variable contains the name of the custom tool for google chrome.

The screenshot shows the 'Custom tool' configuration page for Google Chrome. It includes fields for 'Name' (Chrome-stable), 'Custom Tool Configuration...', 'Install automatically' (checked), 'Run Shell Command' (with a command to install Google Chrome via sudo su and wget), 'Tool Home' (/opt/chrome), and a 'Delete Installer' button.

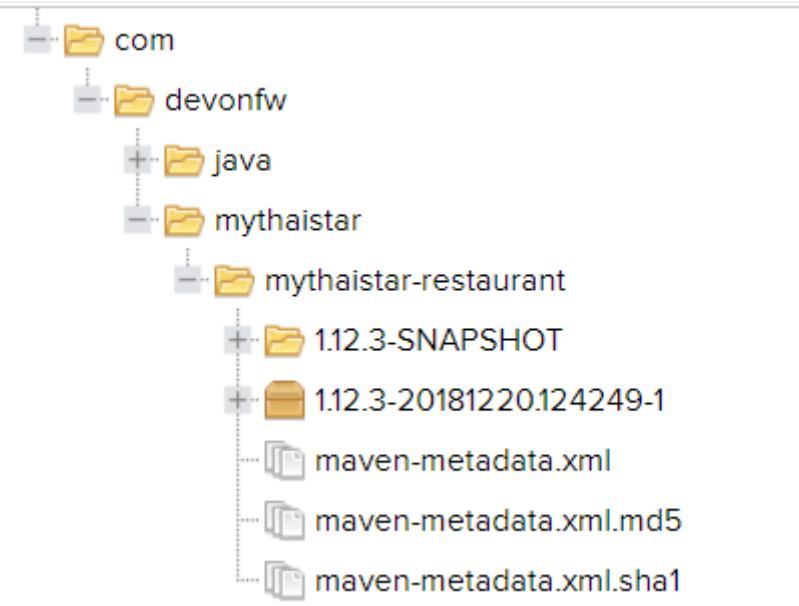
- **sonarTool:** Name of the sonarQube scanner installation.

The screenshot shows the 'SonarQube Scanner' configuration page. It includes fields for 'Name' (SonarQube), 'Install automatically' (checked), 'Install from Maven Central' (Version SonarQube Scanner 3.2.0.1227), and a 'Delete Installer' button.

- **sonarEnv:** Name of the sonarQube environment. SonarQube is the default value for PL.

The screenshot shows the 'SonarQube servers' configuration page. It includes sections for 'Environment variables' (checkbox for enabling injection), 'SonarQube installations' (Name: SonarQube, Server URL: http://sonarqube-core:9000/sonarqube, Server authentication token: masked), and a 'Delete SonarQube Scanner' button.

- **groupId:** Group id of the application. It will be used to storage the application in nexus3



- **repositoryId**: Id of the nexus3 repository. It must be defined at maven global config file.

Server Credentials	ServerId	Credentials	
	pl-nexus	admin/******** (Admin credentials to access Nexus)	

- **repositoryUrl**: The url of the repository.
- **globalSettingsId**: The id of the global settings file.

The configuration	
ID	MavenSettings
Name	MyGlobalSettings
Comment	global settings
Replace All	<input checked="" type="checkbox"/>

- **mavenInstallation**: The name of the maven tool.

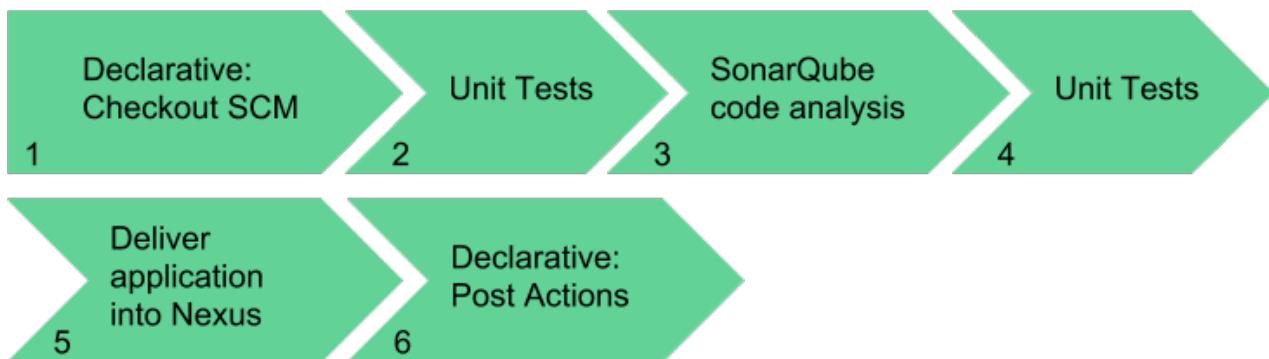
Maven											
Maven installations	<table border="1"> <tr> <td></td> <td>Maven</td> </tr> <tr> <td>Name</td> <td>Maven3</td> </tr> <tr> <td colspan="2"><input checked="" type="checkbox"/> Install automatically</td> </tr> <tr> <td colspan="2"></td> </tr> <tr> <td>Version</td> <td>3.6.0</td> </tr> </table>		Maven	Name	Maven3	<input checked="" type="checkbox"/> Install automatically				Version	3.6.0
	Maven										
Name	Maven3										
<input checked="" type="checkbox"/> Install automatically											
Version	3.6.0										

88.1.7. Java CI

The Java server-side of My Thai Star is an **devon4j**-based application. As long as **Maven** and a **Java 8** are going to be needed, the Pipeline should have those tools available as well.

Pipeline

This Pipeline is called **MyThaiStar_SERVER_BUILD**, and it is located exactly in the same PL instance's folder than **MyThaiStar_FRONTEND_BUILD**. Let's see how the Pipeline's flow behaves.



Check those Pipeline stages with more detail:

1. Declarative: Checkout SCM

Gets the code from <https://github.com/devonfw/my-thai-star>. This step is not defined directly in our pipeline, but as it is loaded from the repository this step should always be done at the beginning.

2. Declarative: Tool Install

The My Thai Star application works with JDK11. In this step, if JDK11 is not installed, we install it and then put the JDK folder into PATH.

```

tools {
    jdk 'OpenJDK11'
}

```

3. Loading Custom Tools

In this step we load the tools that can not be loaded in the previous step. As My Thai Star is delivered as docker container, in this step we load docker as custom tool.

```
tool dockerTool
```

4. Install dependencies

This step will download all project dependencies.

```
mvn clean install -Dmaven.test.skip=true
```

5. Unit Tests

This step will execute the project unit test with maven.

```
mvn clean test
```

6. Dependency Checker

Execute the OWASP Dependency Checker in order to validate the project dependencies. It will generate a report that can be used in SonarQube

```
dependencyCheck additionalArguments: '--project "mtsj" --scan java/mtsj --format XML', odcInstallation: 'dependency-check'
dependencyCheckPublisher pattern: ''
```

7. SonarQube analysis

The code is evaluated using the integrated PL instance's SonarQube. Also, it will wait for the quality gate status. If the status is failing, the pipeline execution will be stopped.

```
withSonarQubeEnv(sonarEnv) {
    sh "mvn sonar:sonar"
}

def qg = waitForQualityGate()
if (qg.status != 'OK') {
    error "Pipeline aborted due to quality gate failure: ${qg.status}"
}
```

8. Deliver application into Nexus

Store all artifacts into nexus.

```
mvn deploy -Dmaven.test.skip=true
```

9. Create the Docker image

Create the docker image and then publish the image into a docker registry.

Adjustments

Pipeline Environment

In order to easily reuse the pipeline in other java projects, all variables have been defined in the block environment. All variables have the default values that Production Line uses, so if you're going to work in production line you won't have to change anything. Example:

```

environment {
    // Directory with java project
    javaDir = 'java/mtsj'

    // sonarQube
    // Name of the sonarQube environment
    sonarEnv = "SonarQube"

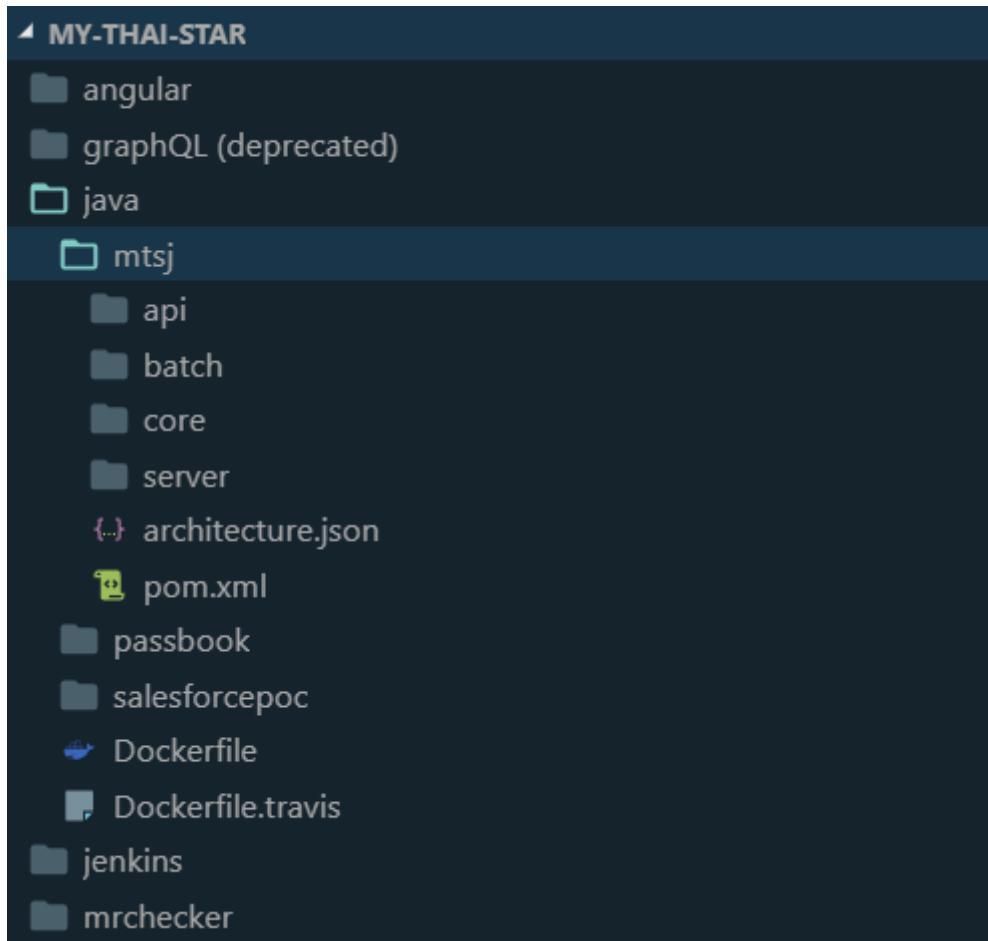
    // Nexus 3
    // Maven global settings configuration ID
    globalSettingsId = 'MavenSettings'
    // Maven tool id
    mavenInstallation = 'Maven3'

    // Docker
    dockerRegistryCredentials = 'nexus-api'
    dockerRegistryProtocol = 'https://'
    dockerTool = 'docker-global'
}

```

Description

- **javaDir:** Relative route to java application. In My Thai Star this is the java/mtsj folder. The actual directory(.) is also allowed.



- **sonarEnv**: Name of the sonarQube environment. SonarQube is the default value for PL.
- **globalSettingsId**: The id of the global settings file. MavenSettings is the default value for PL.

The configuration

ID	MavenSettings
Name	MyGlobalSettings
Comment	global settings
Replace All	<input checked="" type="checkbox"/>

[?](#)

- **mavenInstallation**: The name of the maven tool. Maven3 is the default value for PL.

Maven

Maven installations	<table border="1"> <tr> <td> Maven</td> <td>Name <input type="text" value="Maven3"/></td> </tr> <tr> <td colspan="2"><input checked="" type="checkbox"/> Install automatically</td> </tr> <tr> <td> Install from Apache</td> <td>Version <input type="text" value="3.6.0"/></td> </tr> </table>	 Maven	Name <input type="text" value="Maven3"/>	<input checked="" type="checkbox"/> Install automatically		 Install from Apache	Version <input type="text" value="3.6.0"/>
 Maven	Name <input type="text" value="Maven3"/>						
<input checked="" type="checkbox"/> Install automatically							
 Install from Apache	Version <input type="text" value="3.6.0"/>						
Delete Installer							
Add Installer ▾							
Delete Maven							

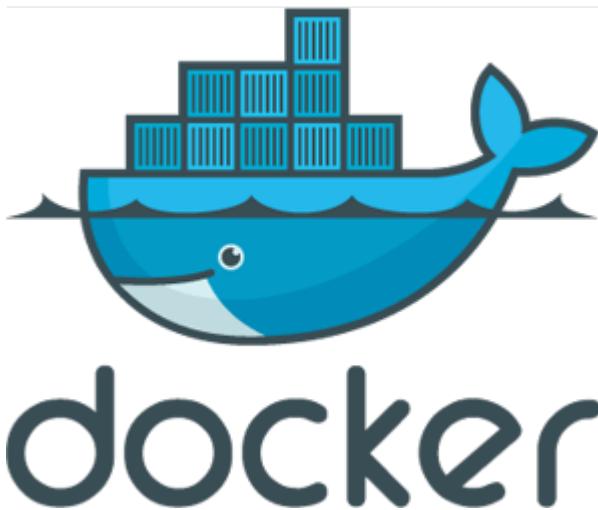
Distribution management

The only *extra* thing that needs to be added to the Java server-side is some information that determines where the artifact of the project is going to be stored in **Nexus**. This is going to be a section in the main **pom.xml** file called **<distributionManagement>**. This section will point to the PL instance's Nexus. Let's have a look at it. It's already configured with the PL default values.

```
<distributionManagement>
  <repository>
    <id>pl-nexus</id>
    <name>PL Releases</name>
    <url>http://nexus3-core:8081/nexus/content/repositories/maven-releases/</url>
  </repository>
  <snapshotRepository>
    <id>pl-nexus</id>
    <name>PL Snapshots</name>
    <url>http://nexus3-core:8081/nexus3/repository/maven-snapshots</url>
  </snapshotRepository>
</distributionManagement>
```

88.2. Deployment

The main deployment tool used for **My Thai Star** is be **Docker**.



It is a tool to run application in isolated environments. Those *isolated environments* will be what we call **Docker containers**. For instance, it won't be necessary any installation of **nginx** or **Apache tomcat** or anything necessary to deploy, because there will be some containers that actually *have* those technologies inside.

88.2.1. Where Docker containers will be running?

Of course, it is necessary to have an external Deployment Server. Every Docker process will run in it. It will be accessed from Production Line pipelines via **SSH**. Thus, the pipeline itself will manage the scenario of, if every previous process like testing passes as OK, stop actual containers and create new ones.

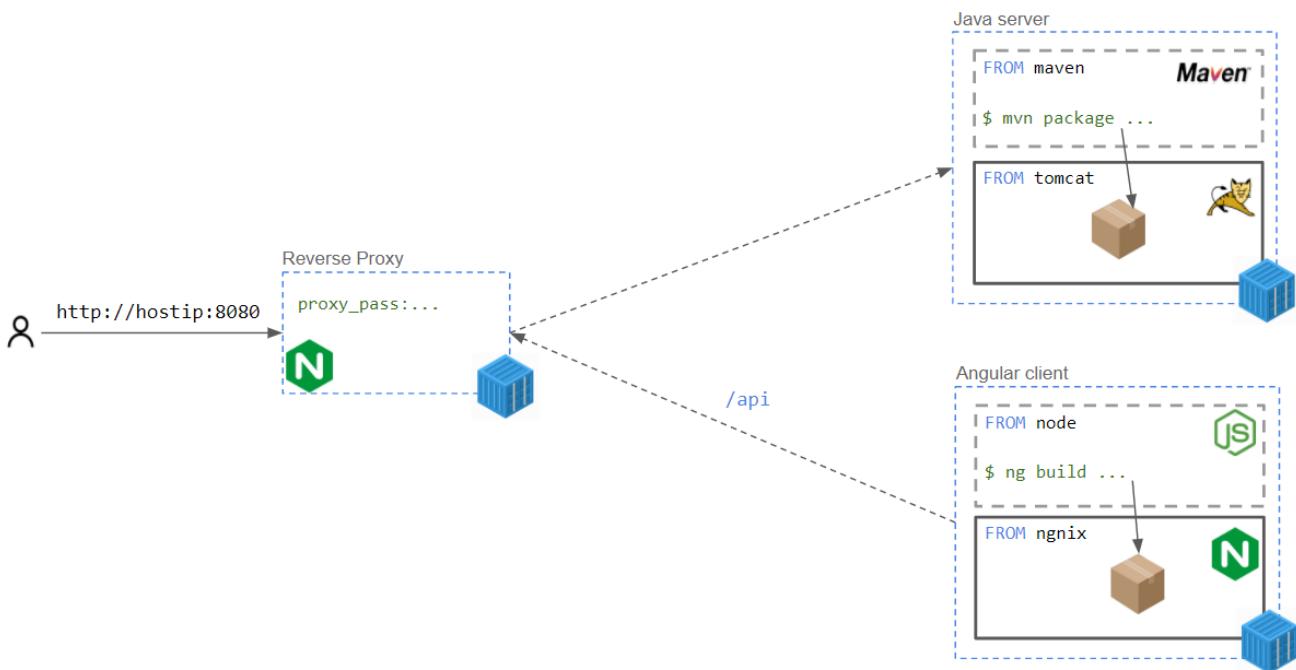
This external server will be located in <http://de-mucdevondepl01> .

88.2.2. Container Schema

3 Docker containers are being used for the deployment of My Thai Star:

1. **nginx** for the Reverse Proxy
2. **tomcat** for the Java Server
3. **nginx** for the Angular Client

The usage of the **Reverse Proxy** will allow the client to call via `/api` every single Java Server's REST operation. Moreover, there will only be 1 port in usage in the remote Docker host, the one mapped for the Reverse Proxy: `8080`. Besides the deployment itself using **nginx** and **tomcat**, both client and server are previously built using **nodejs** and **maven** images. Artifacts produced by them will be pasted in servers' containers using multi-stage docker builds. It will all follow this schema:



This orchestration of all 3 containers will be done by using a `docker-compose.yml` file. To redirect traffic from one container to another (i.e. reverse-proxy to angular client or angular client to java server) will be done by using, as host names, the service name `docker-compose` defines for each of them, followed by the internally exposed port:

- `http://reverse-proxy:80`
- `http://angular:80`
- `http://java:8080`



A implementation using `Traefik` as reverse proxy instead of NGINX is also available.

88.2.3. Run My Thai Star

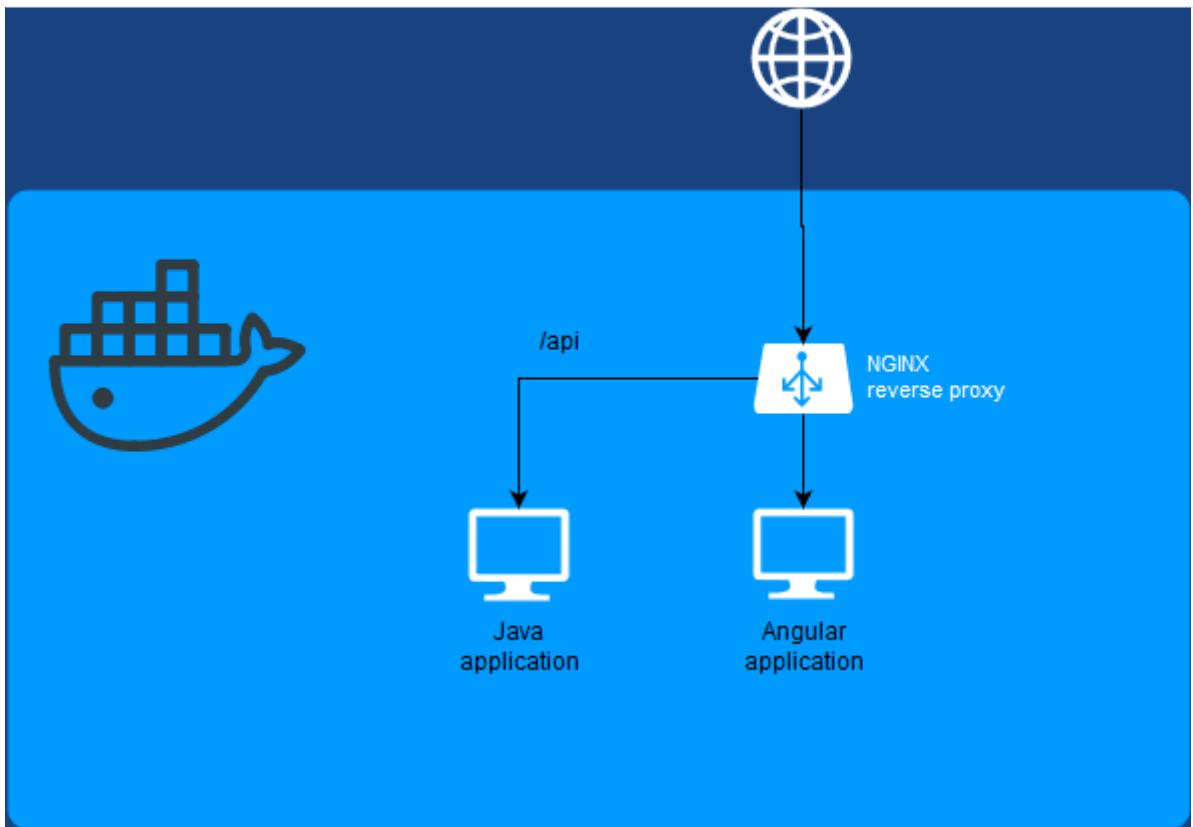
The steps to run **My Thai Star** are:

1. Clone the repository `$ git clone https://github.com/devonfw/my-thai-star.git`
2. Run the docker compose command: `$ docker-compose up`

88.2.4. Deployment Pipelines

As PL does not support deployments, we have created separate pipelines for this purpose. Those pipelines are: `MyThaiStar_REVERSE-PROXY_DEPLOY`, `MyThaiStar_FRONTEND_DEPLOY` and `MyThaiStar_SERVER_DEPLOY`.

The application will be deployed using docker on a remote machine. The architecture is as follows:



The parts to be deployed are: an NGINX reverse proxy, the java application and the angular application.

MyThaiStar_SERVER_DEPLOY Pipeline

Deploys on the server the Java part of My Thai Star.

Parameters

- **registryUrl**: The url to the docker registry where the image is stored.
- **registryCredentialsId**: Credentials to publish/download images from registry.
- **dockerNetwork**: Network of your My Thai Star application. You can deploy several versions of MTS in the same server by changing the dockerNetwork.
- **VERSION**: The version that you can to deploy.

Pipeline steps

- **Create docker network**: Create the docker network with the name provided as parameter.
- **Deploy new image**: Deploy a new java container. If it already exists, first it delete the previous one.

MyThaiStar_FRONTEND_DEPLOY

Deploys on the server the Angular part of My Thai Star

Parameters

- **registryUrl**: The url to the docker registry where the image is stored.

- **registryCredentialsId:** Credentials to publish/download images from registry.
- **dockerNetwork:** Network of your My Thai Star application. You can deploy several versions of MTS in the same server by changing the dockerNetwork.
- **VERSION:** The version that you can to deploy.

Pipeline steps

- **Create docker network:** Create the docker network with the name provided as parameter.
- **Deploy new image:** Deploy a new angular container. If it already exists, first it delete the previous one.

MyThaiStar_REVERSE-PROXY_DEPLOY Pipeline



As reverse proxy connects to the Java and Angular application, both must be deployed before you execute this pipeline.

The MyThaiStar_REVERSE-PROXY_DEPLOY pipeline will deploy the My Thai Star reverse proxy into a remote machine using docker.

Parameters

- **registryUrl:** The url to the docker registry where the image is stored.
- **registryCredentialsId:** Credentials to publish/download images from registry.
- **buildReverseProxy:** If yes, it will build and publish a new version of reverse-proxy.
- **port:** Port of the MTS application. You must ensure that those port is available in the deployment machine.
- **dockerNetwork:** Network of your My Thai Star application. You can deploy several versions of MTS in the same server by changing the port and the dockerNetwork.
- **VERSION:** The version that you can to deploy.

Pipeline steps

- **Create docker network:** Create the docker network with the name provided as parameter.
- **Create the Docker image:** If buildReverseProxy is enabled, this step will create a new docker image and publish it to the docker registry.
- **Deploy new image:** Deploy a new reverse proxy container. If it already exists, first it delete the previous one.

88.2.5. Deployment Strategies

In this chapter different way of deploying My Thai Star are explained. Everything will be based in Docker.

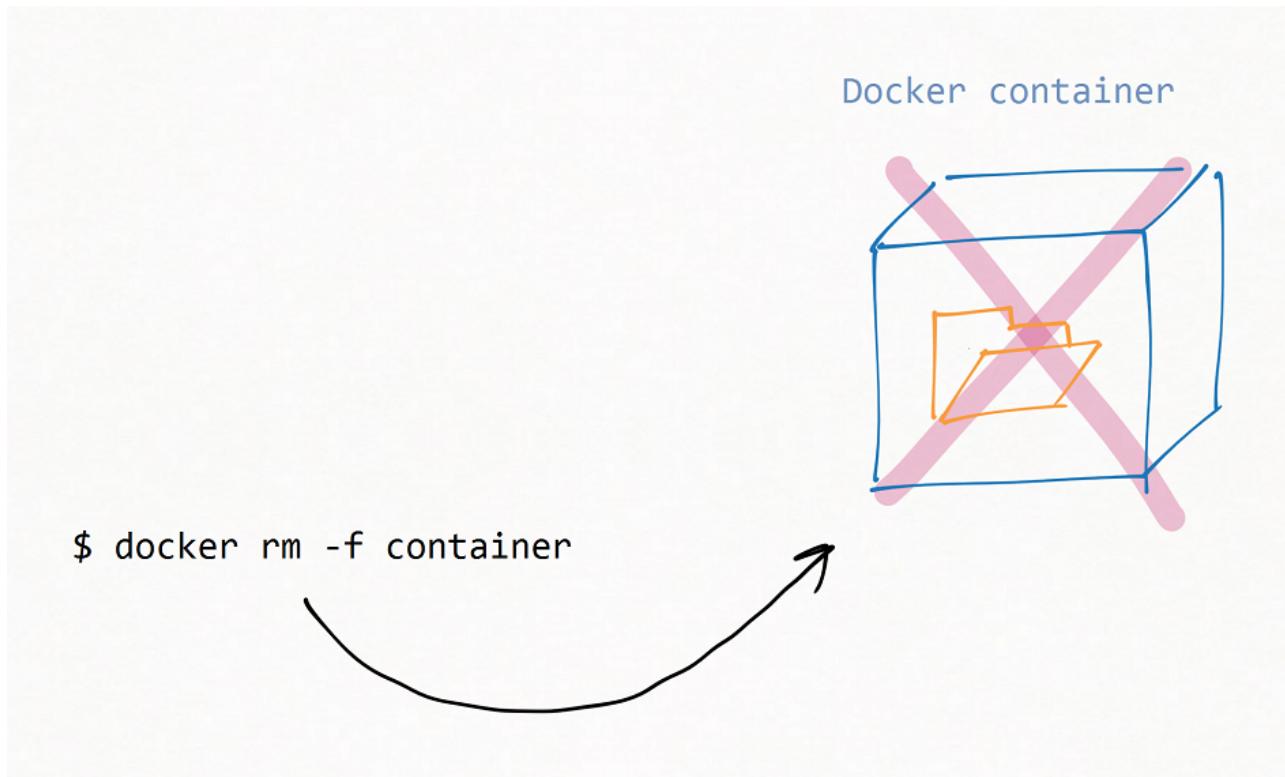
Independent Docker containers

The first way of deployment will use isolated Docker containers. That means that if the client-side

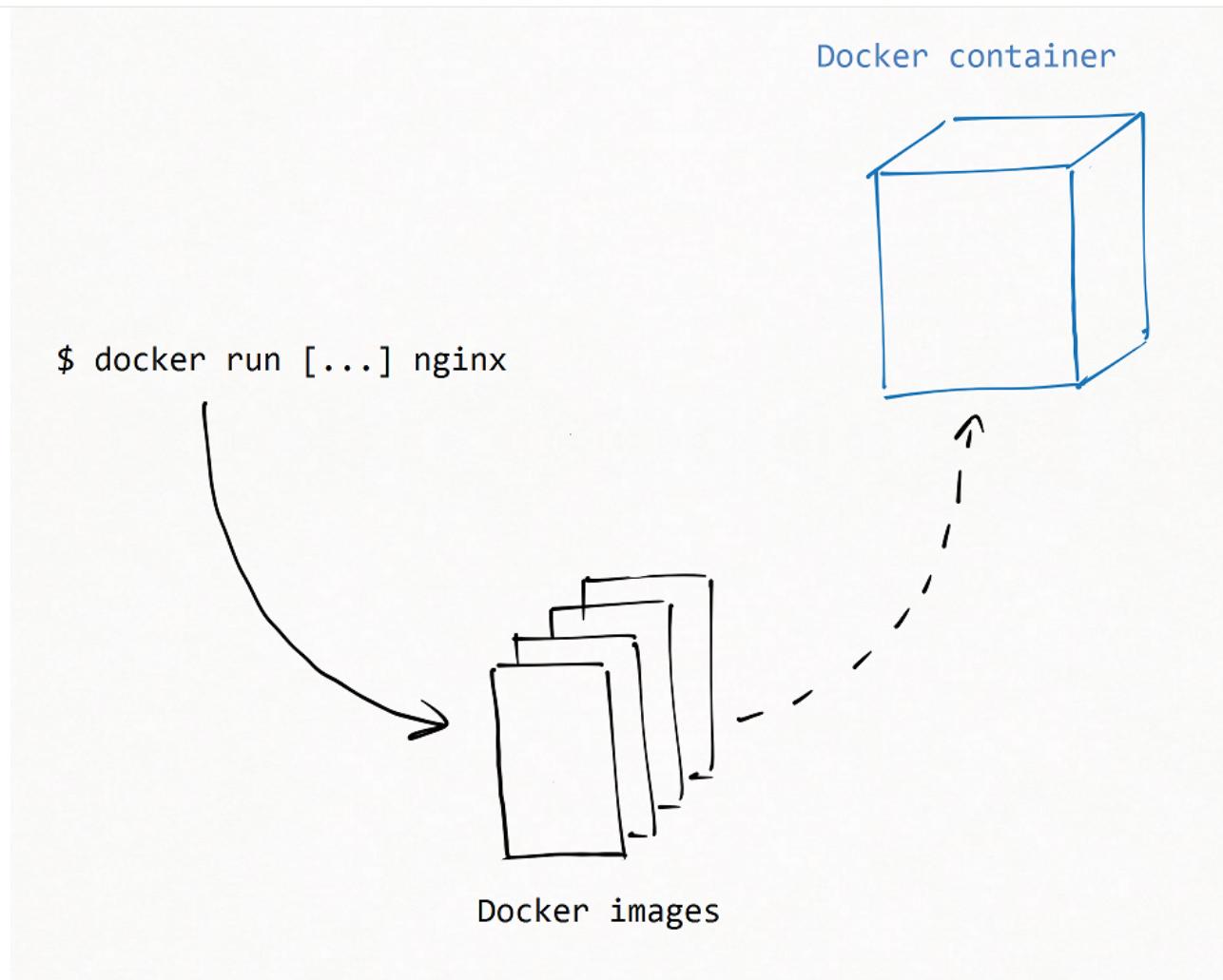
container is deployed, it does not affect the server-side container's life cycle and vice versa.

Let's show how the containers will behave during their life cycle.

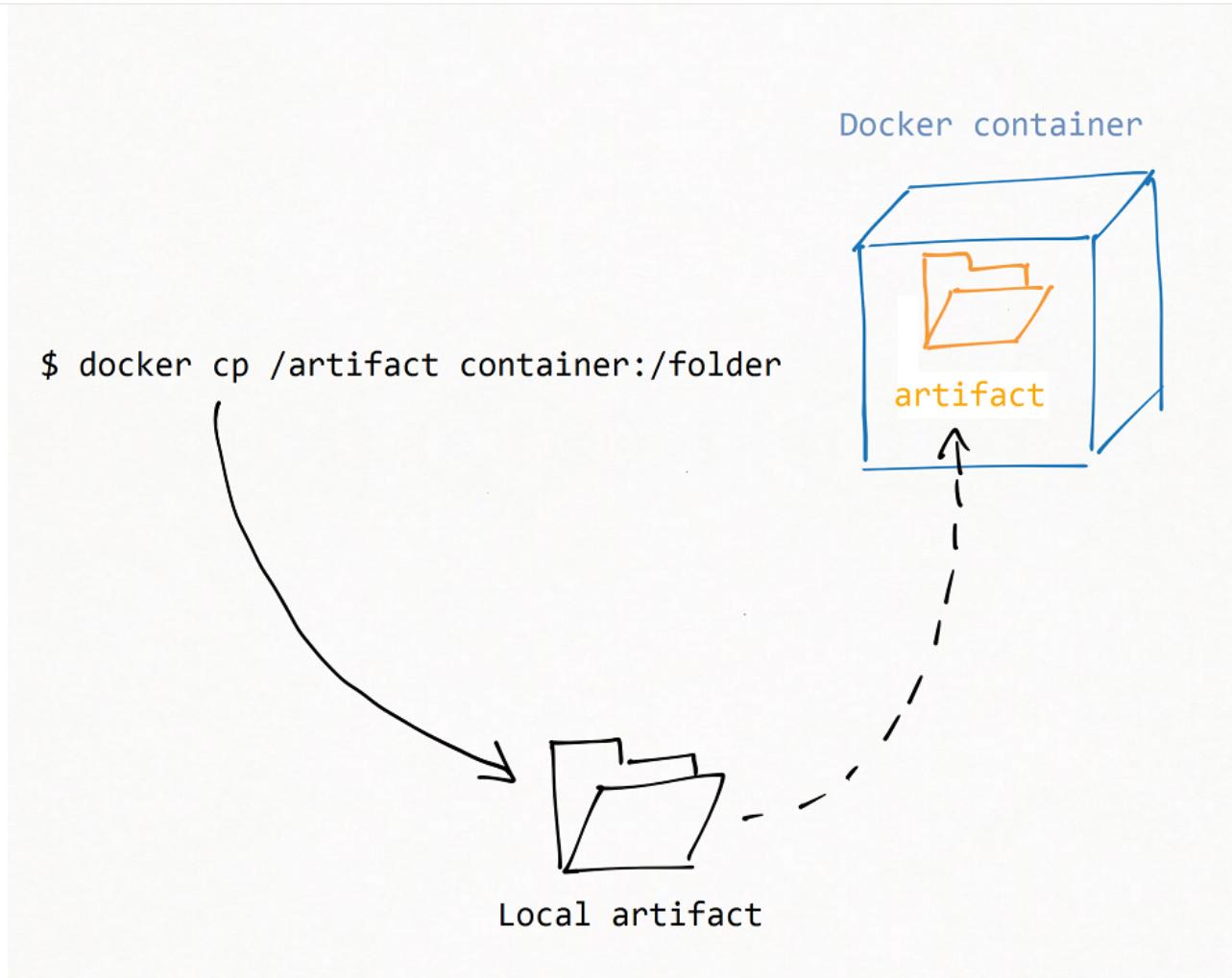
- 0) Copy everything you need into the Deployment Server directory
- 1) Remove existing container (Nginx or Tomcat)



- 2) Run new one from the Docker images collection of the external Deployment Server.



- 3) Add the artifact `/dist` to the "deployable" folder of the Docker container (`/usr/share/nginx/html/`)



Now, let's see how it's being executed in the command line (simplified due to documentation purposes). The next block of code represents what is inside of the last stage of the Pipeline.

```
sshagent (credentials: ['my_ssh_token']) {
    sh """
        // Copy artifact from workspace to deployment server

        // Manage container:
        docker rm -f [mts-container]
        docker run -itd --name=[mts-container] [base_image]
        docker exec [mts-container] bash -C \\\"rm [container_deployment_folder]/*
        \\
        docker cp [artifact] [mts-container]:[container_deployment_folder]
        \\
    """
}
```

For every operation performed in the external Deployment Server, it is necessary to define *where* those commands are going to be executed. So, for each one of previous `docker` commands, this should appear before:

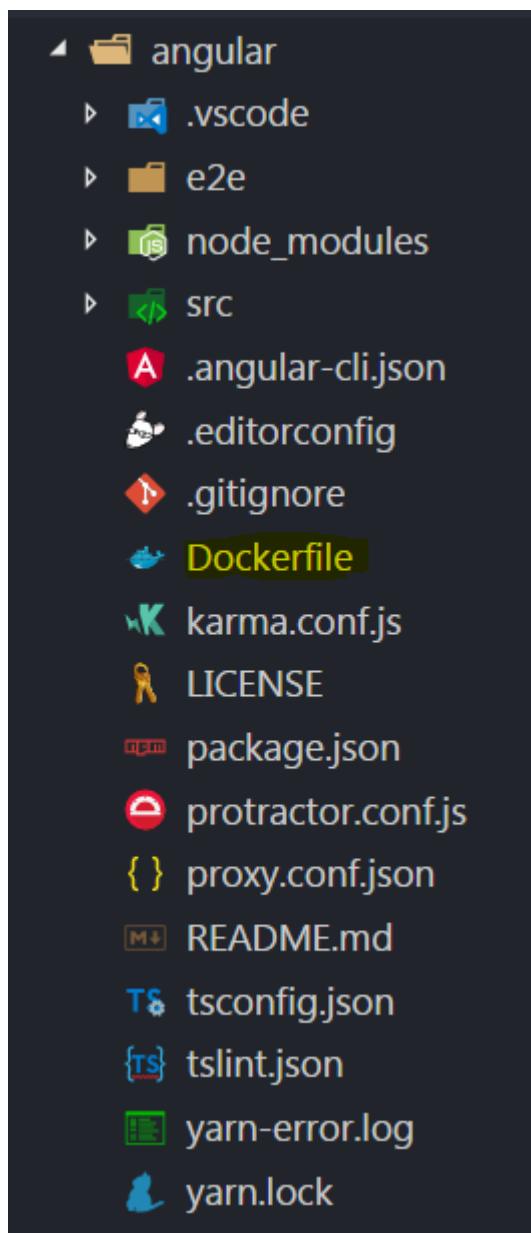
```
ssh -o StrictHostKeyChecking=no root@10.40.235.244
```

Docker Compose

The second way of deployment will be by orchestrating both elements of the application: The Angular client-side and the Java server-side. Both elements will be running in Docker containers as well, but in this case they won't be independent anymore. **Docker Compose** will be in charge of keeping both containers up, or to put them down.

Project adjustment

In order to perform this second way of deployment, some files will be created in the project. The first one is the **Dockerfile** for the Angular client-side. This file will pull (if necessary) an **nginx** Docker image and copy the Angular artifact (`/dist` folder) inside of the deployment folder of the image. It will be located in the main directory of the Angular client-side project.

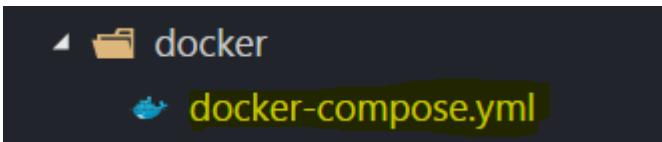


The second file is the **Dockerfile** for the Java server-side. Its function will be quite similar to the Angular one. It will run a **tomcat** Docker image and copy the Java artifact (`mythaistar.war` file) in its deployment folder.



Finally, as long as the **docker-compose** is being used, a file containing its configuration will be necessary as well. A new folder one the main My That Star's directory is created, and it's called **/docker**. Inside there is just a **docker-compose.yml** file. It contains all the information needed to orchestrate the deployment process. For example, which port both containers are going to be published on, and so on. This way of deployment will allow the application to be published or not just with one action.

```
docker-compose rm -f          # down
docker-compose up --build -d    # up fresh containers
```



Let's have a look at the file itself:

```
version: '3'
services:
  client_compose:
    build: "angular"
    ports:
      - "8091:80"
    depends_on:
      - server_compose
  server_compose:
    build: "java"
    ports:
      - "9091:8080"
```

This Orchestrated Deployment will offer some interesting possibilities for [the future of the application](#).

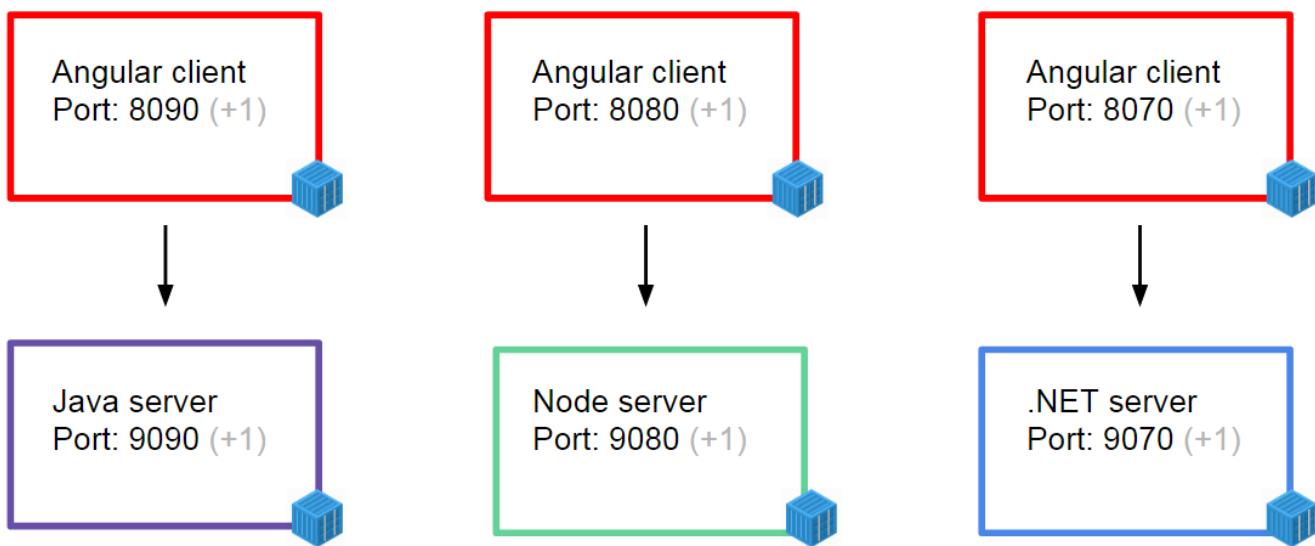
88.2.6. Future Deployment

The **My Thai Star** project is going to be built in many technologies. Thus, let's think about one deployment schema that allow the Angular client to communicate to all three back ends: **Java**, **Node** and **.NET**.

As long as **Docker containers** are being used, it shouldn't be that hard to deal with this "distributed" deployment. The schema represents 6 Docker containers that will have client-side(s) and server-side(s). Each of 3 Angular client containers (those in red) are going to communicate with

different back-ends. So, when the deployment is finished, it would be possible to use all three server-sides just by changing the "port" in the URL.

Let's see how it would look like:



Reverse proxy strategy using Traefik

This implementation is the same as described at [My Thai Star deployment wiki page](#). The only thing that changes is that Traefik is used instead of NGINX.

Using Traefik as reverse proxy, we can define the routes using labels in the docker containers instead of using a nginx.conf file. With this, it is not necessary to modify the reverse proxy container for each application. In addition, as Traefik is listening to the docker daemon, it can detect new containers and create routes for them without rebooting.

Example of labels:

```

labels:
  - "traefik.http.routers.angular.rule=PathPrefix('/')"
  - "traefik.http.services.angular.loadBalancer.healthcheck.path=/health"
  - "traefik.http.services.angular.loadBalancer.healthcheck.interval=10s"
  - "traefik.http.services.angular.loadBalancer.healthcheck.scheme=http"
  
```

How to use it

If you want to build the images from code, change to My Thai Star root folder and execute:

```
$ docker-compose -f docker-compose.traefik.yml up -d --build
```

If you want to build the images from artifacts, change to traefik folder (reverse-proxy/traefik) and execute:

```
$ docker-compose up -d --build
```

After a few seconds, when the healthcheck detects that containers are running, your application will be available at <http://localhost:8090>. Also, the Traefik dashboard is available at <http://localhost:8080>.

If you want to check the behaviour of the application when you scale up the backend, you can execute:

```
$ docker-compose scale java=5
```

With this, the access to the java backend will be using the load balancing method: Weighted Round Robin.

88.3. MyThaiStar on Native Kubernetes as a Service (nKaaS)

The MyThaiStar sample application can be deployed on a nKaaS environment. The required Kubernetes configuration files can be found in the MyThaiStar repository. There are no additional changes required in order to deploy the application.

88.3.1. Setting up the environment

Following the nKaaS guide

After requesting access to the nKaaS platform you'll be greeted with a welcome mail which contains your personal credentials. Make sure to change the given password to a personal one within the 24 hour time period, otherwise the credentials will expire.

After successfully following the [guide](#) mentioned in the welcome mail you should be able to establish a connection to the nKaaS VPN and have access to all their services (Jenkins, BitBucket, etc.). You should also be able to communicate with Kubernetes using kubectl.

Known issues: The nKaaS guide provides a download link for [OpenVPN Connect](#). However, some users experienced connection issues with this client. If you're having issues connecting to the VPN with OpenVPN Connect, you may try out the client by [OVPN](#).

Requesting a namespace

Initially, you won't be able to edit anything on Kubernetes, as you don't have any privileges on any namespace. To request your own namespace you should raise a ticket at the [Customer Support Portal](#) containing your desired name for the namespace.

As soon as the namespace was created you can change your kubectl context:

```
kubectl config set-context --current -namespace=YOUR-NAMESPACE
```

On your own namespace you should have permissions to create/delete deployments/services etc. and perform other actions.

88.3.2. Setting up Harbor

Jenkins will build the MyThaiStar Docker images and push them to the nKaaS [Harbor registry](#). The Jenkinsfile defaults to a Harbor project called "my-thai-star". If there's no such project on Harbor, simply create a new one.

88.3.3. Setting up Jenkins

As MyThaiStar includes all required Jenkinsfiles for nKaaS, almost no configurations have to be performed by the user. Create a new Pipeline on [Jenkins](#) and configure its definition to be a "Pipeline script from SCM". The SCM used is "Git" and the repository URL is the MyThaiStar repository <https://github.com/devonfw/my-thai-star.git> or your fork of it.

The Branch Specifier should point to `*/develop`, the Script Path is `jenkins/nKaaS/Jenkinsfile` as that's where the Jenkinsfile is located at the MyThaiStar repository. Checking the "Lightweight checkout" could speed up the Pipeline.

Note: If you're using the nKaaS [Bitbucket](#) as repository for your MyThaiStar clone you have to perform some additional configurations. First you'll have to create a new SSH keypair, for example with [ssh-keygen](#). Add the public key to the Bitbucket authentication methods and the private key in Jenkins to a new pair of credentials. This step is required for Jenkins to be able to authenticate against Bitbucket. Afterwards, instead of the official MyThaiStar repository, specify your Bitbucket repository:

```
ssh://git@bitbucket.demo.xpaas.io:7999/YOUR-PROJECT/YOUR-MTS-REPO.git
```

Under "Credentials" choose the credentials that contain your Bitbucket private key you've created earlier.

88.3.4. Deploying MTS

After setting up the Jenkins Pipeline, you can simply run it by clicking on the "Build" button. This will trigger the pipeline, Jenkins will:

1. Check out the MTS project
2. Build the docker images
3. Push the docker images to the Harbor registry
4. Deploy the MTS application onto Kubernetes

Finally, the applications should be available at <http://my-thai-star.demo.xpaas.io>.

The first part, `my-thai-star`, is specified in the MTS [ingress](#) configuration at `host`. The second part, `demo.xpaas.io`, is the host of the nKaaS you're working on.

Part XIV: devonfw dashboard

89. Landing page

89.1. Landing page

This is the entry point of the devonfw dashboard. Click on **GET STARTED NOW** to start using it.

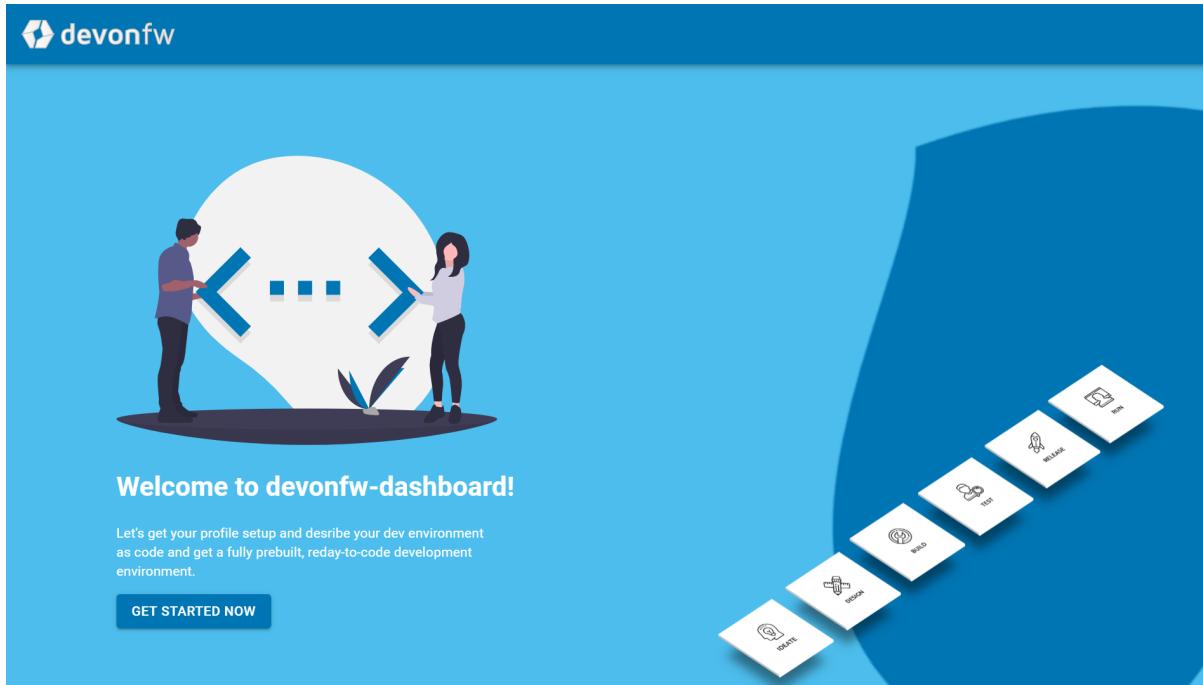
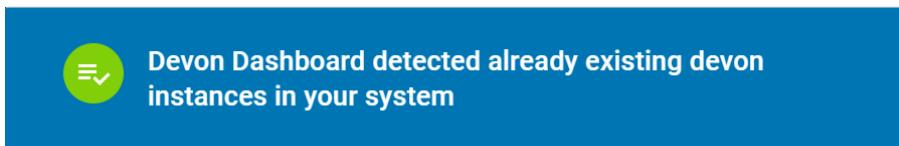


Figure 89. Get Started

89.1.1. Your devonfw distributions

The first time you open the application you will get a dialog with all the devonfw distributions found on your machine. Click on **OK GOT IT** to continue.



The following are the paths of installed devon instances:

Path: c:\Devon\devonfw-ide-scripts-2020.04.004

Version: 2020.04.004

Path: c:\Devon\devonfw-ide-scripts-2020.08.002

Version: 2020.08.002

Path: c:\TEMP

Version: 2020.08.002

OK GOT IT

Figure 90. devon-ide distributions

89.1.2. Profile form

Here you will find a screen that allows you to create a profile. This is just for the purpose of customizing your dashboard.

First off, what's your full name?

Please upload a nice photo of yourself.

Select your avatar for dashboard display.





What's your role in the organisation?

CREATE MY PROFILE **WILL DO IT LATER**

Figure 91. Profile

Fill the data and click on **CREATE MY PROFILE** if you want to create the profile at the moment or click **WILL DO IT LATER** to skip the creation.

90. Home

90.1. Home page

This is the main page that you will find after your profile creation and the page where you will start from henceforth.

It contains three sections:

1. Toolbar
2. Sidebar
3. Content

[[home-page.asciidoc_topbar-+]] == Topbar

This section is at the top of the page, it contains **devonfw instance** dropdown to select devonfw-ide that can be used as a base for the projects.



Next to the **devonfw instance** dropdown, there is a **quick help** icon, clicking on it will open a popup which gives some tips for how to use Devon Dashboard IDE.



[[home-page.asciidoc_sidebar-+]] == Sidebar

The sidebar has divided into two sections:

1. User Profile - Users can see his/her pic, name, and role.
2. Links to access to different sections of the dashboard.



Unknown User
Undefined Role

-  Home
-  Projects
-  IDE's
-  Repositories
-  Wiki
-  Settings

▼

90.1.1. Content Section

The Content section has also divided into three sections:

1. A small introduction about the devonfw IDE
2. A button to **Download latest version** of devonfw IDE
3. A "Project" block which shows the total number of Projects which are available in different devonfw IDE

[[home-page.asciidoc_steps-to-download-and-install-devonfw-ide-+]] == Steps to download and Install devonfw IDE

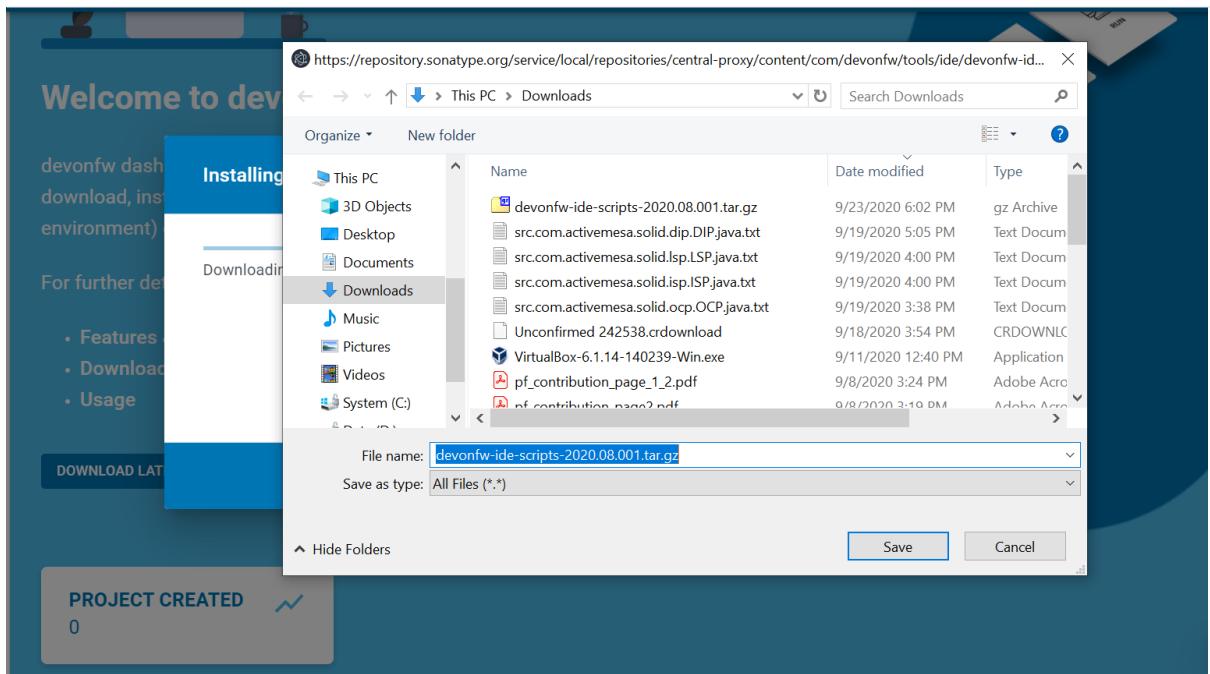
Step 1: Click on **Download latest version** button which is in the Content section. Check the below screen for the reference.

The screenshot shows the devonfw dashboard interface. At the top, there's a header with the devonfw logo and the word "Dashboard". On the left, a sidebar menu includes "Home", "Projects", "IDEs", "Repositories", "Wiki", "Settings", "Account settings", and "Installed versions". The main area features a cartoon character at a desk with a laptop, and the text "Welcome to devonfw dashboard!". Below this, it says "devonfw dashboard uses devonfw-ide, a fantastic tool to automatically download, install, setup and update the IDE (integrated development environment) of your software development projects." It also lists "For further details visit the following links:" with "Features & motivation", "Download & setup", and "Usage". A "DOWNLOAD LATEST VERSION" button is present. A "PROJECT CREATED" box shows "0" with a green checkmark icon.

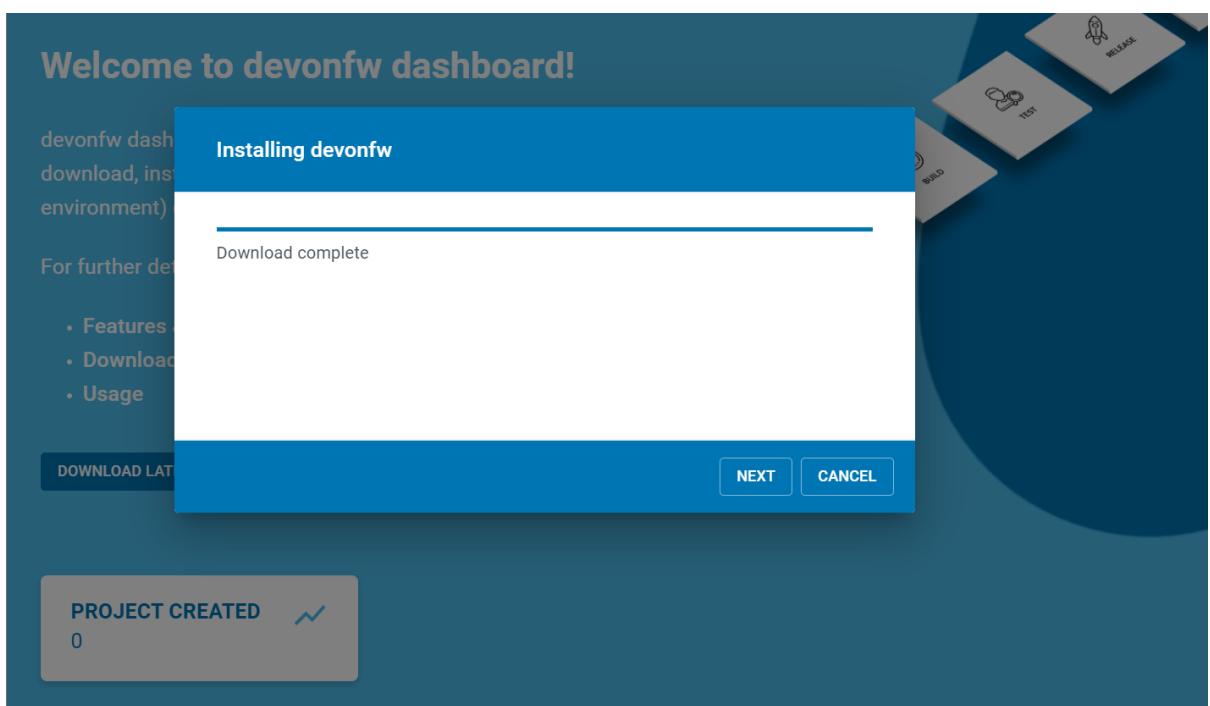
Step 2: By clicking **Download latest version** button, **Installing devonfw** popup will open.

The screenshot shows the "Installing devonfw" progress dialog. The title bar says "Installing devonfw". The main content area shows a progress bar with the text "Downloading...". At the bottom right are "NEXT" and "CANCEL" buttons. The background of the dialog is white, while the rest of the dashboard has a dark blue theme. The "PROJECT CREATED" box at the bottom of the dashboard shows "0" with a green checkmark icon.

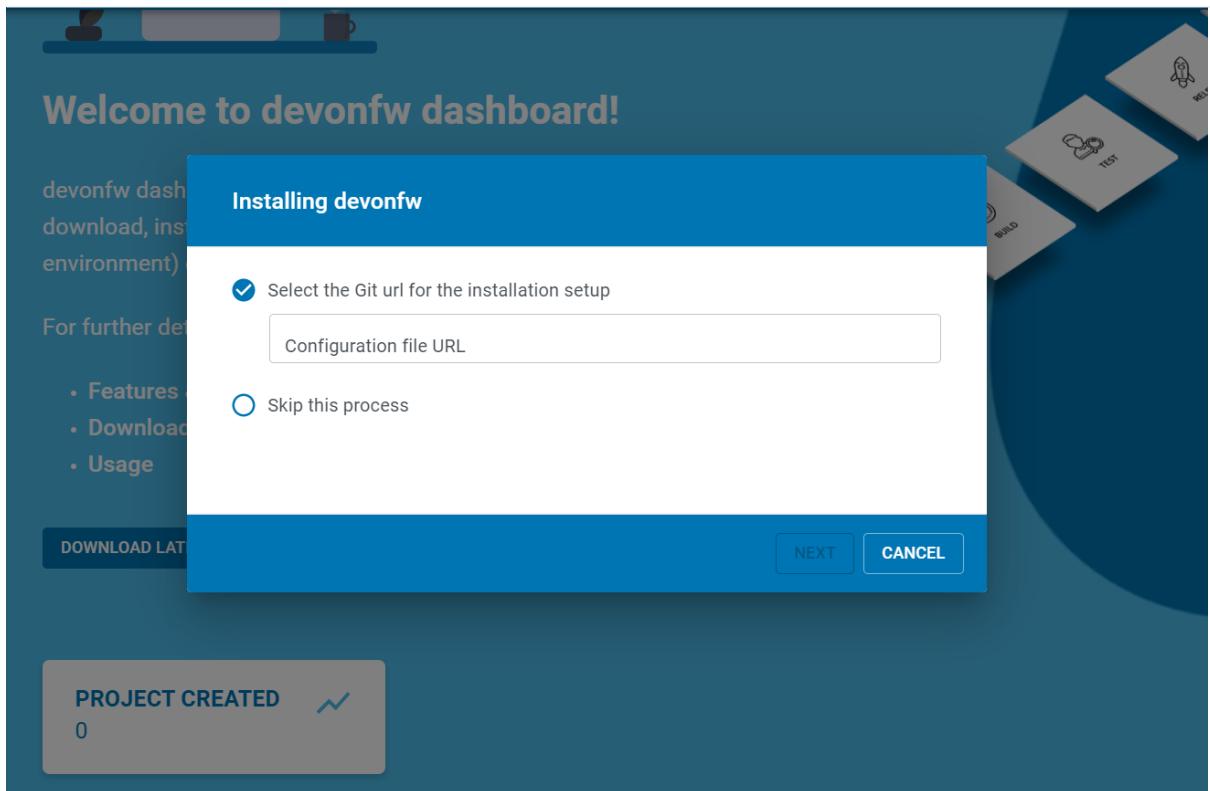
Step 3: **Installing devonfw** popup will automatically trigger one more popup to specify the location for downloading Devonfw IDE. Specify the location and click the **Save** button to download.



Step 3: Once the download completes successfully, the **Next** button will be enabled for the further installation process.



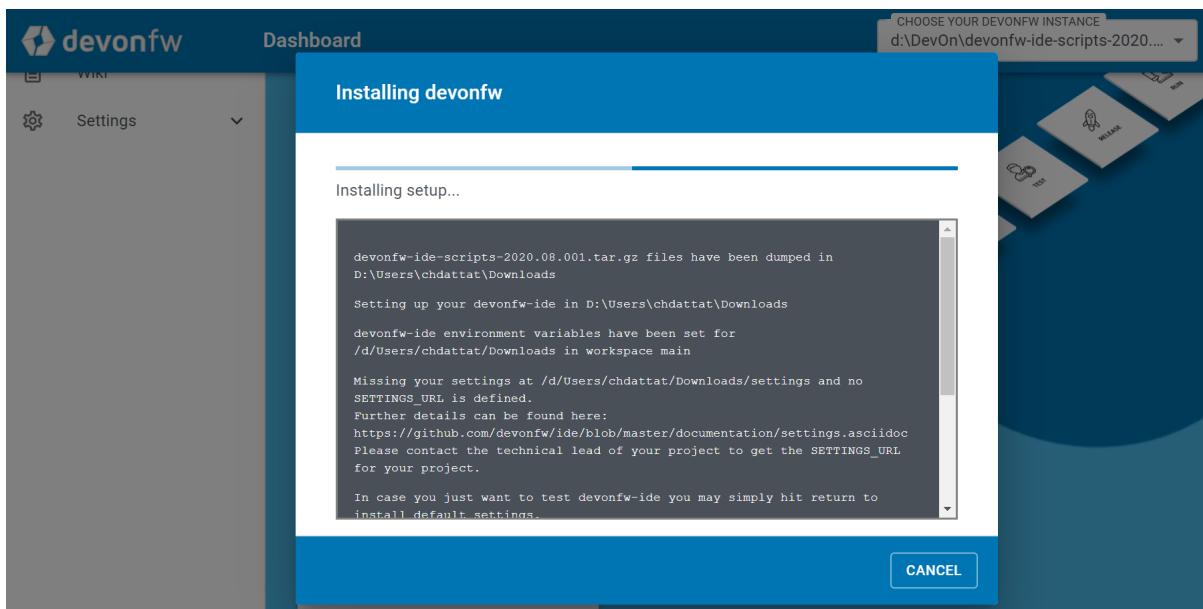
Step 4: By Clicking **Next** button in the **Installing devonfw** pop up, two options are shown:
 1: Select the Git url for the installation setup.
 2: Skip this process.



Step 5: Select one of the above options.

- If the selection is **Git url**, then **Configuration file url** should be filled in the input box and needs to click **Next** button to start the further installation process.
- In case the user doesn't have **Git url**, then simply **Skip the process** and click the **Next** button to start the further installation process.

Step 6: Click on the **Next** button for the final installation process. Wait for some time to complete the installation setup. Once the installation setup completes, the **Close** button will appear. Just click on it and go to the specified folder location.



91. Projects

91.1. Introduction to project management in the dashboard

- The **dashboard** manages multiple projects in multiple workspaces that include Angular, JAVA, and Node.
- The **dashboard** provides rich UI for creating multiple projects, abstracting all the functionality which is usually required while creating an application like opening a command terminal, specifying workspace, and executing commands.
- The **dashboard** makes it easy to see all the projects which are in different devonfw-ide workspace, just by changing the "devonfw Instance" dropdown.
- The **dashboard** makes it very easy to open a project in a different IDE like Visual Studio or Eclipse respectively just by right click on the Project folder and open option.
- The **dashboard** also makes it easy to delete the project, explore the project location.

91.2. Projects

Click on the **Projects** link on the sidebar to navigate to the project's screen. The screen displays all the projects in the currently selected devonfw-ide, grouped by the workspaces in which they exist.
Note: Currently it only displays projects created through the dashboard.

The screenshot shows the devonfw IDE dashboard with the following details:

- Header:** CHOOSE YOUR DEVONFW INSTANCE: c:\Devon\devonfw-ide-scripts-2020... ▾ ?
- User Profile:** Unknown User Undefined Role
- Left Sidebar:**
 - Home
 - Projects**
 - IDE's
 - Repositories
 - Wiki
 - Settings

LATEST VERSION 2020.12.001 ⓘ
Get IDE fixes and more Features
UPDATE NOW
- Dashboard Content:**
 - 4 Projects**
 - Add New Project** button
 - workspace main**: devon4ng-mat-layout-scss2, Last Updated 4/11/2020
 - workspace stable** (highlighted):
 - buffer**, Last Updated 10/12/2020
 - super-important-ng-** (context menu open):
 - Open
 - In Visual Studio Code
 - Enclosing Folder
 - Delete
 - ngtest**, Last Updated 17/10/2020
 - workspace test**
- Bottom:** Projects per page: 10 ▾ 1-4 of 4 < >

- It shows the total number of projects available in each **devonfw-ide**.
- Filtering and searching the projects.
- Add New Project** - For creating a Project.
- Project folder which gives information about the project like which technology the project belongs to, the name of the project, and when it has created.
- There are many operations that are available on right-click on **Project folder** they are :
 - Opening a project in different IDE (Visual Studio or Eclipse)
 - Enclosing Folder, and
 - Deleting the project.
- Users can see projects of different **devonfw-ide** workspace just by changing the option in the **devonfw instance** dropdown which is set globally at the top of the screen.

Click on **Add New Project** to start creating a new project.

91.3. How to create a project

Three main steps are involved in creating any devonfw project. They are:

Step 1. Project Type

In this first step the user has to choose the language technology to start the project with, e.g. **Angular**, **Java** or **Node** and click the **Next** button for to continue to the next step.

The screenshot shows the 'Projects > New Project' screen. At the top, there's a navigation bar with 'Dashboard' and a dropdown for 'CHOOSE YOUR DEVONFW INSTANCE'. On the left, a sidebar shows 'Unknown User' and 'Undefined Role' with links for Home, Projects, IDE's, Repositories, Wiki, and Settings. The main area has a title 'Project type' with three numbered steps: 1 (Project type), 2 (Project data), and 3 (Execution). Below the steps is a note: 'Choose the technology in the below section, dolor sit amet, consectetur adipiscing elit. Phasellus non tincidunt velit. Quisque laoreet, lectus id tincidunt fringilla, eros est bibendum felis, sit amet lobortis ante sem non diam. Donec viverra a nisi eu eleifend. Mauris vel leo tempor, commodo felis in, sollicitudin velit.' Three cards are shown: 'DEVON ANGULAR' with an 'A' icon, 'DEVON JAVA' with a coffee cup icon, and 'DEVON NODE' with a node.js logo icon. At the bottom are 'BACK' and 'NEXT' buttons.

Step 2. Project Data

After the **Project type** selection, the second screen will appear for the user to fill up all the required fields. User can select the workspace in the active devonfw-ide for the project in this step. Once the user enters all the required fields, the **Next** button will be enabled for the final step.

The screenshot shows the 'Project data' step of the wizard. A checkmark is next to the 'Project type' step. The 'Project data' step is highlighted with a blue circle containing the number '2'. The 'Execution' step is shown with a grey circle containing the number '3'. Below the steps is a note: 'Fill in the data to configure your new project. The dashboard will create the project with the values you put here.' On the left, there's a 'DEVON ANGULAR' icon with an 'A' inside a shield shape. The form fields include: 'Project name *' (empty input field), 'Routing?' (dropdown menu showing 'Yes'), 'Styling?' (dropdown menu showing 'SCSS'), and 'Workspace' (dropdown menu showing 'main'). At the bottom are 'BACK' and 'NEXT' buttons.

User can change the **devonfw-ide** workspace where the project is going to generate, just by changing the option in the **devonfw instance** dropdown which is set globally in the header of the dashboard.



Step 3. Execution

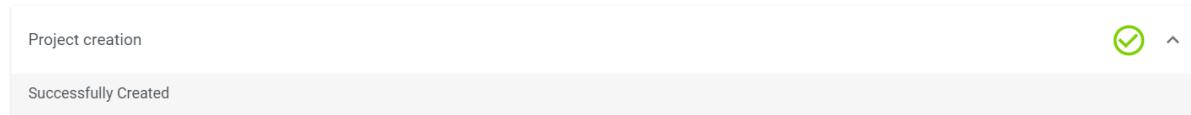
The execution step takes all the user entered data from the **Project Data** step and executes the respective commands to generate the project.

Execution has divided into two sections:

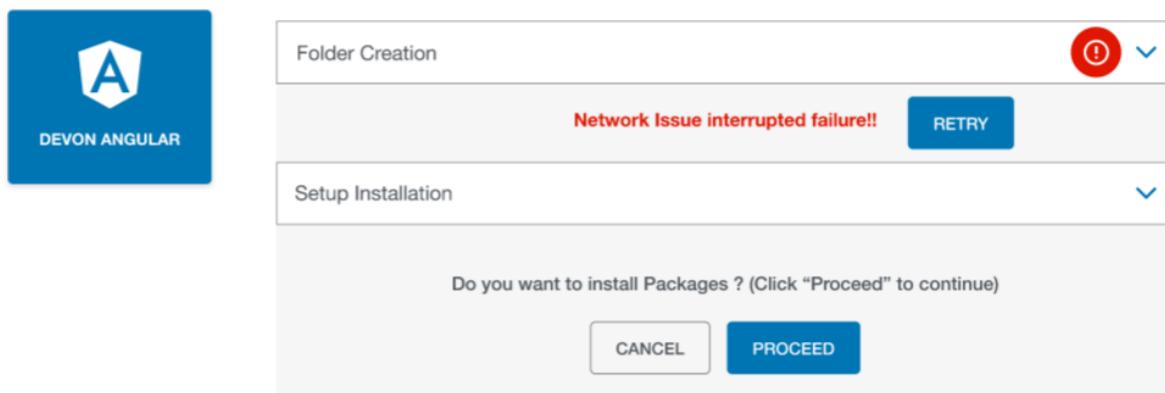
- Creation
- Setup Installation

3.1 Creation

- Creates only source code and notify the user if the project creation fails or success.



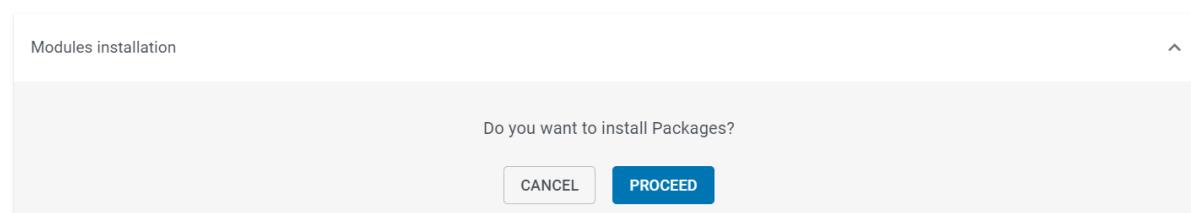
- In case any network issue or technical issue and the user wants to re-run the **Project execution** process, then the **Retry** button will help to start the process again.



3.2 Setup installation

Allows user to install the dependencies of application (maven modules for java, node modules for node, angular) by clicking **Proceed** button.

The installation can be skipped by clicking **cancel** button.



Step 4. Click on **Finish** button to go to **Project Details Screen**.

92. Repositories

92.1. Repositories

This page lists the different repositories under devonfw organization.

The screenshot shows the devonfw dashboard interface. On the left, there is a sidebar with navigation links: Home, Projects, IDE's, **Repositories**, Wiki, and Settings. A message at the top of the sidebar says "LATEST VERSION 2020.08.001 ⓘ Get IDE fixes and more Features" and a "UPDATE NOW" button. The main area is titled "Dashboard" and contains a search bar labeled "Search your repository". Below the search bar is a list of repositories, each with a name, a brief description, a "COPY GITHUB URL" button, and an "OPEN REPOSITORY" button. The repositories listed are:

- devon4j**: devonfw Java stack - create enterprise-grade business apps in Java safe and fast. Description: devonfw Java stack - create enterprise-grade business apps in Java safe and fast.
- my-thai-star**: My Thai Star reference application for devonfw. Description: My Thai Star reference application for devonfw.
- cobigen**: Code-based Incremental Generator. Description: Code-based Incremental Generator.
- mrchecker**: End to End (E2E) test framework - MrChecker. Description: End to End (E2E) test framework - MrChecker.
- jump-the-queue**: Source code used in the different devonfw tutorials. Description: Source code used in the different devonfw tutorials.
- devon4ng**: devonfw Angular stack - create enterprise-grade business apps in Angular safe and fast. Description: devonfw Angular stack - create enterprise-grade business apps in Angular safe and fast.
- devon4node**: devonfw node.js stack - create enterprise-grade business apps in node.js safe and fast. Description: devonfw node.js stack - create enterprise-grade business apps in node.js safe and fast.

Figure 92. Repositories

The list updates as you type in the search bar.

The screenshot shows a search interface for repositories. At the top, there is a search bar with the placeholder "Search your repository" and a blue outline around the input field. Below the search bar, the word "ide" is typed into it. The search results are displayed in a list of cards:

- ide**: Tool to automate setup and update of development environment (esp. for Java projects).
COPY GITHUB URL **OPEN REPOSITORY**
- ide-settings**: Settings for devonfw-ide
COPY GITHUB URL **OPEN REPOSITORY**
- swtbot**: devonfw ide integration tests
COPY GITHUB URL **OPEN REPOSITORY**
- katacoda-scenarios-ide-settings**: Minimized IDE Settings for Katacoda Scenarios
COPY GITHUB URL **OPEN REPOSITORY**

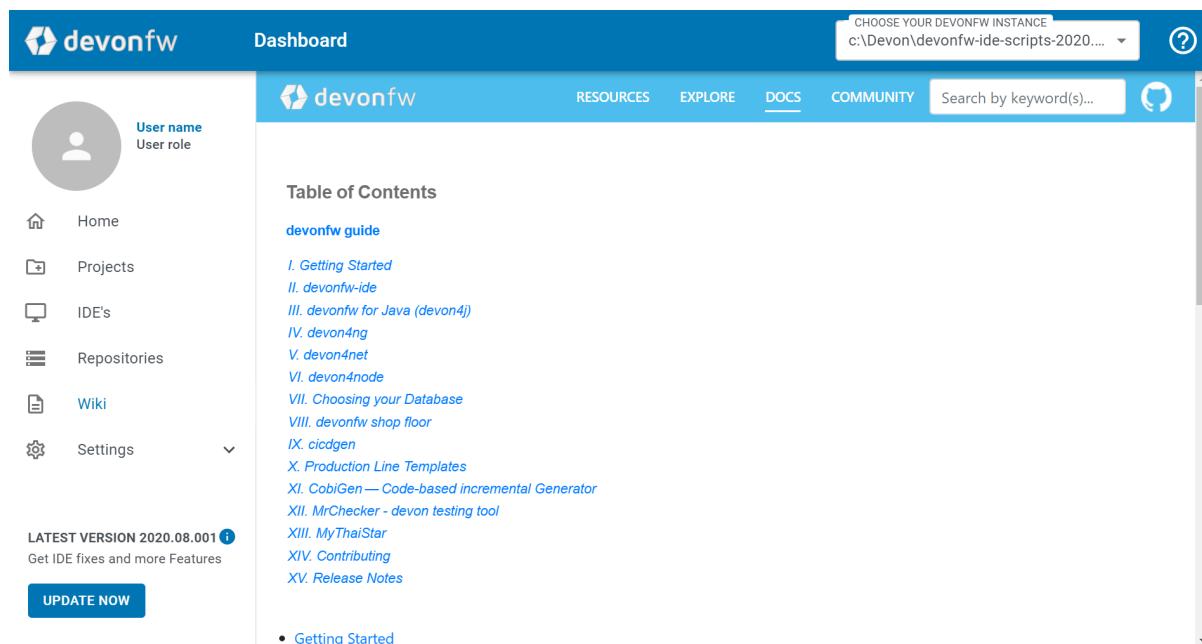
Figure 93. Search Repositories

- You can click **COPY GITHUB URL** for any of the repository list item to copy its github URL to your clipboard and clone it locally.
- You can also click the **OPEN REPOSITORY** button to view its github repository page in your default browser.

93. Wiki

93.1. Wiki page.

This page displays the documentation of devonfw. You can also find it at <https://devonfw.com/>



The screenshot shows the devonfw Wiki interface. On the left, there's a sidebar with navigation links: Home, Projects, IDE's, Repositories, Wiki (which is selected), and Settings. Below these are notices for 'LATEST VERSION 2020.08.001' and a 'UPDATE NOW' button. The main content area has a header 'Table of Contents' and lists various sections of the wiki guide, such as 'devonfw guide', 'Getting Started', 'devonfw for Java (devon4j)', 'devon4ng', 'devon4net', 'devon4node', 'Choosing your Database', 'devonfw shop floor', 'cicdgen', 'Production Line Templates', 'CobiGen — Code-based incremental Generator', 'MrChecker - devon testing tool', 'MyThaiStar', 'Contributing', and 'Release Notes'. A search bar at the top right contains the placeholder 'Search by keyword(s)...'.

Figure 94. Wiki

94. Settings

94.1. Settings

94.1.1. Account settings

Here you get a screen that allows you to create a profile. This is the same screen which you see during the intial setup of the dashboard. It is completely optional.

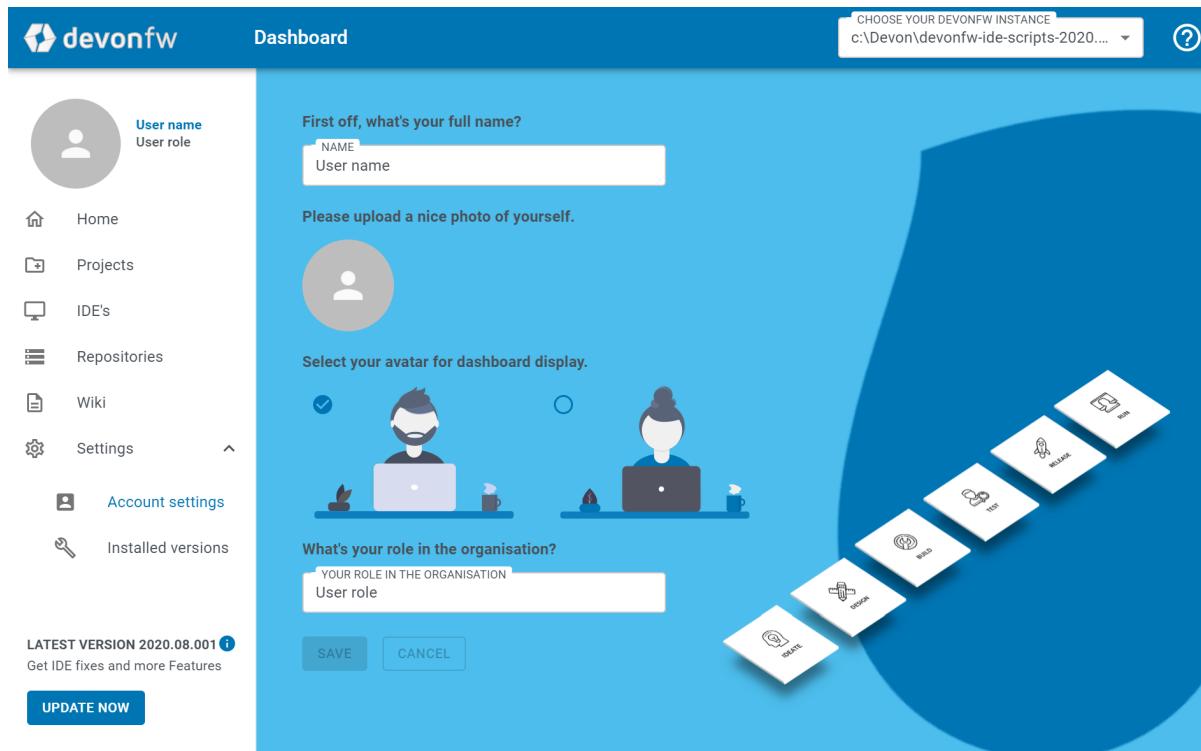


Figure 95. Account settings

Fill the data and click on **Save** if you want to create the profile.

94.1.2. Installed versions

The installed versions subsection allows you to manage the different versions of devonfw-ide available.

The screenshot shows the 'Dashboard' page of the devonfw IDE. On the left sidebar, there's a user profile icon and a list of navigation items: Home, Projects, IDE's, Repositories, Wiki, Settings (with a dropdown arrow), Account settings, and Installed versions (which is currently selected). Below these is a message about the latest version (2020.08.001) and a 'UPDATE NOW' button. The main content area is titled 'INSTALLED TOOLS devonfw IDE versions'. It contains a table with columns: VERSION NAMES, RELEASE DATE, RELEASE NOTES, UPDATE, and DOWNLOAD. The table lists several versions, each with a tooltip eye icon, a 'Consolidated list of features' link, and 'UPDATE' or 'UNINSTALL' buttons. A search bar at the top right says 'Search versions'. At the bottom of the table, there are pagination controls for 'Rows per page' (set to 5), '1-5 of 11', and arrows.

VERSION NAMES	RELEASE DATE	RELEASE NOTES	UPDATE	DOWNLOAD
2020.04.004	28-Jul-2020	Consolidated list of features	UPDATE	UNINSTALL
2020.04.003	11-Jul-2020	Consolidated list of features	UPDATE	UNINSTALL
2020.08.001	31-Aug-2020	Consolidated list of features	UPDATE	DOWNLOAD
2020.04.004	28-Jul-2020	Consolidated list of features	UPDATE	DOWNLOAD
2020.04.003	11-Jul-2020	Consolidated list of features	UPDATE	DOWNLOAD

Figure 96. Installed versions

- It lists the devonfw-ide you have installed in your system, along with the ones available for download from our maven repository
- If you want to install specific version, you can search it here and **DOWNLOAD** it
- To check the release notes for a version, simply click on **Consolidated list of features**
- For the installed versions:
 - Hovering over the eye icon shows you the path for the devonfw-ide in a tooltip
 - You can view it in your system explorer by clicking the eye icon
 - You can update its settings and softwares by clicking on **UPDATE**
 - You can also **UNINSTALL** an installed version, after which the dashboard will no longer keep track of the projects and IDEs belonging to that devonfw-ide

Part XV: Solicitor User Guide

SPDX-License-Identifier: Apache-2.0

95. Introduction

Todays software projects often make use of large amounts of Open Source software. Being compliant with the license obligations of the used software components is a prerequisite for every such project. This results in different requirements that the project might need to fulfill. Those requirements can be grouped into two main categories:

- Things that need to be done to actually fulfill license obligations
- Things that need to be done to monitor / report fulfillment of license obligations

Most of the above activities share common points:

- The need to have an inventory of used (open source) components and their licenses
- Some rule based evaluation and reporting based on this inventory

In practice these easy looking tasks might get complex due to various aspects:

- The number of open source components might be quite large (>> 100 for a typical webapplication based on state of the art programming frameworks)
- Agile development and rapid changes of used components result in frequent changes of the inventory
- Open Source usage scenarios and license obligations might be OK in one context (e.g. in the relation between a software developer and his client) but might be completely unacceptable in another context (e.g. when the client distributes the same software to end customers)
- Legal interpretation of license conditions often differ from organisation to organisation and result in different compliance rules to be respected.
- License information for components is often not available in a standardized form which would allow automatic processing
- Tools for supporting the license management processes are often specific to a technology or build tool and do not support all aspects of OSS license management.

Of course there are specific commercial tool suites which address the IP rights and license domain. But due to high complexity and license costs those tools are out of reach for most projects - at least for permanent use.

Solicitor tries to address some of the issues hightlighted above. In its initial version it is a tool for programmatically executing a process which was originally defined as an Excel-supported manual process.

When running *Solicitor* three subsequent processing steps are executed:

- Creating an initial component and license inventory based on technology specific input files
- Rule based normalization and evaluation of licenses
- Generation of output documents



Solicitor comes with a set of **sample** rules for the normalization and evaluation of licenses. Even though these included rules are not "intentionally wrong" they are only samples and **you should never rely on these builtin rules without checking and possibly modifying their content and consulting your lawyer.** *Solicitor* is a tool for technically supporting the management of OSS licenses within your project. *Solicitor* neither gives legal advice nor is a replacement for a lawyer.

95.1. Licensing of Solicitor

The *Solicitor* code and accompanying resources (including this userguide) as stored in the GIT Repository <https://github.com/devonfw/solicitor> are licensed as Open Source under Apache 2 license (<https://www.apache.org/licenses/LICENSE-2.0>).



Specifically observe the "Disclaimer of Warranty" and "Limitation of Liability" which are part of the license.



The executable JAR file which is created by the Maven based build process includes numerous other Open Source components which are subject to different Open Source licenses. Any distribution of the *Solicitor* executable JAR file needs to comply with the license conditions of all those components. If you are running *Solicitor* from the executable JAR you might use the `-eug` option to store detailed license information as file `solicitor_licenseinfo.html` in your current working directory (together with a copy of this user guide).

96. Architecture

The following picture show a business oriented view of *Solicitor*.

[domain model]

Raw data about the components and attached licenses within an application is gathered by *scanning* with technology and build chain specific tools. This happens outside *Solicitor*.

The *import* step reads this data and transforms it into a common technology independent internal format.

In the *normalization* step the license information is completed and unified. Information not contained in the raw data is added. Where possible the applicable licenses are expressed by [SPDX-IDs](#).

Many open source components are available via multi licensing models. Within *qualification* the finally applicable licenses are selected.

In the *legal assessment* the compliance of applicable licenses will be checked based on generic rules defined in company wide policies and possibly project specific project specific extensions. Defining those rules is considered as "legal advice" and possibly needs to be done by lawyers which are authorized to do so. For this step *Solicitor* only provides a framework / tool to support the process here but does not deliver any predefined rules.

The final *export* step produces documents based on the internal data model. This might be the list of licenses to be forwarded to the customer or a license compliance report. Data might also be fed into other systems.

A more technical oriented view of *Solicitor* is given below.

[solution]

There are three major technical components: The *reader* and *writer* components are performing import and export of data. The business logic - doing *normalization*, *qualification* and *legal assessment* is done by a *rule engine*. Rules are mainly defined via *decision tables*. *Solicitor* comes with a starting set of rules for *normalization* and *qualification* but these rulesets need to be extended within the projects. Rules for legal evaluation need to be completely defined by the user.

Solicitor is working without additional persisted data: When being executed it generates the output directly from the read input data after processing the business rules.

96.1. Data Model

[datamodel]

The internal business data model consists of 6 entities:

- **ModelRoot**: root object of the business data model which holds metadata about the data processing

- **Engagement:** the masterdata of the overall project
- **Application:** a deliverable within the **Engagement**
- **ApplicationComponent:** component within an **Application**
- **RawLicense:** License info attached to an **ApplicationComponent** as it is read from the input data
- **NormalizedLicense:** License info attached to an **ApplicationComponent** processed by the business rules

96.1.1. ModelRoot

Property	Type	Description
modelVersion	int	version number of the data model
executionTime	String	timestamp when the data was processed
solicitorVersion	String	Solicitor version which processed the model
solicitorGitHash	String	buildnumber / GitHash of the Solicitor build
solicitorBuildDate	String	build date of the Solicitor build
extensionArtifactId	String	artifactId of the active <i>Solicitor</i> Extension ("NONE" if no extension)
extensionVersion	String	Version of the active Extension (or "NONE")
extensionGitHash	String	Buildnumber / GitHash of the Extension (or "NONE")

96.1.2. Engagement

Property	Type	Description
engagementName	String	the engagement name
engagementType	Engage mentTyp e	the engagement type; possible values: INTERN, EXTERN
clientName	String	name of the client
goToMarketModel	GoToMa rketMod el	the go-to-market-model; possible values: LICENSE
contractAllowsOss	boolean	does the contract explicitly allow OSS?
ossPolicyFollowed	boolean	is the companies OSS policy followed?
customerProvidesOss	boolean	does the customer provide the OSS?

96.1.3. Application

Property	Type	Description
applicationName	String	the name of the application / deliverable

Property	Type	Description
releaseId	String	version identifier of the application
releaseDate	String	release date of the application
sourceRepo	String	URL of the source repo of the application (should be an URL)
programmingEcosystem	String	programming ecosystem (e.g. Java8; Android/Java, iOS / Objective C)

96.1.4. ApplicationComponent

Property	Type	Description
usagePattern	UsagePattern	possible values: DYNAMIC_LINKING, STATIC_LINKING, STANDALONE_PRODUCT
ossModified	boolean	is the OSS modified?
ossHomepage	String	URL of the OSS homepage
groupId	String	component identifier: maven groupId
artifactId	String	component identifier: maven artifactId
version	String	component identifier: Version
repoType	String	component identifier: RepoType

96.1.5. RawLicense

Property	Type	Description
declaredLicense	String	name of the declared license
licenseUrl	String	URL of the declared license
trace	String	detail info of history of this data record
specialHandling	boolean	(for controlling rule processing)

96.1.6. NormalizedLicense

Property	Type	Description
declaredLicense	String	name of the declared license (copied from RawLicense)
licenseUrl	String	URL of the declared license (copied from RawLicense)
declaredLicenseContent	String	resolved content of licenseUrl
normalizedLicenseType	String	type of the license, see License types
normalizedLicense	String	name of the license in normalized form (SPDX-Id) or special "pseudo license id", see Pseudo License Ids
normalizedLicenseUrl	String	URL pointing to a normalized form of the license

Property	Type	Description
normalizedLicenseType	String	type of the license, see License types
effectiveNormalizedLicense Type	String	type of the effective license, see License types
effectiveNormalizedLicense	String	effective normalized license (SPDX-Id) or "pseudo license id"; this is the information after selecting the right license in case of multi licensing or any license override due to a component being redistributed under a different license
effectiveNormalizedLicense Url	String	URL pointing to the effective normalized license
effectiveNormalizedLicense Content	String	resolved content of effectiveNormalizedLicenseUrl
legalPreApproved	String	indicates whether the license is pre approved based on company standard policy
copyLeft	String	indicates the type of copyleft of the license
licenseCompliance	String	indicates if the license is compliant according to the default company policy
licenseRefUrl	String	URL to the reference license information (TBD)
licenseRefContent	String	resolved content of licenseRefUrl
includeLicense	String	does the license require to include the license text ?
includeSource	String	does the license require to deliver source code of OSS component ?
reviewedForRelease	String	for which release was the legal evaluation done?
comments	String	comments on the component/license (mainly as input to legal)
legalApproved	String	indicates whether this usage is legally approved
legalComments	String	comments from legal, possibly indicating additional conditions to be fulfilled
trace	String	detail info of history of this data record (rule executions)

For the mechanism how *Solicitor* resolves the content of URLs and how the result might be influenced see [Resolving of License URLs](#)

License types

Defines the type of license

- **OSS-SPDX** - An OSS license which has a corresponding SPDX-Id
- **OSS-OTHER** - An OSS license which has no SPDX-Id
- **COMMERCIAL** - Commercial (non OSS) license; this might also include code which is owned by the project

-
- **UNKNOWN**- License is unknown
 - **IGNORED**- license will be ignored (non selected license in multi licensing case; only to be used as "Effective Normalized License Type")

Pseudo License Ids

A "normalized" license id might be either a SPDX-Id or a "pseudo license id" which is used to indicate a specific situation. The following pseudo license ids are used:

- **OSS specific** - a nonstandard OSS license which could not be mapped to a SPDX-Id
- **PublicDomain** - any form of public domain which is not represented by an explicit SPDX-Id
- **Ignored** - license will be ignored (non selected license in multi licensing case; only to be used as "Effective Normalized License")
- **NonOSS** - commercial license, not OSS

97. Usage

97.1. Executing Solicitor

Solicitor is a standalone Java (Spring Boot) application. Prerequisite for running it is an existing Java 8 or 11 runtime environment. If you do not yet have the *Solicitor* executable JAR (`solicitor.jar`) you need to build it as given on the project GitHub homepage <https://github.com/devonfw/solicitor>.

Solicitor is executed with the following command:

```
java -jar solicitor.jar -c <configfile>
```

where `<configfile>` is to be replaced by the location of the [Project Configuration File](#).

To get a first idea on what *Solicitor* does you might call

```
java -jar solicitor.jar -c classpath:samples/solicitor_sample.cfg
```

This executes *Solicitor* with default configuration on its own list of internal components and produces sample output.

To get an overview of the available command line options use

```
java -jar solicitor.jar -h
```

Addressing of resources

For unique addressing of resources **to be read** (configuration files, input data, rule templates and decision tables) *Solicitor* makes use of the Spring ResourceLoader functionality, see <https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#resources-resourceloader>. This allows to load from the classpath, the filesystem or even via http get.

If you want to reference a file in the filesystem you need to write it as follows: `file:path/to/file.txt`

Note that this only applies to resources being read. Output files are addressed without that prefix.

97.2. Project Configuration File

The project configuration of *Solicitor* is done via a configuration file in JSON format. This configuration file defines the engagements and applications master data, configures the readers for

importing component and license information, references the business rules to be applied and defines the exports to be done.

The config file has the following skeleton:

```
{
  "version" : 1,
  "comment" : "Sample Solicitor configuration file",
  "engagementName" : "DevonFW", ①
  .
  .
  .
  "applications" : [ ... ], ②
  "rules" : [ ... ], ③
  "writers" : [ ... ] ④
}
```

① The leading data defines the engagement master data, see [Header and Engagement Master Data](#)

② `applications` defines the applications within the engagement and configures the readers to import the component/license information, see [Applications](#)

③ `rules` references the rules to apply to the imported data, see [Business Rules](#)

④ `writers` configures how the processed data should be exported, see [Writers / Reporting](#)



The following section describes all sections of the *Solicitor* configuration file format. Often the configuration of `writers` and especially `rules` will be identical for projects. To facilitate the project specific configuration setup *Solicitor* internally provides a base configuration which contains reasonable defaults for the `rules` and `writers` section. If the project specific configuration file omits the `rules` and/or `writers` sections then the corresponding settings from the base configuration will be taken. For details see [Default Base Configuration](#).



If locations of files are specified within the configuration files as relative pathnames then this is always evaluated relative to the current working directory (which might differ from the location of the configuration file). If some file location should be given relative to the location of the configuration file this might be done using the special placeholder `#{cfgdir}` as described in the following.

97.2.1. Placeholders within the configuration file

Within certain parts of the configuration file (path and filenames) special placeholders might be used to parameterize the configuration. These areas are explicitly marked in the following description.

These placeholders are available:

- `#{project}` - A simplified project name (taking the engagement name, removing all non-word characters and converting to lowercase).

- `#{cfgdir}` - If the config file was loaded from the filesystem this denotes the directory where the config file resides, `.` otherwise. This can be used to reference locations relative to the location of the config file.

97.2.2. Header and Engagement Master Data

The leading section of the config file defines some metadata and the engagement master data.

```
"version" : 1, ①
"comment" : "Sample Solicitor configuration file", ②
"engagementName" : "DevonFW", ③
"engagementType" : "INTERN", ④
"clientName" : "none", ⑤
"goToMarketModel" : "LICENSE", ⑥
"contractAllowsOss" : true, ⑦
"ossPolicyFollowed" : true, ⑧
"customerProvidesOss" : false, ⑨
```

① version of the config file format (currently needs to be 1)

② is a free text comment (no further function at the moment)

③ the engagement name (any string)

④ the engagement type; possible values: INTERN, EXTERN

⑤ name of the client (any string)

⑥ the go-to-market-model; possible values: LICENSE

⑦ does the contract explicitly allow OSS? (boolean)

⑧ is the companies OSS policy followed? (boolean)

⑨ does the customer provide the OSS? (boolean)

97.2.3. Applications

Within this section the different applications (=deliverables) of the engagement are defined. Furtheron for each application at least one reader needs to be defined which imports the component and license information.

```

"applications" : [ {
  "name" : "Devon4J", ①
  "releaseId" : "3.1.0-SNAPSHOT", ②
  "sourceRepo" : "https://github.com/devonfw/devon4j.git", ③
  "programmingEcosystem" : "Java8", ④
  "readers" : [ { ⑤
    "type" : "maven", ⑥
    "source" : "classpath:samples/licenses_devon4j.xml", ⑦ ⑩
    "usagePattern" : "DYNAMIC_LINKING", ⑧
    "repoType" : "maven" ⑨
  } ]
}, ],

```

- ① The name of the application / deliverable (any string)
- ② Version identifier of the application (any string)
- ③ URL of the source repo of the application (string; should be an URL)
- ④ programming ecosystem (any string; e.g. Java8; Android/Java, iOS / Objective C)
- ⑤ multiple readers might be defined per application
- ⑥ the type of reader; for possible values see [Reading License Information with Readers](#)
- ⑦ location of the source file to read (ResourceLoader-URL)
- ⑧ usage pattern; possible values: DYNAMIC_LINKING, STATIC_LINKING, STANDALONE_PRODUCT
- ⑨ repoType: Repository to download the sources from: currently possible values: maven, npm; if omitted then "maven" will be taken as default
- ⑩ *placeholder patterns might be used here*

The different readers are described in chapter [Reading License Information with Readers](#)

97.2.4. Business Rules

Business rules are executed within a [Drools](#) rule engine. They are defined as a sequence of rule templates and corresponding XLS files which together represent decision tables.

```

"rules" : [ {
    "type" : "dt", ①
    "optional" : false, ②
    "ruleSource" : "classpath:samples/LicenseAssignmentSample.xls", ③ ⑦
    "templateSource" : "classpath:com/.../rules/rule_templates/LicenseAssignment.drt",
    ④ ⑦
    "ruleGroup" : "LicenseAssignment", ⑤
    "description" : "setting license in case that no one was detected" ⑥
},
.
.
.
,
{
    "type" : "dt",
    "optional" : false,
    "ruleSource" : "classpath:samples/LegalEvaluationSample.xls",
    "templateSource" : "classpath:com/.../rules/rule_templates/LegalEvaluation.drt",
    "ruleGroup" : "LegalEvaluation",
    "decription" : "final legal evaluation based on the rules defined by legal"
} ],

```

- ① type of the rule; only possible value: `dt` which stands for "decision table"
- ② if set to `true` the processing of this group of rules will be skipped if the XLS with table data (given by `ruleSource`) does not exist; if set to `false` a missing XLS table will result in program termination
- ③ location of the tabular decision table data
- ④ location of the drools rule template to be used to define the rules together with the decision table data
- ⑤ id of the group of rules; used to reference it e.g. when doing logging
- ⑥ some textual description of the rule group
- ⑦ *placeholder patterns might be used here*

When running, *Solicitor* will execute the rules of each rule group separately and in the order given by the configuration. Only if there are no more rules to fire in a group *Solicitor* will move to the next rule group and start firing those rules.

Normally a project will only customize (part of) the data of the decision tables and thus will only change the `ruleSource` and the data in the XLS. All other configuration (the different templates and processing order) is part of the *Solicitor* application itself and should not be changed by end users.

See [Working with Decision Tables](#) and [Standard Business Rules](#) for further information on the business rules.

97.2.5. Writers / Reporting

The writer configuration defines how the processed data will be exported and/or reported.

```

"writers" : [ {
    "type" : "xls", ①
    "templateSource" : "classpath:samples/Solicitor_Output_Template_Sample.xlsx", ②
    ⑥
    "target" : "OSS-Inventory-DevonFW.xlsx", ③ ⑥
    "description" : "The XLS OSS-Inventory document", ④
    "dataTables" : { ⑤
        "ENGAGEMENT" :
    "classpath:com/devonfw/tools/solicitor/sql/allden_engagements.sql",
        "LICENSE" :
    "classpath:com/devonfw/tools/solicitor/sql/allden_normalizedlicenses.sql"
    }
}
]

```

① type of writer to be selected; possible values: `xls`, `velo`

② path to the template to be used

③ location of the output file

④ some textual description

⑤ reference to SQL statements used to transform the internal data model to data tables used for reporting

⑥ *placeholder patterns might be used here*

For details on the writer configuration see [Reporting / Creating output documents](#).

97.3. Starting a new project

To simplify setting up a new project *Solicitor* provides an option to create a project starter configuration in a given directory.

```
java -jar solicitor.jar -wiz some/directory/path
```

Besides the necessary configuration file this includes also empty XLS files for defining project specific rules which amend the builtin rules. Furtheron a sample `license.xml` file is provided to directly enable execution of solicitor and check functionality.

This configuration then serves as starting point for project specific configuration.

97.4. Exporting the Builtin Configuration

When working with *Solicitor* it might be necessary to get access to the builtin base configuration, e.g. for reviewing the builtin sample rules or using builtin reporting templates as starting point for the creation of own templates.

The command

```
java -jar solicitor.jar -ec some/directory/path
```

will export all internal configuration to the given directory. This includes:

- The base configuration file, which defines standard settings inherited by the [Project Configuration File](#)
- The Drools Rule Templates
- The builtin decision tables which are referenced in the base configuration, see [Standard Business Rules](#)
- The SQL statements which are used for [SQL transformation and filtering](#)
- The referenced templates for the [Velocity Writer](#) and [Excel Writer](#)

97.5. Configuration of Technical Properties

Besides the project configuration done via the above described file there are a set of technical settings in *Solicitor* which are done via properties. *Solicitor* is implemented as a [Spring Boot Application](#) and makes use of the [standard configuration mechanism provided by the Spring Boot Platform](#) which provides several ways to define/override properties.

The default property values are given in [Built in Default Properties](#).

In case that a property shall be overridden when executing *Solicitor* this can easiest be done via the command line when executing *Solicitor*:

```
java -Dsome.property.name1=value -Dsome.property.name2=another_value -jar  
solicitor.jar <any other arguments>
```

98. Reading License Information with Readers

Different Readers are available to import raw component / license information for different technologies. This chapter describes how to setup the different build / dependency management systems to create the required input and how to configure the corresponding reader.

98.1. Maven

For the export of the licenses from a maven based project the license-maven-plugin is used, which can directly be called without the need to change anything in the pom.xml.

To generate the input file required for *Solicitor* the License Plugin needs to be executed with the following command:

```
mvn org.codehaus.mojo:license-maven-plugin:1.14:aggregate-download-licenses
-Dlicense.excludedScopes=test,provided
```

The generated output file named `licenses.xml` (in the directory specified in the plugin config) should look like the following:

```
Unresolved directive in solicitor.wiki/master-solicitor.asciidoc -
include::licenses.xml.asciidoc[]
```

In *Solicitor* the data is read with the following reader config:

```
"readers" : [ {
  "type" : "maven",
  "source" : "file:target/generated-resouces/licenses.xml",
  "usagePattern" : "DYNAMIC_LINKING"
} ]
```

(the above assumes that *Solicitor* is executed in the maven projects main directory)

98.2. CSV

The CSV input is normally manually generated and should look like this (The csv File is ";" separated):

```
Unresolved directive in solicitor.wiki/master-solicitor.asciidoc -
include::csvlicenses.csv.asciidoc[]
```

In *Solicitor* the data is read with the following part of the config

```
"readers" : [ {
    "type" : "csv",
    "source" : "file:path/to/the/file.csv",
    "usagePattern" : "DYNAMIC_LINKING"
} ]
```

The following 5 columns need to be contained:

- groupId
- artifactId
- version
- license name
- license URL

In case that a component has multiple licenses attached, there needs to be a separate line in the file for each license.

98.3. NPM

For NPM based projects either the NPM License Crawler (<https://www.npmjs.com/package/npm-license-crawler>) or the NPM License Checker (<https://www.npmjs.com/package/license-checker>) might be used. The NPM License Crawler can process several node packages in one run.

98.3.1. NPM License Crawler

To install the NPM License Crawler the following command needs to be executed.

```
npm i npm-license-crawler -g
```

To get the licenses, the crawler needs to be executed like the following example

```
npm-license-crawler --dependencies --csv licenses.csv
```

The export should look like the following (The csv file is "," separated)

```
Unresolved directive in solicitor.wiki/master-solicitor.asciidoc -
include::licenses.csv.asciidoc[]
```

Source: <https://www.npmjs.com/package/npm-license-crawler>

In *Solicitor* the data is read with the following part of the config

```
"readers" : [ {
    "type" : "npm-license-crawler-csv",
    "source" : "file:path/to/licenses.csv",
    "usagePattern" : "DYNAMIC_LINKING",
    "repoType" : "npm"
} ]
```

98.3.2. NPM License Checker

To install the NPM License Checker the following command needs to be executed.

```
npm i license-checker -g
```

To get the licenses, the checker needs to be executed like the following example (we require JSON output here)

```
license-checker --json > /path/to/licenses.json
```

The export should look like the following

```
Unresolved directive in solicitor.wiki/master-solicitor.asciidoc -
include::licensesNpmLicenseChecker.json.asciidoc[]
```

Source: <https://www.npmjs.com/package/license-checker>

In *Solicitor* the data is read with the following part of the config

```
"readers" : [ {
    "type" : "npm-license-checker",
    "source" : "file:path/to/licenses.json",
    "usagePattern" : "DYNAMIC_LINKING",
    "repoType" : "npm"
} ]
```

98.4. Gradle (Windows)

For the export of the licenses from a Gradle based project the Gradle License Plugin is used.

To install the plugin some changes need to be done in **build.gradle**, like following example

```

buildscript {
    repositories {
        maven { url 'https://oss.jfrog.org/artifactory/oss-snapshot-local/' }
    }

    dependencies {
        classpath 'com.jaredsburrows:gradle-license-plugin:0.8.5-SNAPSHOT'
    }
}

apply plugin: 'java-library'
apply plugin: 'com.jaredsburrows.license'

```

Afterwards execute the following command in the console:

For Windows (Java Application)

```
gradlew licenseReport
```

The Export should look like this:

```
Unresolved directive in solicitor.wiki/master-solicitor.asciidoc -  
include::licenses.json.asciidoc[]
```

Source: <https://github.com/jaredsburrows/gradle-license-plugin>

In *Solicitor* the data is read with the following part of the config

```
"readers" : [ {
    "type" : "gradle2",
    "source" : "file:path/to/licenses.json",
    "usagePattern" : "DYNAMIC_LINKING"
} ]
```



The former reader of type `gradle` is deprecated and should no longer be used. See [List of Deprecated Features](#).

98.5. Gradle (Android)

For the Export of the the Licenses from a Gradle based Android Projects the Gradle License Plugin is used.

To install the Plugin some changes need to be done in the build.gradle of the Project, like following example

```

buildscript {
    repositories {
        jcenter()
    }

    dependencies {
        classpath 'com.jaredsburrows:gradle-license-plugin:0.8.5'
    }
}

```

Also there is a change in the build.gradle of the App. Add the line in the second line

```
apply plugin: 'com.android.application'
```

Afterwards execute the following command in the Terminal of Android studio: For Windows(Android Application)

```
gradlew licenseDebugReport
```

The Export is in the following folder

```
$Projectfolder\app\build\reports\licenses
```

It should look like this:

```
Unresolved directive in solicitor.wiki/master-solicitor.asciidoc -  
include::licenseDebugReport.json.asciidoc[]
```

Source: <https://github.com/jaredsburrows/gradle-license-plugin>

In Solicitor the Data is read with the following part of the config

```

"readers" : [ {
    "type" : "gradle2",
    "source" : "file:$input/licenses.json",
    "usagePattern" : "DYNAMIC_LINKING"
} ]

```



The former reader of type **gradle** is deprecated and should no longer be used. See [List of Deprecated Features](#).

99. Working with Decision Tables

Solicitor uses the Drools rule engine to execute business rules. Business rules are defined as "extended" decision tables. Each such decision table consists of two artifacts:

- A rule template file in specific drools template format
- An Excel (XLSX) table which defines the decision table data

When processing, *Solicitor* will internally use the rule template to create one or multiple rules for every record found in the Excel sheet. The following points are important here:

- Rule templates:
 - Rule templates should be regarded as part of the *Solicitor* implementation and should not be changed on an engagement level.
- Excel decision table data
 - The Excel tables might be extended or changed on a per project level.
 - The rules defined by the tabular data will have decreasing "salience" (priority) from top to bottom
 - In general multiple rules defined within a table might fire for the same data to be processed; the definition of the rules within the rule template will normally ensure that once a rule from the decision table was processed no other rule from that table will be processed for the same data
 - The excel tables contain header information in the first row which is only there for documentation purposes; the first row is completely ignored when creating rules from the xls
 - The rows starting from the second row contain decision table data
 - The first "empty" row (which does not contain data in any of the defined columns) ends the decision table
 - Decision tables might use multiple condition columns which define the data that a rule matches. Often such conditions are optional: If left free in the Excel table the condition will be omitted from the rule conditions. This allows to define very specific rules (which only fire on exact data patterns) or quite general rules which get activated on large groups of data. Defining general rules further down in the table (with lower salience/priority) ensures that more specific rules get fired earlier. This even allows to define a default rule at the end of the table which gets fired if no other rule could be applied.
- *rule groups*: Business rules are executed within groups. All rules resulting from a single decision table are assigned to the same *rule group*. The order of execution of the *rule groups* is defined by the sequence of declaration in the config file. Processing of the current group will be finished when there are no more rules to fire in that group. Processing of the next group will then start. Rule groups which have been finished processing will not be resumed even if rules within that group might have been activated again due to changes of the facts.

99.1. Extended comparison syntax

By default any conditions given in the fields of decision tables are simple textual comparisons: The condition is true if the property of the model is identical to the given value in the XLS sheet.

Depending on the configuration of the rule templates for some fields, an extended syntax might be available. For those fields the following syntax applies:

- If the given value of the XLS field starts with the prefix **NOT**: then the outcome of the remaining condition is logically negated, i.e. this field condition is **true** if the rest of the condition is NOT fulfilled.
- A prefix of **REGEX**: indicates that the remainder of the field defines a [Java Regular Expression](#). For the condition to become true the whole property needs to match the given regular expression.
- The prefix **RANGE**: indicates that the remainder of the field defines a [Maven Version Range](#). Using this makes only sense on the artifact version property.
- If no such prefix is detected, then the behavior is identical to the normal (verbatim) comparison logic

Fields which are subject to this extended syntax are marked explicitly in the following section.

100. Standard Business Rules

The processing of business rules is organized in different phases. Each phase might consist of multiple decision tables to be processed in order.

100.1. Phase 1: Determining assigned Licenses

In this phase the license data imported via the readers is cleaned and normalized. At the end of this phase the internal data model should clearly represent all components and their assigned licenses in normalized form.

The phase itself consists of two decision tables / rule groups:

100.1.1. Decision Table: Explicitely setting Licenses

With this decision table it is possible to explicitly assign NormalizedLicenses to components. This will be used if the imported RawLicense data is either incomplete or incorrect. Items which have been processed by rules of this group will not be reprocessed by the next rule group.

- LHS conditions:
 - Engagement.clientName
 - Engagement.engagementName
 - Application.applicationName
 - ApplicationComponent.groupId 
 - ApplicationCompomment.artifactId 
 - ApplicationComponent.version 
 - RawLicense.declaredLicense 
 - RawLicense.url 
- RHS result:
 - NormalizedLicense.normalizedLicenseType
 - NormalizedLicense.normalizedLicense
 - NormalizedLicense.normalizedLicenseUrl
 - NormalizedLicense.comment

 On these fields the [Extended comparison syntax](#) might be used

All RawLicenses which are in scope of fired rules will be marked so that they do not get reprocessed by the following decision table.

100.1.2. Decision Table: Detecting Licenses from Imported Data

With this decision table the license info from the RawLicense is mapped to the NormalizedLicense. This is based on the name and/or URL of the license as imported via the readers.

- LHS conditions:

- RawLicense.declaredLicense 
- RawLicense.url 
- RHS result:
 - NormalizedLicense.normalizedLicenseType
 - NormalizedLicense.normalizedLicense

 On these fields the [Extended comparison syntax](#) might be used

100.2. Phase 2: Selecting applicable Licenses

Within this phase the actually applicable licenses will be selected for each component.

This phase consists of two decision tables.

100.2.1. Choosing specific License in case of Multi-Licensing

This group of rules has the speciality that it might match to a group of NormalizedLicenses associated to an ApplicationComponent. In case that multiple licenses are associated to an ApplicationComponent one of them might be selected as "effective" license and the others might be marked as [Ignored](#).

- LHS conditions:
 - ApplicationComponent.groupId 
 - ApplicationComponent.artifactId 
 - ApplicationComponent.version 
 - NormalizedLicense.normalizedLicense (licenseToTake; mandatory)
 - NormalizedLicense.normalizedLicense (licenseToIgnore1; mandatory)
 - NormalizedLicense.normalizedLicense (licenseToIgnore2; optional)
 - NormalizedLicense.normalizedLicense (licenseToIgnore3; optional)
- RHS result
 - license matching "licenseToTake" will get this value assigned to [effectiveNormalizedLicense](#)
 - licenses matching "licenseToIgnoreN" will get [IGNORED](#) assigned to [effectiveNormalizedLicenseType](#) [Ignored](#) assigned to [effectiveNormalizedLicense](#)

 On these fields the [Extended comparison syntax](#) might be used

It is important to note that the rules only match, if all licenses given in the conditions actually exist and are assigned to the same ApplicationComponent.

100.2.2. Selecting / Overriding applicable License

The second decision table in this group is used to define the [effectiveNormalizedLicense](#) (if not already handled by the decision table before).

- LHS conditions:

- `ApplicationComponent.groupId` 
- `ApplicationComponent.artifactId` 
- `ApplicationComponent.version` 
- `NormalizedLicense.normalizedLicenseType`
- `NormalizedLicense.normalizedLicense`

- RHS result:

- `NormalizedLicense.effectiveNormalizedLicenseType` (if empty in the decision table then the value of `normalizedLicenseType` will be taken)
- `NormalizedLicense.effectiveNormalizedLicense` (if empty in the decision table then the value of `normalizedLicense` will be taken)
- `NormalizedLicense.effectiveNormalizedLicenseUrl` (if empty in the decision table then the value of `normalizedLicenseUrl` will be taken)

 On these fields the [Extended comparison syntax](#) might be used

100.3. Phase 3: Legal evaluation

The third phase ist the legal evaluation of the licenses and the check, whether OSS usage is according to defined legal policies. Again this phase comprises two decision tables.

100.3.1. Pre-Evaluation based on common rules

Within the pre evaluation the license info is checked against standard OSS usage policies. This roughly qualifies the usage and might already determine licenses which are OK in any case or which need to be further evaluated. Furtheron they qualify whether the license text or source code needs to be included in the distribution. The rules in this decision table are only based on the `effectiveNormalizedLicense` and do not consider any project, application or component information.

- LHS condition:

- `NormalizedLicense.effectiveNormalizedLicenseType`
- `NormalizedLicense.effectiveNormalizedLicense`

- RHS result:

- `NormalizedLicense.legalPreApproved`
- `NormalizedLicense.copyLeft`
- `NormalizedLicense.licenseCompliance`
- `NormalizedLicense.licenseRefUrl`
- `NormalizedLicense.includeLicense`
- `NormalizedLicense.includeSource`

100.3.2. Final evaluation

The decision table for final legal evaluation defines all rules which are needed to create the result of the legal evaluation. Rules here might be general for all projects or even very specific to a project

if the rule can not be applied to other projects.

- LHS condition:

- Engagement.clientName
- Engagement.engagementName
- Engagement.customerProvidesOss
- Application.applicationName
- ApplicationComponent.groupId 
- ApplicationComponent.artifactId 
- ApplicationComponent.version 
- ApplicationComponent.usagePattern
- ApplicationComponent.ossModified
- NormalizedLicense.effectiveNormalizedLicenseType
- NormalizedLicense.effectiveNormalizedLicense

- RHS result:

- NormalizedLicense.legalApproved
- NormalizedLicense.legalComments

 On these fields the [Extended comparison syntax](#) might be used

100.4. Amending the builtin decision tables with own rules

The standard process as described before consists of 6 decision tables / rule groups to be processed in sequence. When using the builtin default base configuration all those decision tables use the internal sample data / rules as contained in *Solicitor*.

To use your own rule data there are three approaches:

- Include your own `rules` section in the project configuration file (so not inheriting from the builtin base configuration file) and reference your own decision tables there.
- Create your own "Solicitor Extension" which might completely redefine/replace the builtin `Solicitor` setup including all decision tables and the base configuration file. See [Extending Solicitor](#) for details.
- Make use of the optional project specific decision tables which are defined in the default base configuration: For every builtin decision table there is an optional external decision table (expected in the filesystem) which will be checked for existence. If such external decision table exists it will be processed first - before processing the builtin decision table. Thus is it possible to amend / override the builtin rules by project specific rules. When you create the starter configuration of your project as described in [Starting a new project](#), those project specific decision tables are automatically created.

101. Reporting / Creating output documents

After applying the business rules the resulting data can be used to create reports and other output documents.

Creating such reports consists of three steps:

- transform and filter the model data by using an embedded SQL database
- determining difference to previously stored model (optional)
- Template based reporting via
 - Velocity templates (for textual output like e.g. HTML)
 - Excel templates

101.1. SQL transformation and filtering

101.1.1. Database structure

After the business rules have been processed (or a Solicitor data model has been loaded via command line option `-1`) the model data is stored in a dynamically created internal SQL database.

- For each type of model object a separate table is created. The tablename is the name of model object type written in uppercase characters. (E.g. type `NormalizedLicense` stored in table `NORMALIZEDLICENSE`)
- All properties of the model objects are stored as strings in fields named like the properties within the database table. Field names are case sensitive (see note below for handling this in SQL statements).
- An additional primary key is defined for each table, named `ID_<TABLENAME>`.
- For all model elements that belong to some parent in the object hierarchy (i.e. all objects except `ModelRoot`) a foreign key field is added named `PARENT_<TABLENAME>` which contains the unique key of the corresponding parent

101.1.2. SQL queries for filtering and transformation

Each Writer configuration (see [Writers / Reporting](#)) includes a section which references SQL select statements that are applied on the database data. The result of the SQL select statements is made accessible for the subsequent processing of the Writer via the `dataTable` name given in the configuration.

101.1.3. Postprocessing of data selected from the database tables

Before the result of the SQL select statement is handed over to the Writer the following postprocessing is done:

- a `rowCount` column is added to the result which gives the position of the entry in the result set (starting with 1).

- Columns named `ID_<TABLENAME>` are replaced with columns named `OBJ_<TABLENAME>`. The fields of those columns are filled with the corresponding original model objects (java objects).



The result table column `OBJ_<TABLENAME>` gives access to the native Solicitor data model (java objects), e.g. in the Velocity writer. As this breaks the decoupling done via the SQL database using this feature is explicitly discouraged. It should only be used with high caution and in exceptional situations. The feature might be discontinued in future versions without prior notice.

101.2. Determining difference to previously stored model

When using the command line option `-d` *Solicitor* can determine difference information between two different data models (e.g. the difference between the licenses of the current release and a former release.) The difference is calculated on the result of the above described SQL statements:

- First the internal reporting database is created for the current data model and all defined SQL statements are executed
- Then the internal database is recreated for the "old" data model and all defined SQL statements are executed again
- Finally for each defined result table the difference between the current result and the "old" result is calculated

To correctly correlate corresponding rows of the two different versions of table data it is necessary to define explicit correlation keys for each table in the SQL select statement. It is possible to define up to 10 correlation keys named `CORR_KEY_X` with X in the range from 0 to 9. `CORR_KEY_0` has highest priority, `CORR_KEY_9` has lowest priority.

The correlation algorithm will first try to match rows using `CORR_KEY_0`. It will then attempt to correlate unmatched rows using `CORR_KEY_1` e.t.c.. Correlation will stop, when

- all correlations keys `CORR_KEY_0` to `CORR_KEY_9` have been processed OR
- the required correlation key column does not exist in the SQL select result OR
- there are no unmatched "new" rows OR
- there are no unmatched "old" rows

The result of the correlation / difference calculation is stored in the reporting table data structure. For each row the status is accessible if

- The row is "new" (did not exist in the old data)
- The row is unchanged (no changes in the field values representing the properties of the *Solicitor* data model)
- The row is changed (at least one field corresponding to the *Solicitor* data model changed)

For each field of "changed" or "unchanged" rows the following status is available:

- Field is "changed"
- Field is "unchanged"

For each field of such rows it is furtheron possible to access the new and the old field value.

101.3. Sample SQL statement

The following shows a sample SQL statement showing some join over multiple tables and the use of correlations keys.

```
Unresolved directive in solicitor.wiki/master-solicitor.asciidoc - include::../src
/main/resources/com/devonfw/tools/solicitor/sql/allden_normalizedlicenses.sql.asciidoc
[]
```



Above example also shows how the case sensitive column names have to be handled within the SQL

101.4. Writers

The above described SQL processing is identical for all Writers. Writers only differ in the way how the output document is created based on a template and the reporting table data obtained by the SQL transformation.

101.4.1. Velocity Writer

The Velocity Writer uses the [Apache Velocity Templating Engine](#) to create text based reports. The reporting data tables created by the SQL transformation are directly put to the into Velocity Context.

For further information see the

- Velocity Documentation
- The *Solicitor* JavaDoc (which also includes details on how to access the diff information for rows and fields of reporting data tables)
- The samples included in *Solicitor*

101.4.2. Excel Writer

Using Placeholders in Excel Spreadsheets

Within Excel spreadsheet templates there are two kinds of placeholders / markers possible, which control the processing:

Iterator Control

The templating logic searches within the XLSX workbook for fields containing the names of the

reporting data tables as defined in the Writer configuration like e.g.:

- #ENGAGEMENT#
- #LICENSE#

Whenever such a string is found in a cell this indicates that this row is a template row. For each entry in the respective reporting data table a copy of this row is created and the attribute replacement will be done with the data from that reporting table. (The pattern #…# will be removed when copying.)

Attribute replacement

Within each row which was copied in the previous step the templating logic searches for the string pattern \$someAttributeName\$ where `someAttributeName` corresponds to the column names of the reporting table. Any such occurrence is replaced with the corresponding data value.

Representation of Diff Information

In case that a difference processing (new vs. old model data) was done this will be represented as follows when using the XLS templating:

- For rows that are "new" (so no corresponding old row available) an Excel note indicating that this row is new will be attached to the field that contained the #…# placeholder.
- Fields in non-new rows that have changed their value will be marked with an Excel note indicating the old value.

102. Resolving of License URLs

Resolving of the content of license texts which are referenced by the URLs given in `NormalizedLicense` is done in the following way:

- If the content is found as a resource in the classpath under `licenses` this will be taken. (The *Solicitor* application might include a set of often used license texts and thus it is not necessary to fetch those via the net.) If the classpath does not contain the content of the URL the next step is taken.
- If the content is found as a file in subdirectory `licenses` of the current working directory this is taken. If no such file exists the content is fetched via the net. The result will be written to the file directory, so any content will only be fetched once. (The user might alter the files in that directory to change/correct its content.) A file of length zero indicates that no content could be fetched.

102.1. Encoding of URLs

When creating the resource or filename for given URLs in the above steps the following encoding scheme will be applied to ensure that always a valid name can be created:

All "non-word" characters (i.e. characters outside the set `[a-zA-Z_0-9]`) are replaced by underscores ("`_`").

103. Feature Deprecation

Within the lifecycle of the *Solicitor* development features might be discontinued due to various reasons. In case that such discontinuation is expected to break existing projects a two stage deprecation mechanism is used:

- Stage 1: Usage of a deprecated feature will produce a *warning only* giving details on what needs to be changed.
- Stage 2: When a deprecated feature is used *Solicitor* by default will terminate with an error message giving information about the deprecation.

By setting the property `solicitor.deprecated-features-allowed` to `true` (e.g. via the command line, see [Configuration of Technical Properties](#)), even in second stage the feature will still be available and only a warning will be logged. The project setup should in any case ASAP be changed to no longer use the feature as it might soon be removed without further notice.



Enabling the use of deprecated feature via the above property should only be a temporary workaround and not a standard setting.



If usage of a feature should be discontinued immediately (e.g. because it might lead to wrong/misleading output) the first stage of deprecation will be skipped.

103.1. List of Deprecated Features

The following features are deprecated via the above mechanism:

- Reader of type "gradle" (use Reader of type "gradle2" instead); Stage 2 from Version 1.0.5 on; see <https://github.com/devonfw/solicitor/issues/58>
- Reader of type "npm" (use type "npm-license-crawler-csv" instead); Stage 1 from Version 1.0.8 on; see <https://github.com/devonfw/solicitor/issues/62>

Appendix A: Default Base Configuration

The builtin default base configuration contains settings for the `rules` and `writers` section of the `Solicitor` configuration file which will be used if the project specific config file omits those sections.

Listing 131. Default Configuration

```
Unresolved directive in solicitor.wiki/master-solicitor.asciidoc - include::../src/main/resources/com/devonfw/tools/solicitor/config/solicitor_base.cfg.asciidoc[lines=23..169]
```

Appendix B: Built in Default Properties

The following lists the default settings of technical properties as given by the built in [application.properties](#) file.

If required these values might be overridden on the command line when starting [Solicitor](#):

```
java -Dpropertyname1=value1 -Dpropertyname2=value2 -jar solicitor.jar <any other arguments>
```

Listing 132. application.properties

```
Unresolved directive in solicitor.wiki/master-solicitor.asciidoc -  
include:.../src/main/resources/application.properties.asciidoc[]
```

Appendix C: Extending Solicitor

Solicitor comes with a sample rule data set and sample reporting templates. In general it will be required to correct, supplement and extend this data sets and templates. This can be done straightforward by creating copies of the appropriate resources (rule data XLS and template files), adopting them and furtheron referencing those copies instead of the original resources from the project configuration file.

Even though this approach is possible it will result in hard to maintain configurations, especially in the case of multiple projects using **Solicitor** in parallel.

To support such scenarios **Solicitor** provides an easy extension mechanism which allows to package all those customized configurations into a single archive and reference it from the command line when starting **Solicitor**.

This facilitates configuration management, distribution and deployment of such extensions.

C.1. Format of the extension file

The extensions might be provided as JAR file or even as a simple ZIP file. There is only one mandatory file which contains (at least metadata) about the extension and which needs to be included in this archive *in the root folder*.

Listing 133. application-extension.properties

```
Unresolved directive in solicitor.wiki/master-solicitor.asciidoc -  
include:.../src/main/resources/samples/application-extension.properties.asciidoc[]
```

This file is included via the standard Spring Boot profile mechanism. Besides containing naming and version info on the extension this file might override any property values defined within **Solicitor**.

Any other resources (like rule data or templates) which need to be part of the Extension can be included in the archive as well - either in the root directory or any subdirectories. If the extension is active those resources will be available *on the classpath* like any resources included in the **Solicitor** jar.

Overriding / redefining the default base configuration within the Extension enables to update all rule data and templates without the need to touch the projects configuration file.

C.2. Activating the Extension

The Extension will be activated by referencing it as follows when starting **Solicitor**:

```
java -Dloader.path=path/to/the/extension.zip -jar solicitor.jar <any other arguments>
```

Appendix D: Release Notes

Changes in 1.3.0

Changes in 1.2.0

- Added some license name mapping rules in LicenseNameMappingSample.xls
- <https://github.com/devonfw/solicitor/issues/71>: New "Quality Report" which might be helpful in validating the outcome of the *Solicitor* run. Currently this report contains a list of all application components which have more than one effective license attached. This might be helpful for spotting cases where appropriate rules for selecting the applicable license in case of dual-/multilicensing is missing.

Changes in 1.1.1

- Corrected order of license name mapping which prevented Unlicense, The W3C License, WTFPL, Zlib and Zope Public License 2.1 to be mapped.

Changes in 1.1.0

- <https://github.com/devonfw/solicitor/issues/67>: Inclusion of detailed license information for the dependencies included in the executable JAR. Use the '-eug' command line option to store this file (together with a copy of the user guide) in the current work directory.
- Additional rules for license name mappings in decision table LicenseNameMappingSample.xls.
- <https://github.com/devonfw/solicitor/pull/61>: Solicitor can now run with Java 8 or Java 11.

Changes in 1.0.8

- <https://github.com/devonfw/solicitor/issues/62>: New Reader of type `npm-license-checker` for reading component/license data collected by NPM License Checker (<https://www.npmjs.com/package/license-checker>). The type of the existing Reader for reading CSV data from the NPM License Crawler has been changed from `npm` to `npm-license-crawler-csv`. (`npm` is still available but deprecated.) Projects should adopt their Reader configuration and replace type `npm` by `npm-license-crawler-csv`.

Changes in 1.0.7

- <https://github.com/devonfw/solicitor/issues/56>: Enable continuing analysis in multiapplication projects even if some license files are unavailable.
- Described simplified usage of license-maven-plugin without need to change pom.xml. (Documentation only)
- Ensure consistent sorting even in case that multiple "Ignored" licenses exist for a component

Part XVI: Contributing

Unresolved directive in general/master-contributing.asciidoc
include:../../github/CONTRIBUTING.asciidoc[leveloffset=1]

Unresolved directive in general/master-contributing.asciidoc
include:../../github/CODE_OF_CONDUCT.asciidoc[leveloffset=1]

104. OSS Compliance

This chapter helps you to gain transparency on OSS usage and reach OSS compliance in your project.

104.1. Preface

devonfw, as most Java software, makes strong use of Open Source Software (OSS). It is using about 150 OSS products on the server only and on the client even more. Using a platform like devonfw to develop your own custom solution requires handling contained OSS correctly, i.e acting *OSS-compliant*.

Please read the Open Source policy of your company first, e.g. the [Capgemini OSS Policy](#) which contains a short, comprehensive and well written explanation on relevant OSS-knowledge. Make sure you:

- understand the copyleft effect and its effect in commercial projects
- understand the 3 license categories: "permissive", "weak copyleft" and "strong copyleft"
- know prominent license types as e.g. "Apache-2.0" or "GPL-3.0" and what copyleft-category they are in
- are aware that some OSS offer dual/multi-licenses
- Understand that OSS libraries often come with sub-dependencies of other OSS carrying licenses themselves

To define sufficient OSS compliance measures, contact your IP officer or legal team as early as possible, especially if you develop software for clients.

104.2. Obligations when using OSS

If you create a custom solution containing OSS, this in legal sense is a "derived" work. If you distribute your derived work to your business client or any other legal entity in binary packaged form, the license obligations of contained OSS get into effect. Ignoring these leads to a license infringement which can create high damage.

To carefully handle these obligations you must:

- maintain an OSS inventory (to gain transparency on OSS usage and used licenses)
- check license conformity depending on usage/distribution in a commercial scenario
- check license compatibility between used OSS-licenses
- fulfill obligations defined by the OSS-licenses

Obligations need to be checked per license. Frequent obligations are:

- deliver the license terms of all used versions of the OSS licenses
- not to change any copyright statements or warranty exclusions contained in the used OSS

components

- deliver the source code of the OSS components (e.g. on a data carrier)
- when modifying OSS, track any source code modification (including date and name of the employee/company)
- display OSS license notice in a user frontend (if any)
- other obligations depending on individual license

104.3. Automate OSS handling

Carefully judging the OSS usage in your project is a MANUAL activity! However, collecting OSS information and fulfilling license obligations should be automated as much as possible. A prominent professional tool to automate OSS compliance is the commercial software "Black Duck". Unfortunately it is rather expensive - either purchased or used as SaaS.

The most recommended lightweight tooling is a combination of Maven plugins. We will mainly use the [Mojo Maven License Plugin](#).

104.4. Configure the Mojo Maven License Plugin

You can use it from command line but this will limit the ability to sustainably configure it (shown later). Therefore we add it permanently as a build-plugin to the project parent-pom like this (already contained in OASP-parent-pom):

```

<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>license-maven-plugin</artifactId>
  <version>1.14</version>

  <configuration>
    <outputDirectory>${project.build.directory}/generated-resources</outputDirectory>
    <sortArtifactByName>true</sortArtifactByName>
    <includeTransitiveDependencies>true</includeTransitiveDependencies>
    <!-- the "missing file" declares licenses for dependencies that could not be
detected automatically -->
    <useMissingFile>true</useMissingFile>
    <!-- find the "missing files" in all child-projects at the following location -->
    <missingFile>src/license/THIRD-PARTY.properties</missingFile>
    <!-- if the "missing files" are not yet existing in child-projects they will be
created automatically -->
    <failOnMissing>false</failOnMissing>
    <overrideFile>src/license/override-THIRD-PARTY.properties</overrideFile>
    <!-- harmonize different ways of writing license names -->
    <licenseMerges>
      <licenseMerge>Apache-2.0|Apache 2.0</licenseMerge>
      <licenseMerge>Apache-2.0|Apache License, Version 2.0</licenseMerge>
      <licenseMerge>Apache-2.0|Apache Software License, Version 2.0</licenseMerge>
      <licenseMerge>Apache-2.0|The Apache Software License, Version 2.0</licenseMerge>
    </licenseMerges>
    <encoding>utf-8</encoding>
  </configuration>
</plugin>

```

In the config above there are several settings that help to permanently improve the result of an automated OSS scan. We explain these now.

104.4.1. Declare additional licenses

Sometimes the licenses of used OSS cannot be resolved automatically. That is not the mistake of the maven-license-tool, but the mistake of the OSS author who didn't make the respective license-information properly available.

Declare additional licenses in a "missing file" within *each* maven-subproject: /src/license/THIRD-PARTY.properties.

```
# Generated by org.codehaus.mojo.license.AddThirdPartyMojo
#-----
# Already used licenses in project :
# - ASF 2.0
# - Apache 2
...
#-----
# Please fill the missing licenses for dependencies :
...
dom4j--dom4j--1.6.1=BSD 3-Clause
javax.servlet--jstl--1.2=CDDL
...
```

In case the use of "missing files" is activated, but the THIRD-PARTY.properties-file is not yet existing, the first run of an "aggregate-add-third-party" goal (see below) will fail. Luckily the license-plugin just helped us and created the properties-files automatically (in each maven-subproject) and prefilled it with:

- a list of all detected licenses within the maven project
- all OSS libraries where a license could not be detected automatically.

You now need to fill in missing license information and rerun the plugin.

104.4.2. Redefine wrongly detected licenses

In case automatically detected licenses proof to be wrong by closer investigation, this wrong detection can be overwritten. Add a configuration to declare alternative licenses within each maven-subproject: /src/license/override-THIRD-PARTY.properties

```
com.sun.mail--javax.mail--1.5.6=Common Development and Distribution License 1.1
```

This can be also be useful for OSS that provides a multi-license to make a decision which license to actually choose .

104.4.3. Merge licenses

You will see that many prominent licenses come in all sorts of notations, e.g. Apache-2.0 as: "Apache 2" or "ASL-2.0" or "The Apache License, Version 2.0". The Mojo Maven License Plugin allows to harmonize different forms of a license-naming like this:

```
<!-- harmonize different ways of writing license names -->
<licenseMerges>
    <licenseMerge>Apache-2.0|Apache 2.0</licenseMerge>
    <licenseMerge>Apache-2.0|Apache License, Version 2.0</licenseMerge>
    <licenseMerge>Apache-2.0|Apache Software License, Version 2.0</licenseMerge>
    <licenseMerge>Apache-2.0|The Apache Software License, Version 2.0</licenseMerge>
</licenseMerges>
```

License-names will be harmonized in the OSS report to one common term. We propose to harmonize to short-license-IDs defined by the [SPDX](#) standard.

104.5. Retrieve licenses list

For a quick initial judgement of OSS license situation run the following maven command from command line:

```
$ mvn license:license-list
```

You receive the summary list of all used OSS licenses on the cmd-out.

104.6. Create an OSS inventory

To create an OSS inventory means to report on the overall bill of material of used OSS and corresponding licenses. Within the parent project, run the following maven goal from command line.

```
$ mvn license:aggregate-download-licenses -Dlicense.excludedScopes=test,provided
```

Running the aggregate-download-licenses goal creates two results.

1. a license.xml that contains all used OSS dependencies (even sub-dependencies) with respective license information
2. puts all used OSS-license-texts as html files into folder target/generated resources

Carefully validate and judge the outcome of the license list. It is recommended to copy the license.xml to the project documentation and hand it over to your client. You may also import it into a spreadsheet to get a better overview.

104.7. Create a THIRD PARTY file

Within Java software it is a common practice to add a "THIRD-PARTY" text file to the distribution. Contained is a summary-list of all used OSS and respective licenses. This can also be achieved with the Mojo Maven License Plugin.

Within the parent project, run the following maven goal from command line.

```
$ mvn license:aggregate-add-third-party -Dlicense.excludedScopes=test,provided
```

Find the THIRD-PARTY.txt in the folder: target\generated-resources. The goal aggregate-add-third-party also profits from configuration as outlined above.

104.8. Download and package OSS SourceCode

Some OSS licenses require handing over the OSS source code which is packaged with your custom software to the client the solution is distributed to. It is a good practice to hand over the source code of *all* used OSS to your client. Collecting all source code can be accomplished by another Maven plugin: Apache Maven Dependency Plugin.

It downloads all OSS Source Jars into the folder: \target\sources across the parent and all child maven projects.

You configure the plugin like this:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>3.0.2</version>

  <configuration>
    <classifier>sources</classifier>
    <failOnMissingClassifierArtifact>false</failOnMissingClassifierArtifact>
    <outputDirectory>${project.build.directory}/sources</outputDirectory>
  </configuration>
  <executions>
    <execution>
      <id>src-dependencies</id>
      <phase>package</phase>
      <goals>
        <!-- use unpack-dependencies instead if you want to explode the sources -->
        <goal>copy-dependencies</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

You run the plugin from command line like this:

```
$ mvn dependency:copy-dependencies -Dclassifier=sources
```

The plugin provides another goal that also unzips the jars, which is not recommended, since contents get mixed up.

Deliver the OSS source jars to your client with the release of your custom solution. This has been

done physically - e.g. on DVD.

104.9. Handle OSS within CI-process

To automate OSS handling in the regular build-process (which is not recommended to start with) you may declare the following executions and goals in your maven-configuration:

```
<plugin>
  ...
<executions>
  <execution>
    <id>aggregate-add-third-party</id>
    <phase>generate-resources</phase>
    <goals>
      <goal>aggregate-add-third-party</goal>
    </goals>
  </execution>

  <execution>
    <id>aggregate-download-licenses</id>
    <phase>generate-resources</phase>
    <goals>
      <goal>aggregate-download-licenses</goal>
    </goals>
  </execution>
</executions>
</plugin>
```

Note that the build may fail in case the OSS information was not complete. Check the build-output to understand and resolve the issue - like e.g. add missing license information in the "missing file".

Part XVII: Release Notes

105. devonfw Release notes 2020.12

105.1. Introduction

We are proud to announce the release of devonfw version 2020.12.

This release includes lots of addition of new features, updates and bug fixes but it is very important to highlight the following improvements:

105.2. devonfw IDE

The consolidated list of features for this devonfw IDE release is as it follows.

105.2.1. 2020.12.001

Update with the following bugfixes and improvements:

- [#495](#): Documentation corrections
- [#491](#): Consider lombok support
- [#489](#): Update node to v12.19.0 and VS Code to 1.50.1
- [#470](#): reverse merge of workspace settings not sorting properties anymore
- [#483](#): Error during installation when npm is already installed
- [#415](#): documentation to customize settings
- [#479](#): Error for vscode plugin installation
- [#471](#): Preconfigure Project Explorer with Hierarchical Project Presentation

The full list of changes for this release can be found in [milestone 2020.12.001](#).

105.2.2. 2020.08.001

Update with the following bugfixes and improvements:

- [#454](#): update to eclipse 2020.06
- [#442](#): update nodejs and vscode
- [#432](#): vsCode settings are not updated
- [#446](#): intellij: doConfigureEclipse: command not found
- [#440](#): Software update may lead to inconsistent state due to windows file locks
- [#427](#): release: keep leading zeros
- [#450](#): update settings
- [#431](#): devon build command not working correct for yarn or npm
- [#449](#): update to devon4j 2020.08.001

The full list of changes for this release can be found in [milestone 2020.08.001](#).

105.2.3. 2020.04.004

Minor update with the following bugfixes and improvements:

- [#433](#): Windows: devon command line sets wrong environment variables (with tilde symbol)
- [#435](#): fix variable resolution on bash

The full list of changes for this release can be found in [milestone 2020.04.004](#).

105.2.4. 2020.04.003

Minor update with the following bugfixes and improvements:

- [#395](#): variable from devon.properites unset if value is in double quotes
- [#429](#): Added script to create a meta file in the users directory after setup

The full list of changes for this release can be found in [milestone 2020.04.003](#).

105.2.5. 2020.04.002

Minor update with the following bugfixes and improvements:

- [#418](#): Make projects optional
- [#421](#): update devon4j to 2020.04.002
- [#413](#): Update Eclipse to 2020-03
- [#424](#): Strange errors on windows if devon.properties contains mixed line endings
- [#399](#): launching of IntelliJ fails with No such file or directory error.
- [#410](#): fix jsonmerge for boolean and null values

The full list of changes for this release can be found in [milestone 2020.04.002](#).

105.3. devon4j

The consolidated list of features for this devon4j release is as it follows.

105.3.1. 2020.12.001

New release of [devon4j](#) with pluggable web security (CSRF starter) and [CompletableFuture](#) support for async REST service client as well as other improvements:

- [#283](#): Support for CompletableFuture in async service client
- [#307](#): Fix CSRF protection support
- [#287](#): spring-boot update to 2.3.3
- [#288](#): Update jackson to 2.11.2

- [#293](#): Update owasp-dependency-check plugin version to 5.3.2
- [#302](#): added guide for project/app structure
- [#315](#): devon4j documentation correction
- [#306](#): improve documentation to launch app

Documentation is available at [devon4j guide 2020.12.001](#). The full list of changes for this release can be found in [milestone devon4j 2020.12.001](#).

105.3.2. 2020.08.001

New release of [devon4j](#) with async REST service client support and other improvements:

- [#279](#): support for async service clients
- [#277](#): Update Security-Guide to recent OWASP Top (2017)
- [#281](#): cleanup documentation

Documentation is available at [devon4j guide 2020.08.001](#). The full list of changes for this release can be found in [milestone devon4j 2020.08.001](#).

105.3.3. 2020.04.002

Minor update of [devon4j](#) with the following bugfixes and small improvements:

- [#261](#): JUnit4 backward compatibility
- [#267](#): Fix JWT permission expansion
- [#254](#): JWT Authentication support for devon4j-kafka
- [#258](#): archetype is still lacking a .gitignore
- [#273](#): Update libs
- [#271](#): Do not enable resource filtering by default
- [#255](#): Kafka: Support different retry configuration for different topics

Documentation is available at [devon4j guide 2020.04.002](#). The full list of changes for this release can be found in [milestone devon4j 2020.04.002](#).

105.4. devon4node

New [devon4node](#) version is published, the changes are:

On this release we have deprecated devon4node cli, now we use nest cli, and we have added a GraphQL sample.

- [#375](#): GraphQL Sample.
- [#257](#): D4N cli remove

105.5. CobiGen

Various bugfixes were made as well as consolidating behavior of eclipse vs maven vs cli by properly sharing more code across the different clients. Also properly takes into account a files line delimiter instead of defaulting to those of the host system.

[CobiGen CLI v7.1.0](#) [CobiGen Maven Plug-in v7.1.0](#) [CobiGen Eclipse Plug-in v7.1.0](#)

105.5.1. Templates

- Removed environment.ts from the crud_angular_client_app/CRUD devon4ng Angular App increment since Cobigen did not make any changes in it
- Removed cross referencing between template increments since there is currently no useful use case for it and it leads to a few problems
- [v2020.12.001](#)

105.5.2. Java Plug-in

- Now properly merges using the input files line delimiters instead of defaulting to those of the host system.
- [v7.1.0](#)

105.5.3. TypeScript Plug-in

- Fixed NPE Added the option to read a path from an object input
- [v7.1.0](#)

105.5.4. Property Plug-in

- Now properly merges using the input files line delimiters instead of defaulting to those of the host system.
- [v7.1.0](#)

105.5.5. OpenAPI Plug-in

- Fixed an issue where nullable enums lead to errors
- [7.1.0](#)

105.5.6. Textmerger

- Now properly merges using the input files line delimiters instead of defaulting to those of the host system.
- [v7.1.0](#)
- [v7.1.1](#)

105.6. Sonar devon4j plugin

With this release, we made the package structure configurable and did some other improvements and fixes:

- [#117](#): Rule from checkstyle plugin could not be instantiated in our quality profile
- [#118](#): NPE during project analysis
- [#97](#): Custom configuration for architecture
- [#92](#): Display warnings on the 'devonfw' config page in the 'Administration' section of SonarQube
- [#95](#): Add 3rd Party rule to avoid Immutable annotation from wrong package
- [#94](#): Add 3rd Party rule to avoid legacy date types
- [#93](#): Improve devonfw Java quality profile
- [#114](#): Deleted unused architecture config from SonarQube settings to avoid confusion

Changes for this release can be found in [milestone 2020.12.001](#) and [milestone 2020.12.002](#)

105.7. devon4net

The consolidated list of features for **devon4net** is as follows:

- LiteDb: - Support for LiteDB - Provided basic repository for CRUD operations.
- RabbitMq: - Use of EasyQNet library to perform CQRS main functions between different microservices - Send commands / Subscribe queues with one C# sentence - Events management: Handled received commands to subscribed messages - Automatic messaging backup when sent and handled (Internal database via LiteDB and database backup via Entity Framework)
- MediatR: - Use of MediatR library to perform CQRS main functions in memory - Send commands / Subscribe queues with one C# sentence - Events management: Handled received commands to subscribed messages - Automatic messaging backup when sent and handled (Internal database via LiteDB and database backup via Entity Framework)
- SmaxHcm: - Component to manage Microfocus SMAX for cloud infrastructure services management
- CyberArk: - Manage safe credentials with CyberArk
- AnsibleTower: - Ansible automates the cloud infrastructure. devon4net integrates with Ansible Tower via API consumption endpoints
- gRPC+Protobuf: - Added Client + Server basic templates sample gRPC with Google's Protobuf protocol using devon4net
- Kafka: - Added Apache Kafka support for deliver/consume messages and create/delete topics as well
- AWS support
 - AWS Template to create serverless applications with auto generation of an APIGateway using AWS base template

- AWS template to create pure Lambda functions and manage SQS Events, SNS Events, Generic Events, CloudWatch, S3 Management, AWS Secrets management as a configuration provider in .NET life cycle
- AWS CDK integration component to create/manage AWS infrastructures (Infra As Code): Database, Database cluster, VPC, Secrets, S3 buckets, Roles...
- Minor performance and stability improvements such Entity framework migration integration
- Updated to the latest .net Core 3.1 TLS

105.8. dashboard (beta version)

We are adding dashboard beta version as part of this release. Dashboard is a tool that allows you to create and manage devonfw projects. It makes it easy to onboard a new person with devonfw.

- Dashboard list all ide available on user system or if no ide is available it will provide option to download latest version of ide.
- Project creation and management: Project page list all projects created by user using dashboard. User will be able to create devon4j, devon4ng and devon4node projects using dashboard.
- Support for Eclipse and VSCode IDE
- Integrated devonfw-ide usage guide from the website

105.9. Solicitor

Solicitor is a tool which helps managing Open Source Software used within projects. Below is consolidated feature list of solicitor:

- Standalone Command Line Java Tool
- Importers for component/license information from
 - Maven
 - Gradle
 - NPM
- CSV (e.g. for manual entry of data)
- Rules processing (using Drools Rule Engine) controls the different phases:
 - Normalizing / Enhancing of license information
 - Handling of multilicensing (including selection of applicable licenses) and re-licensing
 - Legal evaluation
 - Rules to be defined as Decision Tables
 - Sample Decision Tables included
 - Automatic download and file based caching of license texts
 - Allows manual editing / reformatting of license text
 - Output processing

- Template based text (Velocity) and XLS generation
- SQL based pre-processor (e.g. for filtering, aggregation)
- Audit log which documents all applied rules for every item might be included in report
- "Diff Mode" allows to mark data which has changed as compared to a previous run of Solicitor (in Velocity and XLS reporting)
- Customization
- Project specific configuration (containing e.g. reporting templates, decision tables) allows to override/amend builtin configuration
- Builtin configuration might be overridden/extended by configuration data contained in a single extension file (ZIP format)
- This allows to safely provide organization specific rules and reporting templates to all projects of an organization (e.g. to reflect the specific OSS usage policy of the organization)

105.10. MrChecker

MrChecker Test Framework is an end to end test automation framework written in Java. It is an automated testing framework for functional testing of web applications, API web services, Service Virtualization, Security, native mobile apps and, in the near future, databases. All modules have tangible examples of how to build resilient integration test cases based on delivered functions. Below is consolidated list of updates in MrChecker:

- Migration of core module to junit5
- Extension of MrCheckers tests harness
- Migration of mrchecker-example-module to junit 5
- Migration guide https://devonfw.com/website/pages/docs/master-mrchecker.asciidoc_migration-from-junit4-to-junit5.html
- Upgrade to cucumber 6.7.0
- Release of the 3.0.1 version to maven-central

105.11. Trainings/tutorials

- Katakoda tutorials : <https://katacoda.com/devonfw>
- Youtube tutorials : <https://www.youtube.com/channel/UCtb1p-24jus-QoXy49t9Xzg>

106. devonfw Release notes 2020.08

106.1. Introduction

We are proud to announce the release of devonfw version 2020.08.

This release includes lots of addition of new features, updates and bug fixes but it is very important to highlight the following improvements:

106.2. devonfw IDE

The consolidated list of features for this devonfw IDE release is as it follows.

106.2.1. 2020.08.001

Update with the following bugfixes and improvements:

- [#454](#): update to eclipse 2020.06
- [#442](#): update nodejs and vscode
- [#432](#): vsCode settings are not updated
- [#446](#): intellij: doConfigureEclipse: command not found
- [#440](#): Software update may lead to inconsistent state due to windows file locks
- [#427](#): release: keep leading zeros
- [#450](#): update settings
- [#431](#): devon build command not working correct for yarn or npm
- [#449](#): update to devon4j 2020.08.001

The full list of changes for this release can be found in [milestone 2020.08.001](#).

106.2.2. 2020.04.004

Minor update with the following bugfixes and improvements:

- [#433](#): Windows: devon command line sets wrong environment variables (with tilde symbol)
- [#435](#): fix variable resolution on bash

The full list of changes for this release can be found in [milestone 2020.04.004](#).

106.2.3. 2020.04.003

Minor update with the following bugfixes and improvements:

- [#395](#): variable from devon.properites unset if value is in double quotes
- [#429](#): Added script to create a meta file in the users directory after setup

The full list of changes for this release can be found in [milestone 2020.04.003](#).

106.2.4. 2020.04.002

Minor update with the following bugfixes and improvements:

- [#418](#): Make projects optional
- [#421](#): update devon4j to 2020.04.002
- [#413](#): Update Eclipse to 2020-03
- [#424](#): Strange errors on windows if devon.properties contains mixed line endings
- [#399](#): launching of IntelliJ fails with No such file or directory error.
- [#410](#): fix jsonmerge for boolean and null values

The full list of changes for this release can be found in [milestone 2020.04.002](#).

106.3. devon4j

The consolidated list of features for this devon4j release is as it follows.

106.3.1. 2020.08.001

New release of [devon4j](#) with async REST service client support and other improvements:

- [#279](#): support for async service clients
- [#277](#): Update Security-Guide to recent OWASP Top (2017)
- [#281](#): cleanup documentation

Documentation is available at [devon4j guide 2020.08.001](#). The full list of changes for this release can be found in [milestone devon4j 2020.08.001](#).

106.3.2. 2020.04.002

Minor update of [devon4j](#) with the following bugfixes and small improvements:

- [#261](#): JUnit4 backward compatibility
- [#267](#): Fix JWT permission expansion
- [#254](#): JWT Authentication support for devon4j-kafka
- [#258](#): archetype is still lacking a .gitignore
- [#273](#): Update libs
- [#271](#): Do not enable resource filtering by default
- [#255](#): Kafka: Support different retry configuration for different topics

Documentation is available at [devon4j guide 2020.04.002](#). The full list of changes for this release can be found in [milestone devon4j 2020.04.002](#).

106.4. devon4ng

This release is focused mainly on the **Angular 10 upgrade**:

- [#176](#): Template submodules updated to Angular 10 and NgRx 10.
- [#167](#), [#168](#), [#174](#) and [#175](#): Updated electron (sample and documentation).
- [#166](#): Update error handler.
- [#165](#): Cypress sample.
- [#164](#): Update to Angular 10 (samples and documentation).

106.5. devon4node

New **devon4node** version is published, the changes are:

- Updated dependencies.
- Solved bug when you introduce a name with dashes in new command.
- Add more options to the non-interactive new command.

106.6. CobiGen

CobiGen version numbers have been consolidated to now represent plug-in compatibility in the major release number (7.x.x).

106.6.1. CLI

- CLI increments can be referenced by name and description.
- Ability to configure logging.
- Fixed error on code formatting.
- Improved Performance by lazy plug-in loading.
- Possibility to prefer custom plug-ins over CobiGen ones.
- Fixed bug, which broke whole CobiGen execution in case a custom CobiGen Plug-in was throwing an arbitrary exception.

106.6.2. Eclipse

- Improved Performance by lazy plug-in loading.
- Possibility to prefer custom plug-ins over CobiGen ones.
- Fixed bug, which broke whole CobiGen execution in case a custom CobiGen Plug-in was throwing an arbitrary exception.

106.6.3. Maven

- Fixed bug to properly load template util classes.
- Improved Performance by lazy plug-in loading.
- Possibility to prefer custom plug-ins over CobiGen ones.
- Fixed bug, which broke whole CobiGen execution in case a custom CobiGen Plug-in was throwing an arbitrary exception.

106.6.4. XML Plug-in

- Added ability to provide custom merge schemas as part of the template folder.
- Added further merge strategies for merging including XML validation.

106.6.5. Java Plug-in

- Fixed NPE for annotated constructors.
- Fixed line separator handling to now prefer the file's one instead of the system ones.
- Fixed unwanted new lines in constructors after merging.
- Fixed annotation formatting after merge.

106.6.6. TypeScript Plug-in

- Fixed issue on automatic update of the ts-merger bundle.

106.7. Sonar devon4j plugin

The consolidated list of features for this [Sonar devon4j plugin](#) release is as it follows.

With this release, we added our own quality profile:

- [#16](#): Install devon4j quality profile

Changes for this release can be found in [milestone 2020.08.001](#)

106.8. My Thai Star with Microservices and ISTIO Service Mesh Implementation

As always, our reference application, [My Thai Star](#) now has been implemented with Microservices and ISTIO Service Mesh features:

- devon4j - Java
 - My Thai Star now has a sample version on Microservices architecture.
 - The github repository for the microservices version of My Thai Star is hosted at [My Thai Star with Microservices](#)

-
- My Thai Star Microservices now has a multi stage docker build which generates the respective docker images for all the My Thai Star services.
 - My Thai Star microservices has the Kubernetes artifacts available to be able to deploy into Kubernetes pods.
 - My Thai Star microservices has ISTIO the service mesh implementation.
 - Check out the guides to implement or configure ISTIO features such as Traffic Routing, Network Resiliency features(RequestRouting, RequestTimeouts, Fault Injection, Circuit Breaker), Canary Deployments.

107. devonfw Release notes 2020.04

107.1. Introduction

We are proud to announce the immediate release of devonfw version 2020.04. This version is the first one with the new versioning that will make easier to the community to identify when it was released since we use the year and month as many other software distributions.

This release includes lots of bug fixes and many version updates, but it is very important to highlight the following improvements:

- New devonfw IDE auto-configure project feature.
- Improved devonfw IDE plugin configuration.
- New devon4j kafka module.
- New devon4j JWT module.
- New devon4j authorization of batches feature.
- Dozer replaced with Orika in devon4j.
- Support for composite keys in devon4j and CobiGen.
- Multiple enhancements for project specific plugin development and usage of project specific template sets in CobiGen.
- Ability to adapt your own templates by making use of CobiGen CLI.
- Better responsiveness in eclipse and bugfixes in all assets in CobiGen.
- devon4ng updated to Angular 9, NgRx 9 and Ionic 5, including documentation, samples and templates.
- Yarn 2 support in devon4ng.
- devon4node updated to NestJS 7 (packages, samples and documentation)
- devon4node TSLint replaced with ESLint.
- @devon4node/config package added.
- devon4net updated to latest .NET Core 3.1.3 LTS version.
- Update of the Production Line templates for devonfw projects in devonfw shop floor.
- New merge feature included in the devonfw shop floor cicdgen tool.
- Updated sonar-devon4j-plugin:
 - Improved coloring and other visual cues to our rule descriptions to highlight good and bad code examples.
 - Improved the locations of issues thrown on method- and class-level.

Please check the detailed list below.

This would have not been possible without the commitment and hard work of the devonfw core team, German, Indian and ADCenter Valencia colleagues and collaborators as, among many others,

the Production Line team.

107.2. devonfw IDE

The consolidated list of features for this devonfw IDE release is as it follows.

107.2.1. 2020.04.001

Starting with this release we have changed the versioning schema in `devonfw` to `yyyy.mm.NNN` where `yyyy.mm` is the date of the planned milestone release and `NNN` is a running number increased with every bug- or security-fix update.

- #394 variable from `devon.properties` not set if not terminated with newline
- #399 launching of IntelliJ fails with No such file or directory error.
- #371 Eclipse plugin installation broke
- #390 maven get/set-version buggy
- #397 migration support for `devon4j` 2020.04.001
- #400 allow custom args for release

The full list of changes for this release can be found in [milestone 2020.04.001](#).

107.2.2. 3.3.1

New release with bugfixes and new ide plugin feature:

- #343: Setup can't find Bash nor Git
- #369: Fix flattening of POMs
- #386: Feature/clone recursive
- #379: Use own extensions folder in `devonfw-ide`
- #381: Add ability to configure VS Code plugins via settings
- #376: Improve Eclipse plugin configuration
- #373: Fix project import on windows
- #374: Rework build on import

The full list of changes for this release can be found in [milestone 3.3.1](#).

107.2.3. 3.3.0

New release with bugfixes and new project import feature:

- #343: Detect non-admin GitForWindows and Cygwin
- #175: Ability to clone projects and import into Eclipse automatically
- #346: `devon eclipse add-plugin` parameters swapped

- [#363](#): devon ide update does not pull latest project settings
- [#366](#): update java versions to latest fix releases

The full list of changes for this release can be found in [milestone 3.3.0](#).

107.3. devon4j

The consolidated list of features for this devon4j release is as it follows.

107.3.1. 2020.04.001

Starting with this release we have changed the versioning schema in `devonfw` to `yyyy.mm.NNN` where `yyyy.mm` is the date of the planned milestone release and `NNN` is a running number increased with every bug- or security-fix update.

The following changes have been incorporated in devon4j:

- [#233](#): Various version updates
- [#241](#): Add module to support JWT and parts of OAuth
- [#147](#): Switch from dozer to orika
- [#180](#): Cleanup archetype
- [#240](#): Add unreferenced guides
- [#202](#): Architecture documentation needs update for components
- [#145](#): Add a microservices article in the documentation
- [#198](#): Deploy SNAPSHOTs to OSSRH in travis CI
- [#90](#): Authorization of batches
- [#221](#): Wrote monitoring guide
- [#213](#): Document logging of custom field in json
- [#138](#): Remove deprecated RevisionMetadata[Type]
- [#211](#): Archetype: security config broken
- [#109](#): LoginController not following devon4j to use JAX-RS but uses spring-webmvc instead
- [#52](#): Improve configuration
- [#39](#): Ability to log custom fields via SLF4J
- [#204](#): Slf4j version
- [#190](#): Rework of spring-batch integration
- [#210](#): Rework documentation for blob support
- [#191](#): Rework of devon4j-batch module
- [#209](#): Include performance info in separate fields
- [#207](#): Use more specific exception for not found entity

- [#208](#): Remove unnecesary clone
- [#116](#): Bug in JSON Mapping for ZonedDateTime
- [#184](#): Fixed BOMs so devon4j and archetype can be used again
- [#183](#): Error in executing the project created with devon4j
- [#177](#): Switch to new maven-parent
- [169](#): Provide a reason, why unchecked exceptions are used in devon4j

Documentation is available at [devon4j guide 2020.04.001](#). The full list of changes for this release can be found in [milestone devon4j 2020.04.001](#).

107.4. devon4ng

The consolidated list of features for this devon4ng release is as it follows.

107.4.1. 2020.04.001

Starting with this release we have changed the versioning schema in [devonfw](#) to [yyyy.mm.NNN](#) where [yyyy.mm](#) is the date of the planned milestone release and [NNN](#) is a running number increased with every bug- or security-fix update.

- [#111](#): Yarn 2 support included
- [#96](#): devon4ng upgrade to Angular 9
 - Templates and samples updated to Angular 9, NgRx 9 and Ionic 5.
 - New internationalization module.
 - Documentation updates and improvements.
- [#95](#): Added token management info in documentation

107.5. devon4net

The consolidated list of features for this devon4net release is as it follows:

- Updated to latest .NET Core 3.1.3 LTS version
- Dependency Injection Autoregistration for services and repositories
- Added multiple role managing claims in JWT
- Added custom headers to circuit breaker
- Reviewed default log configuration
- Added support to order query results from database via lambda expression
- Updated template and nuget packages

107.6. devon4node

The consolidated list of features for this devon4node release is as it follows:

- Upgrade to NestJS 7 (packages, samples and documentation)
- TSLint replaced with ESLint
- Add lerna to project to manage all the packages
- Add @devon4node/config package
- Add new schematics: Repository
- Improve WinstonLogger
- Improve documentation
- Update dependencies to latest versions

107.7. Cobigen

New release with updates and bugfixes:

- devonfw templates:
 - [#1063](#): Upgrade devon4ng Ionic template to latest version
 - [#1065](#): devon4ng templates for devon4node
 - [#1128](#): update java templates for composite keys
 - [#1130](#): Update template for devon4ng application template
 - [#1131](#): Update template for devon4ng NgRx template
 - [#1149](#): .NET templates
 - [#1146](#): Dev ionic template update bug fix
- TypeScript plugin:
 - [#1126](#): OpenApi parse/merge issues (ionic List templates)
- Eclipse plugin:
 - [#412](#): Write UI Test for HealthCheck use
 - [#867](#): Cobigen processbar
 - [#1069](#): #953 dot path
 - [#1099](#): NPE on HealthCheck
 - [#1100](#): 1099 NPE on health check
 - [#1101](#): #867 fix import of core and api
 - [#1102](#): eclipse_plugin doesn't accept folders as input
 - [#1134](#): (Eclipse-Plugin) Resolve Template utility classes from core
 - [#1142](#): #1102 accept all kinds of input

- CobiGen core:
 - [#429](#): Reference external template files
 - [#1143](#): Abort generation if external trigger does not match
 - [#1125](#): Generation of templates from external increments does not work
 - [#747](#): Variable assignment for external increments throws exception
 - [#1133](#): Bugfix/1125 generation of templates from external increments does not work
 - [#1127](#): #1119 added new TemplatesUtilsClassesUtil class to core
 - [#953](#): NPE bug if foldername contains a dot
 - [#1067](#): Feature/158 lat variables syntax
- CobiGen CLI:
 - [#1111](#): Infinity loop in mmm-code (MavenDependencyCollector.collectWithReactor)
 - [#1113](#): cobigen-cli does not seem to properly resolve classes from dependencies
 - [#1120](#): Feature #1108 custom templates folder
 - [#1115](#): Fixing CLI bugs related to dependencies and custom templates jar
 - [#1108](#): CobiGen CLI: Allow easy use of user's templates
 - [#1110](#): FileSystemNotFoundException blocking cobigen-cli
 - [#1138](#): #1108 dev cli feature custom templates folder
 - [#1136](#): (Cobigen-CLI) Resolve Template utility classes from core

107.8. devonfw-shop-floor

- Add documentation for deploy jenkins slaves
- Improve documentation
- Add devon4net Openshift template
- Add nginx docker image for devon4ng
- Add Openshift provisioning
- Production Line:
 - Updated MTS template: add step for dependency check and change the deployment method
 - Add template utils: initialize instance, openshift configuration, docker configuration and install sonar plugin
 - Add devon4net template
 - Add from existing template
 - Improve documentation
 - Refactor the documentation in order to follow the devonfw wiki workflow
 - Update devon4j, devon4ng, devon4net and devon4node in order to be able to choose the deployment method: none, docker or openshift.

- Update the tools version in order to use the latest.
- Production Line Shared Lib
 - Add more functionality to the existing classes.
 - Add classes: DependencyCheckConfiguration, DockerConfiguration and OpenshiftConfiguration
- CICDGEN
 - Add devon4net support
 - Update tools versions in Jenkinsfiles to align with Production Line templates
 - Add merge strategies: error, keep, override, combine
 - Add lerna to the project
 - Minor improvements in the code
 - Add GitHub actions workflow to validate the new changes
 - Improve documentation
 - Breaking changes:
 - Remove the following parameters: plurl, ocurl
 - Add the following parameters: dockerurl, dockercertid, registryurl, ocname and merge

107.9. Sonar devon4j plugin

The consolidated list of features for this Sonar devon4j plugin release is as it follows.

107.9.1. 2020.04.001

This is the first version using our new versioning scheme. Here, the following issues were resolved:

- #60: Fixed a bug in the naming check for Use-Case implementation classes
- #67: Fixed a bug where the whole body of a method or a class was marked as the issue location. Now only the method / class headers will be highlighted.
- #68: Made our rule descriptions more accessible by using better readable colors as well as alternative visual cues
- #71: Fixed a bug where a NPE could be thrown
- #74: Fixed a bug where a method always returned null

Unrelated to any specific issues, there was some refactoring and cleaning up done with the following two PRs:

- PR #66: Refactored the prefixes of our rule names from 'Devon' to 'devonfw'
- PR #65: Sorted security-related test files into their own package

Changes for this release can be found in [milestone 2020.04.001](#).

107.10. My Thai Star

As always, our reference application, My Thai Star, contains some interesting improvements that come from the new features and bug fixes from the other assets. The list is as it follows:

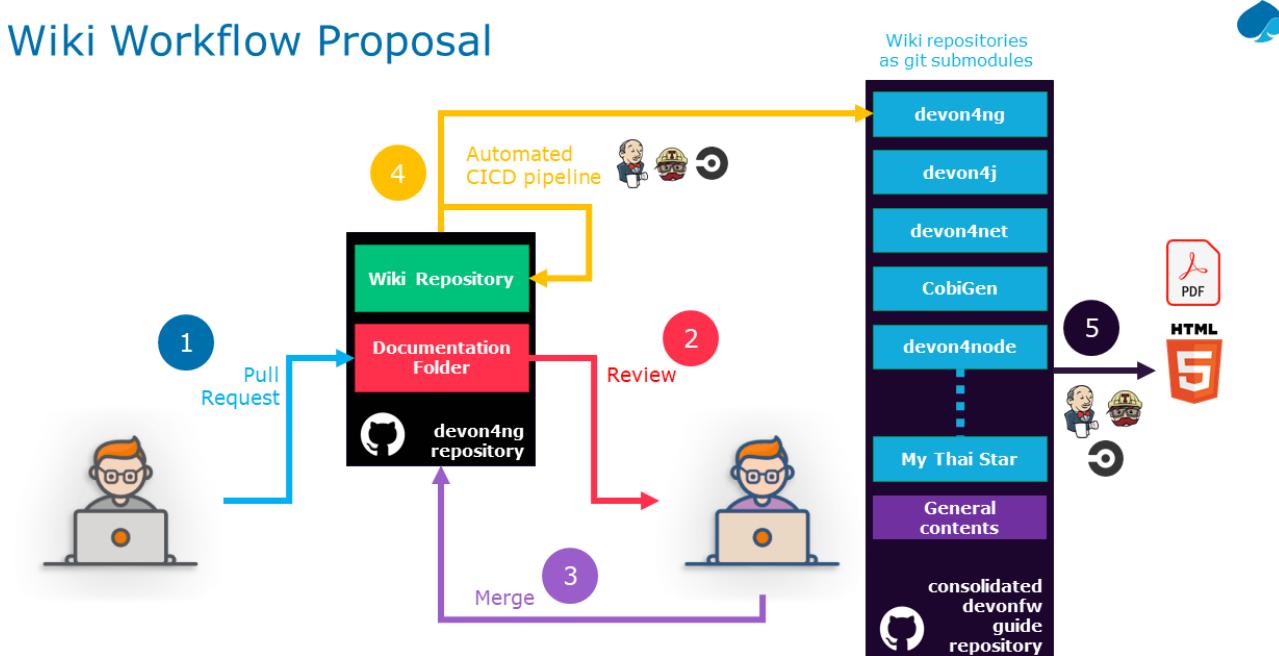
- devon4j - Java
 - Implement example batches with modified devon-batch
 - Upgrade spring boot version to 2.2.6 and devon4j 2020.004.001
 - Migrate from dozer to orika
- devon4ng - Angular
 - Move configuration to NgRx store
- devonfw shop floor - Jenkins
 - Update tools versions in order to align with Production Line templates
 - Add dependency check step (using dependency checker and yarn audit)
 - Send dependency checker reports to SonarQube
 - Changed deployment pipelines. Now pipelines are able to deploy docker containers using docker directly. No more ssh connections to execute commands in a remote machine are required.
 - Update documentation in order to reflect all changes
- devon4nde - Node.js
 - Upgrade to NestJS 7
 - Add custom repositories
 - Add exceptions and exception filters
 - Add tests (missing in the previous version)
 - Split logic into use cases in order to make the test process easier
 - Minor patches and improvements
 - Documentation updated in order to reflect the new implementation

108. devonfw Release notes 3.2 “Homer”

108.1. Introduction

We are proud to announce the immediate release of devonfw version 3.2 (code named “Homer” during development). This version is the first one that contains the **new devonfw IDE** by default, so there is no need to download a huge ZIP with the whole distribution regardless of the use to which it will be put. The new devonfw IDE CLI will allow any user to setup a customized development environment completely configured with access to all the devonfw features, frameworks and tools. As we access to the official IDEs this is also the first version macOS compatible.

This release consolidates the documentation workflow adding the contents dynamically to the new devonfw website at the same time the PDF is generated. This have been achieved using a new GitHub action that takes the contents and builds the HTML files for the documentation section of the website. The documentation workflow proposed in the following picture is now complete:



This release also includes the **first version of devon4node**. We consider that node.js should be a first-class citizen inside the devonfw platform and for that reason we have included the latest development technologies for this ecosystem. The devon4node CLI, schematics and other tools will allow our users to create powerful node.js applications with the same philosophy you may find in the other languages and frameworks included. More information at its section below.

The new **devon4net 3.2.0 version** is also included in this release. Based on the .NET Core 3.0 and containing lots of new features gathered from important and recent projects, it represents a great improvement and an intermediate step to provide support for the incoming .NET Core 3.1 LTS. More information at its section below.

This release includes the final version of the **new CobiGen CLI** and completely integrated with the new devonfw IDE. Now using commands, you will be able to generate code the same way as you do with Eclipse. This means that you can use CobiGen on other IDEs like Visual Studio Code or IntelliJ.

Besides the Update command has been implemented. Now you will be able to update easily all your CobiGen plug-ins and templates inside the CLI.

On the other hand, the refactoring process has been completely developed, improving the mergers and including input readers for any other languages and frameworks, allowing the creation of models to generate code from them. Last, but not least, this new version includes the new templates for devon4net, devon4ng and devon4j generation.

And as always, **My Thai Star** has been updated to the latest versions of devon4j, devon4node and devon4net including completely State Management with NgRx in its devon4ng implementation upgrade.

This is the last release with the current semantic versioning number and without a fixed release calendar. From now on the new devonfw releases will happen in April, August and December and will be named **YYYY.MM.NN**, being the first release of the next year the 2020.04.00.

108.2. Changes and new features

108.2.1. devonfw-ide

We have entirely rewritten our automated solution for your local IDE (integrated desktop environment). The former oasp4j-ide and devonfw distributions with their extra-large gigabyte zip files are not entirely replaced with devonfw-ide. This new solution is provided as a small *.tar.gz file that is publicly available. It works on all platforms and has been tested on Windows, MacOS, and Linux. After extraction you only need to run a **setup** script. Here you provide a settings git URL for your customer project or simply hit return for testing or small projects. After reading and confirming the terms of use it will download all required tools in the proper versions for your operating system and configure them. Instead of various confusing scripts there is now only one CLI command **devon** for all use-cases what gives a much better user experience.

To get started go to the [home page](#). There is even a [migration-guide](#) if you are currently used to the old approach and want to quickly jump into the new solution.

108.2.2. My Thai Star Sample Application

The new release of My Thai Star has focused on the following improvements:

- Release 3.2.0.
- devon4j:
 - devon4j 3.2.0 integrated.
 - Spring Boot 2.1.9 integrated.
 - SAP 4/HANA prediction use case.
 - Bug fixes.
- devon4ng:
 - SAP 4/HANA prediction use case.

- 2FA toggleable (two factor authentication).
- NgRx full integrated (PR #285).
- devon4net
 - devon4net for dotnet core 3.0 updated
 - Updated the API contract compatible with the other stacks
 - JWT implementation reviewed to increase security
 - ASP.NET user database dependencies removed
 - HTTP2 support
 - Clearer CRUD pattern implementation
- devon4node
 - TypeScript 3.6.3.
 - Based on Nest framework.
 - Configuration Module
 - Added cors and security headers
 - Added mailer module and email templates.
 - Built in winston logger
 - Custom ClassSerializerInterceptor
- MrChecker
 - Example cases for end-to-end test.
 - Production line configuration.
- CICD
 - Improved integration with Production Line
 - New Traefik load balancer and reverse proxy
 - New deployment from artifact
 - New CICD pipelines
 - New deployment pipelines
 - Automated creation of pipelines in Jenkins

108.2.3. Documentation updates

This release addresses the new documentation workflow, being now possible to keep the documentation synced with any change. The new documentation includes the following contents:

- Getting started
- devonfw ide
- devon4j documentation
- devon4ng documentation

-
- devon4net documentation
 - devon4node documentation
 - CobiGen documentation
 - devonfw-shop-floor documentation
 - cicdgen documentation
 - devonfw testing with MrChecker
 - My Thai Star documentation
 - Contribution guide
 - Release notes

108.2.4. devon4j

The following changes have been incorporated in devon4j:

- Completed full support from Java8 to Java11
- Several security fixes
- Upgrade to Spring Boot 2.1.9
- Upgrade to Spring 5.1.8
- Upgrade to JUnit 5 (requires migration via devonfw-ide)
- Improved JPA support for IdRef
- Improved auditing metadata support
- Many improvements to documentation (added JDK guide, architecture-mapping, JMS, etc.)
- For all details see [milestone](#).

108.2.5. devon4ng

The following changes have been incorporated in devon4ng:

- Angular CLI 8.3.1,
- Angular 8.2.11,
- Angular Material 8.2.3,
- Ionic 4.11.1,
- Capacitor 1.2.1 as Cordova replacement,
- NgRx 8.3 support for State Management,
- devon4ng Angular application template updated to Angular 8.2.11 with visual improvements and bugfixes <https://github.com/devonfw/devon4ng-application-template>
- devon4ng Ionic application template updated to 4.11.1 and improved <https://github.com/devonfw/devon4ng-ionic-application-template>
- Improved devon4ng Angular application template with state management using Angular 8 and

NgRx 8 <https://github.com/devonfw/devon4ng-ngrx-template>

- Documentation and samples updated to latest versions:
 - Web Components with Angular Elements
 - Initial configuration with App Initializer pattern
 - Error Handling
 - PWA with Angular and Ionic
 - Lazy Loading
 - Library construction
 - Layout with Angular Material
 - Theming with Angular Material

108.2.6. devon4net

The following changes have been incorporated in devon4net:

- Updated to latest .net core 3.0 version
- Template
 - Global configuration automated. devon4net can be instantiated on any .net core application template with no effort
 - Added support for HTTP2
 - Number of libraries minimized
 - Architecture layer review. More clear and scalable
 - Added red button functionality (aka killswitch) to stop attending API request with custom error
 - Improved API error management
 - Added support to only accept request from clients with a specific client certificate on Kestrel server. Special thanks to Bart Rozendaal (Capgemini NL)
 - All components use IOptions pattern to be set up properly
 - Swagger generation compatible with OpenAPI v3
- Modules
 - The devon4net netstandard libraries have been updated to netstandard 2.1
 - JWT:
 - Added token encryption (token cannot be decrypted anymore by external parties). Now You can choose the encryption algorithm depending on your needs
 - Added support for secret key or certificate encryption
 - Added authorization for swagger portal
 - Circuit breaker
 - Added support to bypass certificate validation

- Added support to use a certificate for https communications using Microsoft's httpclient factory
- Unit of Work
 - Repository classes unified and reviewed for increasing performance and reduce the consumed memory
 - Added support for different database servers: In memory, Cosmos, MySQL + MariaDB, Firebird, PostgreSQL, Oracle, SQLite, Access, MS Local.

108.2.7. devon4node

The following changes have been incorporated in devon4node:

- TypeScript 3.6.3.
- Based on Nest framework.
- Complete backend implementation.
- New devon4node CLI. It will provide you some commands
 - new: create a new devon4node interactively
 - generate: generate code based on schematics
 - db: manage the database
- New devon4node schematics
 - application: create a new devon4node application
 - config-module: add a configuration module to the project
 - mailer: install and configure the devon4node mailer module
 - typeorm: install TypeORM in the project
 - auth-jwt: add users and auth-jwt modules to the project
 - swagger: expose an endpoint with the auto-generated swagger
 - security: add cors and other security headers to the project.
 - crud: create all CRUD for an entity
 - entity: create an entity
- New mailer module
- New common library
- Build in winston logger
- Custom ClassSerializerInterceptor
- Extendable base entity
- New application samples

108.2.8. CobiGen

- CobiGen core new features:
 - CobiGen CLI: Update command implemented. Now you will be able to update easily all your CobiGen plug-ins and templates inside the CLI. Please take a look into the [documentation](#) for more info.
 - CobiGen CLI is now JDK11 compatible.
 - CobiGen CLI commandlet for devonfw-ide has been added. You can use it to setup easily your CLI and to run CobiGen related commands.
 - Added a version provider so that you will be able to know all the CobiGen plug-ins versions.
 - Added a process bar when the CLI is downloading the CobiGen plug-ins.
 - CobiGen refactoring finished: With this refactoring we have been able to decouple CobiGen completely from the target and input language. This facilitates the creation of parsers and mergers for any language. For more information please take a look [here](#).
 - New TypeScript input reader: We are now able to parse any TypeScript class and generate code using the parsed information. We currently use [TypeORM](#) entities as a base for generation.
- Improving CobiGen templates:
 - Updated devon4ng-NgRx templates to NgRx 8.
 - Generation of an Angular client using as input a [TypeORM](#) entity. This is possible thanks to the new TypeScript input reader.
 - .Net templates have been upgraded to .Net Core 3.0
- CobiGen for Eclipse is now JDK11 compatible.
- Fixed bugs when adapting templates and other bugs on the CobiGen core.

108.2.9. devonfw shop floor

- Added devon4ng OpenShift templates
- Added devon4j OpenShift templates
- Added devon4node OpenShift templates
- Added more methods to link <https://github.com/devonfw-forge/devon-production-line-shared-lib> [devonfw Production Line shared library]
- Updated link: [devonfw Production Line templates](#)

cicdgen

- Patched minor bugs

108.2.10. sonar-devon4j-plugin

sonar-devon4j-plugin is a SonarQube plugin for architecture governance of devon4j applications. It

verifies the architecture and conventions of devon4j, the Java stack of devonfw. The following changes have been incorporated:

- * Plugin was renamed from sonar-devon-plugin to sonar-devon4j-plugin
- * Rules/checks have been added to verify naming conventions
- * New rule for proper JPA datatype mapping
- * Proper tagging of rules as architecture-violation and not as bug, etc.
- * Several improvements have been made to prepare the plugin to enter the SonarQube marketplace, what will happen with the very next release.
- * Details can be found here: <https://github.com/devonfw/sonar-devon4j-plugin/milestone/2?closed=1>

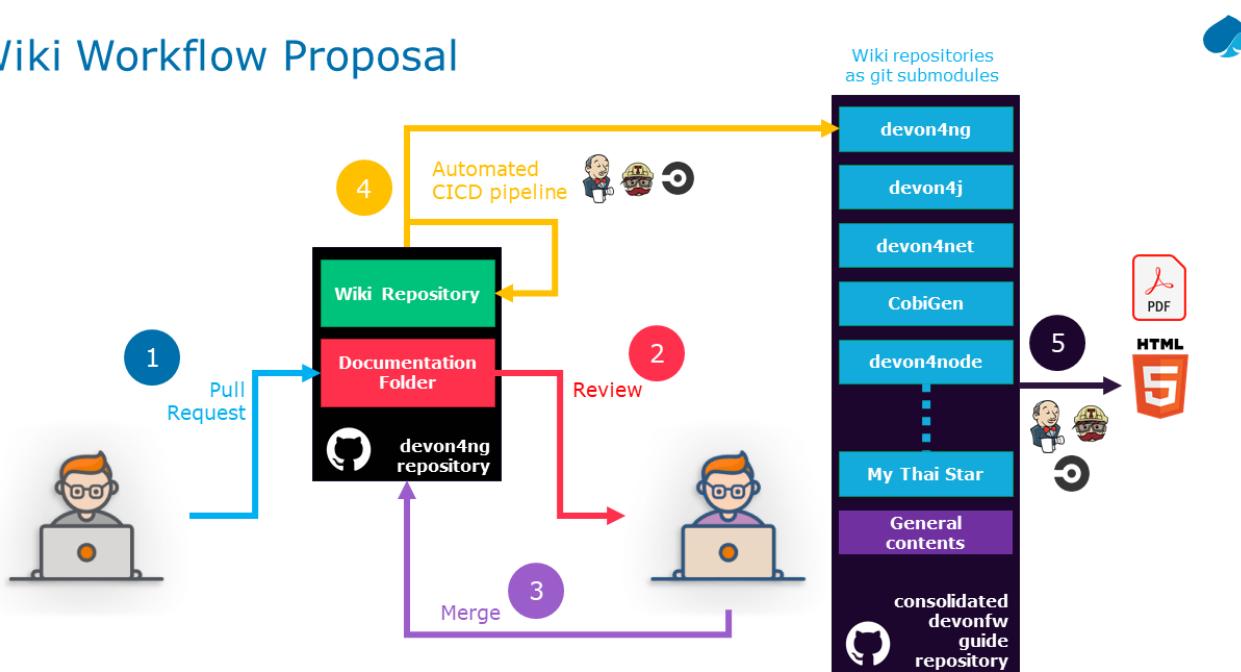
109. devonfw Release notes 3.1 “Goku”

109.1. Introduction

We are proud to announce the immediate release of devonfw version 3.1 (code named “Goku” during development). This version is the first one that implements our new documentation workflow, that will allow users to get the updated documentation at any moment and not to wait for the next devonfw release.

This is now possible as we have established a new workflow and rules during development of our assets. The idea behind this is that all the repositories contain a **documentation** folder and, in any pull request, the developer must include the related documentation change. A new Travis CI configuration added to all these repositories will automatically take the changes and publish them in the wiki section of every repository and in the new devonfw-guide repository that consolidates all the changes from all the repositories. Another pipeline will take changes from this consolidated repository and generate dynamically the devonfw guide in PDF and in the next weeks in HTML for the new planned devonfw website. The following schema explains this process:

Wiki Workflow Proposal



This release includes the very first version of the new CobiGen CLI. Now using commands, you will be able to generate code the same way as you do with Eclipse. This means that you can use CobiGen on other IDEs like Visual Studio Code or IntelliJ. Please take a look at https://github.com/devonfw/cobigen/wiki/howto_Cobigen-CLI-generation for more info.

The devonfw-shop-floor project has got a lot of updates in order to make even easier the creation of devonfw projects with CICD pipelines that run on the Production Line, deploy on Red Hat OpenShift Clusters and in general Docker environments. See the details below.

This release includes the very first version of our devonfw-ide tool that will allow users to automate devonfw setup and update the development environment. This tool will become the default devonfw setup tool in future releases. For more information please visit the repository

[https://github.com/devonfw/devon-ide.](https://github.com/devonfw/devon-ide)

Following the same collaboration model we used in order to improve the integration of devonfw with Red Hat OpenShift and which allowed us to get the Red Hat Open Shift Primed certification, we have been working alongside with SAP HANA developers in order to support this database in the devon4j. This model was based on the contribution and review of pull requests in our reference application My Thai Star. In this case, SAP developers collaborated with us in the following two new use cases:

- Prediction of future demand
- Geospatial analysis and clustering of customers

More info at <https://blogs.sap.com/2019/06/17/introducing-devonfw-support-for-sap-hana/>.

Last but not least the devonfw extension pack for VS Code has been improved with the latest extensions and helpers for this IDE. Among many others you can now use:

- Remote development on Docker containers and VMs <https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remote-extensionpack>
- Dependency Analysis for maven and npm <https://marketplace.visualstudio.com/items?itemName=redhat.fabric8-analytics>
- React Native Tools <https://marketplace.visualstudio.com/items?itemName=msjsdiag.vscode-react-native>
- NgRx Snippets <https://marketplace.visualstudio.com/itemdetails?itemName=hardikpthv.NgRxSnippets>

Also it is worth the try of the updated support for Java and Spring Boot development in VS Code. Check it out for yourself!

More information at <https://marketplace.visualstudio.com/items?itemName=devonfw.devonfw-extension-pack>. Also, you can contribute to this extension in this GitHub repository <https://github.com/devonfw/devonfw-extension-pack-vscode>.

109.2. Changes and new features

109.2.1. Devonfw dist

- Eclipse 2018.12 integrated
 - CheckStyle Plugin updated.
 - SonarLint Plugin updated.
 - Git Plugin updated.
 - FindBugs Plugin updated.
 - CobiGen plugin updated.
- Other Software
 - Visual Studio Code latest version included and pre-configured with the devonfw Platform

Extension Pack.

- Ant updated to latest.
- Maven updated to latest.
- Java updated to latest.
- Nodejs LTS updated to latest.
- @angular/cli included.
- @devonfw/cicdgen included.
- Yarn package manager updated.
- Python3 updated.
- Spyder3 IDE integrated in python3 installation updated.
- devon4ng-application-template for Angular 8 at workspaces/examples
- devon4ng-ionic-application-template for Ionic 4 at workspace/samples

109.2.2. My Thai Star Sample Application

The new release of My Thai Star has focused on the following improvements:

- Release 3.1.0.
- devon4j:
 - devon4j 3.1.0 integrated.
 - Spring Boot 2.1.6 integrated.
 - SAP 4/HANA prediction use case.
 - Bug fixes.
- devon4ng:
 - SAP 4/HANA prediction use case.
 - 2FA toggleable (two factor authentication).
 - NgRx integration in process (PR #234).
- devon4node
 - TypeScript 3.1.3.
 - Based on Nest framework.
 - Aligned with devon4j.
 - Complete backend implementation.
 - TypeORM integrated with SQLite database configuration.
 - Webpack bundler.
 - Nodemon runner.
 - Jest unit tests.
- Mr.Checker

- Example cases for end-to-end test.
- Production line configuration.
- CICD
- Improved integration with Production Line
- New Traefik load balancer and reverse proxy
- New deployment from artifact
- New CICD pipelines
- New deployment pipelines
- Automated creation of pipelines in Jenkins

109.2.3. Documentation updates

This release addresses the new documentation workflow, being now possible to keep the documentation synced with any change. The new documentation includes the following contents:

- Getting started
- Contribution guide
- Devcon
- Release notes
- devon4j documentation
- devon4ng documentation
- devon4net documentation
- devonfw-shop-floor documentation
- cicdgen documentation
- devonfw testing with MrChecker
- My Thai Star documentation

109.2.4. devon4j

The following changes have been incorporated in devon4j:

- Added Support for Java8 up to Java11
- Upgrade to Spring Boot 2.1.6.
- Upgrade to Spring 5.1.8
- Upgrade to JPA 2.2
- Upgrade to Hibernate 5.3
- Upgrade to Dozer 6.4.1 (ATTENTION: Requires Migration, use devon-ide for automatic upgrade)
- Many improvements to documentation (added JDK guide, architecture-mapping, JMS, etc.)
- Completed support (JSON, Beanmapping) for pagination, IdRef, and java.time

-
- Added MasterCto
 - For all details see [milestone](#).

109.2.5. devon4ng

The following changes have been incorporated in devon4ng:

- Angular CLI 8,
- Angular 8,
- Angular Material 8,
- Ionic 4,
- Capacitor 1.0 as Cordova replacement,
- NgRx 8 support for State Management,
- devon4ng Angular application template updated to Angular 8 with visual improvements and bugfixes <https://github.com/devonfw/devon4ng-application-template>
- devon4ng Ionic application template updated and improved <https://github.com/devonfw/devon4ng-ionic-application-template>
- New devon4ng Angular application template with state management using Angular 8 and NgRx 8 <https://github.com/devonfw/devon4ng-ngrx-template>
- New devon4ng library <https://github.com/devonfw/devon4ng-library> that includes the following libraries:
 - Cache Module for Angular 7+ projects.
 - Authorization Module for Angular 7+ projects.
- New use cases with documentation and samples:
 - Web Components with Angular Elements
 - Initial configuration with App Initializer pattern
 - Error Handling
 - PWA with Angular and Ionic
 - Lazy Loading
 - Library construction
 - Layout with Angular Material
 - Theming with Angular Material

109.2.6. devon4net

The following changes have been incorporated in devon4net:

- New circuit breaker component to communicate microservices via HTTP
- Resolved the update packages issue

109.2.7. AppSec Quick Solution Guide

This release incorporates a new Solution Guide for Application Security based on the state of the art in OWASP based application security. The purpose of this guide is to offer quick solutions for common application security issues for all applications based on devonfw. It's often the case that we need our systems to comply to certain sets of security requirements and standards. Each of these requirements needs to be understood, addressed and converted to code or project activity. We want this guide to prevent the wheel from being reinvented over and over again and to give clear hints and solutions to common security problems.

- The wiki can be accessed here: <https://github.com/devonfw/devonfw-security/wiki>
- The PDF can be accessed here: <https://github.com/devonfw/devonfw-security>

109.2.8. CobiGen

- CobiGen core new features:
 - CobiGen CLI: New command line interface for CobiGen. Using commands, you will be able to generate code the same way as you do with Eclipse. This means that you can use CobiGen on other IDEs like Visual Studio Code or IntelliJ. Please take a look into the documentation for more info.
 - Performance improves greatly in the CLI thanks to the lack of GUI.
 - You will be able to use path globs for selecting multiple input files.
 - We have implemented a search functionality so that you can easily search for increments or templates.
 - First steps taken on CobiGen refactoring: With the new refactoring we will be able to decouple CobiGen completely from the target and input language. This will facilitate the creation of parsers and mergers for any language.
 - NashornJS has been deprecated: It was used for executing JavaScript code inside JVM. With the refactoring, performance has improved on the TypeScript merger.
 - Improving CobiGen templates:
 - Removed Covalent from Angular templates as it is not compatible with Angular 8.
 - Added devon4ng-NgRx templates that implement reactive state management. Note: The TypeScript merger is currently being improved in order to accept NgRx. The current templates are set as overridable by default.
 - Test data builder templates now make use of Lambdas and Consumers.
 - CTOs and ETOs increments have been correctly separated.
 - TypeScript merger has been improved: Now it is possible to merge comments (like tsdoc) and enums.
 - OpenAPI parsing extended to read enums. Also fixed some bugs when no properties were set or when URLs were too short.
 - Java static and object initializers now get merged.
 - Fixed bugs when downloading and adapting templates.

109.2.9. Devcon

A new version of Devcon has been released. Fixes and new features include:

- Updated to match current devon4j
- Update to download Linux distribution.
- Custom modules creation improvements.
- Code Migration feature added.
- Bugfixes.

109.2.10. Devonfw OSS Modules

Modules upgraded to be used in new devon4j projects:

- Reporting module
- WinAuth AD Module
- WinAuth SSO Module
- I18n Module
- Async Module
- Integration Module
- Microservice Module
- Compose for Redis Module See: <https://github.com/devonfw/devon/wiki#devonfw-modules>

109.2.11. devonfw shop floor

- Industrialization oriented to configure the provisioning environment provided by Production Line and deploy applications on an OpenShift cluster.
- Added Jenkinsfiles to configure automatically OpenShift environments to deploy devonfw applications.
- Industrialization to start new projects and configure them with CICD.
- Upgrade the documentation with getting started guide to configure CICD in any devonfw project and deploy it.
- Added new tool cicdgen to generate CICD code/files.

cicdgen

cicdgen is a devonfw tool to generate all code/files related to CICD in your project. It's based on angular schematics and it has its own CLI. More information [here](#).

- CICD configuration for devon4j, devon4ng and devon4node projects
- Option to deploy devonfw projects with Docker
- Option to deploy devonfw projects with OpenShift

109.2.12. Devonfw Testing

Mr.Checker

The Mr.Checker Test Framework is an automated testing framework for functional testing of web applications, API web services, Service Virtualization, Security and in coming future native mobile apps, and databases. All modules have tangible examples of how to build resilient integration test cases based on delivered functions. Mr.Checker updates and improvements:

- Examples available under embedded project “MrChecker-App-Under-Test” and in project wiki:
<https://github.com/devonfw/devonfw-testing/wiki>
- How to install:
 - Wiki : <https://github.com/devonfw/devonfw-testing/wiki/How-to-install>
- Release Note:
 - module selenium - 3.8.2.1:
 - possibility to define version of driver in properties.file
 - automatic driver download if the version is not specified
 - possibility to run with different browser options
 - module webAPI – 1.2.1:
 - possibility to connect to the remote WireMock server

110. devonfw Release notes 3.0 “Fry”

110.1. Introduction

We are proud to announce the immediate release of devonfw version 3.0 (code named “Fry” during development). This version is the consolidation of Open Source, focused on the major namespace change ever in the platform, removing the OASP references and adopting the new devonfw names for each technical stack or framework.

The new stack names are the following:

- devon4j, former OASP4J, is the new name for Java.
- devon4ng, former OASP4JS, is the new one for Angular.
- devon4net, is the new .NET stack.
- devon4X, is the new stack for Xamarin development.
- devon4node, is the new devonfw incubator for node.js.

The new devon4j version was created directly from the latest oasp4j version (3.0.0). Hence it brings all the features and values that oasp4j offered. However, the namespace migration was used to do some housekeeping and remove deprecated code as well as reduce dependencies. Therefore your data-access layer will no longer have to depend on any third party except for devon4j as well as of course the JPA. We also have improved the application template that now comes with a modern JSON logging ready for docker and logstash based environments.

To help you upgrading we introduced a migration feature in devcon. This can automatically migrate your code from oasp4j (even older versions starting from 2.4.0) to the latest version of devon4j. There might be some small manual changes left to do but 90% of the migration will be done automatically for you.

Besides, the first version of the devonfw plugin for SonarQube has been released. It extends SonarQube with the ability to validate your code according to the devon4j architecture. More details at <https://github.com/devonfw/sonar-devon-plugin>.

This is the first release that integrates the new devonfw .NET framework, called devon4net, and Xamarin for mobile native development, devon4X. devon4NET and devon4X are the Capgemini standard frameworks for .NET and Xamarin software development. With the two new family members devonfw provides guidance and acceleration for the major software development platforms in our industry. Their interoperability provides you the assurance your multichannel solution will be consistent across web and mobile channels.

“Fry” release contains lots of improvements in our Mr.Checker E2E Testing Framework, including a complete E2E sample inside our reference application My Thai Star. Besides Mr.Checker, we include as an incubator Testar, a test tool (and framework) to test applications at the GUI level whose objective is to solve part of the maintenance problem affecting tests by automatically generating test cases based on a structure that is automatically derived from the GUI. Testar is not included to replace Mr.Checker but rather to provide development teams with a series of interesting options which go beyond what Mr.Checker already provides.

Apart from Mr.Checker, engagements can now use Testar as an extra option for testing. This is a tool that enables the automated system testing of desktop, web and mobile applications at the GUI level. Testar has been added as an incubator to the platform awaiting further development during 2019.

The new incubator for node.js, called devon4node, has been included and implemented in several internal projects. This incubator is based on the Nest framework <https://www.nestjs.com/>. Nest is a framework for building efficient, scalable Node.js server-side applications. It uses progressive JavaScript, is built with TypeScript (preserves compatibility with pure JavaScript) and combines elements of OOP (Object Oriented Programming), FP (Functional Programming), and FRP (Functional Reactive Programming). Under the hood, Nest makes use of Express, but also provides compatibility with a wide range of other libraries (e.g. Fastify). This allows for easy use of the myriad third-party plugins which are available.

In order to facilitate the utilization of Microsoft Visual Studio Code in devonfw, we have developed and included the new devonfw Platform Extension Pack with lots of features to develop and test applications with this IDE in languages and frameworks such as TypeScript, JavaScript, .NET, Java, Rust, C++ and many more. More information at <https://marketplace.visualstudio.com/items?itemName=devonfw.devonfw-extension-pack>. Also, you can contribute to this extension in this GitHub repository <https://github.com/devonfw/devonfw-extension-pack-vscode>.

There is a whole range of new features and improvements which can be seen in that light. The My Thai Star sample app has now been upgraded to devon4j and devon4ng, a new devon4node backend implementation has been included that is seamless interchangeable, an E2E MrChecker sample project, CICD and deployment scripts and lots of bugs have been fixed.

Last but not least, the projects wikis and the devonfw Guide has once again been updated accordingly before the big refactor that will be addressed in the following release in 2019.

110.2. Changes and new features

110.2.1. Devonfw dist

- Eclipse 2018.9 integrated
 - CheckStyle Plugin updated.
 - SonarLint Plugin updated.
 - Git Plugin updated.
 - FindBugs Plugin updated.
 - CobiGen plugin updated.
- Other Software
 - Visual Studio Code latest version included and pre-configured with the devonfw Platform Extension Pack.
 - Ant updated to latest.
 - Maven updated to latest.

- Java updated to latest.
- Nodejs LTS updated to latest.
- @angular/cli included.
- Yarn package manager updated.
- Python3 updated.
- Spyder3 IDE integrated in python3 installation updated.
- devon4ng-application-template for Angular 7 at workspaces/examples
- devon4ng-ionic-application-template for Ionic 3.20 at workspace/samples

110.2.2. My Thai Star Sample Application

The new release of My Thai Star has focused on the following improvements:

- Release 1.12.2.
- devon4j:
 - devon4j 3.0.0 integrated.
 - Spring Boot 2.0.4 integrated.
 - Spring Data integration.
 - New pagination and search system.
 - Bug fixes.
- devon4ng:
 - Client devon4ng updated to Angular 7.
 - Angular Material and Covalent UI frameworks updated.
 - Electron framework integrated.
- devon4node
 - TypeScript 3.1.3.
 - Based on Nest framework.
 - Aligned with devon4j.
 - Complete backend implementation.
 - TypeORM integrated with SQLite database configuration.
 - Webpack bundler.
 - Nodemon runner.
 - Jest unit tests.
- Mr.Checker
 - Example cases for end-to-end test.
 - Production line configuration.
 - CICD

- Improved integration with Production Line
- New deployment from artifact
- New CICD pipelines
- New deployment pipelines
- Automated creation of pipelines in Jenkins

110.2.3. Documentation updates

The following contents in the devonfw guide have been updated:

- Upgrade of all the new devonfw named assets.
 - devon4j
 - devon4ng
 - Mr.Checker
- Electron integration cookbook.
- Updated cookbook about Swagger.
- Removed deprecated entries.

Apart from this the documentation has been reviewed and some typos and errors have been fixed.

The current development of the guide has been moved to <https://github.com/devonfw-forge/devon-guide/wiki> in order to be available as the rest of OSS assets.

110.2.4. devon4j

The following changes have been incorporated in devon4j:

- Spring Boot 2.0.4 Integrated.
- Spring Data layer Integrated.
- Decouple mmm.util.*
- Removed depreciated restaurant sample.
- Updated Pagination support for Spring Data
- Add support for hana as dbType.
- Bugfixes.

110.2.5. devon4ng

The following changes have been incorporated in devon4ng:

- New client application architecture guide <https://github.com/devonfw/devon4ng/wiki>
- Angular CLI 7,
- Angular 7,

-
- Angular Material 7 and Covalent 2.0.0-beta.7,
 - Ionic 3.20.0,
 - Cordova 8.0.0,
 - devon4ng Angular application template updated to Angular 7 with visual improvements and bugfixes <https://github.com/devonfw/devon4ng-application-template>
 - devon4ng Ionic application template updated and improved <https://github.com/devonfw/devon4ng-ionic-application-template>
 - PWA enabled.
 - Electron integrated to run My Thai Star as a desktop application in Windows, Linux or macOS.

110.2.6. devon4net

Some of the highlights of devon4net 1.0 are:

- External configuration file for each environment.
- .NET Core 2.1.X working solution (Latest 2.1.402).
- Packages and solution templates published on nuget.org.
- Full components customization by config file.
- Docker ready (My Thai Star sample fully working on docker).
- Port specification by configuration.
- Dependency injection by Microsoft .NET Core.
- Automapper support.
- Entity framework ORM (Unit of work, async methods).
- .NET Standard library 2.0 ready.
- Multi-platform support: Windows, Linux, Mac.
- Samples: My Thai Star back-end, Google API integration, Azure login, AOP with Castle.
- Documentation site.
- SPA page support.

And included the following features:

- Logging:
 - Text File.
 - Sqlite database support.
 - Serilog Seq Server support.
 - Graylog integration ready through TCP/UDP/HTTP protocols.
 - API Call params interception (simple and compose objects).
 - API error exception management.
- Swagger:

- Swagger auto generating client from comments and annotations on controller classes.
- Full swagger client customization (Version, Title, Description, Terms, License, Json endpoint definition).
- JWT:
 - Issuer, audience, token expiration customization by external file configuration.
 - Token generation via certificate.
 - MVC inherited classes to access JWT user properties.
 - API method security access based on JWT Claims.
- CORS:
 - Simple CORS definition ready.
 - Multiple CORS domain origin definition with specific headers and verbs.
- Headers:
 - Automatic header injection with middleware.
 - Supported header definitions: AccessControlExposeHeader, StrictTransportSecurityHeader, XFrameOptionsHeader, XssProtectionHeader, XContentTypeOptionsHeader, ContentSecurityPolicyHeader, PermittedCrossDomainPoliciesHeader, ReferrerPolicyHeader.
- Reporting server:
 - Partial implementation of reporting server based on My-FyiReporting (now runs on linux container).
- Testing:
 - Integration test template with sqlite support.
 - Unit test template.
 - Moq, xunit frameworks integrated.

110.2.7. devon4X

Some of the highlights of the new devonfw Xamarin framework are:

- Based on Excalibur framework by Hans Harts (<https://github.com/Xciles/Excalibur>).
- Updated to latest MVVMCross 6 version.
- My Thai Star Excalibur forms sample.
- Xamarin Forms template available on nuget.org.

110.2.8. AppSec Quick Solution Guide

This release incorporates a new Solution Guide for Application Security based on the state of the art in OWASP based application security. The purpose of this guide is to offer quick solutions for common application security issues for all applications based on devonfw. It's often the case that we need our systems to comply to certain sets of security requirements and standards. Each of these requirements needs to be understood, addressed and converted to code or project activity. We

want this guide to prevent the wheel from being reinvented over and over again and to give clear hints and solutions to common security problems.

- The wiki can be accessed here: <https://github.com/devonfw/devonfw-security/wiki>
- The PDF can be accessed here: <https://github.com/devonfw/devonfw-security>

110.2.9. CobiGen

- CobiGen core new features:
 - CobiGen_Templates will not need to be imported into the workspace anymore. However, If you want to adapt them, you can still click on a button that automatically imports them for you.
 - CobiGen_Templates can be updated by one-click whenever the user wants to have the latest version.
 - Added the possibility to reference external increments on configuration level. This is used for reducing the number of duplicated templates.
- CobiGen_Templates project and docs updated:
 - Spring standard has been followed better than ever.
 - Interface templates get automatically relocated to the api project. Needed for following the new devon4j standard.
- CobiGen Angular:
 - Angular 7 generation improved based on the updated application template.
 - Pagination changed to fit Spring standard.
- CobiGen Ionic: Pagination changed to fit Spring standard.
- CobiGen OpenAPI plugin released with multiple bug-fixes and other functionalities like:
 - Response and parameter types are parsed properly when they are a reference to an entity.
 - Parameters defined on the body of a request are being read correctly.

110.2.10. Devcon

A new version of Devcon has been released. Fixes and new features include:

- Updated to match current devon4j
- Update to download Linux distribution.
- Custom modules creation improvements.
- Code Migration feature added
- Bugfixes.

110.2.11. Devonfw OSS Modules

Modules upgraded to be used in new devon4j projects:

- Reporting module
- WinAuth AD Module
- WinAuth SSO Module
- I18n Module
- Async Module
- Integration Module
- Microservice Module
- Compose for Redis Module

See: <https://github.com/devonfw/devon/wiki#devonfw-modules>

110.2.12. Devonfw Testing

Mr.Checker

The Mr.Checker Test Framework is an automated testing framework for functional testing of web applications, API web services, Service Virtualization, Security and in coming future native mobile apps, and databases. All modules have tangible examples of how to build resilient integration test cases based on delivered functions. Mr.Checker updates and improvements:

- Examples available under embedded project “MrChecker-App-Under-Test” and in project wiki: <https://github.com/devonfw/devonfw-testing/wiki>
- How to install:
 - Wiki : <https://github.com/devonfw/devonfw-testing/wiki/How-to-install>
- Release Note:
 - module selenium - 3.8.1.13:
 - headless browser
 - enable browser options
 - module DevOps :
 - Jenkinsfile align with ProductionLine

Testar

We have added Test*, Testar, as an incubator to the available test tools within devonfw. This ground-breaking tool is being developed by the Technical University of Valencia (UPV). In 2019 Capgemini will co-develop Testar with the UPV.

Testar is a tool that enables the automated system testing of desktop, web and mobile applications at the GUI level.

With Testar, you can start testing immediately. It automatically generates and executes test sequences based on a structure that is automatically derived from the UI through the accessibility API. Testar can detect the violation of general-purpose system requirements and you can use

plugins to customize your tests.

You do not need test scripts and maintenance of it. The tests are random and are generated and executed automatically.

If you need to do directed tests you can create scripts to test specific requirements of your application.

Testar is included in the devonfw distro or can be downloaded from <https://testar.org/download/>.

The Github repository can be found at o: <https://github.com/TESTARtool/TESTAR>.

111. devonfw Release notes 2.4 “EVE”

111.1. Introduction

We are proud to announce the immediate release of devonfw version 2.4 (code named “EVE” during development). This version is the first one that fully embraces Open Source, including components like the documentation assets and CobiGen. Most of the IP (Intellectual Property or proprietary) part of devonfw are now published under the Apache License version 2.0 (with the documentation under the Creative Commons License (Attribution-NoDerivatives)). This includes the GitHub repositories where all the code and documentation is located. All of these repositories are now open for public viewing as well.

“EVE” contains a slew of new features but in essence it is already driven by what we expect to be the core focus of 2018: strengthening the platform and improving quality.

This release is also fully focused on deepening the platform rather than expanding it. That is to say: we have worked on improving existing features rather than adding new ones and strengthen the qualitative aspects of the software development life cycle, i.e. security, testing, infrastructure (CI, provisioning) etc.

“EVE” already is very much an example of this. This release contains the Allure Test Framework (included as an incubator in version 2.3) update called MrChecker Test Framework. MrChecker is an automated testing framework for functional testing of web applications, API web services, Service Virtualization, Security and in coming future native mobile apps, and databases. All modules have tangible examples of how to build resilient integration test cases based on delivered functions.

Another incubator being updated is the devonfw Shop Floor which intended to be a compilation of DevOps experiences from the devonfw perspective. A new part of the release is the new Solution Guide for Application Security based on the state of the art in OWASP based application security.

There is a whole range of new features and improvements which can be seen in that light. OASP4j 2.6 changes and improves the package structure of the core Java framework. The My Thai Star sample app has now been upgraded to Angular 6, lots of bugs have been fixed and the devonfw Guide has once again been improved.

Last but not least, this release contains the formal publication of the devonfw Methodology or The Accelerated Solution Design - an Industry Standards based solution design and specification (documentation) methodology for Agile (and less-than-agile) projects.

111.2. Changes and new features

111.2.1. devonfw 2.4 is Open Source

This version is the first release of devonfw that fully embraces Open Source, including components like the documentation assets and CobiGen. This is done in response to intensive market pressure and demands from the MU’s (Public Sector France, Netherlands)

Most of the IP (Intellectual Property or proprietary) part of devonfw are now published under the Apache License version 2.0 (with the documentation under the Creative Commons License (Attribution-NoDerivatives)).

So you can now use the devonfw distribution (the "zip" file), CobiGen, the devonfw modules and all other components without any worry to expose the client unwittingly to Capgemini IP.

Note: there are still some components which are IP and are not published under an OSS license. The class room trainings, the Sencha components and some CobiGen templates. But these are not included in the distribution nor documentation and are now completely maintained separately.

111.2.2. devonfw dist

- Eclipse Oxygen integrated
 - CheckStyle Plugin updated.
 - SonarLint Plugin updated.
 - Git Plugin updated.
 - FindBugs Plugin updated.
 - CobiGen plugin updated.
- Other Software
 - Visual Studio Code latest version included and pre-configured with <https://github.com/oasp/oasp-vscode-ide>
 - Ant updated to latest.
 - Maven updated to latest.
 - Java updated to latest.
 - Nodejs LTS updated to latest.
 - @angular/cli included.
 - Yarn package manager updated.
 - Python3 updated.
 - Spyder3 IDE integrated in python3 installation updated.
 - OASP4JS-application-template for Angular 6 at workspaces/examples

111.2.3. My Thai Star Sample Application

The new release of My Thai Star has focused on the following improvements:

- Release 1.6.0.
- Travis CI integration with Docker. Now we get a valuable feedback of the current status and when collaborators make pull requests.
- Docker compose deployment.
- OASP4J:

- Flyway upgrade from 3.2.1 to 4.2.0
- Bug fixes.
- OASP4JS:
 - Client OASP4JS updated to Angular 6.
 - Frontend translated into 9 languages.
 - Improved mobile and tablet views.
 - Routing fade animations.
 - Compodoc included to generate dynamically frontend documentation.

111.2.4. Documentation updates

The following contents in the devonfw guide have been updated:

- devonfw OSS modules documentation.
- Creating a new OASP4J application.
- How to update Angular CLI in devonfw.
- Include Angular i18n.

Apart from this the documentation has been reviewed and some typos and errors have been fixed.

The current development of the guide has been moved to <https://github.com/oasp-forge/devon-guide/wiki> in order to be available as the rest of OSS assets.

111.2.5. OASP4J

The following changes have been incorporated in OASP4J:

- Integrate batch with archetype.
- Application module structure and dependencies improved.
- Issues with Application Template fixed.
- Solved issue where Eclipse maven template oasp4j-template-server version 2.4.0 produced pom with missing dependency spring-boot-starter-jdbc.
- Solved datasource issue with project archetype 2.4.0.
- Decouple archetype from sample (restaurant).
- Upgrade to Flyway 4.
- Fix for issue with Java 1.8 and QueryDSL #599.

111.2.6. OASP4JS

The following changes have been incorporated in OASP4JS:

- First version of the new client application architecture guide <https://github.com/oasp-forge/oasp4js-wiki/wiki>

- Angular CLI 6,
- Angular 6,
- Angular Material 6 and Covalent 2.0.0-beta.1,
- Ionic 3.20.0,
- Cordova 8.0.0,
- OASP4JS Angular application template updated to Angular 6 with visual improvements and bugfixes <https://github.com/oasp/oasp4js-application-template>
- OASP4JS Ionic application template updated and improved <https://github.com/oasp/oasp4js-ionic-application-template>
- PWA enabled.

111.2.7. AppSec Quick Solution Guide

This release incorporates a new Solution Guide for Application Security based on the state of the art in OWASP based application security. The purpose of this guide is to offer quick solutions for common application security issues for all applications based on devonfw. It's often the case that we need our systems to comply to certain sets of security requirements and standards. Each of these requirements needs to be understood, addressed and converted to code or project activity. We want this guide to prevent the wheel from being reinvented over and over again and to give clear hints and solutions to common security problems.

- The wiki can be accessed here: <https://github.com/devonfw/devonfw-security/wiki>
- The PDF can be accessed here: <https://github.com/devonfw/devonfw-security>

111.2.8. CobiGen

- CobiGen_Templates project and docs updated.
- CobiGen Angular 6 generation improved based on the updated application template
- CobiGen Ionic CRUD App generation based on Ionic application template. Although a first version was already implemented, it has been deeply improved:
 - Changed the code structure to comply with Ionic standards.
 - Added pagination.
 - Pull-to-refresh, swipe and attributes header implemented.
 - Code documented and JSDoc enabled (similar to Javadoc)
- CobiGen TSPlugin Interface Merge support.
- CobiGen XML plugin comes out with new cool features:
 - Enabled the use of XPath within variable assignment. You can now retrieve almost any data from an XML file and store it on a variable for further processing on the templates. Documented here.
 - Able to generate multiple output files per XML input file.
 - Generating code from UML diagrams. XMI files (standard XML for UML) can be now read

and processed. This means that you can develop templates and generate code from an XMI like class diagrams.

- CobiGen OpenAPI plugin released with multiple bug-fixes and other functionalities like:
 - Assigning global and local variables is now possible. Therefore you can set any string for further processing on the templates. For instance, changing the root package name of the generated files. Documented here.
 - Enabled having a class with more than one relationship to another class (more than one property of the same type).
- CobiGen Text merger plugin has been extended and now it is able to merge text blocks. This means, for example, that the generation and merging of AsciiDoc documentation is possible. Documented here.

111.2.9. Devcon

A new version of Devcon has been released. Fixes and new features include:

- Now Devcon is OSS, with public repository at <https://github.com/devonfw/devcon>
- Updated to match current OASP4J
- Update to download Linux distribution.
- Custom modules creation improvements.
- Bugfixes.

111.2.10. devonfw OSS Modules

- Existing devonfw IP modules have been moved to OSS.
 - They can now be accessed in any OASP4J project as optional dependencies from Maven Central.
 - The repository now has public access <https://github.com/devonfw/devon>
- Starters available for modules:
 - Reporting module
 - WinAuth AD Module
 - WinAuth SSO Module
 - I18n Module
 - Async Module
 - Integration Module
 - Microservice Module
 - Compose for Redis Module

See: <https://github.com/devonfw/devon/wiki#devonfw-modules>

111.2.11. devonfw Shop Floor

- devonfw Shop Floor 4 Docker
 - Docker-based CI/CD environment
 - docker-compose.yml (installation file)
 - dsf4docker.sh (installation script)
 - Service Integration (documentation in Wiki)
 - devonfw projects build and deployment with Docker
 - Dockerfiles (multi-stage building)
 - Build artifact (NodeJS for Angular and Maven for Java)
 - Deploy built artifact (NGINX for Angular and Tomcat for Java)
 - NGINX Reverse-Proxy to redirect traffic between both Angular client and Java server containers.
- devonfw Shop Floor 4 OpenShift
 - devonfw projects deployment in OpenShift cluster
 - s2i images
 - OpenShift templates
 - Video showcase (OpenShift Origin 3.6)

This incubator is intended to be a compilation of DevOps experiences from the devonfw perspective. “How we use our devonfw projects in DevOps environments”. Integration with the Production Line, creation and service integration of a Docker-based CI environment and deploying devonfw applications in an OpenShift Origin cluster using devonfw templates. See: <https://github.com/devonfw/devonfw-shop-floor>

111.2.12. devonfw Testing

The MrChecker Test Framework is an automated testing framework for functional testing of web applications, API web services, Service Virtualization, Security and in coming future native mobile apps, and databases. All modules have tangible examples of how to build resilient integration test cases based on delivered functions.

- Examples available under embedded project “MrChecker-App-Under-Test” and in project wiki: <https://github.com/devonfw/devonfw-testing/wiki>
- How to install:
 - Wiki : <https://github.com/devonfw/devonfw-testing/wiki/How-to-install>
- Release Note:
 - module core - 4.12.0.8:
 - fixes on getting Environment values
 - top notch example how to keep vulnerable data in repo , like passwords

- module selenium - 3.8.1.8:
 - browser driver auto downloader
 - list of out of the box examples to use in any web page
- module webAPI - ver. 1.0.2 :
 - api service virtualization with REST and SOAP examples
 - api service virtualization with dynamic arguments
 - REST working test examples with page object model
- module security - 1.0.1 (security tests against My Thai Start)
- module DevOps :
 - dockerfile for Test environment execution
 - CI + CD as Jenkinsfile code

111.2.13. devonfw methodology: Accelerated Solution Design

One of the prime challenges in Distributed Agile Delivery is the maintenance of a common understanding and unity of intent among all participants in the process of creating a product. That is: how can you guarantee that different parties in the client, different providers, all in different locations and time zones during a particular period of time actually understand the requirements of the client, the proposed solution space and the state of implementation.

We offer the Accelerated Solution Design as a possible answer to these challenges. The ASD is carefully designed to be a practical guideline that fosters and ensures the collaboration and communication among all team members.

The Accelerated Solution Design is:

- A practical guideline rather than a “methodology”
- Based on industry standards rather than proprietary methods
- Consisting of an evolving, “living”, document set rather than a static, fixed document
- Encapsulating the business requirements, functional definitions as well as Architecture design
- Based on the intersection of Lean, Agile, DDD and User Story Mapping

And further it is based on the essential belief or paradigm that ASD should be:

- Focused on the design (definition) of the “externally observable behavior of a system”
- Promoting communication and collaboration between team members
- Guided by prototypes

For more on the devonfw Methodology / ASD, see: https://github.com/devonfw/devon-methodology/blob/master/design-guidelines/Accelerated_Solution_Design.adoc

112. devonfw Release notes 2.3 "Dash"

112.1. Release: improving & strengthening the Platform

We are proud to announce the immediate release of **devonfw version 2.3** (code named “*Dash*” during development). This release comes with a bit of a delay as we decided to wait for the publication of OASP4j 2.5. “*Dash*” contains a slew of new features but in essence it is already driven by what we expect to be the core focus of 2018: strengthening the platform and improving quality.

After one year and a half of rapid expansion, we expect the next release(s) of the devonfw 2.x series to be fully focused on deepening the platform rather than expanding it. That is to say: we should work on improving existing features rather than adding new ones and strengthen the qualitative aspects of the software development life cycle, i.e. testing, infrastructure (CI, provisioning) etc.

“*Dash*” already is very much an example of this. This release contains the Allure Test Framework as an incubator. This is an automated testing framework for functional testing of web applications. Another incubator is the devonfw Shop Floor which intended to be a compilation of DevOps experiences from the devonfw perspective. And based on this devonfw has been *OpenShift Primed* (“certified”) by Red Hat.

There is a whole range of new features and improvements which can be seen in that light. OASP4j 2.5 changes and improves the package structure of the core Java framework. The My Thai Star sample app has now been fully integrated in the different frameworks and the devonfw Guide has once again been significantly expanded and improved.

112.2. An industrialized platform for the ADcenter

Although less visible to the overall devonfw community, an important driving force was (meaning that lots of work has been done in the context of) the creation of the ADcenter concept towards the end of 2017. Based on a radical transformation of on/near/offshore software delivery, the focus of the ADcenters is to deliver agile & accelerated “Rightshore” services with an emphasis on:

- Delivering Business Value and optimized User Experience
- Innovative software development with state of the art technology
- Highly automated devops; resulting in lower costs & shorter time-to-market

The first two ADcenters, in Valencia (Spain) and Bangalore (India), are already servicing clients all over Europe - Germany, France, Switzerland and the Netherlands - while ADcenter aligned production teams are currently working for Capgemini UK as well (through Spain). Through the ADcenter, Capgemini establishes industrialized innovation; designed for & with the user. The availability of platforms for industrialized software delivery like devonfw and the Production Line has allowed us to train and make available over a 150 people in very short time.

The creation of the ADcenter is such a short time is visible proof that we’re getting closer to a situation where devonfw and Production Line are turning into the default development platform

for APPS2, thereby standardizing all aspects of the software development life cycle: from training and design, architecture, devops and development, all the way up to QA and deployment.

112.3. Changes and new features

112.3.1. devonfw dist

The **devonfw dist**, or distribution, i.e. the central zip file which contains the main working environment for the devonfw developer, has been significantly enhanced. New features include:

- Eclipse Oxygen integrated
 - CheckStyle Plugin installed and configured
 - SonarLint Plugin installed and configured
 - Git Plugin installed
 - FindBugs replaced by SpotBugs and configured
 - Tomcat8 specific Oxygen configuration
 - CobiGen Plugin installed
- Other Software
 - Cmdr integrated (when console.bat launched)
 - Visual Studio Code latest version included and pre-configured with <https://github.com/devonfw/extension-pack-vscode>
 - Ant updated to latest.
 - Maven updated to latest.
 - Java updated to latest.
 - Nodejs LTS updated to latest.
 - @angular/cli included.
 - Yarn package manager included.
 - Python3 integrated
 - Spyder3 IDE integrated in python3 installation
 - OASP4JS-application-template for Angular5 at workspaces/examples
 - Devon4sencha starter templates updated

112.3.2. OASP4j 2.5

Support for JAX-RS & JAX-WS clients

With the aim to enhance the ease in consuming RESTful and SOAP web services, JAX-RS and JAX-WS clients have been introduced. They enable developers to concisely and efficiently implement portable client-side solutions that leverage existing and well-established client-side HTTP connector implementations. Furthermore, the getting started time for consuming web services has been

considerably reduced with the default configuration out-of-the-box which can be tweaked as per individual project requirements.

See: <https://github.com/oasp/oasp4j/issues/358>

Separate security logs for OASP4J log component

Based on OWASP(Open Web Application Security Project), OASP4J aims to give developers more control and flexibility with the logging of security events and tracking of forensic information. Furthermore, it helps classifying the information in log messages and applying masking when necessary. It provides powerful security features while based on set of logging APIs developers are already familiar with over a decade of their experience with Log4J and its successors.

See: <https://github.com/oasp/oasp4j/issues/569>

Support for Microservices

Integration of an OASP4J application to a Microservices environment can now be leveraged with this release of OASP4J. Introduction of service clients for RESTful and SOAP web services based on Java EE give developers agility and ease to access microservices in the Devon framework. It significantly cuts down the efforts on part of developers around boilerplate code and stresses more focus on the business code improving overall efficiency and quality of deliverables.

See: <https://github.com/oasp/oasp4j/pull/589/commits>

112.3.3. Cobigen

A new version of Cobigen has been included. New features include:

- Swagger/Yaml Plugin for CobiGen. CobiGen is able to read a swagger definition file that follows the OpenAPI 3.0 spec and generate code. A preliminary release was already included in 2.2.1 but the current version is much more mature and stable. See: https://github.com/devonfw/cobigen/wiki/howto_openapi_generation
- Integration of CobiGen into Maven build process. This already existed but has been improved. It consists mainly of documentation + better log output and bug fixes. See: https://github.com/devonfw/cobigen/wiki/cobigen-maven_configuration
- CobiGen Ionic CRUD App generation based on <https://github.com/oasp/oasp4js-ionic-application-template>
- Cobigen_Templates project and docs updated
- Bugfixes and Hardening

112.3.4. My Thai Star Sample Application

From this release on the My Thai Star application has been fully integrated in the different frameworks in the platform. Further more, a more modularized approach has been followed in the current release of My Thai star application to decouple client from implementation details. Which provides better encapsulation of code and dependency management for API and implementation classes. This has been achieved with creation of a new “API” module that contain interfaces for

REST services and corresponding Request/Response objects. With existing “Core” module being dependent on “API” module. To read further you can follow the link <https://github.com/oasp/my-thai-star/wiki/java-design#basic-architecture-details>

Furthermore: an email and Twitter micro service were integrated in my-thai-star. This is just for demonstration purposes. A full micro service framework is already part of oasp4j 2.5.0

112.3.5. Documentation refactoring

The complete devonfw guide is restructured and refactored. Getting started guides are added for easy start with devonfw. Integration of the new Tutorial with the existing devonfw Guide whereby existing chapters of the previous tutorial were converted to Cookbook chapters. Asciidoc is used for devonfw guide PDF generation. See: <https://github.com/devonfw/devon-guide/wiki>

112.3.6. OASP4JS

The following changes have been incorporated in OASP4JS:

- Angular CLI 1.6.0,
- Angular 5.1,
- Angular Material 5 and Covalent 1.0.0 RC1,
- PWA enabled,
- Core and Shared Modules included to follow the recommended Angular projects structure,
- Yarn and NPM compliant since both lock files are included in order to get a stable installation.

112.3.7. Admin interface for oasp4j apps

The new version includes an Integration of an admin interface for oasp4j apps (Spring Boot). This module is based on CodeCentric’s Spring Boot Admin (<https://github.com/codecentric/spring-boot-admin>). See: <https://github.com/devonfw/devon-guide/wiki/Spring-boot-admin-Integration-with-devon4j>

112.3.8. Devcon

A new version of Devcon has been released. Fixes and new features include:

- Renaming of system Commands.
- New menu has been added - “other modules”, if menus are more than 10, other modules will display some menus.
- A progress bar has been added for installing the distribution

112.3.9. devonfw Modules

Existing devonfw modules can now be accessed with the help of starters following namespace devonfw-<module_name>-starter. Starters available for modules:

- Reporting module

- WinAuth AD Module
- WinAuth SSO Module
- I18n Module
- Async Module
- Integration Module
- Microservice Module
- Compose for Redis Module

See: <https://github.com/devonfw/devon/wiki#ip-modules>

112.3.10. devonfw Shop Floor

This incubator is intended to be a compilation of DevOps experiences from the devonfw perspective. “How we use our devonfw projects in DevOps environments”. Integration with the Production Line, creation and service integration of a Docker-based CI environment and deploying devonfw applications in an OpenShift Origin cluster using devonfw templates.

See: <https://github.com/devonfw/devonfw-shop-floor>

112.3.11. devonfw-testing

The Allure Test Framework is an automated testing framework for functional testing of web applications and in coming future native mobile apps, web services and databases. All modules have tangible examples of how to build resilient integration test cases based on delivered functions.

- Examples available under embedded project “Allure-App-Under-Test” and in project wiki: <https://github.com/devonfw/devonfw-testing/wiki>
- How to install: <https://github.com/devonfw/devonfw-testing/wiki/How-to-install>
- Release Notes:
 - Core Module – ver.4.12.0.3:
 - Test report with logs and/or screenshots
 - Test groups/tags
 - Data Driven (inside test case, external file)
 - Test case parallel execution
 - Run on independent Operating System (Java)
 - Externalize test environment (DEV, QA, PROD)
 - UI Selenium module – ver. 3.4.0.3:
 - Malleable resolution (Remote Web Design, Mobile browsers)
 - Support for many browsers(Internet Explorer, Edge, Chrome, Firefox, Safari)
 - User friendly actions (elementCheckBox, elementDropdown, etc.)
 - Ubiquese test execution (locally, against Selenium Grid through Jenkins)

- Page Object Model architecture
- Selenium WebDriver library ver. 3.4.0

See: <https://github.com/devonfw/devonfw-testing/wiki>

112.3.12. DOT.NET Framework incubators

The .NET Core and Xamarin frameworks are still under development by a workgroup from The Netherlands, Spain, Poland, Italy, Norway and Germany. The 1.0 release is expected to be coming soon but the current incubator frameworks are already being used in several engagements. Some features to highlight are:

- Full .NET implementation with multi-platform support
- Detailed documentation for developers
- Docker ready
- Web API server side template :
 - Swagger auto-generation
 - JWT security
 - Entity Framework Support
 - Advanced log features
- Xamarin Templates based on Excalibur framework
- My Thai Star implementation:
 - Backend (.NET Core)
 - FrontEnd (Xamarin)

112.3.13. devonfw has been Primed by Red Hat for OpenShift

OpenShift is a supported distribution of Kubernetes from Red Hat for container-based software deployment and management. It is using Docker containers and DevOps tools for accelerated application development. Using OpenShift allows Capgemini to avoid Cloud Vendor lock-in. OpenShift provides devonfw with a state of the art CI/CD environment (devonfw Shop Floor), providing devonfw with a platform for the whole development life cycle: from development to staging / deploy.

See <https://hub.openshift.com/primed/120-capgemini> and <https://github.com/oasp/s2i>

112.3.14. Harvested components and modules

The devonfw Harvesting process continues to add valuable components and modules to the devonfw platform. The last months the following elements were contributed:

Service Client support (for Micro service Projects).

This client is for consuming microservices from other application. This solution is already very

flexible and customizable. As of now, this is suitable for small and simple project where two or three microservices are invoked. Donated by Jörg Hohwiller. See: <https://github.com/devonfw/devon-microservices>

JHipster devonfw code generation

This component was donated by the ADcenter in Valencia. It was made in order to comply with strong requirements (especially from the French BU) to use jHipster for code generation.

JHipster is a code generator based on Yeoman generators. Its default generator generator-jhipster generates a specific JHipster structure. The purpose of generator-jhipster-DevonModule is to generate the structure and files of a typical OASP4j project. It is therefore equivalent to the standard OASP4j application template based CobiGen code generation.

See: <https://github.com/devonfw/devon-guide/wiki/cookbook-devon-jhipster-module>

Simple Jenkins task status dashboard

This component has been donated by, has been harvested from system in use by, Capgemini Valencia. This dashboard, apart from an optional gamification element, allows the display of multiple Jenkins instances. See: https://github.com/oasp/jenkins_view

112.3.15. And lots more, among others:

- OASP4J/devonfw docker based build IN a docker process. See: <https://github.com/devonfw/devon-guide/wiki/Dockerfile-for-the-maven-based-spring.io-projects>
- CI test boot archetype. This is for unit testing. This will create a sample project and add sample web service to it. A Jenkins job will start oasp4j server and will call web service. See: <https://github.com/devonfw/devonfw-shop-floor/tree/master/testing/Oasp4jTestingScripts>
- CI test Angular starterTemplate. Testing automation for Angular applications (My Thai Star) in Continuous Integration environments by using Headless browsers and creating Node.js scripts. See: <https://github.com/oasp/my-thai-star/blob/develop/angular/package.json#L8-L12> and <https://github.com/oasp/my-thai-star/blob/develop/angular/karma.conf.js>

113. devonfw Release notes 2.2 "Courage"

113.1. Production Line Integration

devonfw is now fully supported on the Production Line v1.3 and the coming v2.0. Besides that, we now "eat our own dogfood" as the whole devonfw project, all "buildable assets", now run on the Production Line.

113.2. OASP4js 2.0

The main focus of the Courage release is the renewed introduction of "OASP for JavaScript", or OASP4js. This new version is a completely new implementation based on Angular (version 4). This new "stack" comes with:

- New application templates for Angular 4 application (as well as Ionic 3)
- A new reference application
- A new tutorial (and Architecture Guide following soon)
- Component Gallery
- New CobiGen templates for generation of both Angular 4 and Ionic 3 UI components ("screens")
- Integration of Covalent and Bootstrap offering a large number of components
- my-thai-star, a showcase and reference implementation in Angular of a real, responsive usable app using recommended architecture and patterns
- A new Tutorial using my-thai-star as a starting point

See: <https://github.com/oasp/oasp4js-application-template> <https://github.com/oasp/oasp4js-angular-catalog> <https://github.com/oasp/my-thai-star/tree/develop/angular>

113.3. A new OASP Portal

As part of the new framework(s) we have also done a complete redesign of the OASP Portal website at <http://oasp.io/> which should make all things related with OASP more accessible and easier to find.

113.4. New Cobigen

Major changes in this release:

- Support for multi-module projects
- Client UI Generation:
 - New Angular 4 templates based on the latest - angular project seed
 - Basic Typescript Merger
 - Basic Angular Template Merger
 - JSON Merger

- Refactored oasp4j templates to make use of Java template logic feature
- Bugfixes:
 - Fixed merging of nested Java annotations including array values
 - more minor issues
- Under the hood:
 - Large refactoring steps towards language agnostic templates formatting sensitive placeholder descriptions automatically formatting camelCase to TrainCase to snake-case, etc.
- Easy setup of CobiGen IDE to enable fluent contribution
- CI integration improved to integrate with GitHub for more valuable feedback

See: <https://github.com/devonfw/cobigen/releases>

113.5. MyThaiStar: New Restaurant Example, reference implementation & Methodology showcase

A major part of the new devonfw release is the incorporation of a new application, "my-thai-star" which among others:

- serve as an example of how to make a "real" devonfw application (i.e. the application could be used for real)
- Serves as an attractive showcase
- Serves as a reference application of devonfw patterns and practices as well as the standard example in the new devonfw tutorial
- highlights modern security option like JWT Integration

The application is accompanied by a substantial new documentation asset, the devonfw methodology, which described in detail the whole lifecycle of the development of a devonfw application, from requirements gathering to technical design. Officially my-thai-star is still considered to be an incubator as especially this last part is still not as mature as it could be. But the example application and tutorial are 100% complete and functional and form a marked improvement over the "old" restaurant example app. My-Thai-star will become the standard example app from devonfw 3.0 onwards.

See: <https://github.com/oasp/my-thai-star> <https://github.com/oasp/my-thai-star/wiki>

113.6. The new OASP Tutorial

The OASP Tutorial is a new part of the combined OASP / devonfw documentation which changes the focus of how people can get started with the platform

There are tutorials for OASP4j, OASP4js (Angular), OASP4fn and more to come. My-Thai-Star is used throughout the tutorial series to demonstrate the basic principles, architecture, and good practices of the different OASP "stacks". There is an elaborated exercise where the readers get to write their

own application "JumpTheQueue".

We hope that the new tutorial offers a better, more efficient way for people to get started with devonfw. Answering especially the question: how to make a devonfw application.

Oasp4j tutorial: <https://github.com/oasp/oasp-tutorial-sources/wiki/OASP4jGettingStartedHome>
Oasp4js tutorial: <https://github.com/oasp/oasp-tutorial-sources/wiki/OASP4jsGettingStartedHome>
Oasp4fn tutorial: <https://github.com/oasp/oasp-tutorial-sources/wiki/OASP4FnGettingStartedHome>

113.7. OASP4j 2.4.0

"OASP for Java" or OASP4j now includes updated versions of the latest stable versions of Spring Boot and the Spring Framework and all related dependencies. This allows guaranteed, stable, execution of any devonfw 2.X application on the latest versions of the Industry Standard Spring stack. Another important new feature is a new testing architecture/infrastructure. All database options are updated to the latest versions as well as guaranteed to function on all Application Servers which should cause less friction and configuration time when starting a new OASP4j project.

Details:

- Spring Boot Upgrade to 1.5.3
- Updated all underlying dependencies
- Spring version is 4.3.8
- Exclude Third Party Libraries that are not needed from sample restaurant application
- Bugfix:Fixed the 'WhiteLabel' error received when tried to login to the sample restaurant application that is deployed onto external Tomcat
- Bugfix:Removed the API `api.org.apache.catalina.filters.SetCharacterEncodingFilter` and used spring framework's API `org.springframework.web.filter.CharacterEncodingFilter` instead
- Bugfix:Fixed the error "class file for javax.interceptor.InterceptorBinding not found" received when executing the command 'mvn site' when trying to generate javadoc using Maven javadoc plugin
- Removed `io.oasp.module.web.common.base.PropertiesWebApplicationContextInitializer` the deprecated API
- Documentation of the usage of `UserDetailsService` of Spring Security

See: <https://github.com/oasp/oasp4j>

Wiki: <https://github.com/oasp/oasp4j/wiki>

113.8. Microservices Netflix

devonfw now includes a microservices implementation based on Spring Cloud Netflix. It provides a Netflix OSS integrations for Spring Boot apps through auto-configuration and binding to the Spring Environment. It offers microservices archetypes and a complete user guide with all the details to

start creating microservices with devonfw.

See: <https://github.com/devonfw-forge/devon-guide/wiki/devon-microservices>

113.9. devonfw distribution based on Eclipse OOMPH

The new Eclipse devonfw distribution is now based on Eclipse OOMPH, which allows us, an any engagement, to create and manage the distribution more effectively by formalizing the setup instructions so they can be performed automatically (due to a blocking issue postponed to devonfw 2.2.1 which will be released a few weeks after 2.2.0)

113.10. Visual Studio Code / Atom

The devonfw distro now contains Visual Studio Code alongside Eclipse in order to provide a default, state of the art, environment for web based development.

See: <https://github.com/oasp/oasp-vscode-ide>

113.11. More I18N options

The platform now contains more documentation and a conversion utility which makes it easier to share i18n resource files between the different frameworks.

See: <https://github.com/devonfw/devon/wiki/cookbook-i18n-resource-converter>

113.12. Spring Integration as devonfw Module

This release includes a new module based on the Java Message Service (JMS) and Spring Integration which provides a communication system (sender/subscriber) out-of-the-box with simple channels (only to send and read messages), request and reply channels (to send messages and responses) and request & reply asynchronously channels.

See: <https://github.com/devonfw/devon/wiki/cookbook-integration-module>

113.13. devonfw Harvest contributions

devonfw contains a whole series of new components obtained through the Harvesting process. Examples are :

- New backend IP module Compose for Redis: management component for cloud environments. Redis is an open-source, blazingly fast, key/value low maintenance store. Compose's platform gives you a configuration pre-tuned for high availability and locked down with additional security features. The component will manage the service connection and the main methods to manage the key/values on the storage. The library used is "lettuce".
- Sencha component for extending GMapPanel with the following functionality :
 - Markers management

- Google Maps options management
 - Geoposition management
 - Search address and coordinates management
 - Map events management
 - Map life cycle and behavior management
- Sencha responsive Footer that moves from horizontal to vertical layout depending on the screen resolution or the device type. It is a simple functionality but we consider it very useful and reusable.

See: <https://github.com/devonfw/devon/wiki/cookbook-compose-for-redis-module>

113.14. More Deployment options to JEE Application Servers and Docker/CloudFoundry

The platform now fully supports deployment on the latest version of Weblogic, WebSphere, Wildfly (JBoss) as well as Docker and Cloud Foundry.

See: <https://github.com/devonfw/devon/wiki/Deployment-on-WebLogic> <https://github.com/devonfw/devon/wiki/cookbook-Deployment-on-WebSphere> <https://github.com/devonfw/devon/wiki/cookbook-Deployment-on-Wildfly>

113.15. Devcon on Linux

Devcon is now fully supported on Linux which, together with the devonfw distro running on Linux, makes devonfw fully multi-platform and Cloud compatible (as Linux is the default OS in the Cloud!)

See: <https://github.com/devonfw/devcon/releases>

113.16. New OASP Incubators

From different Business Units (countries) have contributed "incubator" frameworks:

- OASP4NET (Stack based on .NET Core / .NET "Classic" (4.6))
- OASP4X (Stack based on Xamarin)
- OASP4Fn (Stack based on Node.js/Serverless): <https://github.com/oasp/oasp4fn>

An "incubator" status means that the frameworks are production ready, all are actually already used in production, but are still not fully compliant with the OASP definition of a "Minimally Viable Product".

During this summer the OASP4NET and OASP4X repos will be properly installed. In the mean time, if you want to have access to the source code, please contact the *devonfw Core Team*.

114. Release notes devonfw 2.1.1 "Balu"

114.1. Version 2.1.2: OASP4J updates & some new features

We've released the latest update release of devonfw in the *Balu* series: version 2.1.2. The next major release, code named *Courage*, will be released approximately the end of June. This current release contains the following items:

114.1.1. OASP4j 2.3.0 Release

Friday the 12th of May 2017 OASP4J version 2.3.0 was released. Major features added are :

- Database Integration with PostGres, MSSQL Server, MariaDB
- Added docs folder for gh pages and added oomph setups
- Refactored Code
- Refactored Test Infrastructure
- Added Documentation on debugging tests
- Added Two Batch Job tests in the restaurant sample
- Bugfix: Fixed the error received when the Spring Boot Application from sample application that is created from maven archetype is launched
- Bugfix: Fix for 404 error received when clicked on the link '1. Table' in index.html of the sample application created from maven archetype

More details on features added can be found at <https://github.com/oasp/oasp4j/milestone/23?closed=1>. The OASP4j wiki and other documents are updated for release 2.3.0.

114.1.2. CobiGen Enhancements

Previous versions of CobiGen are able to generate code for REST services only. Now it is possible to generate the code for SOAP services as well. There are two use cases available in CobiGen:

- SOAP without nested data
- SOAP nested data

The "nested data" use case is when there are 3 or more entities which are interrelated with each other. CobiGen will generate code which will return the nested data. Currently CobiGen services return ETO classes, CobiGen has been enhanced as to return CTO classes (ETO + relationship).

Apart from the SOAP code generation, the capability to express nested relationships have been added to the existing ReST code generator as well.

See: <https://github.com/devonfw/devon-guide/wiki/cookbook-cobigen-advanced-use-cases-soap-and-nested-data>

114.1.3. Micro services module (Spring Cloud/Netflix OSS)

To make it easier for devonfw users to design and develop applications based on microservices, this release provides a series of archetypes and resources based on *Spring Cloud Netflix* to automate the creation and configuration of microservices.

New documentation in the devonfw Guide contains all the details to start [creating microservices with devonfw](#)

114.1.4. Spring Integration Module

Based on the *Java Message Service* (JMS) and *Spring Integration*, the devonfw *Integration module* provides a communication system (sender/subscriber) out-of-the-box with simple channels (only to send and read messages), request and reply channels (to send messages and responses) and request & reply asynchronously channels. You can find more details about the implementation in the [devonfw guide](#).

114.1.5. WebSphere & Wildfly deployment documentation

The new version of devonfw contains more elaborate and updated documentation about deployment on [WebSphere](#) and [Wildfly](#).

114.2. Version 2.1.1 Updates, fixes & some new features

114.2.1. CobiGen code-generator fixes

The CobiGen incremental code generator released in the previous version contained a regression which has now been fixed. Generating services in Batch mode whereby a package can be given as an input, using all Entities contained in that package, works again as expected.

For more information see: [The CobiGen documentation](#) and the corresponding change in the [devonfw Guide](#)

114.2.2. Devcon enhancements

In this new release we have added devcon to the devonfw distribution itself so one can directly use devcon from the console.bat or ps-console.bat windows. It is therefore no longer necessary to independently install devcon. However, as devcon is useful outside of the devonfw distribution, this remains a viable option.

114.2.3. Devon4Sencha

in Devon4Sencha there are changes in the sample application. It now complies fully with the architecture which is known as "universal app", so now it has screens custom tailored for desktop and mobile devices. All the basic logic remains the same for both versions. (The StarterTemplate is still only for creating a desktop app. This will be tackled in the next release.)

114.2.4. New Winauth modules

The original *winauth* module that, in previous Devon versions, implemented the *Active Directory* authentication and the *Single Sign-on* authentication now has been divided in two independent modules. The *Active Directory* authentication now is included in the new *Winauth-ad* module whereas the *Single Sign-on* implementation is included in a separate module called *Winauth-sso*. Also some improvements have been added to *Winauth-sso* module to ease the way in which the module can be injected.

For more information about the update see: [The Sencha docs within the devonfw Guide](#)

114.2.5. General updates

There are a series of updates to the devonfw documentation, principally the devonfw Guide. Further more, from this release on, you can find the devonfw guide in the *doc* folder of the distribution.

Furthermore, the OASP4J and devonfw source-code in the "examples" workspace, have been updated to the latest version.

114.3. Version 2.1 New features, improvements and updates

114.3.1. Introduction

We are proud to present the new release of devonfw, version "2.1" which we've baptized "Balu". A major focus for this release is developer productivity. So that explains the name, as Balu is not just big, friendly and cuddly but also was very happy to let Mowgli do the work for him.

114.3.2. Cobigen code-generator UI code generation and more

The Cobigen incremental code generator which is part of devonfw has been significantly improved. Based on a single data schema it can generate the JPA/Hibernate code for the whole service layer (from data-access code to web services) for all CRUD operations. When generating code, Cobigen is able to detect and leave untouched any code which developers have added manually.

In the new release it supports Spring Data for data access and it is now capable of generating the whole User Interface as well: data-grids and individual rows/records with support for filters, pagination etc. That is to say: Cobigen can now generate automatically all the code from the server-side database access layer all the way up to the UI "screens" in the web browser.

Currently we support Sencha Ext JS with support for Angular 2 coming soon. The code generated by Cobigen can be opened and used by Sencha Architect, the visual design tool, which enables the programmer to extend and enhance the generated UI non-programmatically. When Cobigen regenerates the code, even those additions are left intact. All these features combined allow for an iterative, incremental way of development which can be up to an order of magnitude more productive than "programming manual"

Cobigen can now also be used for code-generation within the context of an engagement. It is easily extensible and the process of how to extend it for your own project is well documented. This becomes already worthwhile ("delivers ROI") when having 5+ identical elements within the project.

For more information see: [The Cobigen documentation](#) and the corresponding chapter in the [devonfw Guide](#) and

114.3.3. Angular 2

With the official release of Angular 2 and TypeScript 2, we're slowly but steadily moving to embrace these important new players in the web development scene. We keep supporting the Angular 1 based OASP4js framework and are planning a migration of this framework to Angular 2 in the near future. For "Balu" we've decided to integrate "vanilla" Angular 2.

We have migrated the Restaurant Sample application to serve as a, documented and supported, blueprint for Angular 2 applications. Furthermore, we support three "kickstarter" projects which help engagement getting started with Angular2 - either using Bootstrap or Google's Material Design - or, alternatively, Ionic 2 (the mobile framework on top of Angular 2). For more information see: [Angular 2 Kickstarter](#) and [Ionic 2 Kickstarter](#)

114.3.4. OASP4J 2.2.0 Release

A new release of OASP4J, version 2.2.0, is included in this release of devonfw. This release mainly focuses on server side of oasp. i.e oasp4j.

Major features added are :

- Upgrade to Spring Boot 1.3.8.RELEASE
- Upgrade to Apache CXF 3.1.8
- Database Integration with Oracle 11g
- Added Servlet for HTTP-Debugging
- Refactored code and improved JavaDoc
- Bugfix: mvn spring-boot:run executes successfully for oasp4j application created using oasp4j template
- Added subsystem tests of SalesmanagementRestService and several other tests
- Added Tests to test java packages conformance to OASP conventions

More details on features added can be found at <https://github.com/oasp/oasp4j/milestone/19?closed=1>(here). The OASP4j wiki and other documents are updated for release 2.2.0.

114.3.5. Devon4Sencha

Devon4Sencha is an alternative view layer for web applications developed with devonfw. It is based on Sencha Ext JS. As it requires a license for commercial applications it is not provided as Open Source and is considered to be part of the IP of Capgemini.

These libraries provide support for creating SPA (Single Page Applications) with a very rich set of

components for both desktop and mobile. In the new version we extend this functionality to support for "Universal Apps", the Sencha specific term for true multi-device applications which make it possible to develop a single application for desktop, tablet as well as mobile devices. In the latest version Devon4Sencha has been upgraded to support Ext JS 6.2 and we now support the usage of Cobigen as well as Sencha Architect as extra option to improve developer productivity. For more information about the update see: [The Sencha docs within the devonfw Guide](#)

114.3.6. Devcon enhancements

The Devon Console, Devcon, is a cross-platform command line tool running on the JVM that provides many automated tasks around the full life-cycle of Devon applications, from installing the basic working environment and generating a new project, to running a test server and deploying an application to production. It can be used by the engagements to integrate with their proprietary tool chain.

In this new release we have added an optional graphical user interface (with integrated help) which makes using Devcon even easier to use. Another new feature is that it is now possible to easily extend it with commands just by adding your own or project specific Javascript files. This makes it an attractive option for project task automation. You can find more information in the [Devcon Command Developers Guide](#)

114.3.7. Ready for the Cloud

devonfw is in active use in the Cloud, with projects running on IBM Bluemix and on Amazon AWS. The focus is very much to keep Cloud-specific functionality decoupled from the devonfw core. The engagement can choose between - and easily configure the use of - either CloudFoundry or Spring Cloud (alternatively, you can run devonfw in Docker containers in the Cloud as well. See elsewhere in the release notes). For more information about how to configure devonfw for use in the cloud see: [devonfw on Docker](#) and [devonfw in IBM Bluemix](#)

114.3.8. Spring Data

The java server stack within devonfw, OASP4J, is build on a very solid DDD architecture which uses JPA for its data access layer. We now offer integration of Spring Data as an alternative or to be used in conjunction with JPA. Spring Data offers significant advantages over JPA through its query mechanism which allows the developer to specify complex queries in an easy way. Overall working with Spring Data should be quite more productive compared with JPA for the average or junior developer. And extra advantage is that Spring Data also allows - and comes with support for - the usage of NoSQL databases like MongoDB, Cassandra, DynamoDB etc. THis becomes especially critical in the Cloud where NoSQL databases typically offer better scalability than relational databases. For more information see: [Integrating Spring Data in OASP4J](#)

114.3.9. Videos content in the devonfw Guide

The devonfw Guide is the single, authoritative tutorial and reference ("cookbook") for all things devonfw, targeted at the general developer working with the platform (there is another document for Architects). It is clear and concise but because of the large scope and wide reach of devonfw, it comes with a hefty 370+ pages. For the impatient - and sometimes images do indeed say more than

words - we've added 17 videos to the Guide which significantly speed up getting started with the diverse aspects of devonfw.

For more information see: [Video releases on TeamForge](#)

114.3.10. Containerisation with Docker and the Production Line

Docker (see: <https://www.docker.com/>) containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. Docker containers resemble virtual machines but are far more resource efficient. Because of this, Docker and related technologies like Kubernetes are taking the Enterprise and Cloud by storm. We have certified and documented the usage of devonfw on Docker so we can now firmly state that "devonfw is Docker" ready. All the more so as the iCSD Production Line is now supporting devonfw as well. The Production Line is a Docker based set of methods and tools that make possible to develop custom software to our customers on time and with the expected quality. By having first-class support for devonfw on the Production Line, iCSD has got an unified, integral solution which covers all the phases involved on the application development cycle from requirements to testing and hand-off to the client.

See: [devonfw on Docker](#) and [devonfw on the Production Line](#)

114.3.11. Eclipse Neon

devonfw comes with its own pre configured and enhanced Eclipse based IDE: the Open Source "OASP IDE" and "devonfw Distr" which falls under Capgemini IP. We've updated both versions to the latest stable version of Eclipse, Neon. From Balu onwards we support the IDE on Linux as well and we offer downloadable versions for both Windows and Linux.

See: [The Devon IDE](#)

114.3.12. Default Java 8 with Java 7 compatibility

From version 2.1. "Balu" onwards, devonfw is using by default Java 8 for both the tool-chain as well as the integrated development environments. However, both the framework as well as the IDE and tool-set remain fully backward compatible with Java 7. We have added documentation to help configuring aspects of the framework to use Java 7 or to upgrade existing projects to Java 8. See: [Compatibility guide for Java7, Java8 and Tomcat7, Tomcat8](#)

114.3.13. Full Linux support

In order to fully support the move towards the Cloud, from version 2.1. "Balu" onwards, devonfw is fully supported on Linux. Linux is the de-facto standard for most Cloud providers. We currently only offer first-class support for Ubuntu 16.04 LTS onward but most aspects of devonfw should run without problems on other and older distributions as well.

114.3.14. Initial ATOM support

Atom is a text editor that's modern, approachable, yet hackable to the core - a tool you can customize to do anything but also use productively without ever touching a config file. It is turning

into a standard for modern web development. In devonfw 2.1 "Balu" we provide a script which installs automatically the most recent version of Atom in the devonfw distribution with a pre-configured set of essential plugins. See: [OASP/devonfw Atom editor \("IDE"\) settings & packages](#)

114.3.15. Database support

Through JPA (and now Spring Data as well) devonfw supports many databases. In Balu we've extended this support to prepared configuration, extensive documentations and supporting examples for all major "Enterprise" DB servers. So it becomes even easier for engagements to start using these standard database options. Currently we provide this extended support for Oracle, Microsoft SQL Server, MySQL and PostgreSQL. For more information see: [OASP Database Migration Guide](#)

114.3.16. File upload and download

File up and download was supported in previous version of the framework, but as these operations are common but complex, we've extended the base functionality and improved the available documentation so it becomes substantially easier to offer both File up- as well as download in devonfw based applications. See: [devonfw Guide Cookbook: File Upload and Download](#)

114.3.17. Internationalisation (I18N) improvements

Likewise, existing basic Internationalisation (I18N) support has been significantly enhanced through an new devonfw module and extended to support Ext JS and Angular 2 apps as well. This means that both server as well as client side applications can be made easily to support multiple languages ("locales"), using industry standard tools and without touching programming code (essential when working with teams of translators). For more information see: [The I18N \(Internationalization\) module](#) and [Client GUI Sencha i18n](#)

114.3.18. Asynchronous HTTP support

Asynchronous HTTP is an important feature allowing so-called "long polling" HTTP Requests (for streaming applications, for example) or with requests sending large amounts of data. By making HTTP Requests asynchronous, devonfw server instances can better support these types of use-cases while offering far better performance. Documentation about how to include the new devonfw module implementing this feature can be found at: [The devonfw async module](#)

114.3.19. Security and License guarantees

In devonfw security comes first. The components of the framework are designed and implemented according to the recommendations and guidelines as specified by OWASP in order to confront the top 10 security vulnerabilities.

From version 2.1 "Balu" onward we certify that devonfw has been scanned by software from "Black Duck". This verifies that devonfw is based on 100% Open Source Software (non Copyleft) and demonstrates that at moment of release there are no known, critical security flaws. Less critical issues are clearly documented.

114.3.20. Documentation improvements

Apart from the previously mentioned additions and improvements to diverse aspects of the devonfw documentation, principally the devonfw Guide, there are a number of other important changes. We've incorporated the Devon Modules Developer's Guide which describes how to extend devonfw with its Spring-based module system. Furthermore we've significantly improved the Guide to the usage of web services. We've included a Compatibility Guide which details a series of considerations related with different version of the framework as well as Java 7 vs 8. And finally, we've extended the F.A.Q. to provide the users with direct answers to common, Frequently Asked Questions.

114.3.21. Contributors

Many thanks to adrianbielewicz, aferre777, amarinso, arenstedt, azzigeorge, cbeldacap, cmammado, crisjdiaz, csiwiak, Dalgar, drhoet, Drophoff, dumbNickname, EastWindShak, fawinter, fbougeno, fkreis, GawandeKunal, henning-cg, hennk, hohwille, ivanderk, jarek-jpa, jart, jensbartelheimer, jhcore, jkokoszk, julianmetzler, kalmuczakm, kirancvadla, kowalj, lgoerlach, ManjiriBirajdar, MarcoRose, maybeec, mmatczak, nelooo, oelsabba, pablo-parra, patrhel, pawelkorzeniowski, PriyankaBelorkar, RobertoGM, sekaiser, sesslinger, SimonHuber, sjimenez77, sobkowiak, sroeger, ssarmokadam, subashbasnet, szendo, tbialecki, thoptr, tsowada, znazir and anyone who we may have forgotten to add!