



devonfw guide

Table of Contents

I: Getting Started	1
1. devonfw Introduction	2
1.1. Building Blocks of the Platform	2
1.2. devonfw Technology Stack	2
1.3. Custom Tools	3
1.4. devonfw Modules	4
2. Why should I use devonfw?	6
2.1. Objectives	6
2.2. Features	7
3. Download and Setup	8
3.1. Prerequisites	8
3.2. Download	8
3.3. Setup the workspace	8
4. The Devon IDE	14
4.1. Introduction	14
4.2. Cobigen	14
4.3. IDE Plugins:	15
II: Devcon	24
5. Devcon Command Developer's guide	25
5.1. Introduction	25
5.2. Creating your own Devcon modules	26
5.3. Conclusion	41
6. Devcon Command Reference	42
6.1. dist	42
6.2. doc	46
6.3. github	46
6.4. help	47
6.5. devon4j	48
6.6. devon4ng	53
6.7. project	56
6.8. sencha	62
6.9. workspace	65
6.10. system	66
III: devon4j	68
7. Introduction	69
8. Architecture	70
8.1. Key Principles	70
8.2. Architecture Principles	70

8.3. Application Architecture	71
8.4. Components	74
9. Coding	76
9.1. Coding Conventions	76
9.2. Tools	85
10. Layers	86
10.1. Client Layer	86
10.2. Service Layer	87
10.3. Logic Layer	89
10.4. Component Facade	90
10.5. UseCase	92
10.6. Data-Access Layer	96
10.7. Batch Layer	96
11. Guides	121
11.1. Dependency Injection	121
11.2. Configuration	123
11.3. Java Persistence API	129
11.4. Auditing	151
11.5. Transaction Handling	153
11.6. SQL	153
11.7. Database Migration	156
11.8. SAP HANA	158
11.9. Oracle RDBMS	159
11.10. JEE	162
11.11. Logging	164
11.12. Security	168
11.13. Access-Control	170
11.14. Data-permissions	178
11.15. Validation	183
11.16. Aspect Oriented Programming (AOP)	186
11.17. Exception Handling	188
11.18. Internationalization	192
11.19. XML	193
11.20. JSON	194
11.21. REST	197
11.22. SOAP	206
11.23. Service Client	208
11.24. Testing	213
11.25. Transfer-Objects	226
11.26. Bean-Mapping	228
11.27. Datatypes	229

11.28. Accessibility	231
11.29. Caching	231
11.30. CORS support	232
11.31. BLOB support	233
11.32. Java Development Kit	236
11.33. Application Performance Management	239
12. devonfw Development	241
12.1. IDE Setup	241
12.2. Issue creation and resolution	243
12.3. Code contribution	244
12.4. devonfw Documentation	246
13. Tutorials	247
13.1. Introduction	247
13.2. Creating a new application	247
14. For Core-Developers	254
14.1. Creating a Release	254
IV: devon4ng	261
15. Architecture	262
15.1. Overview	262
16. Meta Architecture	264
16.1. Introduction	264
16.2. devonfw Reference Client Architecture	265
16.3. Appendix	270
16.4. References	270
17. Layers	272
17.1. Components Layer	272
17.2. Services Layer	275
18. Guides	280
18.1. Package Managers	280
19. Angular	293
19.1. Accessibility	293
19.2. Lazy loading	312
19.3. An example with Angular	312
19.4. Conclusion	322
19.5. Angular Material Theming	327
19.6. Theming basics	327
19.7. Prebuilt themes	328
19.8. Custom themes	330
19.9. Useful resources	337
19.10. Angular Progressive Web App	337
19.11. Assumptions	337

19.12. Sample Application	338
19.13. APP_INITIALIZER	346
19.14. Component Decomposition	356
19.15. Consuming REST services	361
19.16. File Structure	368
19.17. Internationalization	371
19.18. Routing	375
19.19. Testing	382
19.20. Working with Angular CLI	391
20. Ionic	396
20.1. Ionic: Getting started	396
20.2. Why Ionic?	396
20.3. Basic environment set up	396
20.4. Basic project set up	397
20.5. Ionic: From code to android	398
20.6. Assumptions	398
20.7. From ionic 4 to Android project	398
20.8. From Android project to emulated device	400
20.9. From Android project to real device	402
20.10. Result	405
20.11. Ionic Progressive Web App	405
20.12. Assumptions	406
20.13. Sample Application	406
21. Layouts	412
21.1. Angular Material Layout	412
21.2. Let's begin	412
21.3. Adding Angular Material library to the project	413
21.4. Development	415
21.5. Conclusion	424
22. NgRx	425
22.1. NgRx	425
22.2. Creating a Simple Store	427
23. Cookbook	442
23.1. Abstract Class Store	442
V: devon4net	455
24. Architecture basics	456
25. Coding conventions	465
Camel Case (camelCase)	465
Pascal Case (PascalCase)	465
Underscore Prefix (_underScore)	465
26. Environment	469

27. User guide	471
27.1. OASP4NET Guide	471
28. Packages	493
28.1. Packages overview	493
28.2. The packages.....	493
28.3. Required software.....	510
29. Templates	511
29.1. Templates	511
30. Samples	512
30.1. Samples	512
VI: devonfw shop floor	522
31. What is devonfw shop floor?	523
32. How to use it	524
32.1. Prerequisites - Provisioning environment	524
32.2. Step 1 - Configuration and services integration	524
32.3. Step 2 - Create the project	524
32.4. Step 3 - Deployment	525
32.5. Step 4 - Monitoring	525
33. Provisioning environments	526
33.1. Production Line provisioning environment	526
33.2. dsf4docker provisioning environment.....	526
34. Configuration and services integration	529
34.1. Nexus Configuration	529
34.2. SonarQube Configuration	534
35. Create project	538
35.1. Create and integrate git repository	538
35.2. start new devonfw project.....	538
35.3. cicd configuration	539
36. Deployment environments	552
36.1. OpenShift	552
37. Monitoring	568
37.1. Build monitor view	568
38. Annexes	571
38.1. BitBucket	571
38.2. Basic Selenium Grid setup in OpenShift	583
38.3. Mirabaud CICD Environment Setup	590
38.4. OKD (<i>OpenShift Origin</i>)	618
VII: cicdgen	633
39. CICDGEN	634
39.1. What is angular schematics?	634
39.2. cicdgen CLI	634

39.3. cicdgen Schematics	634
39.4. Usage example	634
40. cicdgen CLI	635
40.1. CICDGEN CLI	635
40.2. cicdgen usage example	636
41. cicdgen Schematics	648
41.1. CICDGEN SCHEMATICS	648
41.2. Devon4j schematic	648
41.3. Devon4ng schematic	652
41.4. Devon4node schematic	656
VIII: CobiGen — Code-based incremental Generator	660
42. Document Description	661
43. Guide to the Reader	662
44. CobiGen - Code-based incremental Generator	663
44.1. Overview	663
44.2. Architecture	663
44.3. Features and Characteristics	663
44.4. Selection of current and past CobiGen applications	664
45. General use cases	665
45.1. devon4j	665
45.2. Register Factory	668
46. CobiGen	669
46.1. Configuration	669
46.2. Plug-ins	679
47. Maven Build Integration	697
47.1. Maven Build Integration	697
48. Eclipse Integration	702
48.1. Installation	702
48.2. Usage	704
48.3. Logging	711
49. Template Development	712
49.1. Helpful links for template development	712
IX: devonfw testing	713
50. Home	714
50.1. Contact	714
50.2. What is E2E Mr Checker Test Framework	714
50.3. Where Mr Checker applies?	714
50.4. Benefits for the project	716
50.5. Road map plan	716
50.6. Wiki Structure	717
51. How to install	718

51.1. Easy out of the box installation	719
51.2. Out of the box installation - with additional steps	720
51.3. Advanced manual step by step installation	720
52. Mr Checker Test Framework modules	724
52.1. Core Test Module	724
52.2. Selenium Test Module	725
52.3. Security Test Module	727
52.4. Mobile Test Module	728
52.5. Standalone Test Module	728
52.6. DevOps Module	728
X: MyThaiStar	751
53. 1.My Thai Star – Agile Framework	752
53.1. 1.1 Team Setup	752
53.2. 1.2 Scrum events	752
53.3. 1.3 Establish Product Backlog	753
54. 2.My Thai Star – Agile Diary	755
54.1. 24.03.2017 Sprint 1 Planning	755
54.2. 27.04.2017 Sprint 1 Review	755
54.3. 03.05.2017 Sprint 2 Planning	755
54.4. 01.06.2017 Sprint 2 Review	756
55. Technical design	757
55.1. Data Model	757
55.2. Server Side	757
55.3. Client Side	785
56. Security	793
56.1. Two-Factor Authentication	793
57. Testing	797
57.1. Server Side	797
57.2. Client Side	799
57.3. End to end	804
58. UI design	806
58.1. Style guide	806
58.2. Low and high fidelity wireframes	806
59. CI/CD	807
59.1. My Thai Star in Production Line	807
59.2. Deployment	821
59.3. Container Schema	822
59.4. Run My Thai Star	822
XI: Contributing Guide	832
60. Code Contributions	833
60.1. Notes on Code Contributions	833

60.2. Introduction to Git and GitHub	833
60.3. Structure of our projects	833
60.4. Contributing to our projects	834
60.5. Reviewing Pull Requests	836
61. Contributor Covenant Code of Conduct	838
61.1. Our Pledge	838
61.2. Our Standards	838
61.3. Our Responsibilities	838
61.4. Scope	838
61.5. Enforcement	839
61.6. Attribution	839
62. Development Guidelines	840
63. Working with forked repositories	841
63.1. Fork a repository	841
63.2. Configuring a remote for a fork	841
63.3. Syncing a fork	842
64. Wiki Contributions	844
64.1. Text styles	844
64.2. Titles	844
64.3. Lists	845
64.4. Tables	845
64.5. Source Code	845
XII: Release Notes	847
65. devonfw Release notes 3.1 "Goku"	848
65.1. Introduction	848
65.2. Changes and new features	849
66. devonfw Release notes 3.0 "Fry"	856
66.1. Introduction	856
66.2. Changes and new features	857
67. devonfw Release notes 2.4 "EVE"	865
67.1. Introduction	865
67.2. Changes and new features	865
68. devonfw Release notes 2.3 "Dash"	872
68.1. Release: improving & strengthening the Platform	872
68.2. An industrialized platform for the ADcenter	872
68.3. Changes and new features	873
69. devonfw Release notes 2.2 "Courage"	879
69.1. Production Line Integration	879
69.2. OASP4js 2.0	879
69.3. A new OASP Portal	879
69.4. New Cobigen	879

69.5. MyThaiStar: New Restaurant Example, reference implementation & Methodology showcase	880
69.6. The new OASP Tutorial	880
69.7. OASP4j 2.4.0	881
69.8. Microservices Netflix	881
69.9. devonfw distribution based on Eclipse OOMPH	882
69.10. Visual Studio Code / Atom	882
69.11. More I18N options	882
69.12. Spring Integration as devonfw Module	882
69.13. devonfw Harvest contributions	882
69.14. More Deployment options to JEE Application Servers and Docker/CloudFoundry	883
69.15. Devcon on Linux	883
69.16. New OASP Incubators	883
70. Release notes devonfw 2.1.1 "Balu"	884
70.1. Version 2.1.2: OASP4J updates & some new features	884
70.2. Version 2.1.1 Updates, fixes & some new features	885
70.3. Version 2.1 New features, improvements and updates	886

Part I: Getting Started

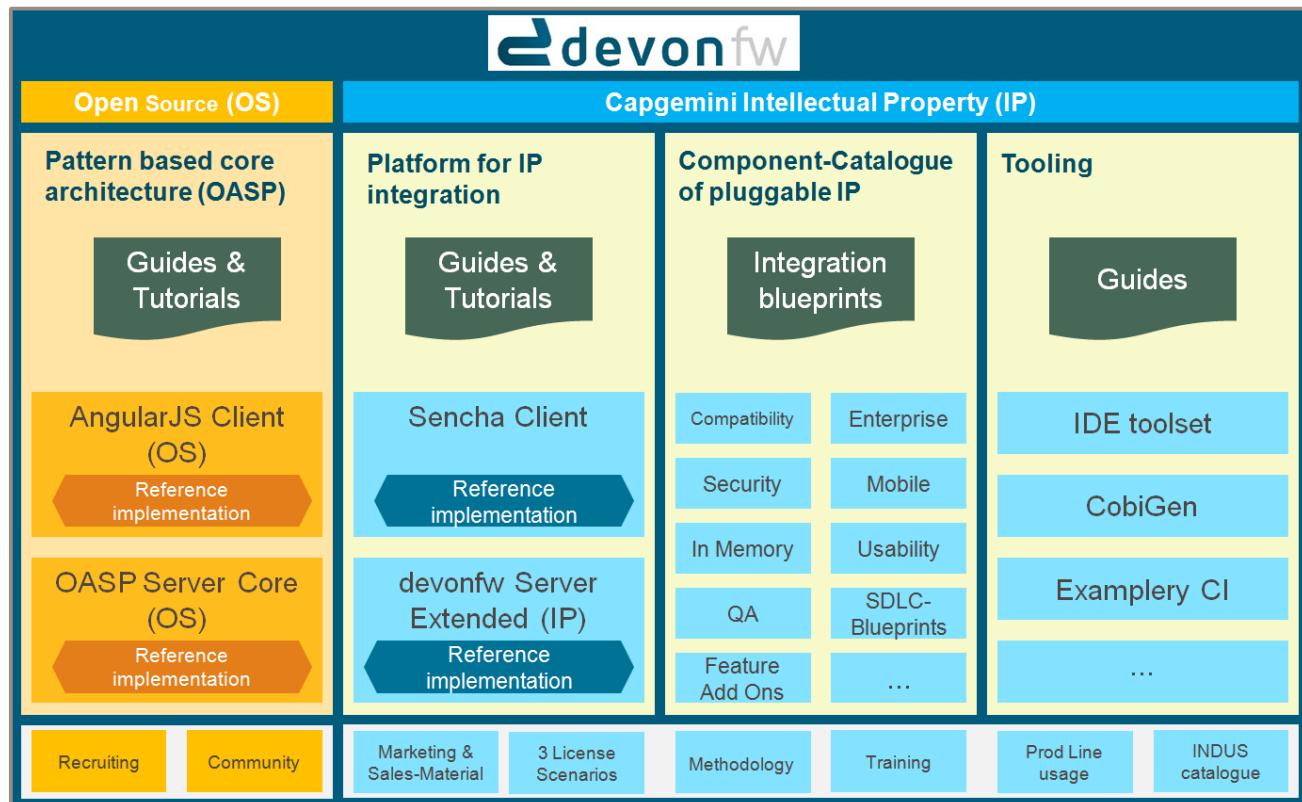
1. devonfw Introduction



Welcome to **devonfw**, the Devon Framework. This is a product of the CSD industrialization effort to bring a standardized platform for custom software development within Capgemini APPS2. This platform is aimed at engagements where clients do not force the use of a determined technology so we can offer a better alternative coming from our experience as a group.

Devon framework is a development platform aiming for standardization of processes and productivity boost, that provides an architecture blueprint for Java/JavaScript applications, alongside a set of tools to provide a fully functional *out-of-the-box* development environment.

1.1. Building Blocks of the Platform



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

1.2. devonfw Technology Stack

devonfw is composed of an Open Source part that can be freely used by other people and proprietary addons which are Capgemini IP and can be used only in Capgemini engagements. The Open Source part of devonfw is called *The Open Application Standard Platform (OASP)*. It consists of

1.2.1. Back-end solutions

- [devon4j](#): server implemented with Java. The OASP platform provides an implementation for Java based on [Spring](#) and [Spring Boot](#).
- [OASP4FN](#): serverless implementation based on [node.js](#).
- *Dot Net* implementation. (Upcoming)

1.2.2. Front-end solutions

For client applications, *devonfw* includes two possible solutions based on *JavaScript*:

- [OASP4JS](#): the *OASP* implementation based on [Angular](#) framework.
- [devon4sencha](#): a client solution based on the [Sencha](#) framework.

Check out the links for more details.

1.3. Custom Tools

1.3.1. Pre-installed Software

- *Eclipse*: pre-configured and fully functional IDE to develop Java based apps.
- *Java*: all the Java environment configured and ready to be used within the distribution.
- *Maven*: to manage project dependencies.
- *Node*: a Node js environment configured and ready to be used within the distribution.
- *Sencha*: *devonfw* also includes a installation of the *Sencha Cmd* tool.
- *Sonarqube*: a code quality tool.
- *Tomcat*: a web server ready to test the deploy of our artifacts.

1.3.2. Devcon

For project management and other life-cycle related tasks, *devonfw* provides also [Devcon](#), a command line and graphic user interface cross platform tool.

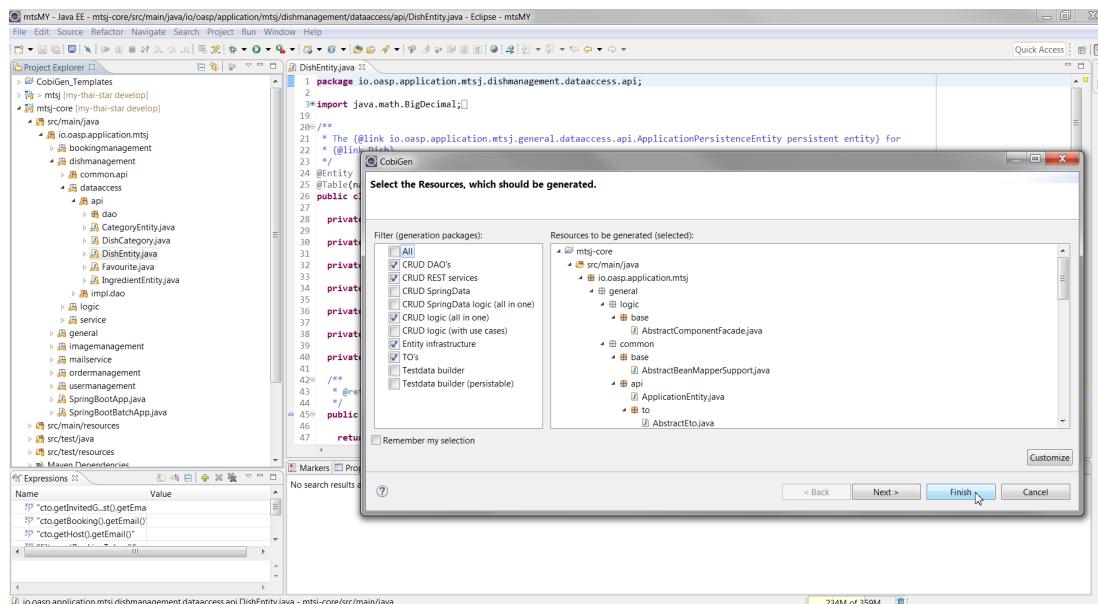
With *Devcon*, users can automate the creation of new projects (both server and client), build and run those and even, for server projects, deploy locally on Tomcat.



All those tasks can be done manually using *Maven*, *Tomcat*, *Sencha Cmd*, *Bower*, *Gulp*, etc. but with *Devcon* users have the possibility of managing the projects without the necessity of dealing with all those different tools.

1.3.3. Cobigen

Cobigen is a code generator included in the context of *devonfw* that allows users to generate all the structure and code of the components, helping to save a lot of time wasted in repetitive tasks.



1.4. devonfw Modules

As a part of the goal of productivity boosting, *devonfw* also provides a set of *modules* to the developers, created from real projects requirements, that can be connected to projects for saving all the work of a new implementation.

The current available modules are:

- *async*: module to manage asynchronous web calls in a *Spring* based server app.
- *i18n*: module for internationalization.
- *integration*: implementation of *Spring Integration*.
- *microservices*: a set of archetypes to create a complete microservices infrastructure based on *Spring Cloud Netflix*.
- *reporting*: a module to create reports based on *Jasper Reports* library.
- *winauth active directory*: a module to authenticate users against an *Active Directory*.
- *winauth single sign on*: module that allows applications to authenticate the users by the Windows credentials.

Find more about devonfw [here](#).

2. Why should I use devonfw?

Devonfw aims at providing a framework which is oriented at development of web applications based on the Java EE programming model using the Spring framework project as the default implementation.

2.1. Objectives

2.1.1. Standardization

It means that to stop reinventing the Wheel in thousands of projects, hundreds of centers, dozens of countries. This also includes rationalize, harmonize and standardize all development assets all over the group and industrialize the software development process

2.1.2. Industrialization of Innovative technologies & “Digital”

devonfw needs to standardize & industrialize. But not just large volume, “traditional” custom software development. devonfw needs to offer a standardized platform which contains a range of state of the art methodologies and technology options. devonfw needs to support agile development by small teams utilizing the latest technologies for Mobile, IoT and the Cloud

2.1.3. Deliver & Improve Business Value



2.1.4. 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 juniors.

2.1.5. Quality

- State of the Art architecture and design.
- Lower cost on maintenance and warranty.

- Technical debt reduction by reuse.
- Risk reduction due to assets continuous improvement.
- Standardized automated quality checks.

2.1.6. Agility

- Focus on business functionality not on technical.
- Shorter release cycles.
- DevOps by design - Infrastructure as Code.
- Continuous Delivery Pipeline.
- On and Off-premise flexibility.
- PoCs and Prototypes in days not months.

2.2. Features

2.2.1. 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*.

2.2.2. devonfw, the package

devonfw package provides:

- Implementation blueprints for a modern cloud-ready server and a choice on JS-Client technologies (either open source AngularJs 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.
- Example application as a reference implementation.
- Support through large community + industrialization services (Standard Platform as a service) available in the iProd service catalog.

To read in details about devonfw features read [here](#)

3. Download and Setup

In this section, you will learn how to setup the devonfw environment and start working on first project based on devonfw.

The devonfw environment contains all software and tools necessary to develop the applications with devonfw.

3.1. Prerequisites

In order to setup the environment, following are the prerequisites:

- Internet connection (including details of your proxy configuration, if necessary)
- 2GB of free disk space
- The ZIP containing the latest devonfw distribution

3.2. Download

The devonfw distributions can be obtained from the [TeamForge releases library](#) and are packaged in a [zip](#) file that includes all the needed tools, software and configurations



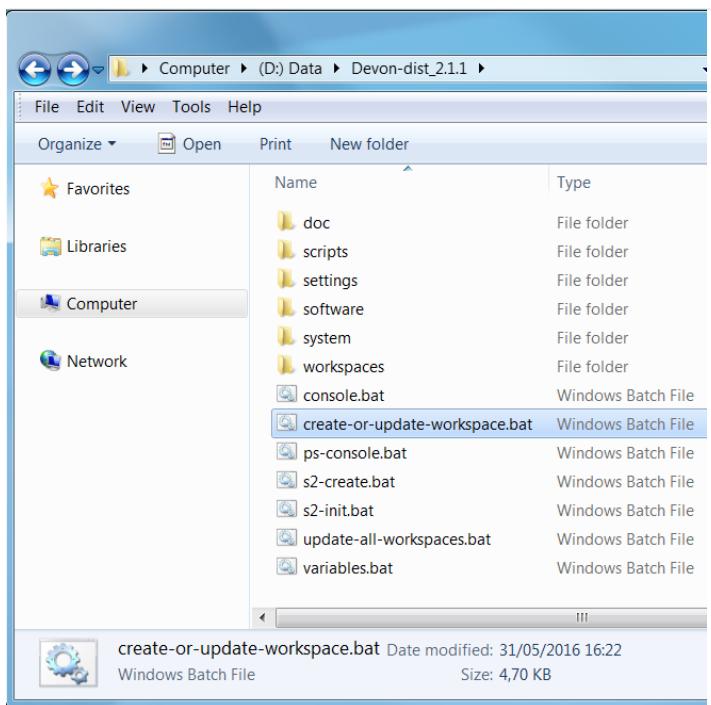
Using devcon



You can do it using devcon with the command `devon dist install`, learn more [here](#). After a successful installation, you can initialize it with the command `devon dist init`, learn more [here](#).

3.3. Setup the workspace

1. Unzip the devonfw distribution into a directory of your choice. **The path to the devonfw distribution directory should contain no spaces**, to prevent problems with some of the tools.
2. Run the batch file "create-or-update-workspace.bat".



This will configure the included tools like Eclipse with the default settings of the devonfw distribution.

The result should be as seen below

```

C:\WINDOWS\system32\cmd.exe
IDE environment has been initialized.
Copied workspaces\main\development\settings\maven\settings.xml to conf\.m2\settings.xml
jun 23, 2015 5:55:37 PM io.oasp.ide.eclipse.configurator.core.Configurator logConfig
INFO: io.oasp.ide.eclipse.configurator.core.Configurator -u
jun 23, 2015 5:55:37 PM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Updating workspace
jun 23, 2015 5:55:37 PM io.oasp.ide.eclipse.configurator.core.Configurator collectWorkspaceFiles
INFO: Collected 54 configuration files.
jun 23, 2015 5:55:37 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 . . .

```

The working devonfw environment is ready!!!

Note : If you use a proxy to connect to the Internet, you have to manually configure it in Maven, Sencha Cmd and Eclipse. Next section explains about it.

3.3.1. Manual Tool Configuration

Maven

Open the file "conf/.m2/settings.xml" in an editor

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- $!ds -->
<settings>
  <!-- If you connect to the internet via a proxy, uncomment the following section and fill out
       host and port values. Delete username and password entries, if your proxy does not require
       authentication. -->
  <!-->
  <proxies>
    <proxy>
      <id>localhttp</id>
      <active>true</active>
      <protocol>http</protocol>
      <host>1.0.5.10</host>
      <port>8080</port>
      <username>capgemini</username>
      <password>capgemini</password>
    </proxy>
    <proxy>
      <id>localhttps</id>
      <active>true</active>
      <protocol>https</protocol>
      <host>1.0.5.10</host>
      <port>8080</port>
      <username>capgemini</username>
      <password>capgemini</password>
    </proxy>
  </proxies>
</-->

```

This screenshot shows the settings.xml file in Notepad++. The XML code defines two proxy configurations: 'localhttp' and 'localhttps'. Both proxies are set to active, use the http/https protocols, and point to host 1.0.5.10 on port 8080. They both have 'capgemini' as the username and password. The XML uses CDATA sections for the proxy definitions.

Remove the comment tags around the <proxy> section at the beginning of the file.

Then update the settings to match your proxy configuration.

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- $!ds -->
<settings>
  <!-- If you connect to the internet via a proxy, uncomment the following section and fill out
       host and port values. Delete username and password entries, if your proxy does not require
       authentication. -->
  <proxies>
    <proxy>
      <id>localhttp</id>
      <active>true</active>
      <protocol>http</protocol>
      <host>1.0.5.10</host>
      <port>8080</port>
      <username>capgemini</username>
      <password>capgemini</password>
    </proxy>
    <proxy>
      <id>localhttps</id>
      <active>true</active>
      <protocol>https</protocol>
      <host>1.0.5.10</host>
      <port>8080</port>
      <username>capgemini</username>
      <password>capgemini</password>
    </proxy>
  </proxies>
<!-- The "localRepository" has to be set to ensure consistent behaviour across command-line and Eclipse. -->

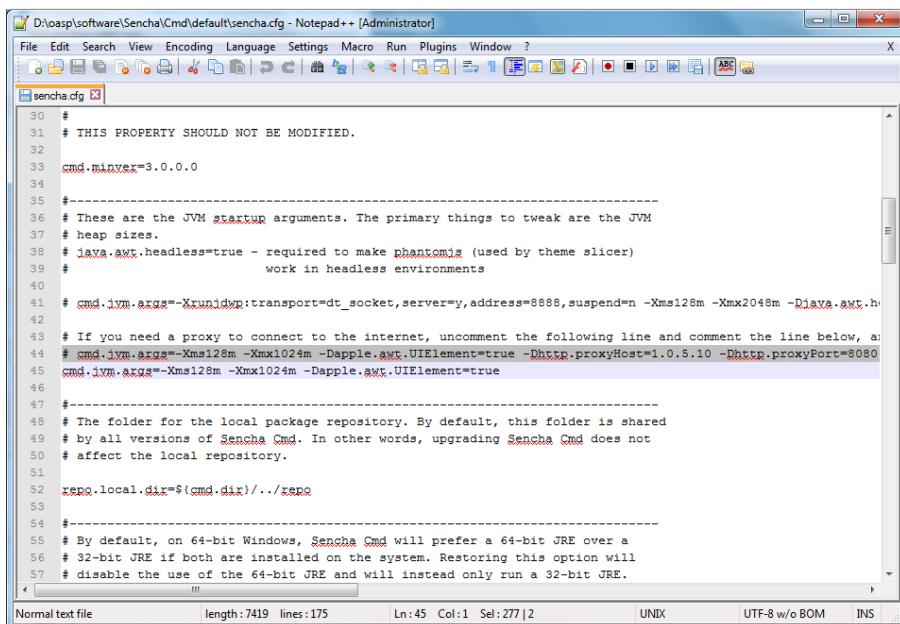
```

This screenshot shows the same settings.xml file after the proxy configuration section has been moved to the top of the file. The XML now starts with the proxy definitions, followed by the original settings section, and ends with a note about the local repository.

If your proxy does not require authentication, simply remove the <username> and <password> lines.

Sencha Cmd

Open the file software/Sencha/Cmd/default/sencha.cfg in an editor



```

D:\oasp\software\Sencha\Cmd\default\sencha.cfg - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
sencha.cfg
30 #
31 # THIS PROPERTY SHOULD NOT BE MODIFIED.
32
33 cmd.minver=3.0.0.0
34
35 #
36 # These are the JVM startup arguments. The primary things to tweak are the JVM
37 # heap sizes.
38 # java.awt.headless=true - required to make phantomjs (used by theme slicer)
39 # work in headless environments
40
41 # cmd.jvm.args=-Xrunjdwp:transport=dt_socket,server=y,address=8888,suspend=n -Xms128m -Xmx2048m -Djava.awt.h
42
43 # If you need a proxy to connect to the internet, uncomment the following line and comment the line below, a
44 # cmd.jvm.args=-Xms128m -Xmx1024m -Dapple.awt.UIElement=true -Dhttp.proxyHost=1.0.5.10 -Dhttp.proxyPort=8080
45 cmd.jvm.args=-Xms128m -Xmx1024m -Dapple.awt.UIElement=true
46
47 #
48 # The folder for the local package repository. By default, this folder is shared
49 # by all versions of Sencha Cmd. In other words, upgrading Sencha Cmd does not
50 # affect the local repository.
51
52 repo.local.dir=$(cmd.dir)/../repo
53
54 #
55 # By default, on 64-bit Windows, Sencha Cmd will prefer a 64-bit JRE over a
56 # 32-bit JRE if both are installed on the system. Restoring this option will
57 # disable the use of the 64-bit JRE and will instead only run a 32-bit JRE.

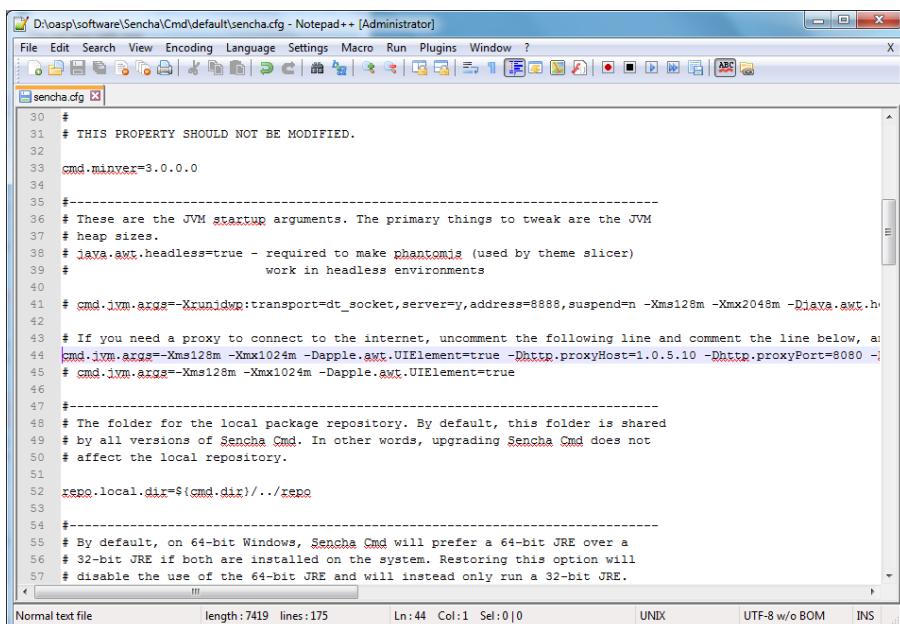
```

Normal text file length: 7419 lines: 175 Ln:45 Col:1 Sel:277|2 UNIX UTF-8 w/o BOM INS

Search for the property definition of "cmd.jvm.args" (around line 45).

Comment the existing property definition and uncomment the line above it.

Then update the settings to match your proxy configuration.



```

D:\oasp\software\Sencha\Cmd\default\sencha.cfg - Notepad++ [Administrator]
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
sencha.cfg
30 #
31 # THIS PROPERTY SHOULD NOT BE MODIFIED.
32
33 cmd.minver=3.0.0.0
34
35 #
36 # These are the JVM startup arguments. The primary things to tweak are the JVM
37 # heap sizes.
38 # java.awt.headless=true - required to make phantomjs (used by theme slicer)
39 # work in headless environments
40
41 # cmd.jvm.args=-Xrunjdwp:transport=dt_socket,server=y,address=8888,suspend=n -Xms128m -Xmx2048m -Djava.awt.h
42
43 # If you need a proxy to connect to the internet, uncomment the following line and comment the line below, a
44 # cmd.jvm.args=-Xms128m -Xmx1024m -Dapple.awt.UIElement=true -Dhttp.proxyHost=1.0.5.10 -Dhttp.proxyPort=8080 -
45 # cmd.jvm.args=-Xms128m -Xmx1024m -Dapple.awt.UIElement=true
46
47 #
48 # The folder for the local package repository. By default, this folder is shared
49 # by all versions of Sencha Cmd. In other words, upgrading Sencha Cmd does not
50 # affect the local repository.
51
52 repo.local.dir=$(cmd.dir)/../repo
53
54 #
55 # By default, on 64-bit Windows, Sencha Cmd will prefer a 64-bit JRE over a
56 # 32-bit JRE if both are installed on the system. Restoring this option will
57 # disable the use of the 64-bit JRE and will instead only run a 32-bit JRE.

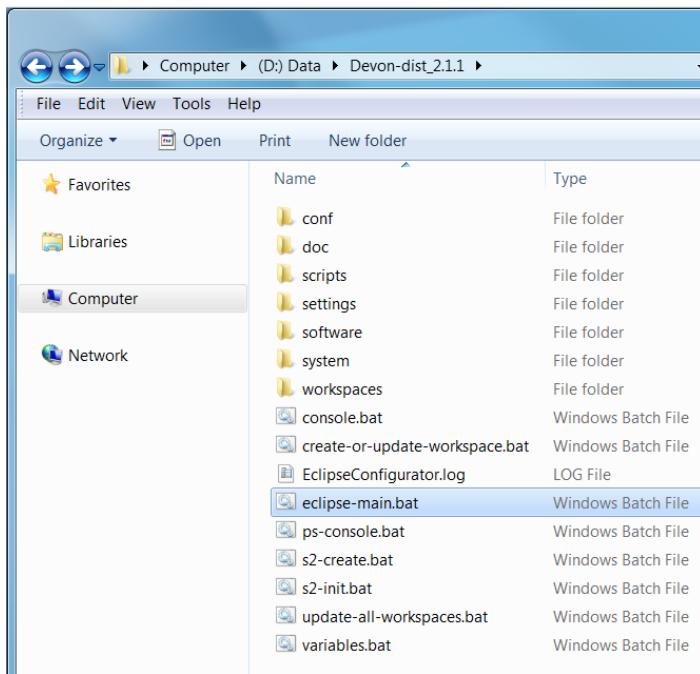
```

Normal text file length: 7419 lines: 175 Ln:44 Col:1 Sel:0|0 UNIX UTF-8 w/o BOM INS

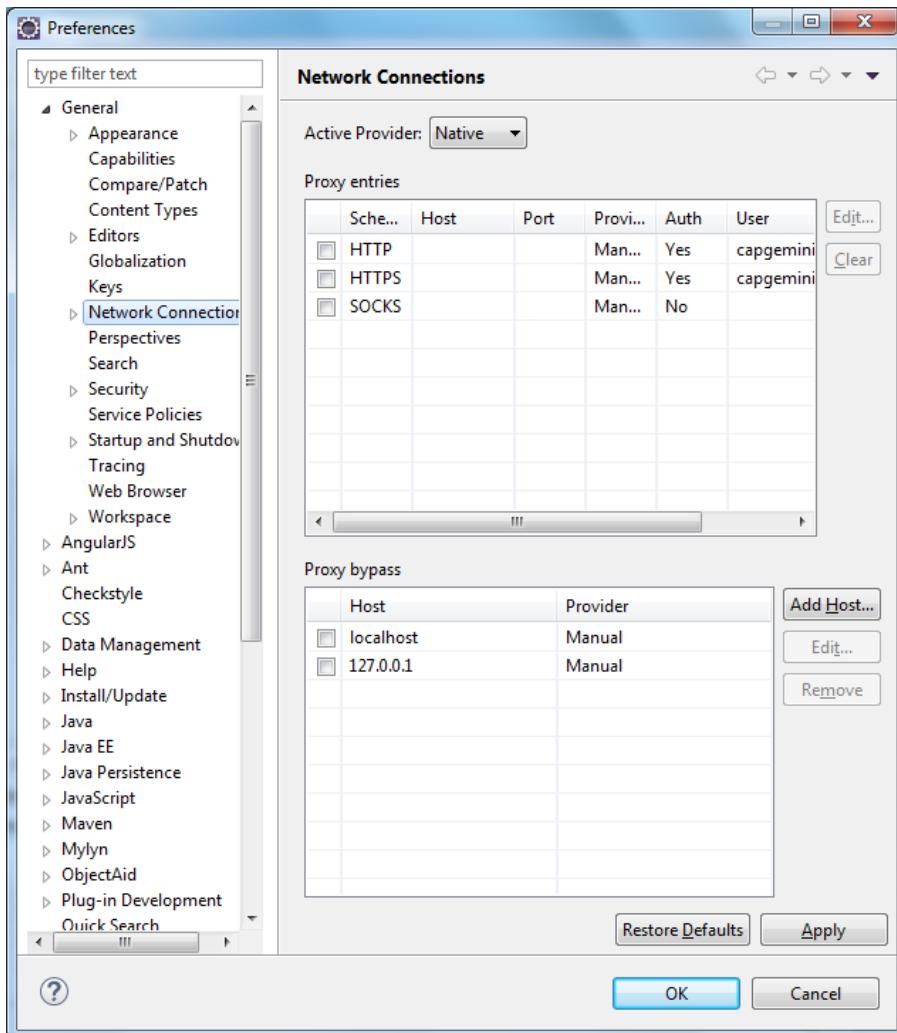
If your proxy does not require authentication, simply remove the "-Dhttp.proxyUser", "-DhttpProxyPassword", "-Dhttps.proxyUser" and "-Dhttps.proxyPassword" parameters.

Eclipse

Open eclipse by executing "eclipse-main.bat".

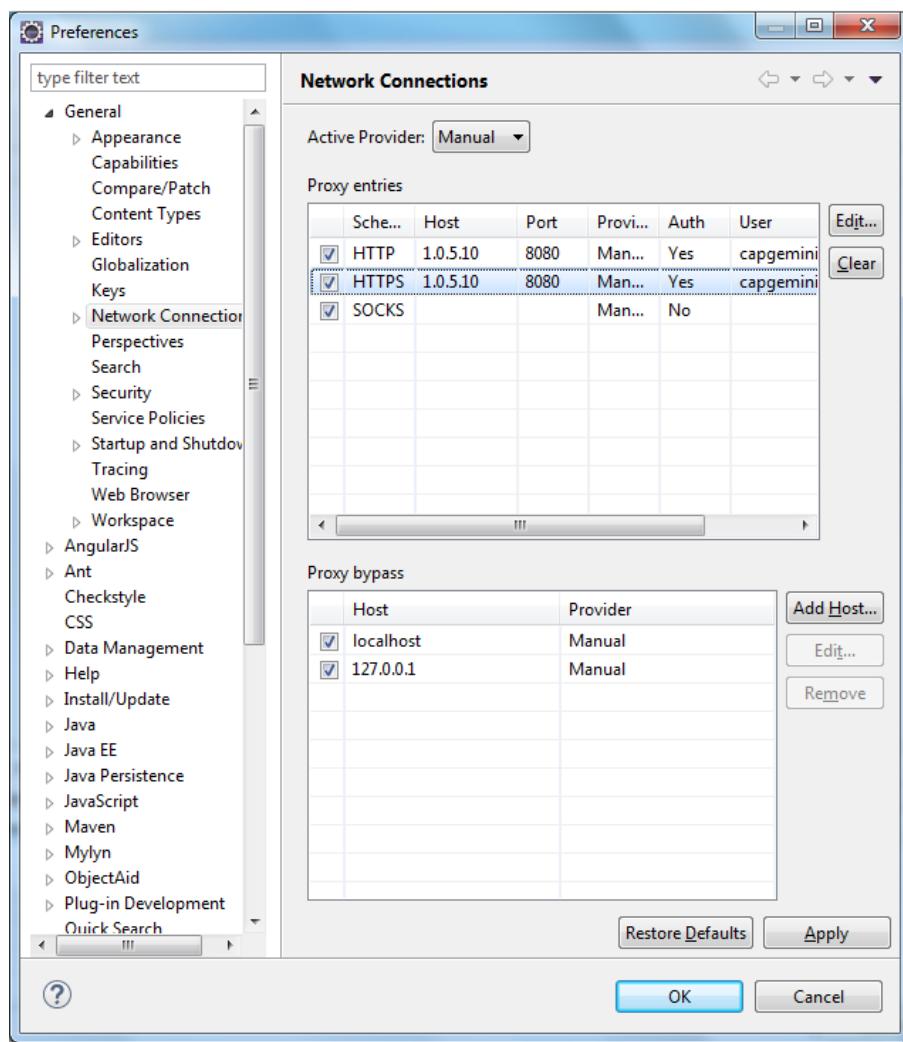


In the Eclipse preferences dialog, go to "General - Network Connection".



Switch from "Native" to "Manual"

Enter your proxy configuration



Thats All!!!

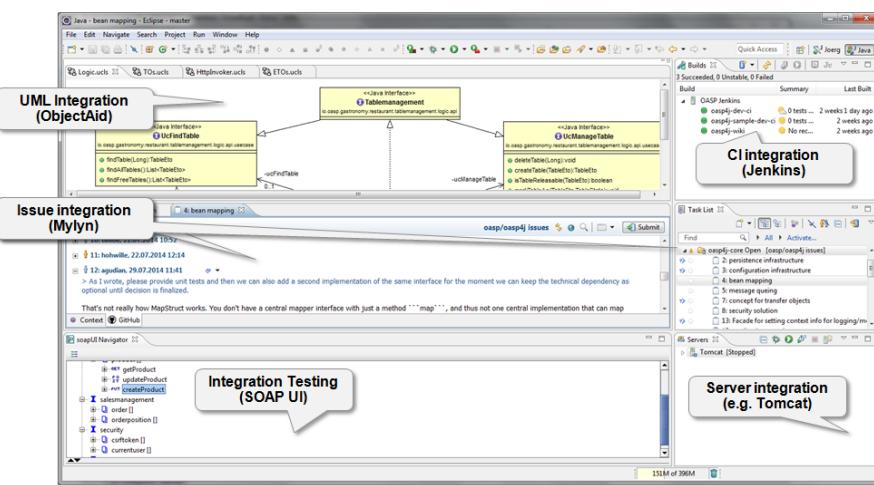
4. The Devon IDE

4.1. Introduction

The Devon IDE is the general name for two distinct versions of a customized Eclipse which comes in a Open Source variant, called devon-ide, and a more extended version included in the "Devon Dist" which is only available for Capgemini engagements.

4.1.1. Features and advantages

devonfw comes with a fully featured IDE in order to simplify the installation, configuration and maintenance of this instrumental part of the development environment. As it is being included in the distribution, the IDE is ready to be used and some specific configuration of certain plugins only takes a few minutes.



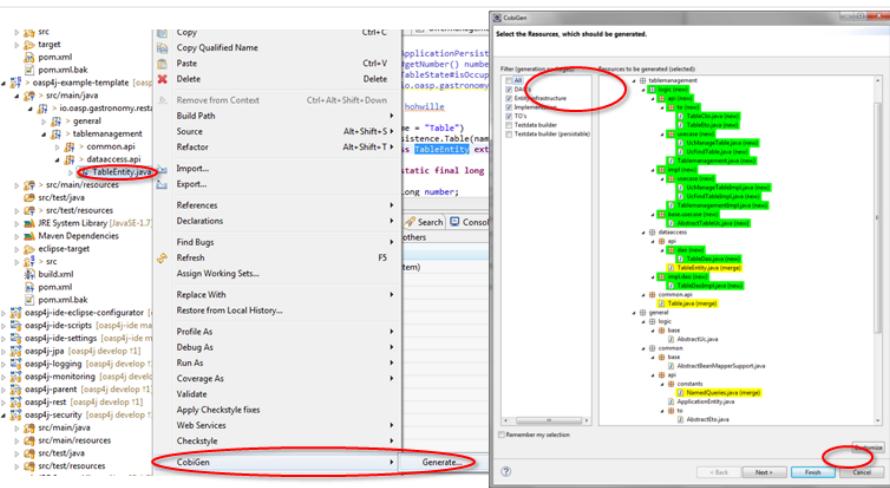
As with the remainder of the distribution, the advantage of this approach is that you can have as many instances of the -ide "installed" on your machine for different projects with different tools, tool versions and configurations. No physical installation and no tweaking of your operating system required. "Installations" of the Devon distribution do not interfere with each other nor with other installed software.

4.1.2. Multiple Workspaces

There is inbuilt support for working with different workspaces on different branches. 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.

4.2. Cobigen

In the Devon distribution we have a code generator to create CRUD code, called **Cobigen**. This 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.



Cobigen is distributed in the Devon distribution as an Eclipse plugin, and is available to all Devon developers for Capgemini engagements. Due to the importance of this component and the scope of its functionality, it is fully described [here](#).

4.3. IDE Plugins:

Since an application's code can greatly vary, and every program can be written in lots of ways without being semantically different, IDE comes with pre-installed and pre-configured plugins that use some kind of a probabilistic approach, usually based on pattern matching, to determine which pieces of code should be reviewed. These hints are a real time-saver, helping you to review incoming changes and prevent bugs from propagating into the released artifacts. Apart from Cobigen mentioned in the previous paragraph, the IDE provides CheckStyle, SonarQube, FindBugs and SOAP-UI. Details of each can be found in subsequent sections.

4.3.1. CheckStyle

What is CheckStyle

[CheckStyle](#) is a Open Source development tool to help you ensure that your Java code adheres to a set of coding standards. Checkstyle does this by inspecting your Java source code and pointing out items that deviate from a defined set of coding rules.

With the Checkstyle IDE Plugin, your code is constantly inspected for coding standard deviations. Within the Eclipse workbench, you are immediately notified with the problems via the Eclipse Problems View and source code annotations similar to compiler errors or warnings. This ensures an extremely short feedback loop right at the developers fingertips.

Why use CheckStyle

If your development team consists of more than one person, then obviously a common ground for coding standards (formatting rules, line lengths etc.) must be agreed upon - even if it is just for practical reasons to avoid superficial, format related merge conflicts. Checkstyle Plugin helps you define and easily apply those common rules.

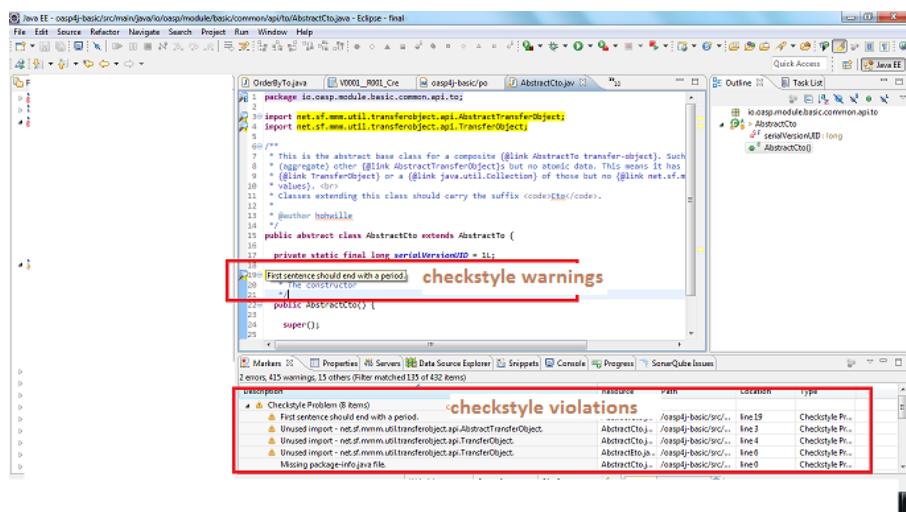
The plugin uses a project builder to check your project files with Checkstyle. Assuming the IDE Auto-Build feature is enabled, each modification of a project file will immediately get checked by

Checkstyle on file save - giving you immediate feedback about the changes you made. To use a simple analogy, the Checkstyle Plug-in works very much like a compiler but instead of producing .class files, it produces warnings where the code violates Checkstyle rules. The discovered deviations are accessible in the Eclipse Problems View, as code editor annotations and via additional Checkstyle violations views.

Installation of CheckStyle

After IDE installation, IDE provides default checkstyle configuration file which has certain check rules specified. The set of rules used to check the code is highly configurable. A Checkstyle configuration specifies which check rules are validated against the code and with which severity violations will be reported. Once defined a Checkstyle configuration can be used across multiple projects. The IDE comes with several pre-defined Checkstyle configurations. You can create custom configurations using the plugin's Checkstyle configuration editor or even use an existing Checkstyle configuration file from an external location.

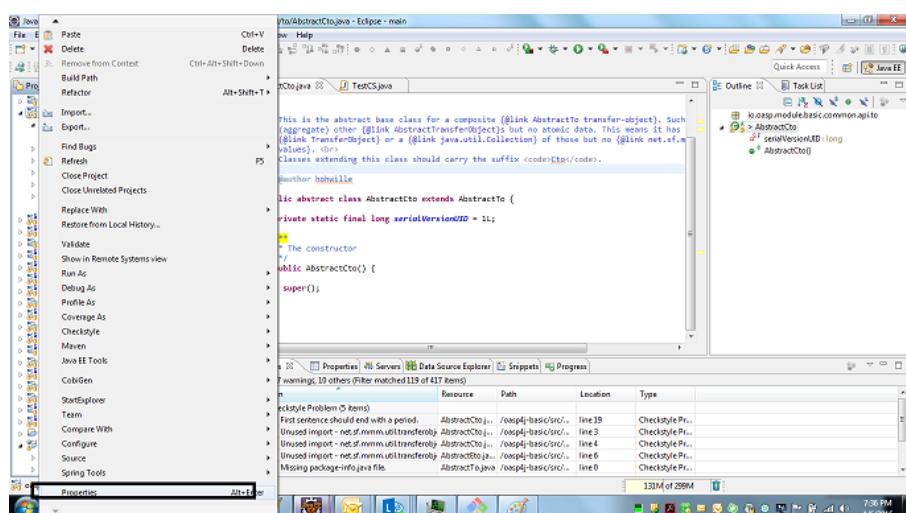
You can see violations in your workspace as shown in below figure.



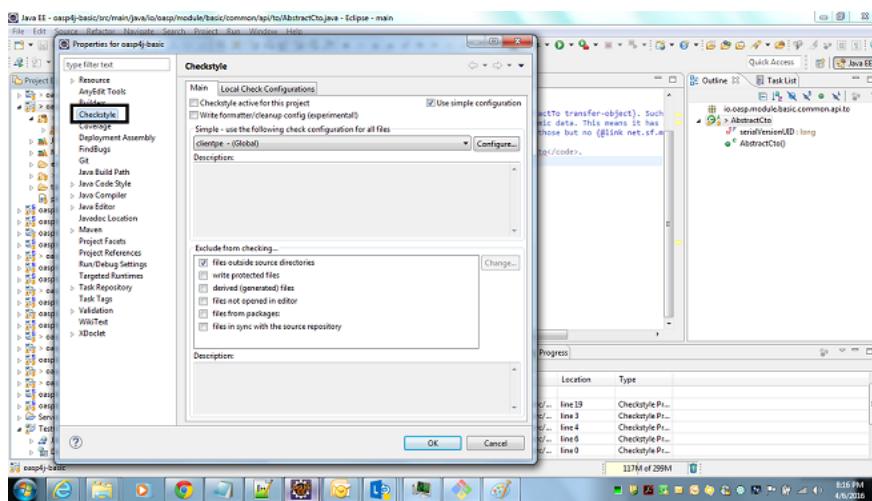
Usage

So, once projects are created, follow steps mentioned below, to activate checkstyle:

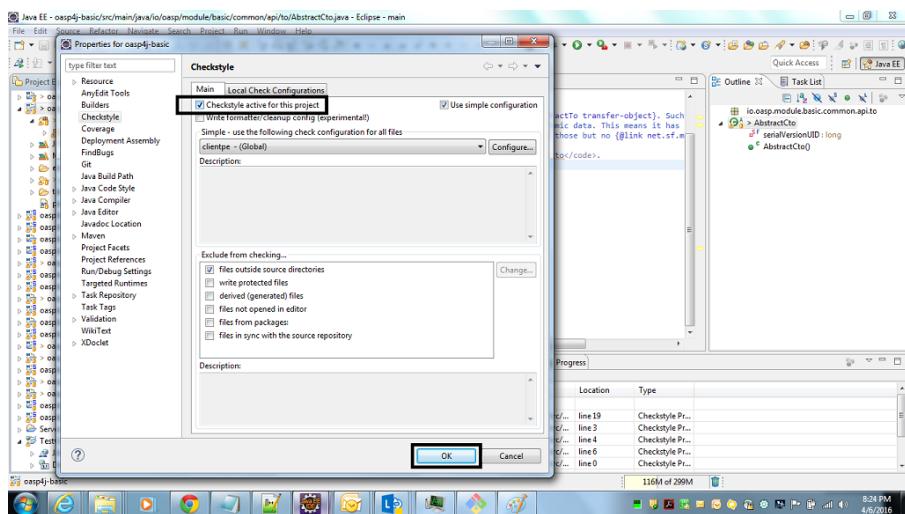
1. Open the properties of the project you want to get checked.



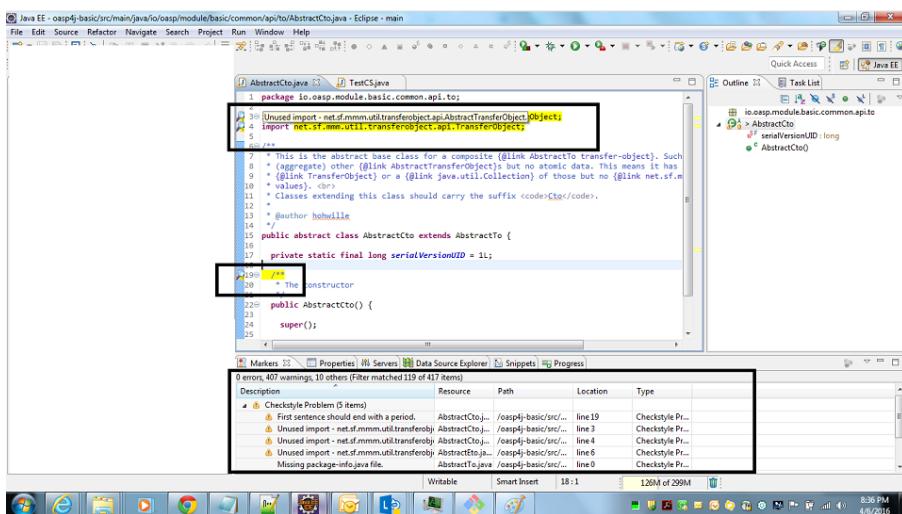
2. Select the Checkstyle section within the properties dialog .



3. Activate Checkstyle for your project by selecting the Checkstyle active for this project check box and press OK



Now Checkstyle should begin checking your code. This may take a while depending on how many source files your project contains. The Checkstyle Plug-in uses background jobs to do its work - so while Checkstyle audits your source files you should be able to continue your work. After Checkstyle has finished checking your code please look into your Eclipse Problems View. There should be some warnings from Checkstyle. These warnings point to the code locations where your code violates the preconfigured Checks configuration.



You can navigate to the problems in your code by double-clicking the problem in you problems view. On the left hand side of the editor an icon is shown for each line that contains a Checkstyle violation. Hovering with your mouse above this icon will show you the problem message. Also note the editor annotations - they are there to make it even easier to see where the problems are.

4.3.2. FindBugs

What is FindBugs

[FindBugs](#) is an open source project for a static analysis of the Java bytecode to identify potential software bugs. Findbugs provides early feedback about potential errors in the code.

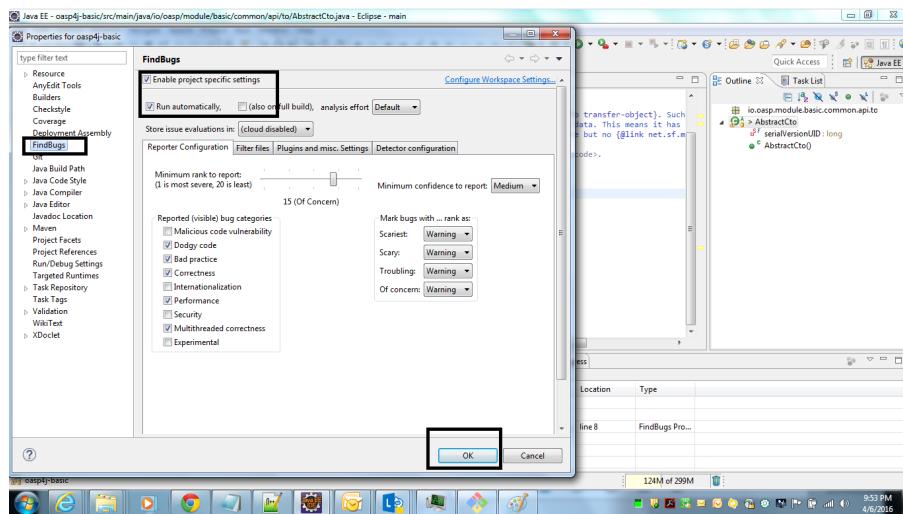
Why use FindBugs

It scans your code for bugs, breaking down the list of bugs in your code into a ranked list on a 20-point scale. The lower the number, the more hardcore the bug. This helps the developer to access these problems early in the development phase.

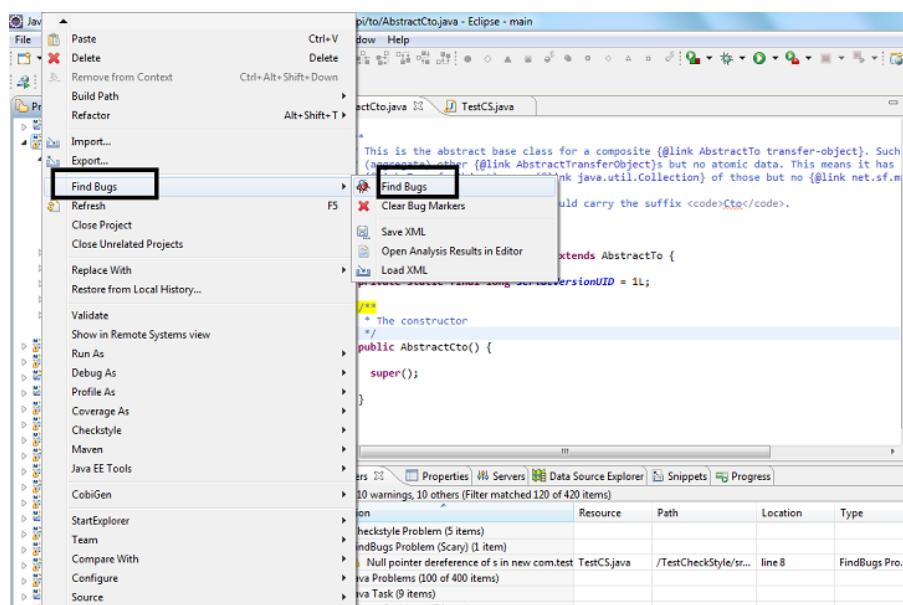
Installation and Usage of FindBugs

IDE comes preinstalled with FindBugs plugin.

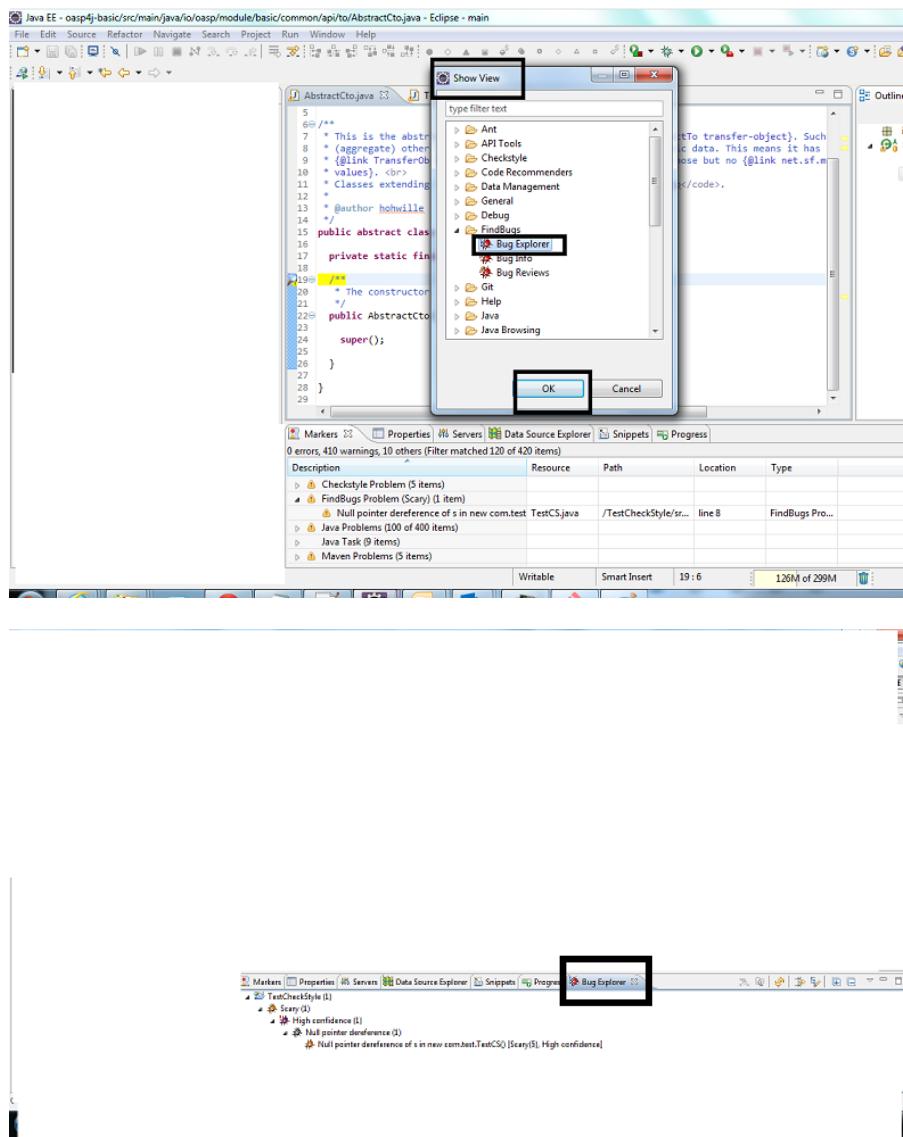
You can configure that FindBugs should run automatically for a selected project. For this right-click on a project and select Properties from the popup menu. via the project properties. Select FindBugs → Run automatically as shown below.



To run the error analysis of FindBugs on a project, right-click on it and select the Find Bugs... → Find Bugs menu entry.



Plugin provides specialized views to see the reported error messages. Select Window → Show View → Other... to access the views. The FindBugs error messages are also displayed in the Problems view or as decorators in the Package Explorer view.



4.3.3. SonarLint

what is SonarLint

SonarLint is an open platform to manage code quality. It provides on-the-fly feedback to developers on new bugs and quality issues injected into their code..

Why use SonarLint

It covers seven aspects of code quality like junits, coding rules, comments, complexity, duplications, architecture and design and potential bugs. SonarLint has got a very efficient way of navigating, a balance between high-level view, dashboard and defect hunting tools. This enables to quickly uncover projects and / or components that are in analysis to establish action plans.

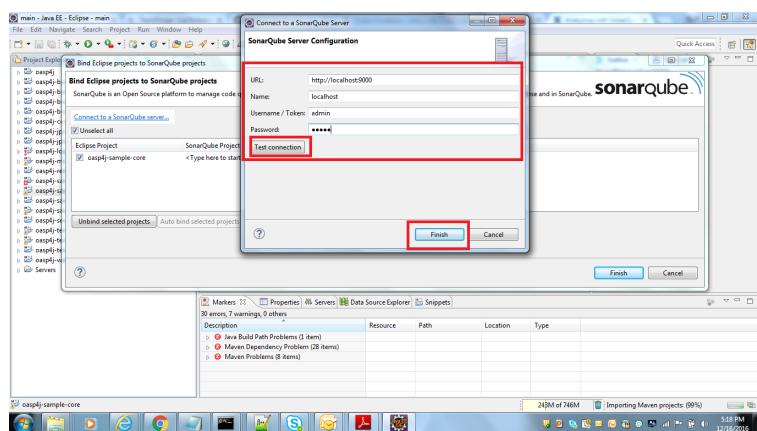
Installation and usage of SonarLint

IDE comes preinstalled with SonarLint. To configure it , please follow below steps:

First of all, you need to start sonar service. For that , go to software folder which is extracted from Devon-dist zip, choose sonarqube → bin → <choose appropriate folder according to your OS> → and execute startSonar bat file.

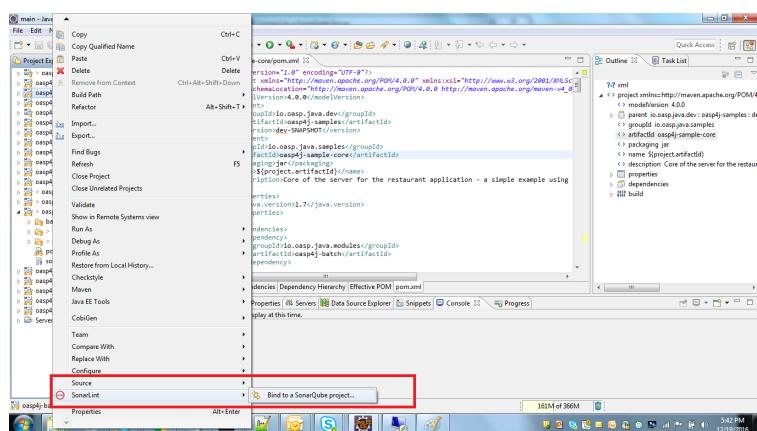
If your project is not already under analysis, you'll need to declare it through the SonarQube web interface as described [here](#). Once your project exists in SonarQube, you're ready to get started with SonarQube in Eclipse.

SonarLint in Eclipse is pre-configured to access a local SonarQube server listening on <http://localhost:9000/>. You can edit this server, delete it or add new ones. By default, user and password is "admin". If sonar service is started properly, test connection will give you successful result.

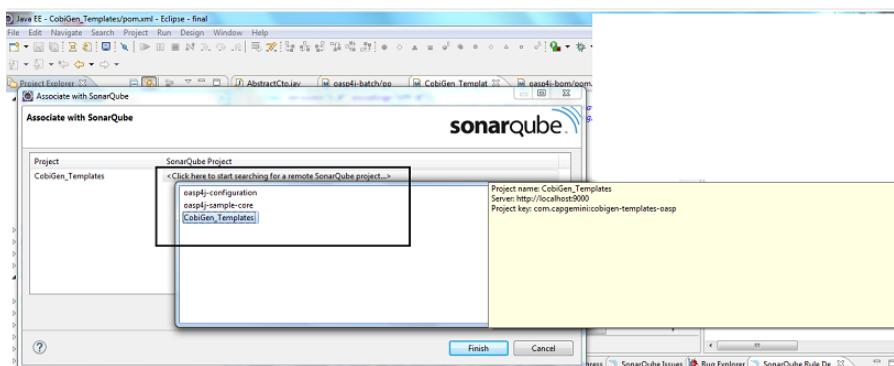


For getting a project analysed on sonar, refer this <http://docs.sonarqube.org/display/SONAR/Analyzing+Source+Code> [link].

Linking a project to one analysed on sonar server.



In the SonarQube project text field, start typing the name of the project and select it in the list box:

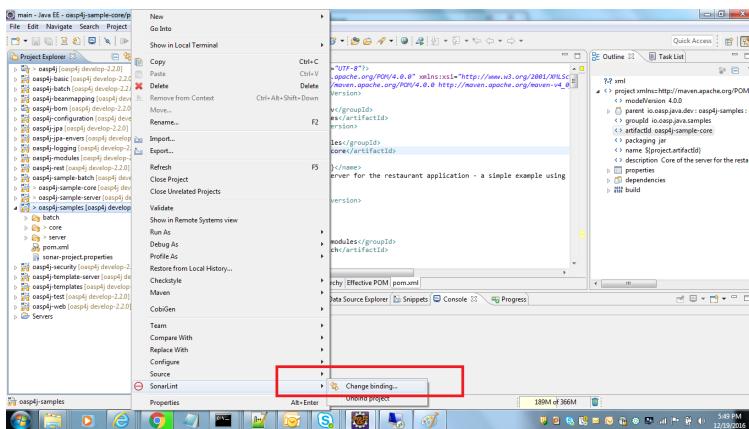


Click on Finish. Your project is now associated to one analyzed on your SonarQube server.

Changing Binding

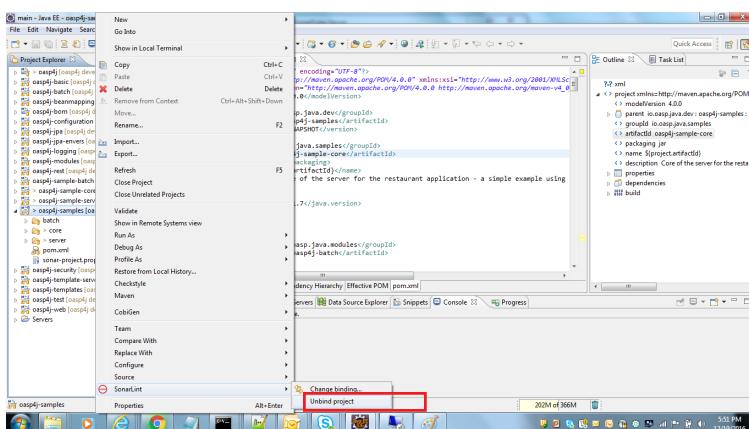
At any time, it is possible to change the project association.

To do so, right-click on the project in the Project Explorer, and then SonarQube > Change Project Association.



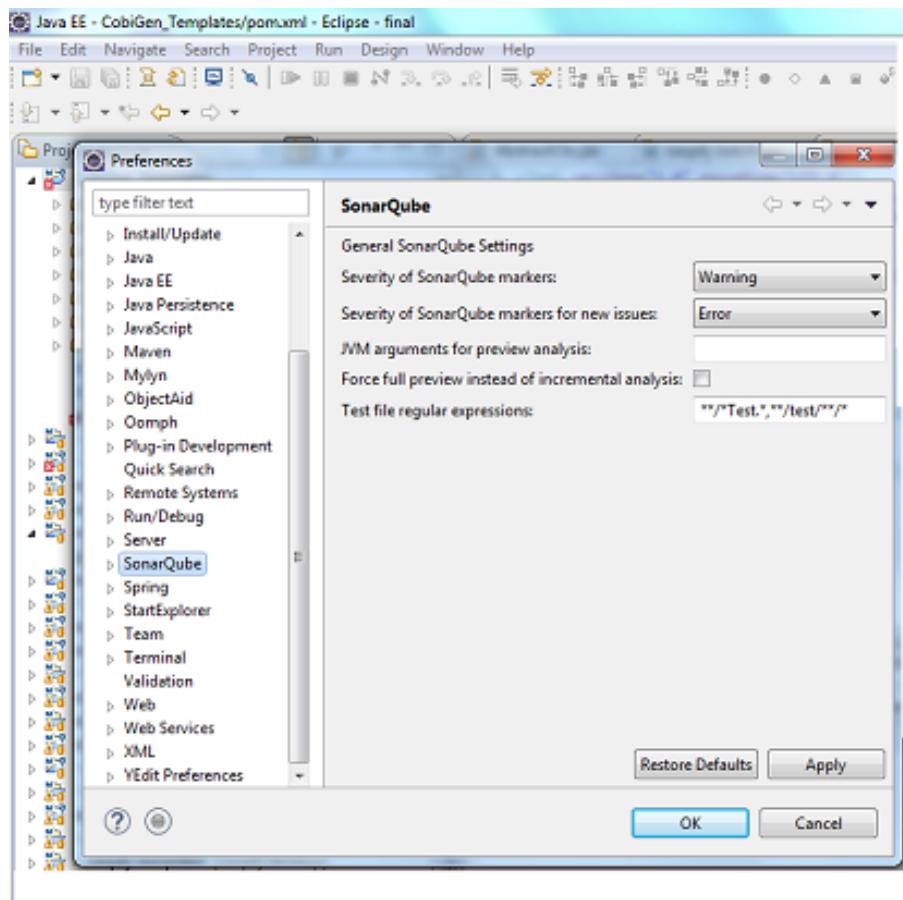
Unbinding a Project

To do so, right-click on the project in the Project Explorer, and then SonarQube > Remove SonarQube Nature.

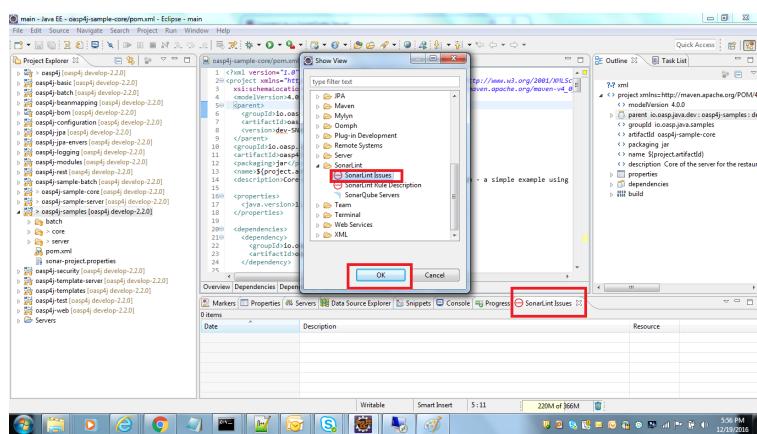


Advanced Configuration

Additional settings (such as markers for new issues) are available through Window > Preferences > SonarLint



To look for sonarqube analysed issue, go to Window → Show View → Others → SonarLint → SonarLint Issues. Now you can see issues in soanrqube issues tab as shown



Or you can go to link <http://localhost:9000> and login with admin as id and admin as password and goto Dashboard.you can see all the statistics of analysis of the configured projects on sonar server.

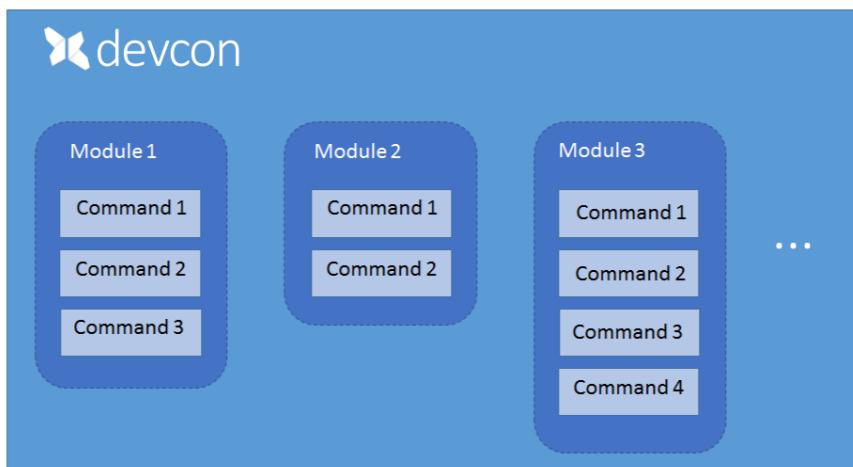
Part II: Devcon

5. Devcon Command Developer's guide

5.1. Introduction

Devcon is a cross-platform command line and GUI tool written in Java that provides many automated tasks around the full life-cycle of devonfw applications.

The structure of Devcon is formed by two main elements: *modules* and *commands*.



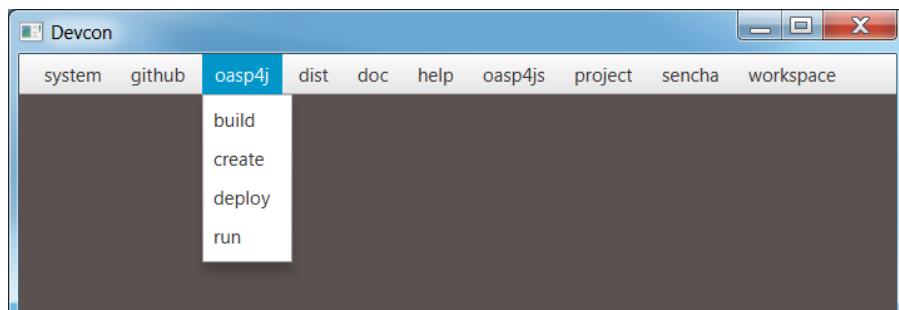
where each module represents an area of Devon and groups commands that are related to some specific task of that area.

There is also a third element with a main spot in Devcon, the *parameters*, we will see them later.

After [installing Devcon](#) you can see the modules and commands available out of the box opening a command line and using the command `devon -g` to launch the Devcon's graphic user interface.



Using the command line and the command `devon -h` (even using only the keyword `devon` or `devcon`) and `devon <module> -h` will show the equivalent information.



The available modules appear in the window bar and clicking over each module a drop down menu shows the list of commands grouped under a particular module.

As showed above the module *devon4j* has four commands related to the *devon4j* projects: *create*, *build*, *run* and *deploy*. Each command takes care of an specific task within the context of that particular module.

5.2. Creating your own Devcon modules

Devcon has been designed to be easily extended with new custom functionalities. Thanks to its structure based on *modules* and *commands* (and *parameters*) the users can cover new tasks simply including new modules and commands to the tool.

We will be able to do that in two ways:

- Adding a new Java module in the *core* of Devcon.
- Adding an external module written in Javascript.

Let's see the basic elements to have in mind before starting with the addition of new modules.

5.2.1. Elements and their keywords

Each main element of Devcon needs to be *registered* to become accessible, to achieve that we annotate each element with a specific *keyword* that will tell Devcon, during the launching process, which elements are available as modules and commands.

module registry

Internally the modules are registered in Devcon's context using the `@CmdModuleRegistry` annotation and providing some *metadata* (like *name*, *description*, etc.) to define the basic details of the module.

In the Javascript approach this annotation will be replaced by a `json` file.

command registry

In the same way, the commands in Devcon are registered using the `@Command` annotation, that also allows to add *metadata* (*name*, *description*, etc.) to provide more information.

parameter registry

In most cases the commands will need parameters to work with. The `@Parameters` and `@Parameter` annotations allow to register those in Devcon. The `@Parameter` annotation also allows to define the basic info of each parameter (*name*, *description*, etc.).

5.2.2. Creating a java module

If you are not interested in create *core* modules and want to focus on external Javascript modules skip this section and go directly to [Javascript modules](#) part.

So once we have the basic definition of the Devcon's elements and we know how to register them, let's see how to add a new module in Devcon's *core* using Java.

In this example we are going to create a new module called *file* in order to manage files. As a second stage we are going to add an *extract* command to extract zip files. To avoid the tricky details we are going to reuse the *unzip* functionality already implemented in the Devcon's utilities.

1 - Get the last Devcon release from <https://github.com/devonfw/devcon/releases>

2 - Unzip it and *Import* the Devcon project using Eclipse.

3 - In `src/main/java/com.devonfw.devcon/modules` folder create a new package *file* for the new module and inside it add a new *File* class.

Module annotations

To define the class as a Devcon module we must provide:

- `@CmdModuleRegistry` annotation with the attributes:
 - *name*: for the module name.
 - *description*: for the module description that will be shown to the users.
 - *visible*: if not provided its default value is *true*. Allows to hide modules during develop time.
 - *sort*: to sort modules, if not provided the default value will be *-1*. If *sort* $>= 0$, it will be sorted by descending value. Modules which do not have any value for sort attribute or which have value < 1 will be omitted from numeric sort and will be sorted alphabetically. This modules will be appended to the modules which are sorted numerically.
- extend the *AbstractCommandModule* to have access to all internal features already implemented for the modules (access to output and input methods, get metadata from the project *devon.json* file, get the directory from which the command has been launched, get the root of the distribution and so forth).

Finally we will have something like

```
@CmdModuleRegistry(name = "file", description = "custom devcon module", sort = -1)
public class File extends AbstractCommandModule {

    ...
}
```

Command annotations

Now is time to define the command *extract* of our new module *file*. In this case we will need to provide:

- `@Command` annotation with attributes:
 - *name*: for the command name.
 - *description*: for the command description that will be shown to the users.
 - *context*: the context in which the command is expected to be launched regarding a project. E.g. think in the *devon4j run* command. In this case the *run* command of the *devon4j* module needs to be launched within the context of an *devon4j* project. We will define that context using this *context* attribute. The options are:
 - *NONE*: if the command doesn't need to be launched within a project context.

- **PROJECT**: if the command is expected to be launched within a project (devon4j, devon4js or Sencha). In these cases this context definition will automatically provide a default *path* parameter to the command parameters alongside some extra project info (see the *devon4j run* implementation.).
- **COMBINEDPROJECT**: if the command needs to be launched within a combined (server & client) project.
 - *proxyParams*: in case you need to configure a proxy this attribute will inject automatically a *host* and *port* parameters as part of the parameters of your command.
 - *sort*: see the *sort* attribute in the previous section.

Parameter annotations

To define the parameters of our *extract* method we must use the following annotations:

- **@Parameters** annotation to group the command parameters
 - *value*: an array with the parameters
 - **@Parameter** annotation for each parameter expected.
 - *name*: the name for the parameter.
 - *description*: the description of the parameter to be shown to the users.
 - *optional*: if the parameter is mandatory or not, by default this attribute has as value *false*, so by default a parameter will be mandatory.
 - *sort*: see the *sort* attribute in the previous section.
 - *inputType*: the type of field related to the parameter to be shown in the graphic user interface of Devcon.
 - *GENERIC* for text field parameters.
 - *PATH* if you want to bind the parameter value to a *directory window*.
 - *PASSWORD* to show a password field.
 - *LIST* to show a dropdown list with predefined options as value for a parameter.

Let's imagine that in our *extract* example we are going to define two parameters *filepath* and *targetpath* (the location of the zip file and the path to the folder to store the extracted files). As our command will extract a zip file we don't need a particular project context so we will use the *ContextType.NONE*.

Finally, importing the package `com.devonfw.devcon.common.utils.Extractor` we will have access to the *unZip* functionality. Also, thanks to the *AbstractCommandModule* class that we have extended we have access to an output object to show info/error messages to the users.

So our example will look like

```

@CmdModuleRegistry(name = "file", description = "custom devcon module", sort = -1)
public class File extends AbstractCommandModule {

    @Command(name = "extract", description = "This command extracts a zip file.",
    context = ContextType.NONE)
    @Parameters(values = {
        @Parameter(name = "filepath", description = "path to the file to be extracted",
        inputType = @InputType(name = InputTypeNames.GENERIC)),
        @Parameter(name = "targetpath", description = "path to the folder to locate the
        extracted files", inputType = @InputType(name = InputTypeNames.PATH)) })
    public void extract(String filepath, String targetpath){
        getOutput().showMessage("Extracting...");
        try {
            Extractor.unZip(filepath, targetpath);
            getOutput().showMessage("Done!");
        } catch (Exception e) {
            getOutput().showError("Ups something went wrong.");
        }
    }
}

```

Generate the jar

Finally, we need to generate a new devcon.jar file containing our new module. To do so, in Eclipse, with right click over the *devcon* project in the *Project Explorer* panel:

- *Export > Runnable JAR file > Next*
- Runnable JAR File Export window:
 - Launch configuration: Devcon (if you don't have any option for that parameter try to launch once the Devcon.java class with right click and *Run as > Java Application* and start again the JAR generation).
 - Export destination: select a location for the jar.
 - Check 'Extract required libraries into generated JAR'.
 - Click *Finish* and click *OK* in the next window prompts.

Once we have the devcon.jar file we have two options depending if we are customizing a Devcon installed locally or the Devcon tool included with the Devon distributions (from version 2.1.1 onwards).

- OPTION1: If you are working over a local installation of Devcon you only need to copy the *devcon.jar* you just created, to *C:\Users\{Your User}\.devcon* replacing the devcon.jar that is inside of that directory with your new *devcon.jar* (be aware that the directory *.devcon* may be placed in another drive like *D*).



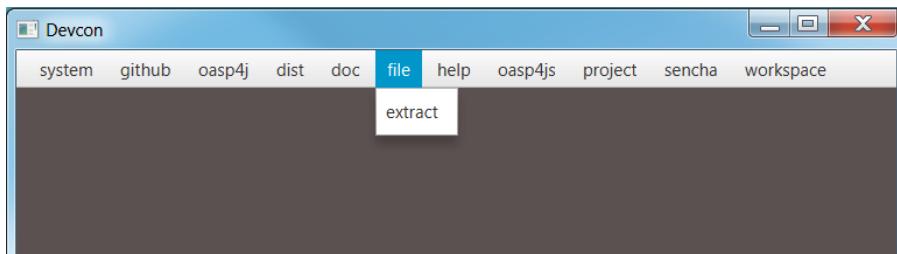
If you don't have Devcon installed you can see how to install it [here](#)

- OPTION 2: In case you are working over the copy of Devcon enabled by default in Devon

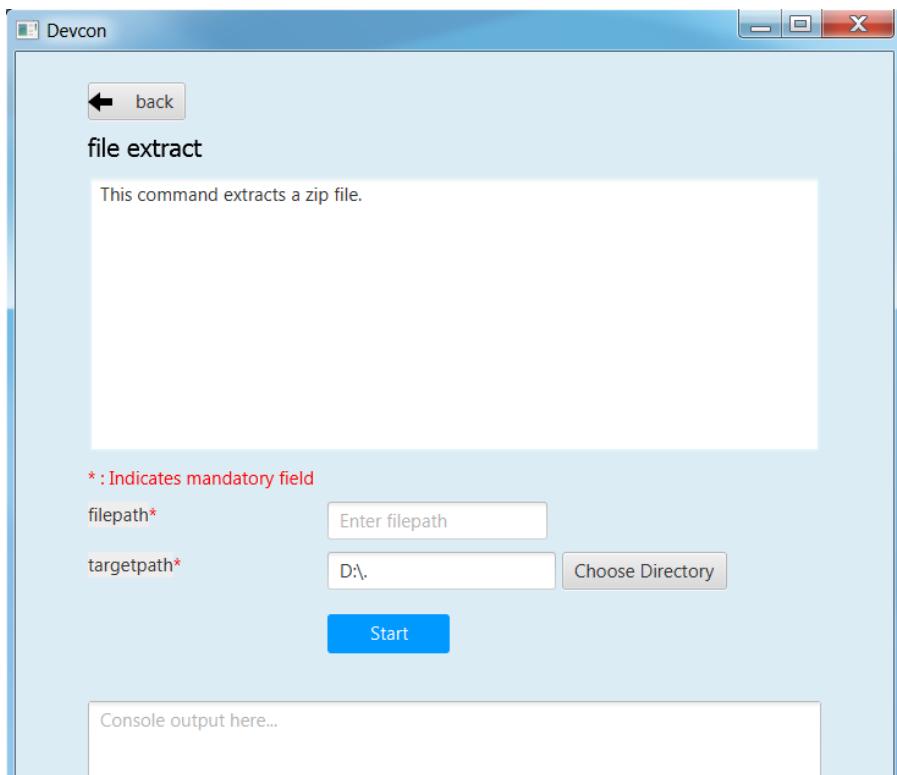
distributions you only need to copy the `devcon.jar` you just created, to `Devon-distribution\software\devcon` replacing the `devcon.jar` that is inside of that directory with your new `devcon.jar`

Once we have installed our customized version of Devcon we can open the Windows command line (for local Devcon installations) or `console.bat` script (for the Devcon included in Devon distributions) and type `devcon -g` or `devcon -h`. The first one will open the Devcon graphic user interface, the second one will show the Devcon basic info in the command line. In both cases we should see our new module as one of the available modules.

In case of the `gui` option we will see



And selecting the `extract` command we can see that the parameters we defined appear as mandatory parameters.



If you want to try the same but using the command line you can use the command `devcon file extract -h`

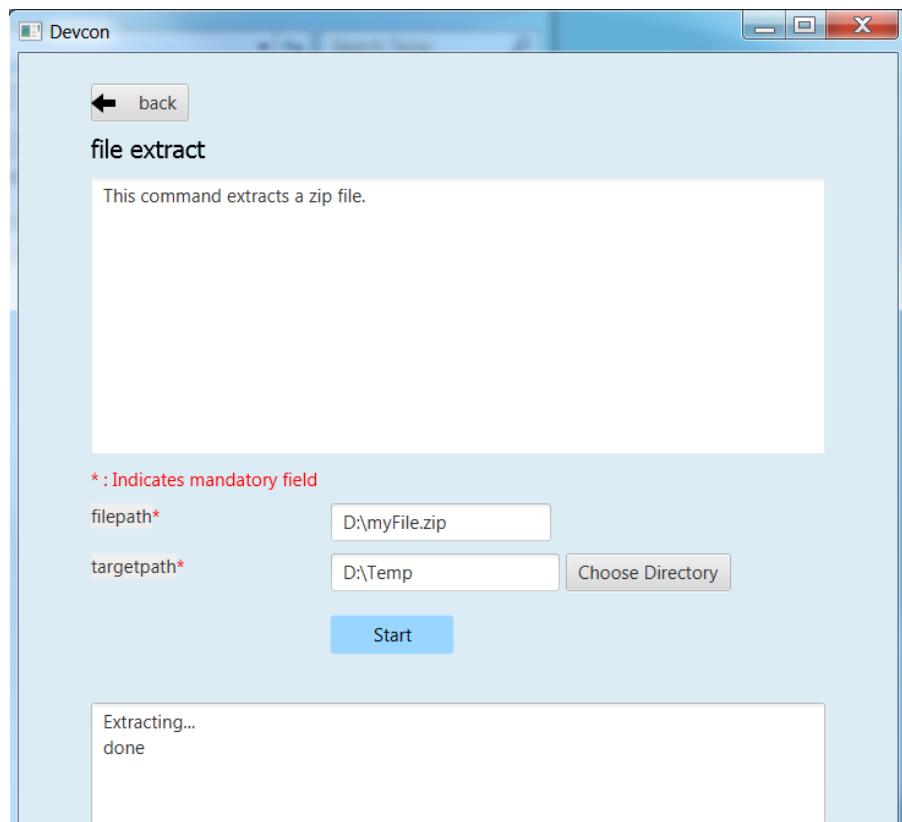
Using our module and command

Finally we want to use the `extract` command of our `file` module to extract a real zip file.

We have a `myFile.zip` in `D:` and want to extract the files into `D:\Temp` directory

with the gui

We will need to provide both mandatory parameters and click *Start* button



with the command line

We would obtain the same result using the command line

```
C:\>devcon file extract -filepath D:\myFile.zip -targetpath D:\Temp
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
Extracting...
file unzip : D:\Temp\myFile\file1.txt
file unzip : D:\Temp\myFile\file2.txt
file unzip : D:\Temp\myFile\file3.txt
file unzip : D:\Temp\myFile\file4.txt
Done

C:\>
```

That's all, with these few steps, we have created and included a new customized module written in Java in the Devcon's core.

5.2.3. Javascript modules

As we mentioned at the beginning of this chapter Devcon allows to be extended with custom modules in an external way by adding modules written in Javascript.



You will need to have installed Java 8 to be able to run Javascript modules.

We have seen how to define the Devcon's elements (modules, commands and parameters) and how to register them (using keywords) so let's see how to add a new module to Devcon using Javascript.

Module structure

The Javascript modules must include two main files:

- the **commands.json** file that contains the definition of the elements of the module (module metadata, commands and parameters).
- a Javascript file <**name of the module**>.js with the logic of the module.

How to register a module

To register a Javascript module we only need to create a directory with that two files and add it to the Devcon's module engine. If you have installed Devcon locally you should add that directory in a *scripts* directory within the `C:\Users\{Your User}\.devcon` folder but if you are customizing the Devcon included by default in the Devon distributions (for versions 2.1.1 or higher) you should add the directory with the *json* and the *js* files in a *scripts* directory within the `Devon-dist\software\devon` folder (we will see it later in more detail).

Module definition

The *commands.json* file located in the Javascript module folder defines the elements included in it, from the module details, as name or description, to the commands and its parameters.

If you have followed the [Creating a Java module](#) section you have seen that for the Java modules we use the `@CmdModuleRegistry` annotation to register a module. In the case of the Javascript modules this is replaced by the *commands.json* file itself so we won't have an equivalent *module registry* keyword.

To define the module in the *commands.json* file we can use the following attributes:

- *name*: for the module name.
- *description*: for the module description that will be shown to the users.
- *visible*: `true/false` attribute. Allows to hide modules in case we don't want them to be available.
- *sort*: to sort modules, if *sort* ≥ 0 , it will be sorted by descending value. Modules which have value < 1 will be omitted from numeric sort and will be sorted alphabetically. This modules will be appended to the modules which are sorted numerically.

An example for a *commands.json* might look like

```
{  
  "name": "myJSmodule",  
  "description": "this is an example of a Devcon Javascript module",  
  "visible": true,  
  "sort": -1,  
  
  ...  
}
```

Command definition

Also in the `commands.json` file we will define the commands of the module and its parameters.

- We will use a **commands** array to enumerate all the commands of a module. Each command will be defined with the following attributes:
 - *name*: for the command name.
 - *path*: path to the *js* file that contains the logic of the module. If this is located in the same folder than the `commands.json` file we can provide only the name of the file, without the path.
 - *description*: for the command description that will be shown to the users.
 - *context*: the context in which the command is expected to be launched regarding a project. E.g. the *run* command of the *devon4j* module needs to be launched within the context of an *devon4j* project. The options to define the context are:
 - *NONE*: if the command doesn't need to be launched within a project context.
 - *PROJECT*: if the command is expected to be launched within a project (*devon4j*, *devon4js* or *Sencha*). In these cases this context definition will automatically provide a default *path* parameter to the command parameters alongside some extra project info (see the *devon4j run* implementation.).
 - *COMBINEDPROJECT*: if the command needs to be launched within a combined (server & client) project.
 - *proxyParams*: in case your command needs to configure a proxy, this attribute will inject automatically a *host* and *port* parameters as part of the parameters of your command.
 - *sort*: see the *sort* attribute in the previous section.

```
{  
  "name": "myJSmodule",  
  "description": "this is an example of a Devcon Javascript module",  
  "visible": true,  
  "sort": -1,  
  "commands": [{  
    "name": "myFirstCommand",  
    "path": "myFirstCommand.js",  
    "description": "this is my first js command",  
    "context": "NONE",  
    "proxyParams": false,  
    ...  
  }]  
}
```

Parameter definition

As part of the *command* object in the *commands.json* file we can define the parameters using the following structure of attributes:

- **parameters** array to group the command parameters. For each parameter we will define the following attributes:
 - *name*: the name for the parameter.
 - *description*: the description of the parameter to be shown to the users.
 - *optional*: a *true/false* attribute to define if the parameter is mandatory or not.
 - *sort*: see the *sort* attribute in the previous section.
 - *inputType*: by default the parameters will be represented in the Devcon's graphic user interface as text boxes but, in case we want the parameter to be a drop down list, a directory picker or a password box, we can specify it using this *inputType* attribute and defining some sub-attributes
 - *drop down list*: `"inputType": {"name":"list", "values":["optionA", "optionB", "optionC"]}`
 - *directory picker*: `"inputType": {"name":"path", "values":[]}`
 - *password box*: `"inputType": {"name":"password", "values":[]}`

In our example we are going to add two parameters, a first one that will be showed as a text box and the second one that will be a drop down with four options. The result will look like the following

```
{
  "name": "myJSmodule",
  "description": "this is an example of a Devcon Javascript module",
  "visible": true,
  "sort": -1,
  "commands": [
    {
      "name": "myFirstCommand",
      "path": "myFirstCommand.js",
      "description": "this is my first js command",
      "context": "NONE",
      "proxyParams": false,
      "parameters": [
        {
          "name": "firstParameter",
          "description": "this is my first parameter",
          "optional": false,
          "sort": -1
        },
        {
          "name": "secondParameter",
          "description": "this is my second parameter",
          "optional": true,
          "sort": -1,
          "inputType": {"name": "list", "values": ["devonfw", "devon4j", "cobigen", "devcon"]}
        }
      ]
    },
    ...
  ],
}

}
```

The commands

Each command will be defined in a separate Javascript file with a name that match the `path` attribute defined in the `commands.json` file of the module. Remember that in case that the js file is in the same directory than the `commands.json` file we only need to provide the name of the js file.

The JavaScript file must have as content either a named or anonymous function which contains the command implementation. The parameters of the funcion contain the parameters in the defined order and the `this` special property points to the Java `CommandModule` context.

So returning to our example we will have a file called `myFirstCommand.js` located in the same directory than the `commands.json`.

The content will be

```
function (firstParameter, secondParameter){
    // Here the content of your module.
}
```

Creating a javascript module

Adding the module structure

We have already seen the structure of a Devcon's Javascript module so let's see how to create one with an example that contains all steps.

In this case we are going to create (again) a command to extract a zip file, so we are going to create a module called *myJSmodule* with a command *extract* that gets two mandatory parameters *filepath* for the path to the zip file and a *targetpath* to define the location of the extracted files.

- 1. The *Devcon Directory* is

- for local installations of Devcon: *C:\Users\{Your User}\.devcon* (if you don't find the *.devcon* directory there try looking in *D:* drive, if the directory is not there neither check your Devcon's installation).
- for the Devcon tool within the Devon distribution (version 2.1.1 or higher): *Devon-dist\software\devon*



If you want to customize a copy of Devcon in a local context and you still don't have Devcon installed you can see how to download and install it [here](#).

- 2. We will need to create the *scripts* folder within the *Devcon Directory*.
- 3. Then we will need to create inside the *scripts* folder the directory for our new module and inside it we need to add
 - a *commands.json* file with the definition of the module
 - and an *extract.js* file with the code for the *extract* command.

So we will end having a structure like *{Devcon Directory}\scripts\myModule*

(C:) System ▶ Users ▶ pparrado ▶ .devcon ▶ scripts ▶ myJSmodule		
New folder		
Name	Date modified	Type
commands.json	28/11/2016 13:29	JSON File
extract.js	28/11/2016 13:22	JScript Script File

Defining the module and the command

To define and register the module and the command we will use the *commands.json* file. First we will add the module metadata (name, description) and then the commands, and its parameters,

inside the `commands` array.

```
{
  "name": "myJSmodule",
  "description": "test module",
  "visible": true,
  "sort": -1,
  "commands": [
    {
      "name": "extract",
      "path": "extract.js",
      "description": "command to extract a file",
      "context": "NONE",
      "proxyParams": false,
      "parameters": [
        {
          "name": "filepath",
          "description": "path to the file to be extracted",
          "optional": false,
          "sort": -1
        },
        {
          "name": "targetpath",
          "description": "path to the folder to locate the extracted
files",
          "optional": false,
          "sort": -1
        }
      ]
    }
  ]
}
```

Adding the command logic

As we have previously mentioned we need to add the code of our command in the `extract.js` file. As we want to extract a file, to avoid a most complicated implementation, we are going to use the `unZip` method that belongs to the `utils` package of Devcon. To access to the method we will need to provide the fully qualified name `com.devonfw.devcon.common.utils.Extractor.unZip`.

So in the `extract.js` file we must add a function that gets the two parameters defined in the `commands.json` (`filepath` and `targetpath`) and uses the Java method `unZip` to extract the file. Also remember that the special property `this` will give us access to the Devcon's module context so we will be able to use the Devcon's output (you can find the entire resources that `this` can provide [here](#))

```
function(filepath, targetpath){  
    this.getOutput().showMessage("extracting...");  
    com.devonfw.devcon.common.utils.Extractor.unZip(filepath, targetpath);  
    this.getOutput().showMessage("Done!");  
}
```

Using the new module and the command

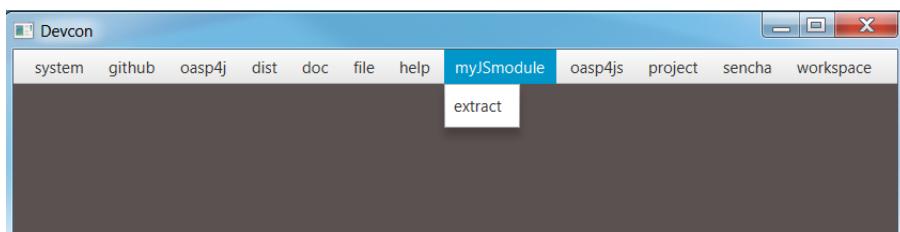
We have finished the implementation of the new Javascript module so now we can start using it.

We have created a module to extract *zip* files so we are going to use a *myFile.zip* located in the **D:** drive and we are going to extract it to the **D:\Temp** directory using our new module.

As you may know if you have followed the Devcon's documentation we can use the tool in two ways: using the command line or using the Devcon's graphic user interface (GUI).

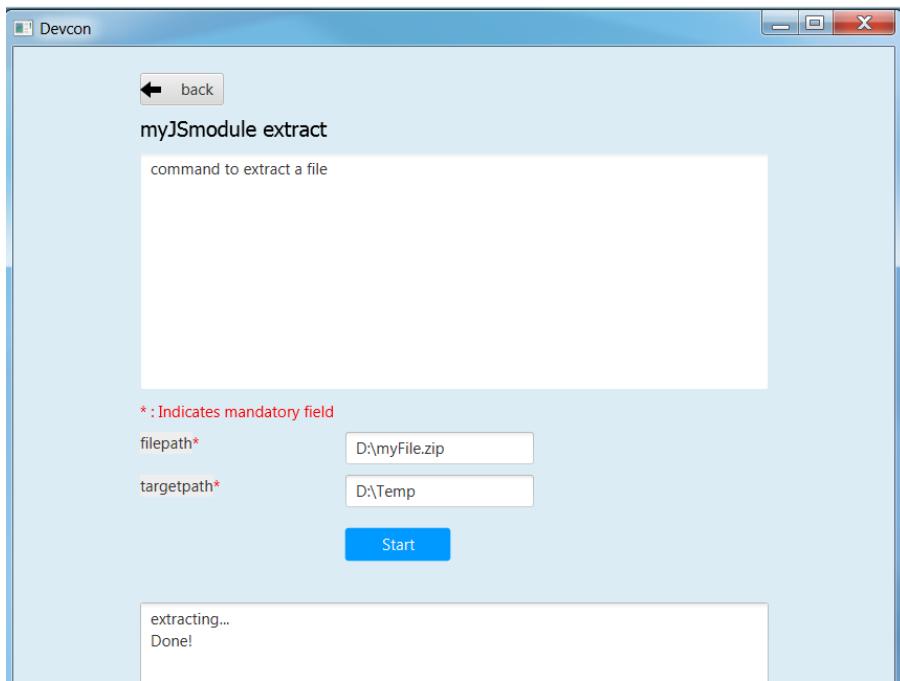
using the gui

To launch Devcon's GUI we must open a command line and use the **devon -g** command. After that the Devcon main window should be opened and we should see our new **myJSmodule** in the list of available modules. Then if we click over the module we should see the **extract** command available.



Then if we click over the **extract** command we should see a window with the name and description we provided in the **commands.json** alongside the parameters that we defined (*filepath* and *targetpath*), both mandatory.

If we provide the parameters and click on the *Start* button the command should be launched and the file should be extracted.



We have extracted the file successfully using our just created Devcon command.

using the command line

If we use the command line the result will be exactly the same.

Open a Windows command line (for local Devcon installations) or *command.line* script (for the Devcon included in Devon distributions) and launch the **devcon** command (**devon** or **devcon -h** will also work)

```

...>devon
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
usage: devon <<module>> <<command>> [parameters...] [-g] [-h] [-p] [-s] [-v]
Devcon is a command line tool that provides many automated tasks around the full
life-cycle of Devon applications.
-g,--gui      show devcon GUI
-h,--help     show help info for each module/command
-p,--prompt   prompt user for parameters
-s,--stacktrace show (if relevant) stack-trace when errors occur
-v,--version   show devcon version
List of available modules:
> dist: Module with general tasks related to the distribution itself
> doc: Module with tasks related with obtaining specific documentation
> file: custom devcon module
> github: Module to get Github repositories related to devonfw.
> help: This module shows help info about devcon
> myJSmodule: test module
> devon4j: devon4j(server project) related commands
> oasp4js: Module to automate tasks related to oasp4js
> project: Module to automate tasks related to the devon projects (server + client)
> sencha: Commands related with Ext JS6/Devon4Sencha projects
> system: Devcon and system-wide commands
> workspace: Module to create a new workspace with all default configuration

```

In the list of available modules you should see our **myJSmodule**.

Now if we ask for the **myJSmodule** information with the command **devcon myJSmodule -h** we can check that our **extract** command is available. Also we can see the needed parameters using the **devcon myJSmodule extract -h** command

```

...>devcon myJSmodule extract -h
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
usage: myJSmodule extract [-filepath] [-targetpath]
command to extract a file
-filename path to the file to be extracted
-targetpath path to the folder to locate the extracted files

```

Finally we can use the **extract** command providing both mandatory parameters

```
...>devcon myJSmodule extract -filepath D:\myFile.zip -targetpath D:\Temp  
Hello, this is Devcon!  
Copyright (c) 2016 Capgemini  
extracting...  
file unzip : D:\Temp\myFile\file1.txt  
file unzip : D:\Temp\myFile\file2.txt  
file unzip : D:\Temp\myFile\file3.txt  
file unzip : D:\Temp\myFile\file4.txt  
Done!
```

5.3. Conclusion

In this section we have seen how easy can be to extend Devcon with new modules. You can either choose to add a Java module into the core of Devcon or achieve the same in an external way creating your own modules with Javascript (remember that you will need Java 8 to run your Javascript modules).

Thanks to the Devcon's structure, in both cases the work is reduced to, first, register the modules and then define each of its elements (commands and parameters) and the modules engine of Devcon will do the rest.

6. Devcon Command Reference



In the introduction to Devcon we mentioned that Devcon is a tool based on modules that group commands so the different functionalities are stored in these modules that act as utilities containers. The current version of devcon has been released with the following modules

- dist
- doc
- github
- help
- devon4j
- devon4ng
- project
- sencha
- system
- workspace



in your Devcon version more modules may have been included. You can list them using the option `devon -h`

6.1. dist

The *dist* module is responsible for the tasks related with the distribution which means all the functionalities surrounding the configuration of the Devon distribution, including the obtention of the distribution itself.

6.1.1. dist install

The *install* command downloads a distribution from a Team Forge repository and after that extracts the file in a location defined by the user.

dist install requirements

A user with permissions to download files from Team Forge repository.

dist install parameters

The *install* parameter needs four parameters to work properly:

- **user**: a Team Forge user with permissions to download files from the repository at least.
- **password**: the Team Forge user password.

- **path:** the path where the distribution must be downloaded.
- **type:** the type of distribution. The options are '*oaspide*' to download a devon4j based distribution or '*devondist*' to download a Devon based distribution.

dist install example of usage

A simple example of usage for this command would be the following

```
D:\>devon dist install -user john -password 1234 -path D:\Temp\MyDistribution -type
devondist
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
[INFO] installing distribution...
[INFO] Downloading Devon-dist_2.0.0.7z (876,16MB). It may take a few minutes.
[=====] 100% downloaded
[INFO] File downloaded successfully.
[...]
[INFO] extracting file...
[INFO] File successfully extracted.
[INFO] The command INSTALL has finished successfully
```

You must have in mind that this process can take a while, specially depending on your connection to the internet.

After downloading and installing the distribution successfully installed. You can now follow the manual steps as described in the devonfw Guide or, alternatively, run 'devon dist init' to initialize the distribution.

6.1.2. dist init

The *init* command initializes a newly downloaded distribution.

dist init requirements

A new, not initialized distribution (running it on a configured distribution has no adverse side-effects).

dist init parameters

The *init* parameter needs one parameter to work properly:

- **path:** location of the Devon distribution (current dir if not given).

6.1.3. dist s2

The *s2* command has been developed to automate the configuration process to use Devon as a Shared Service. This configuration is based on launching two scripts included in the Devon distributions, the *s2-init.bat* and the *s2-create.bat*. The **s2-init.bat** is responsible for configuring the *settings.xml* file (located in the *conf/m2* directory). Basically enables the connection of *Maven* with

the *Artifactory* repository, where the Devon modules are stored, and adds the user credentials for this connection.

The **s2-create.bat** creates a new project in the workspace of the distribution, and optionally does a checkout of a Subversion repository inside this new project. Finally the script creates a Eclipse .bat starter related to the new project.

dist s2 requirements

- The command can be launched from any directory within a Devon distribution version 2.0.1 or higher. The Devon distribution is defined by having a *settings.json* file located in the *conf* directory. This file is a JSON object that defines parameters like the version of the distribution or the type which should be *devon-dist* as is showed below.

```
{"version": "2.0.1", "type": "devon-dist"}
```

- An *Artifactory* user with permissions to download files from the repository.
- In case the optional checkout A Subversion user with permissions to do the checkout of the project specified in the *url* parameter.

The command will search for this file to get the root directory where the scripts are located so is necessary to have this file in its correct location.

Apart from this the *settings.xml* file needs to be compatible with the Shared Services autoconfiguration script (*s2-init.bat*).

dist s2 parameters

So the *s2* command needs six parameters to be able to complete the two phases:

- **user**: the userId for Artifactory provided by S2 for the project.
- **pass**: the password for Artifactory.
- **engagementname**: the name of the repository for the engagement.
- **cias**: if the *settings.xml* must be configured for CIaaS user must set this as TRUE. Is an optional parameter with FALSE as default value.
- **projectname**: the name for the new project.
- **svnuser**: the user for the SVN.
- **svnpass**: the password for the SVN.
- **svnurl**: the url for the SVN provided by S2.

dist s2 example of usage

A simple example of usage for this command would be the followings:

If we only want to configure the *settings.xml* file without using the svn option the simplest usage would be

```
D:\devon-dist\workspaces>devon dist s2 -user john -pass ZMF4AgyhQ5X6Sr9Bd1ohjWcFjL  
-engagementname myEngagement -projectname TestProject  
Hello, this is Devcon!  
Copyright (c) 2016 Capgemini  
[...]  
INFO: Completed  
Eclipse preferences for workspace: "TestProject" have been created/updated  
Created eclipse-TestProject.bat  
Finished creating/updating workspace: "TestProject"
```

After this the `conf/.m2/settings.xml` file should have been configured and a new (and empty) *TestProject* directory must have been created in the *workspaces* directory and in the distribution root a new *eclipse-testproject.bat* script must have been created too.

We also can get the same result and configure the *settings.xml* for CIaaS using the *cias* parameter

```
D:\devon-dist\workspaces>devon dist s2 -user john -pass ZMF4AgyhQ5X6Sr9Bd1ohjWcFjL  
-engagementname myEngagement -projectname TestProject -cias true
```

Using the *svn* option to automate the check out from the repository the usage would be

```
D:\devon-dist\workspaces>devon dist s2 -user john -pass ZMF4AgyhQ5X6Sr9Bd1ohjWcFjL  
-engagementname myEngagement -projectname TestProject -svnurl  
https://coconet...Project/ -svnuser john_svn -svnpass 12345  
Hello, this is Devcon!  
Copyright (c) 2016 Capgemini  
[...]  
[INFO] The checkout has been done successfully.  
[INFO] Creating and updating workspace...  
[...]  
INFO: Completed  
Eclipse preferences for workspace: "TestProject" have been created/updated  
Created eclipse-TestProject.bat  
Finished creating/updating workspace: "TestProject"
```

After this the `conf/.m2/settings.xml` file should have been configured and a new *TestProject* directory must have been created in the *workspaces* directory with all the files checked out from the *svn* repository and in the distribution root a new *eclipse-testproject.bat* script must have been created too.

6.1.4. dist info

The *info* command provides very basic information about the Devon distribution, like type, version and path.

dist info parameters

The *dist info* command has one optional parameter:

- **path**: path to the distro. Uses current directory if not specified.

6.2. doc

With this module we can access in a straightforward way to the documentation to get started with Devon framework. The commands of this module show information related with different components of Devon even opening in the default browser the sites related with them.

- **doc devon**: Opens the Devon site in the default web browser.
- **doc devonguide**: Opens the Devon Guide in the default web browser.
- **doc getstarted**: Opens the 'Getting started' guide of Devon framework.
- **doc links**: Shows a brief description of Devon framework and lists a set of links related to it like the public site, introduction videos, the Yammer group and so forth.
- **doc devon4jguide**: Opens the devon4j guide.
- **doc sencha**: Opens the Sencha Ext JS 6 documentation site.

6.3. github

This module is implemented to facilitate getting the Github code from devon4j and Devon repositories. It has only two commands, one to get the OAPS4J code and other to get the Devon code.

6.3.1. github devon4j

This command clones the devon4j repository to the path that the user specifies in the parameters.

github devon4j parameters

The devon4j command needs only one parameter:

- **path**: the location where the repository should be cloned.
- **proxyHost**: Host parameter for optional Proxy configuration.
- **proxyPort**: Port parameter for optional Proxy configuration.

github devon4j example of usage

A simple example of usage for this command would be the following

```
D:\Projects\devon4j>devon github devon4j
```

Or using the **-path** parameter

```
D:\>devon github devon4j -path C:\Projects\devon4j
```

Also we can define, if necessary, a proxy configuration. The following example shows how configure the connection for Capgemini's proxy in Europe

```
D:\Projects\devon4j>devon github devon4j -proxyHost 1.0.5.10 -proxyPort 8080
```

6.3.2. github devoncode

This command clones the Devon repository to the path specified in the path parameter.

github devoncode requirements

A github user with download permissions over the Devon repository.

github devoncode parameters

The *devoncode* command needs three parameters:

- **path**: the location where the repository must be cloned.
- **username**: the github user (with permission to download).
- **password**: the password of the github user.
- **proxyHost**: Host parameter for optional Proxy configuration.
- **proxyPort**: Port parameter for optional Proxy configuration.

github devoncode example of usage

A simple example of usage for this command would be the following

```
D:\>devon github devoncode -path C:\Projects\devon -user John_g -pass 12345
```

Also we can define, if necessary, a proxy configuration. The following example shows how configure the connection for Capgemini's proxy in Europe

```
D:\>devon github devoncode -path C:\Projects\devon -user John_g -pass 12345 -proxyHost  
1.0.5.10 -proxyPort 8080
```

6.4. help

The help module is responsible for showing the help info to facilitate the user the knowledge to use the tool. It has only one command, the *guide* command, that doesn't need any parameter and that basically prints a summary of the devcon general usage with a list of the global options and a list with the available modules

6.4.1. help example of usage

```
D:\>devon help guide
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
usage: devon <<module>> <<command>> [parameters...]
Devcon is a command line tool that provides many automated tasks around
the full life-cycle of Devon applications.
-h,--help      show help info for each module/command
-v,--version    show devcon version
List of available modules:
> help: This module shows help info about devcon
> sencha: Sencha related commands
> dist: Module with general tasks related to the distribution itself
> doc: Module with tasks related with obtaining specific documentation
> github: Module to create a new workspace with all default configuration
> workspace: Module to create a new workspace with all default configuration
```

If you have follow this guide you can realize that the result is the same that is shown with other options as `devon` or `devon -h`. This is because these options internally are using this module *help*.

6.5. devon4j

This module groups all the devcon functionalities related to the server applications like creating, running and deploying server applications based on the devon4j project.

6.5.1. devon4j create

This command creates a new server project based on the devon4j archetype.

devon4j create requirements

This command needs to be launched from within (or pointing to) a devonfw distribution.

In a second term internally this command uses the *Maven* plugin included in the devonfw distributions so in order to be able to use this plugin we should launch this command from a devonfw command line (use the *console.bat* included in the devonfw distributions).

devon4j create parameters

This command uses five parameters (four of them mandatory).

- **servername:** the name for the new server project.
- **serverpath:** the location for the new server project. Is an optional parameter, if the user does not provide it devcon will use the current directory in its place.
- **packagename:** the name for the project package.
- **groupid:** the groupId for the project.

- **version:** the version for the project.

devon4j create example of usage

A simple example of usage for this command would be the following

```
D:\devon-dist>devon devon4j create -servername MyNewProject -packagename  
io.devon.application.MyNewProject -groupid io.devon.application -version 1.0-SNAPSHOT  
Hello, this is Devcon!  
Copyright (c) 2016 Capgemini  
[INFO] Scanning for projects...  
[...]  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 7.203 s  
[INFO] Finished at: 2016-07-14T13:00:17+01:00  
[INFO] Final Memory: 10M/42M  
[INFO] -----  
D:>
```

Or using the optional *serverpath* parameter to define the location for the project

```
D:>devon devon4j create -servername MyNewProject -serverpath D:\devon-dist\  
-packagename io.devon.application.MyNewProject -groupid io.devon.application -version  
1.0-SNAPSHOT
```

After that we should have a new *MyNewProject* project created in the *devon-dist* directory.

6.5.2. devon4j run

With this command the user can run a server project application from the embedded tomcat server.

devon4j run requirements

The command can be launched within a Devon distribution version 2.0.1 or higher. Also verify that your *devon4j* application has the *devon.json* file well configured.

devon4j run parameters

The *run* command handles two parameters

- **path:** to indicate the location of the core project of the server app. Is an optional parameter and if not provided by the user devcon will take as the path the directory from which the command has been launched.
- **port:** the port from which the app should be accessible.

devon4j run example of usage

A simple example of usage for this command would be the following

```
D:\devon-dist\workspaces\MyApp\core>devon devon4j run -port 8081
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
Application started

[...]

      .----'--_ _ _(_)_ _ _ _ \ \ \ \
( ( )\__|_|_|'|-|_|'`| \| \ \ \
\ \ \ _ __|_|_|_|_|_|(|_|_) ) ) )
' |_____| .__|_|_|_|_|_\__,_| / / /
=====|_|=====|_|/_=/_/_/_/
:: Spring Boot ::          (v1.3.3.RELEASE)

2016-07-01 11:13:59.006 INFO 6116 --- [           main] i.d.application.MyAp
p.SpringBootApp : Starting SpringBootApp on LES002610 with PID 6116 (D:\devon-
alpha\workspaces\MyApp\core\target\classes started by pparrado in D:\devon-al
pha\workspaces\MyApp\core)

[...]

2016-07-01 11:14:18.297 INFO 6116 --- [           main] i.d.application.MyAp
p.SpringBootApp : Started SpringBootApp in 19.698 seconds (JVM running for 35.
789)
```

Or providing the optional *path* parameter

```
D:\>devon devon4j run -port 8081 -path D:\devon-dist\workspaces\MyApp\core
```

6.5.3. **devon4j build**

With this command the user can build a server project, is the equivalent to the `mvn clean install` command

devon4j build requirements

In order to work properly the command must be launched from within (or pointing to) a devon4j project directory (the devon4j project type is defined in a *devon.json* file with parameter 'type' set to 'devon4j').

devon4j build parameters

This command only uses one parameter

-path: the location of the server project. This is an optional parameter and if the user does not provide it devcon will use in its place the current directory from which the command has been launched.

devon4j build example of usage

A simple example of usage for this command would be the following

```
D:\devon-dist\workspaces\MyApp>devon devon4j build
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
projectInfo read...
path D:\devon-dist\workspaces\MyApp project type devon4j

[...]

[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] MyApp ..... SUCCESS [ 0.301 s]
[INFO] MyApp-core ..... SUCCESS [ 12.431 s]
[INFO] MyApp-server ..... SUCCESS [ 3.699 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 16.712 s
[INFO] Finished at: 2016-07-15T11:44:00+01:00
[INFO] Final Memory: 31M/76M
[INFO] -----
D:\devon-dist\workspaces\MyApp>
```

Or using the optional parameter *path*

```
D:\>devon devon4j build -path D:\devon-dist\workspaces\MyApp
```

6.5.4. devon4j migrate

With this command the user can update server project to the latest version

devon4j migrate requirements

The migrate command need only the Project that's needs to migrate to latest version of devon4j

devon4j migrate parameters

This command only uses one parameter

-projectPath: the location of the server project that's needs to be updated.

devon4j migrate example of usage

A simple example of usage for this command would be the following

```
D:\devon-dist\workspaces\MyApp>devon devon4j migrate D:\Devon-dist_2.4.0\testproj
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
*****
Migrating from version oasp4j:2.6.0 to oasp4j:2.6.1 ...
*****
Migrating file: D:\Devon-dist_2.4.0\testproj\pom.xml
*****
Migrating from version oasp4j:2.6.1 to oasp4j:3.0.0 ...
*****
Migrating file: D:\Devon-dist_2.4.0\testproj\core\pom.xml
[.....
.....]
Migrating file: D:\Devon-
dist_2.4.0\testproj\src\main\java\com\company\SpringBootBatchApp.java
Migrating file: D:\Devon-
dist_2.4.0\testproj\src\test\java\com\company\general\batch\base\test\SpringBatchInteg-
rationTest.java
*****
Successfully applied 3 migrations to migrate project from version oasp4j:2.6.0 to
devon4j:3.0.0.
*****
```

After successful upgrade of the project, below are the manual steps that are needed to perform

Add the following in class WebSecurityBeansConfig

```
import org.springframework.security.crypto.factory.PasswordEncoderFactories;
import org.springframework.security.crypto.password.PasswordEncoder;
```

Add the below method

```
@Bean
public PasswordEncoder passwordEncoder() {
    return PasswordEncoderFactories.createDelegatingPasswordEncoder();
}
```

Add the following in class BaseWebSecurityConfig

```
import org.springframework.security.crypto.password.PasswordEncoder;
```

```
@Inject  
private PasswordEncoder passwordEncoder;
```

In method configureGlobal update below

```
auth.inMemoryAuthentication().withUser("waiter").password(this.passwordEncoder.encode("waiter")).roles("Waiter")  
    .and().withUser("cook").password(this.passwordEncoder.encode("cook")).roles("Cook").and().withUser("barkeeper")  
    .password(this.passwordEncoder.encode("barkeeper")).roles("Barkeeper").and()  
.withUser("chief")  
    .password(this.passwordEncoder.encode("chief")).roles("Chief");
```

6.6. devon4ng

The devon4ng module is responsible for automating the tasks related to the client projects based on Angular.

6.6.1. devon4ng create

With this command the user can create a basic devon4ng app.

devon4ng create requirements

This command must be used within a devonfw distribution with version 2.0.0 or higher. You can check your distribution's version looking at the conf/settings.json file.

devon4ng create parameters

This command accepts two parameters:

- **clientname**: the name for the application.
- **clientpath**: the location for the new application. Is an optional parameter and if not provided by the user devcon will take as the path the directory from which the command has been launched.

devon4ng create example of usage

A simple example of usage for this command would be the following

```
D:\devon-dist\workspaces>devon devon4ng create -clientname MyDevon4ngApp
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
Creating project MyDevon4ngApp...
installing ng
  create .editorconfig
  create README.md
  create src\app\app.component.css
  [...]
  create tslint.json
Installing packages for tooling via npm.
Installed packages for tooling via npm.
Project 'MyDevon4ngApp' successfully created.
Adding devon.json file...
Project build successfully

D:\devon-dist\workspaces>
```

If everything goes right a new directory *MyDevon4ngApp* must have been created containing the basic structure of an *devon4ng* app.

The user can also use the next command *devon4ng build* to do that last operation.

6.6.2. devon4ng build

With this command the user can resolve the dependencies of an *devon4ng* app. The *devon4ng build* command is the equivalent to the **ng build** command.

devon4ng build parameters

- **path:** The location of the *devon4ng* app. Is an optional parameter and if not provided devcon will use the current directory from which the command has been launched instead.

devon4ng build example of usage

A simple example of usage for this command would be the following

```
D:\devon-dist\workspaces\MyDevon4ngApp>devon devon4ng build
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
Building project...
Hash: 936deb00dfd88c0d9e56
Hash: 936deb00dfd88c0d9e56
Time: 12735ms
Time: 12735ms
chunk {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 177 kB {4}
[initial] [rendered]
[...]
chunk {4} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry]
[rendered]
Project build successfully
```

Or using the optional parameter *path*

```
D:\devon-dist>devon devon4ng build -path D:\devon-dist\workspaces\MyDevon4ngApp
```

6.6.3. devon4ng run

In order to launch the *devon4ng* apps devcon provides this *run* command that can be launched even without parameters.

devon4ng run parameters

The only parameter needed is the *clientpath* that points to the client app. This is an optional parameter and if not provided devcon will use by default the directory from within the command is launched.

devon4ng run example of usage

A simple example of usage for this command would be the following

```
D:\devon-dist\workspaces\MyDevon4ngApp>devon devon4ng run
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
Project starting
** NG Live Development Server is listening on localhost:4200, open your browser on
http://localhost:4200 **
** NG Live Development Server is listening on localhost:4200, open your browser on
http://localhost:4200 **
Hash: 7f1a11f3e039fd0028ac
Hash: 7f1a11f3e039fd0028ac
Time: 14333ms
Time: 14333ms
chunk {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 177 kB {4}
[initial]
[...]
chunk {4} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry]
[rendered]
webpack: Compiled successfully.
webpack: Compiled successfully.
```

Or using the optional parameter *clientpath*

```
D:\devon-dist>devon devon4ng run -clientpath D:\devon-dist\workspaces\MyDevon4ngApp
```

In both cases, after launching the command, the app should be available through a web browser in url <http://localhost:4200>.

6.7. project

The *project* module groups the functionalities related to the combined server + client projects.

6.7.1. project create

With this command the user can automate the creation of a combined server and client project (Sencha or devon4ng).

project create requirements

If you want to use a Sencha app as client you will need a github user with permissions to download the [devon4sencha](#) repository.

project create parameters

Basically this command needs the same parameters as the 'subcommands' that is using behind ([devon4j create](#), [devon4ng create](#), [sencha workspace](#) and [sencha create](#))

- **combinedprojectpath:** the path to locate the server and client projects. Is an optional parameter and if not provided by the user devcon will take as the path the directory from which

the command has been launched.

- **servername, packagename, groupid, version:** the parameters related to the Server application. You can get more details in the 'devon4j create' command reference in this document.
- **clienttype:** the type for the client app, you can provide *devon4ng* for Angular based client or *devon4sencha* for Sencha based client.
- **clientname:** the name for the client app.
- **clientpath:** the path to locate the client app. Current directory if not provided.
- **createsenchaWS:** is an optional parameter that indicates if the Sencha workspace needs to be created (by default its value is FALSE).

project create example of usage

A simple example of usage for this command would be the following

```
D:\devon-dist\workspaces\combined>devon project create -servername myServerApp  
-groupid com.capgemini.devonfw -packagename com.capgemini.devonfw.myServerApp -version  
1.0 -clientname myClientApp -clienttype devon4ng  
Hello, this is Devcon!  
Copyright (c) 2016 Capgemini  
serverpath is D:\devon-dist\workspaces\combined\  
[INFO] Scanning for projects...  
[INFO]  
[INFO] -----  
[INFO] Building Maven Stub Project (No POM) 1  
[INFO] -----  
[...]  
  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 6.862 s  
[INFO] Finished at: 2016-08-05T09:23:35+01:00  
[INFO] Final Memory: 10M/43M  
[INFO] -----  
Adding devon.json file...  
Project Creation completed successfully  
Creating client project...  
Creating project myClientApp...  
Adding devon.json file...  
Editing java/pom.xml...  
Project created successfully. Please launch 'npm install' to resolve the project  
dependencies.  
Adding devon.json file to combined project...  
Combined project created successfully.
```

With this example we have created a Server + devon4ng app in the `D:\devon-dist\workspaces\combined` directory. So within this folder we should find:

- `myServerApp` folder with the `devon4j` app.
- `myClientApp` folder with the ``devon4ng`` app.
- the `devon.json` file with the following configuration:

```
{"version": "2.0.1",
"type": "COMBINED",
"projects": ["myServerApp", "myClientApp"]}
```

As you can see the 'projects' property points to the 'subprojects' created. In case we had used the `clientpath` parameter to locate it in a different place that 'project' will reflect it pointing to the client path location:

```
{"version": "2.0.1",
"type": "COMBINED",
"projects": ["myServerApp", "D:\\devon-dist\\otherDirectory\\myClientApp"]}
```

Other possible usages

- `D:\devon-dist\TEST>devon project create -servername sss -groupid com.cap -packagename com.cap.sss -version 1.0 -clientname ccc -clienttype devon4sencha -clientpath D:\devon-dist\TESTB`

Will create a server app (sss) in current directory and a Sencha app in the TESTB directory (that must be a Sencha workspace)

- `D:\devon-dist\TEST>devon project create -servername sss -groupid com.cap -packagename com.cap.sss -version 1.0 -clientname ccc -clienttype devon4sencha -clientpath D:\devon-dist\TESTB -createsenchaws true`

Will create a server app (sss) in current directory and a Sencha workspace with a Sencha app inside in the TESTB directory.

- `D:\devon-dist\TEST>devon project create -servername sss -groupid com.cap -packagename com.cap.sss -version 1.0 -clientname ccc -clienttype devon4sencha`

Will create a server app (sss) and a Sencha workspace with a Sencha app inside, all in current directory.

6.7.2. project build

This command will build both client and server project.

project build requirements

In order to work properly, the command must be launched from within (or pointing to) a Devon distribution (the devon4j project type is defined in a *devon.json* file with parameter 'type' set to 'devon4j' in the server project). The directory from where build command is fired should contain client and server project at same level, and directory should contain a *devon.json* which should have project type as *COMBINED*,and client project should contain a *devon.json* file with parameter 'type' set to 'devon4ng' or 'devon4sencha'.

6.7.3. *project build parameters*

The build command takes three parameters and two of them are mandatory.

- **path** : This is an optional paremaeter. It points to server project workspace and if value of this parameter not given, it takes default value as current directory.
- **clienttype** : This parameter shows which type of client is integrated with server i.e devon4ng or sencha. Its a mandatory one.
- **clientpath** : It should point to client directory i.e where the client code is located. Again a mandatory one.

project build example of usage

A simple example of usage for this command would be the following

```
D:\>devon project build -path D:\FIN_IDE\devon4j-ide-all-2.0.0\samplec -clienttype devon4ng -clientpath D:\FIN_IDE\devon4j-ide-all-2.0.0\clientdoc
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
projectInfo read...
path D:\FIN_IDE\devon4j-ide-all-2.0.0\samplecproject type devon4j
Completed
path D:\FIN_IDE\devon4j-ide-all-2.0.0\clientdocproject type devon4ng
Completed
```

6.7.4. project deploy

This command automates all the process described in the [deployment on tomcat](#) section. It creates a new tomcat server associated to the combined server + client project in the *software* directory of the distribution and launches it to make the project available in a browser.

project deploy requirements

The command automates the packaging of the combined Server + Client project but the user must configure those apps to work properly so you need to varify that:

- The client app *points* to the server app: in Sencha projects the 'server' property of *app/Config.js* or *app/ConfigDevelopment.js_* (depending of the type of build) must point to your server app. In case of devon4ng projects we will need to configure the *baseUrl* property of the 'config.json' file to point to our server.
- The server redirects to the client: in the server project the file ... \serverApp\server\src\main\webapp\index.jsp should redirect to **jsclient** profile .index.jsp

```
<%
    response.sendRedirect(request.getContextPath() + "/jsclient/");
%>
```

- The combined project must have a **devon.json** file defining the type (that must be 'combined') and the subprojects (server and client):

```
{"version": "2.0.1",
"type": "COMBINED",
"projects": ["D:\devon-dist\workspaces\SenchaWorkspace\myClientApp", "myServerApp"]
}
```

In the example above that **devon.json** file defines a server app (*myServerApp*) that is located within the combined project directory (so we do not need to provide a path, only the folder name) and a client app (*myClientApp*) located in a Sencha workspace outside the combined project directory (so we need to provide the path).

- Each 'subprojects' (server and client) must have its corresponding **devon.json** file well formed (the 'type' must be *devon4j* for server and for client apps *devon4ng* or *devon4sencha*).
- The command must be launched from within a valid devonfw distribution.

project deploy parameters

- tomcatpath:** the path to the tomcat folder. Devcon will look for the distribution's Tomcat when this parameter is not provided.
- clienttype:** type of client either angular or Sencha (obtained from 'projects' property in **devon.json** when not given).
- clientpath:** path to client project (obtained from 'projects' property in **devon.json** when not given).
- serverpath:** path to server project (obtained from 'projects' property in **devon.json** when not given).
- path:** path for the combined project (current directory when not given).

project deploy example of usage

A simple example of usage for this command would be the following

```
D:\devon-dist\workspaces\MyCombinedProject>devon project deploy
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
[...]
#####
After Tomcat finishes the loading process the app should be available in:
localhost:8080/myServerApp-server-1.0
#####
```

The process will open a new command window for the Tomcat's launching process and finally will shows us the url where the combined app should be accesible.



The url is formed with the name of the .war file generated when packaging the app.

If we use the optional parameter *path*

```
D:\devon-dist>devon project deploy -path D:\devon-dist\workspaces\MyCombinedProject
```

6.7.5. project run

This command runs the server & client project(unified build) in debug mode that is separate client and spring boot server.

6.7.6. project run requirements

Please verify the *devon4j run* and *devon4ng run* or *sencha run* requirements.

6.7.7. project run parameters

- **clienttype** : This parameter shows which type of client is integrated with server i.e devon4ng or sencha and its a mandatory parameter
- **clienttype** : the type of the client app ('devon4ng' or 'devon4sencha').
- **clientpath** : Location of the devon4ng app.
- **serverport** : Port to start server.
- **serverpath** : Path to Server project Workspace (currentDir if not given).

6.7.8. project run example of usage

A simple example of usage for this command (for client type devon4ng) would be the following

```
D:\>devon project run -clienttype devon4ng -clientpath D:\FIN_IDE\devon4j-ide-all-2.0.0\workspaces\main\examples\devon4ng -serverport 8080 -serverpath D:\FIN_IDE\asp4j-ide-all-2.0.0\workspaces\main\code\devon4j\samples\server
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
path before modification D:\FIN_IDE\devon4j-ide-all-2.0.0\workspaces\main\code\oa
sp4j\samples\server
Server project path D:\FIN_IDE\devon4j-ide-all-2.0.0\workspaces\main\code\devon4j\
samples\server
Application started
Starting application
```

After launching the command, a browser should be opened and will show the welcome page of the devon4ng app.

6.8. sencha

Sencha is a pure JavaScript application framework for building interactive cross platform web applications and is the view layer for web applications developed with Devon Framework. This module encapsulates the *Sencha Cmd* functionality that is a command line tool to automate tasks around *Sencha* apps.

6.8.1. sencha run

This command compiles in DEBUG mode and then runs the internal Sencha web server. Is the equivalent to the *Sencha Cmd*'s `sencha app watch` and does not need any parameter.

sencha run requirements

We should launch the command from a Devon4Sencha project which is defined by a `devon.json` file with parameter 'type' set to 'Devon4Sencha'

```
{ "version": "2.0.0",
  "type": "Devon4Sencha"}
```

sencha run example of usage

A simple example of usage for this command would be the following

```
D:\devon-dist\workspaces\senchaProject>devon sencha run
```

6.8.2. sencha workspace

With this command we can generate automatically a fully functional Sencha workspace in a directory of our machine.

sencha workspace requirements

We will need a Github user with permissions to clone the *devon4sencha* repository.

sencha workspace parameters

The *sencha workspace* command needs five parameters and four of them are mandatory.

- **path:** the location where the workspace should be created. This parameter is optional and if the user does not provide it devcon will take the current directory as the location for the Sencha workspace.
- **username:** the github user with permission to download the *devon4sencha* repository.
- **password:** the password of the github user.
- **proxyHost:** Host parameter for optional Proxy configuration.
- **proxyPort:** Port parameter for optional Proxy configuration.

sencha workspace example of usage

A simple example of usage for this command would be the following

```
D:\>devon sencha workspace -path D:\MyProject -username john -password 1234
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
Cloning into 'D:\MyProject\MySenchaWorkspace'...
Having repository: D:\MyProject\MySenchaWorkspace\.git
```

So after that we will have a sencha workspace located in the *D:\MyProject* directory.

Also we can define, if necessary, a proxy configuration. The following example shows how to configure the connection for Capgemini's proxy in Europe

```
D:\>devon sencha workspace -path D:\MyProject -username john -password 1234 -proxyHost
1.0.5.10 -proxyPort 8080
```

6.8.3. sencha copyworkspace

With this command we can make create new Sencha workspace by making a copy from an existing Devon dist to a particular path

sencha copyworkspace requirements

There should be a devonfw distribution present which included the 'workspaces\examples\devon4sencha' folder

sencha copyworkspace parameters

The *sencha copyworkspace* command needs two parameters. Both are optional.

- **workspace:** the path to the workspace. This parameter is optional. Devcon will take the current directory if not provided and in that case it will use the name 'devon4sencha'.
- **distpath:** the path to a devonfw Dist (Current directory if not provided)

6.8.4. sencha build

This command builds a Sencha Ext JS6 project. Is the equivalent to the *Sencha Cmd's* [sencha app build](#).

sencha build parameters

This command only has one parameter and it is optional

- **appDir:** the path to the app to be built. If the user does not provide it devcon will use the current directory as the location of the Sencha app.

sencha build example of usage

A simple example of usage for this command would be the following

```
D:\MySenchaWorkspace\MyApp>devon sencha build
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
OUTPUT:Sencha Cmd v6.1.2.15
OUTPUT:[INF] Processing Build Descriptor : classic
[...]
[INFO] [LOG] Sencha App Watch Started
[INFO] [LOG]Sencha Build Successful
D:\MySenchaWorkspace\MyApp>
```

And using the optional parameter *appDir* to locate the app the usage would be like the following

```
D:>devon sencha build -appDir D:\MySenchaWorkspace\MyApp
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
OUTPUT:Sencha Cmd v6.1.2.15
OUTPUT:[INF] Processing Build Descriptor : classic
[...]
[INFO] [LOG] Sencha App Watch Started
[INFO] [LOG]Sencha Build Successful
D:>
```

6.8.5. sencha create

This command creates a new Sencha Ext JS6 app.

sencha create requirements

The command must be launched within a Sencha workspace or pointing to a Sencha workspace using the optional parameter *workspacepath*. So in order to work properly first we will need to have a Sencha workspace ready in our local machine.

sencha create parameters

The create parameters handles two parameters

- **appname:** the name for the new app.
- **workspacepath:** optionally the user can specify the location of the Sencha workspace. If the user does not provide it the current directory will be used as default.

sencha create example of usage

A simple example of usage for this command would be the following

```
D:\MySenchaWorkspace>devon sencha create -appname MyNewApp
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
OUTPUT:Sencha Cmd v6.1.2.15
OUTPUT:[INF] Loading framework from D:\MySenchaWorkspace\
[...]
[INFO] [LOG]Sencha Ext JS6 app Created
D:\MySenchaWorkspace>
```

And using the optional parameter *workspacepath* to locate the Sencha workspace the command would be like the following

```
D:\>devon sencha create -appname MyNewApp -workspacepath D:\MySenchaWorkspace
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
OUTPUT:Sencha Cmd v6.1.2.15
OUTPUT:[INF] Loading framework from D:\MySenchaWorkspace\
[...]
[INFO] [LOG]Sencha Ext JS6 app Created
D:\>
```

After that we will have a new Sencha app called *MyNewApp* in our Sencha workspace.

6.9. workspace

This module handles all tasks related to distribution workspaces.

6.9.1. workspace create

This command automates the creation of new workspaces within the distribution with the default configuration including a new Eclipse .bat starter related to the new project.

workspace create parameters

The create command needs two parameters:

- **devonpath**: the path where the devon distribution is located.
- **foldername**: the name for the new workspace.

workspace create example of usage

A simple example of usage for this command would be the following

```
D:\>devon workspace create -devonpath C:\MyFolder\devon-dist -foldername newproject
Hello, this is Devcon!
Copyright (c) 2016 Capgemini
[INFO] creating workspace at path D:\devon2-alpha\workspaces\newproject
[...]
```

As a result of that a new folder *newproject* with the default project configuration should be created in the *C:\MyFolder\devon-dist\workspaces* directory alongside an *eclipse-newproject.bat* starter script in the root of the distribution.

6.10. system

This module contains system wide commands related to devcon.

6.10.1. system install

This command installs devcon on user's HOME directory or at an alternative path provided by user.

It should be used as a very first step to install Devcon, [see more here](#)

```
> java -jar devcon.jar system install
```

If you are behind a proxy you must configure the connection using the optional parameters **-proxyHost** and **-proxyPort**. In following example we show how to use the *system install* command for Capgemini's proxy in Europe

```
> java -jar devcon.jar system install -proxyHost 1.0.5.10 -proxyPort 8080
```

6.10.2. system update

Launching this command the user can update the Devcon version installed to the last version available.

system update example of usage

A simple example of usage for this command would be the following

```
D:\>devon system update
```

As occurs with the *system install* command, if you are behind a proxy you will need to use the optional parameters **-proxyHost** and **-proxyPort** to configure the connection. The following example shows how to configure the *system update* with the Capgemini's proxy in Europe

```
D:\>devon system update -proxyHost 1.0.5.10 -proxyPort 8080
```

Part III: devon4j

7. Introduction

The [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.

This document contains the complete compendium of the [devon4j](#), the Java stack of devonfw. From this link you will also find the latest release or nightly snapshot of this documentation.

8. Architecture

There are many different views on what is summarized by the term *architecture*. First we introduce the [key principles](#) and [architecture principles](#) of the devonfw. Then we go into details of the [the architecture of an application](#).

8.1. Key Principles

For the 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](#).

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 his 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 service or presentation logic. A layer is therefore 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

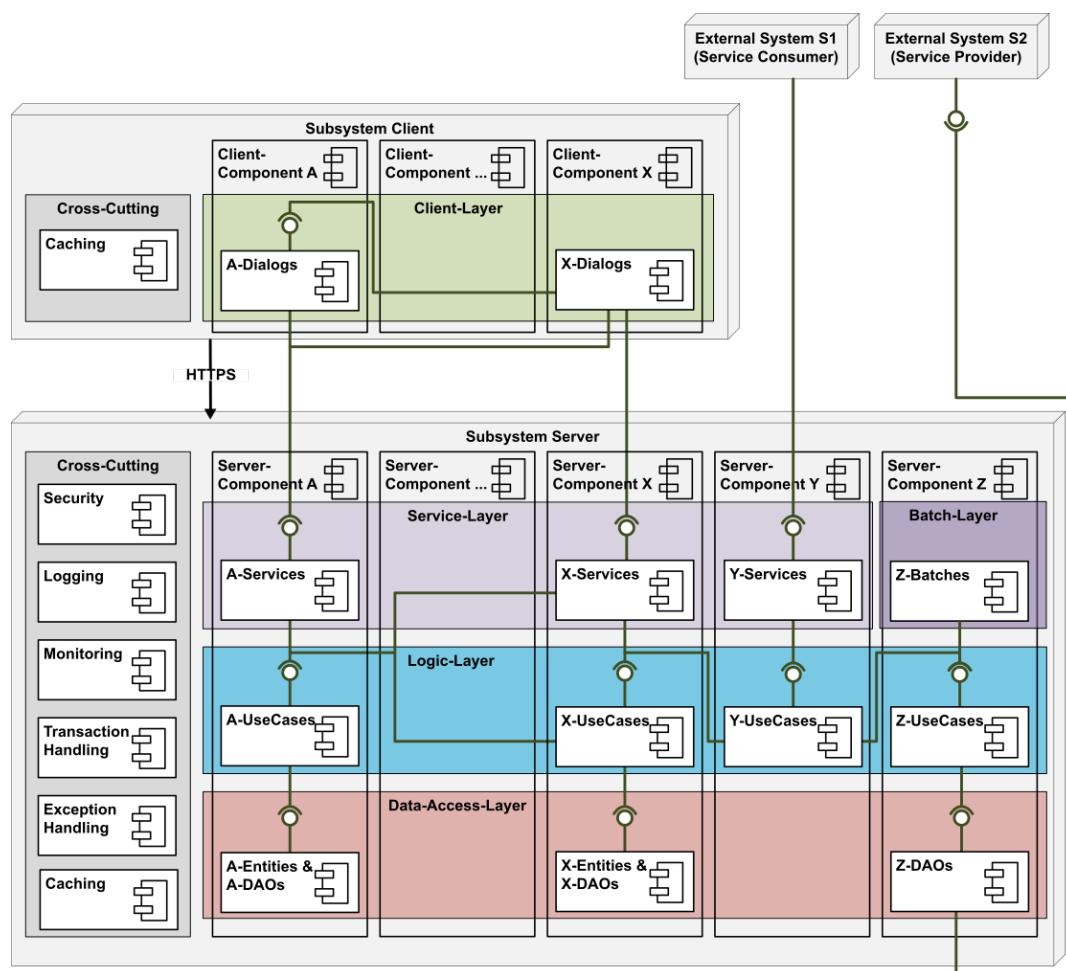


Figure 1. Technical Reference Architecture

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 **X** is allowed to depend on component **A**. Therefore a use-case within the logic component part of **X** is allowed to call a use-case from **A** 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 1. 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
service	SOAP services	JAX-WS	CXF
service	REST services	JAX-RS	CXF

Topic	Detail	Standard	Suggested implementation
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

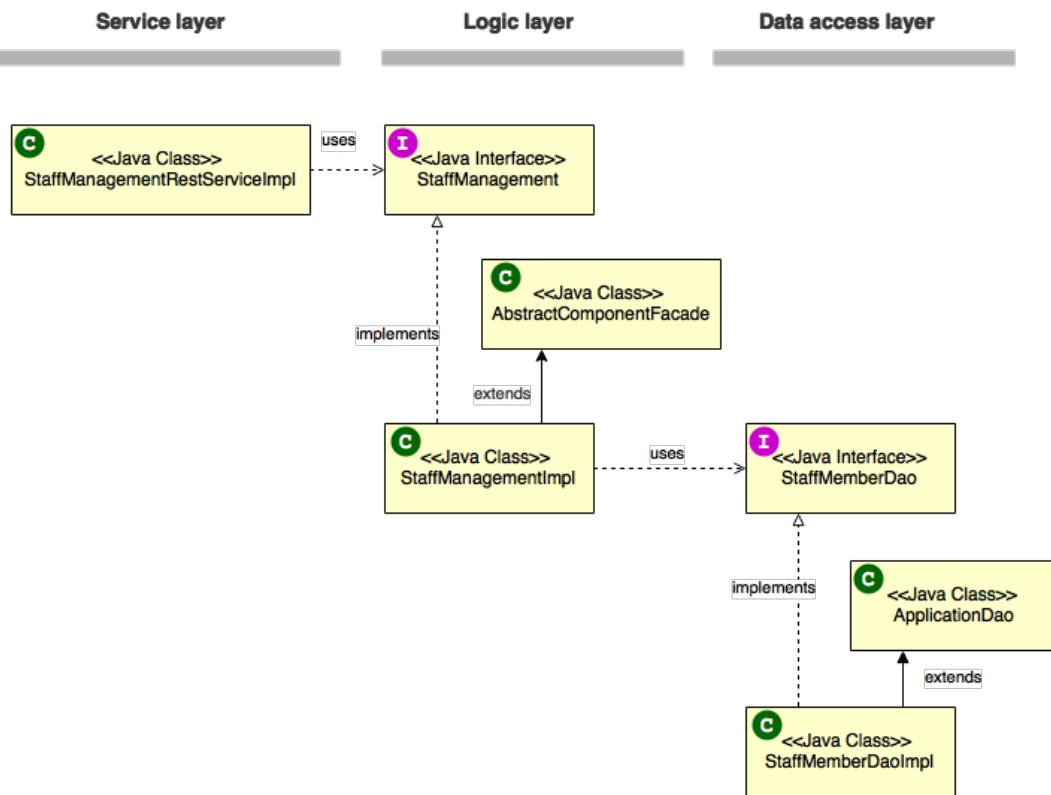
The [business-architecture](#) defines the business components with their allowed dependencies. A small application (microservice) may just have one component and no dependencies making it simple while the same architecture can scale up to large and complex applications (from bigger microservice up to modulith). Tailoring an business domain into applications and applications into components is a tricky task that needs the skills of an experinced architect. Also the tayloring should follow the business and not split by technical reasons or only by size. Size is only an indicator but not a driver of tayloring. Whatever hypes like microservices are telling you, never get misslead 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 other 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`:



Here 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

9.1. 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.

9.1.1. 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`)
- 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.

9.1.2. 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. Further we can use tools such as [SonarQube](#) and [SonarGraph](#) to verify architectural rules.

For an devonfw 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 2. 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`.

9.1.3. 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-devon-plugin](#) to "understand" the code. Also this helps developers going from one devon4j project to the next to quickly understand the codebase. 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 1. Architecture mapped to code

```

<<rootpackage>>.<<application>>
  .<<component>>
    .common
      .api[.<<detail>>]
        .datatype
          .<<Datatype>>
            .<<BusinessObject>>
        .impl[.<<detail>>]
          .<<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
          .UcFind<<BusinessObject>>Impl
          .UcManage<<BusinessObject>>Impl
  
```



9.1.4. 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 do this as part of your change and remove the according tag.

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 <><>
```

A TODO tag is added by the author of the code who is also responsible for completing this task.

FIXME

```
// FIXME <><>
```

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.

REVIEW

```
// REVIEW <>(<>) <>
```

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.

9.1.5. 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.

9.1.6. Code-Style

This section gives you best practices to write better code and avoid pitfalls and mistakes.

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](#).

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 RuntimeIOException(e, IoMode.READ);
}
```

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 RuntimeIOException(e, IoMode.READ);
}
```

Lambdas and Streams

With Java8 you have cool new feautres like lambdas and monads like (`Stream`, `CompletableFuture`, `Optional`, etc.). However, these new features can also be misused or lead 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. Often developers are 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());
}
```

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
```

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 decide 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" what 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)`

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

9.2. Tools

Table 3. Development Tools used for devon4j

Topic	Detail	Suggested Tool
build-management	*	maven
IDE	IDE	Eclipse
IDE	setup & update	devonfw-ide
IDE	code generation	CobiGen
Testing	Unit-Testing	JUnit
Testing	Mocking	Mockito & WireMock
Testing	Integration-Testing	spring-test (arquillian for JEE)
Testing	End-to-end	MrChecker
Quality	Code-Analysis	SonarQube

10. Layers

10.1. 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 backends 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](#).

10.1.1. 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 4. 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.)

10.2. 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](#).

10.2.1. 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 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 5. 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

10.2.2. 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.

10.2.3. 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*.

10.2.4. 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.

10.2.5. Security

Your services are the major entry point to your application. Hence security considerations are important here.

See [REST Security](#).

10.3. 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 devide 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.

10.3.1. Responsibility

The logic layer is responsible for implementation the business logic according to the specified functional demands and requirements. It therefore creates the actual value of the application. The following additional aspects are also in its responsibility:

- [validation](#)
- [authorization](#)
- [transaction-handling](#) (in addition to [service layer](#)).

10.3.2. Security

The logic layer is the heart of the application. It is also responsible for authorization and hence security is important here. 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 at all
- 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 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 depend 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.

10.4. 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 `<<Component>>Impl` (see [implementation](#)).

10.4.1. 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);

}

```

10.4.2. 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.

10.5. 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).

10.5.1. 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);  
  
}
```

10.5.2. 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);  
  
}
```

10.5.3. 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);  
}
```

10.5.4. Implementation

For the implementation of a use-case the same rules apply that are described for the [component-facade implementation](#).

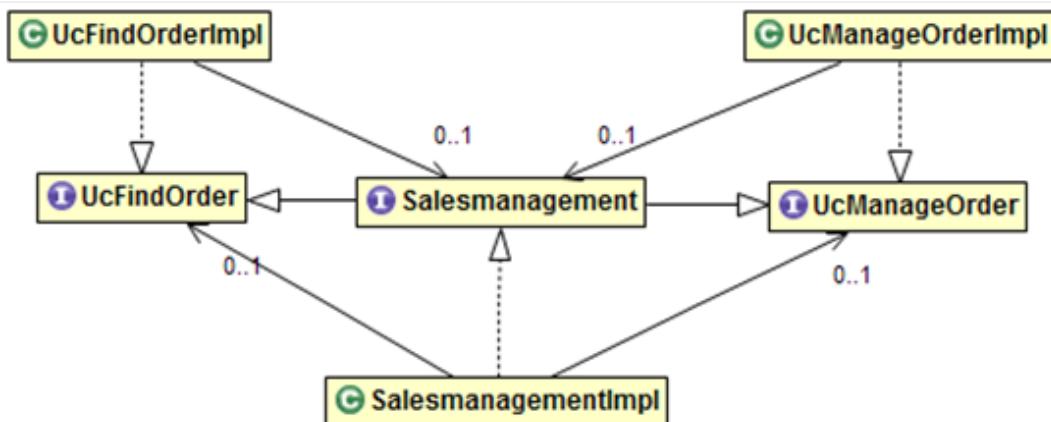
However, when following the use-case approach, your component facade simply changes to:

```
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:



10.5.5. 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 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 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]`.

10.5.6. Injection issues

Technically now you have two implementations of your use-case:

- the direct implementation of the use-case (`Uc*Impl`)
- the component facade implemenation (`<<Component>>Impl`)

When injecting a use-case interface this could cause ambiguities. This is addressed as following:

- In the component facade implemenation (`<<Component>>Impl`) spring is smart enough to resolve the ambiguity as it assumes that a spring bean never wants to inject itself (can already be access via `this`). Therefore only the proper use-case implementation remains as 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 error you need to annotate your according implementation also with `@Lazy` in addition to `@Named`.

10.6. 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.

10.6.1. RDBMS

The classical approach is to use a Relational Database Management System (RDMS). In such case we strongly recommend to follow our [JPA Guide](#). In case you are using Oracle you should also consider the [Oracle guide](#).

10.6.2. NoSQL

In case of specific demands and requirements you may want to choose for a *Not only SQL* database (NoSQL). There are different categories of such products so you should first be aware what fits your requirements best:

- key/value DB
- document DB
- graph DB
- wide-column DB

As there are many such products and the market is evolving very fast, we do not yet give clear recommendations here. If you are doing a devon project and consider NoSQL please contact us for further details.

10.7. Batch Layer

We understand batch processing as 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 batch framework.

This guide explains how Spring Batch is used in devonfw applications. Please note that it is not yet fully consistent concerning batches with the sample application. You should adhere to this guide by

now.

10.7.1. Batch architecture

In this chapter we will describe the overall architecture (especially concerning layering) and how to administer batches.

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 bills. This *billExport* belongs to the salesmanagement business component. So the *billExport* is implemented in the following package:

```
<basepackage>.salesmanagement.batch.impl.billexport.*
```

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.

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 that this pattern is a simplification which could lead to business logic cluttered in different layers which reduces maintainability of your application. It is a typical design decision you have to take when designing your specific batches.

Batch administration and execution

Starting and Stopping Batches

Spring Batch provides a simple command line API for execution and parameterization of batches, the [CommandLineJobRunner](#). It is not yet fully compatible with Spring Boot, however. For those using Spring Boot, devonfw provides the [SpringBootBatchCommandLine](#) with similar functionalities.

Both execute batches as a "simple" standalone process (instantiating a new JVM and creating a new ApplicationContext).

Starting a Batch Job

For starting a batch job, the following parameters are required:

jobPath(s)

The location of the JavaConfig classes (usually annotated with `@Configuration` or `@SpringBootApplication`) and/or XML files that will be used to create an `ApplicationContext`.

The CommandLineJobRunner only accepts one class/file, which must contain everything needed to run a job (potentially by referencing other classes/files), the SpringBootBatchCommandLine, however, expects that there are two paths given: one for the general batch setup and one for the XML file containing the batch job to be executed.

There is an example of a general batch setup for Spring Boot in the [my-thai-star batch module](#). The main class is `SpringBootBatchApp`, which also imports the general configuration class introduced in the chapter on the [general configuration](#). Note that `SpringBootBatchApp` deactivates the evaluation of annotations used for authorization, especially the `@RolesAllowed` annotation. You should of course make sure that only authorized users can start batches, but once the batch is started there is usually no need to check any authorization.

jobName

The name of the job to be run.

All arguments after the job name are considered to be job parameters and must be in the format of `name=value`:

Example for the CommandLineJobRunner:

```
java org.springframework.batch.core.launch.support.CommandLineJobRunner  
classpath:config/app/batch/beans-billexport.xml billExportJob -outputFile=file:out.csv  
date(date)=2015/12/20
```

Example for the SpringBootBatchCommandLine:

```
java com.devonfw.module.batch.common.base.SpringBootBatchCommandLine  
com.devonfw.application.mtsj.SpringBootBatchApp classpath:config/app/batch/beans-  
billexport.xml billExportJob -outputFile=file:out.csv date(date)=2015/12/20
```

The date parameter will be explained in the section on [parameters](#).

Note that when a batch is started with the same parameters as a previous execution of the same batch job, the new execution is considered a restart, see [restarts](#) for further details. Parameters starting with a "-" are ignored when deciding whether an execution is a restart or not (so called non identifying parameters).

When trying to restart a batch that was already complete, there will either be an exception (message: "A job instance already exists and is complete for parameters={…}. If you want to run this job again, change the parameters.") or the batch will simply do nothing (might happen when no or only non identifying parameters are set; in this case the console log contains the following message for every step: "Step already complete or not restartable, so no action to execute: …").

Stopping a Job

The command line option to stop a running execution is as follows:

```
java org.springframework.batch.core.launch.support.CommandLineJobRunner  
classpath:config/app/batch/beans-billexport.xml -stop billExportJob
```

or

```
java com.devonfw.module.batch.common.base.SpringBootBatchCommandLine  
com.devonfw.application.mtsj.SpringBootBatchApp classpath:config/app/batch/beans-  
billexport.xml billExportJob -stop
```

Note that the job is not shutdown immediately, but might actually take some time to stop.

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 batch administration. Likewise your application should not do so.

Batch administration 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.

10.7.2. Implementation

In this chapter we will describe how to properly setup and implement batches.

Main Challenges

At a first glimpse, implementing batches is much like implementing a backend for client processing. There are, however, some points at which batches have to be implemented totally different. This is especially true if large data volumes are to be processed.

The most important points are:

Transaction handling

For processing request made by clients there is usually one transaction for each request. If anything goes wrong, the transaction is rolled back and all changes are reverted.

A naive approach for batches would be to execute a whole batch in one single transaction so that if anything goes wrong, all changes are reverted and the batch could start from scratch. For processing large amounts of data, this is technically not feasible, because the database system would have to be able to undo every action made within this transaction. And the space for storing the undo information needed for this (the so called "undo tablespace") is usually quite limited.

So there is a need of short running transactions. To help programmers to do so, Spring Batch offers the so called chunk processing which will be explained [here](#).

Restarting Batches

In client processing mode, when an exception occurs, the transaction is rolled back and there is no need to worry about data inconsistencies.

This is not true for batches however, due to the fact that you usually can't have just one transaction. When an unexpected error occurs and the batch aborts, the system is in a state where the data is partly processed and partly not and there needs to be some sort of plan on how to continue from there.

Even if a batch was perfectly reliable, there might be errors that are not under the control of the application, e.g. lost connection to the database, so that there is always a need for being able to restart.

The section on [restarts](#) describes how to design a batch that is restartable. What's important is that a programmer has to invest some time upfront for a batch to be able restart after aborts.

Exception handling in Batches

The problem with exception handling is that a single record can cause a whole batch to fail and many records will remain unprocessed. In contrast to this, in client processing mode when processing fails this usually affects only one user.

To prevent this situation, Spring Batch allows to skip data when certain exceptions occur. However, the feature should not be misused in a way that you just skip all exceptions independently of their cause.

So when implementing a batch, you should think about what exceptional situations might occur and how to deal with that and whether it is okay to skip those exceptions or not. When an

unexpected exception occurs, the batch should still fail so that this exception is not ignored but its causes are analyzed.

Another way of handling exceptions in batches is retrying: Simply try to process the data once more and hope that everything works well this time. This approach often works for database problems, e.g. timeouts.

The section on [exception handling](#) explains skipping and retrying in more detail.

Note that exceptions are another reason why you should not execute a whole batch in one transaction. If anything goes wrong, you could either rollback the transaction and start the batch from scratch or you could manually revert all relevant changes. Both are not very good solutions.

Performance issues

In client processing mode, optimizing throughput (and response times) is an important topic as well, of course.

However, a performance that is still considered okay for client processing might be problematic for batches as these usually have to process large volumes of data and the time for their execution is usually quite limited (batches are often executed at night when no one is using the application).

An example: If processing the data of one person takes a second, this is usually still considered OK for client processing (even though performance could be better). However if a batch has to process the data of 100.000 persons in one night and is not executed with multiple threads, this takes roughly 28 hours, which is by far too much.

The section on [performance](#) contains some tips on how to deal with performance problems.

Setup

Database

Spring Batch needs some meta data tables for monitoring batch executions and for restoring state for [restarts](#). Detailed description about needed tables, sequences and indexes can be found in [Spring Batch - Reference Documentation: Appendix B. Meta-Data Schema](#).

It is not recommended to add additional meta data tables, because this easily leads to inconsistencies with what is stored in those tables maintained by Spring Batch. You should rather try to extract all needed information out of the standard tables in case the standard API (especially [JobRepository](#) and [JobExplorer](#), see below) does not fit your needs.

Failure information

`BATCH_JOB_EXECUTION.EXIT_MESSAGE` and `BATCH_STEP_EXECUTION.EXIT_MESSAGE` store a detailed description of how the job exited. In the case of failure, this might include as much of the stack trace as is possible. `BATCH_STEP_EXECUTION_CONTEXT.SHORT_CONTEXT` stores a stringified version of the step's `ExecutionContext` (see [saving and restoring state](#), the rest is stored in a BLOB if needed). The default length of those columns in the sample schema scripts is `2500`.

It is good to increase the length of those columns as far as the database allows it to make it easier to

find out which exception failed a batch (not every exception causes a failure, see [exception handling](#)). Some JDBC drivers cast CLOBs to string automatically. If this is the case, you can use CLOBs instead.

General Configuration

For configuring batches, we recommend not to use annotations (would not work very well for batches) or JavaConfig, but XML, because this makes the whole batch configuration more transparent, as its structure and implementing beans are immediately visible. Moreover the Spring Batch documentation focuses rather on XML based configurations than on JavaConfig.

For explanations on how these XML files are build in general, have a look at the [spring documentation](#).

There is, however, some general configuration needed for all batches, for which we use JavaConfig, as it is also used for the setup of all other layers. You can find an example of such a configuration in the [samples/core](#) project: [BeansBatchConfig](#). In this section, we will explain the most important parts of this class.

The [jobRepository](#) is used to update the meta data tables.

The database type can optionally be set on the [jobRepository](#) for correctly handling database specific things using the [setDatabaseType](#) method. Possible values are oracle, mysql, postgres etc.

If the size of all three columns, which by default have a length limitation of 2500, has been increased as proposed [here](#), the property [maxVarCharLength](#) should be adjusted accordingly using the corresponding setter method in order to actually utilize the additional space.

The [jobExplorer](#) offers methods for reading from the meta data tables in addition to those methods provided by the [jobRepository](#), e.g. getting the last executions of a batch.

The [jobLauncher](#) is used to actually start batches.

We use our own implementation ([JobLauncherWithAdditionalRestartCapabilities](#)) here, which can be found in the module [modules/batch \(devon4j-batch\)](#). It enables a special form of restarting a batch ("restart from scratch", see the section on [restarts](#) for further details).

The [jobRegistry](#) is basically a map, which contains all batch jobs. It is filled by the bean of type [JobRegistryBeanPostProcessor](#) automatically.

A [JobParametersIncremeter](#) (bean [incrementer](#)) can be used to generate unique parameters, see [restarts](#) and [parameters](#) for further details. It should be configured manually for each batch job, see example batch below, otherwise exceptions might occur when starting batches.

Example-Batch

As already mentioned, every batch job consists of one or more batch steps, which internally either use chunk processing or tasklet based processing.

Our bill export batch job consists of the following to steps:

1. Read all (not processed) bills from the database, mark them as processed (additional attribute) and write them into a CSV file (to be further processed by other systems). This step is implemented using chunk processing (see [chunk processing](#)).
2. Delete all bill from the database which are marked as processed. This step is implemented in a tasklet (see [tasklet based processing](#)).

Note that you could also delete the bills directly. However, for being able to demonstrate tasklet based processing, we have created a separate step here.

Also note that in real systems you would usually create a backup of data as important as bills, which is not done here.

The `beans-billexport.xml` configures the batch for exporting the bills.

As you can see, there is a job element (`billExportJob`), which contains the two step elements (`createCsvFile` and `deleteBills`). Note that for every step you have to explicitly specify which step comes next (using the `next` attribute), unless it is the last step.

The step elements always contains a tasklet element, even if chunk processing is used. The `transaction-attributess` element is especially used to set timeout of transactions (in seconds). Note that there is usually more than one transaction per step (see below).

What follows is either a chunk element with `ItemReader`, `ItemProcessor`, `ItemWriter` and a commit interval (see [chunk processing](#)) or the tasklet element containing a reference to a tasklet.

In the example above the `ItemReader` named `unprocessedBillsReader` always reads 1000 ids of unprocessed bills (via a DAO) and returns them one after another. The `ItemProcessor` `processedMarker` reads the corresponding bills from the database (see [chunk processing](#) why we do not read them directly in the `ItemReader`) and marks them as processed. The `ItemWriter` `csvFileWriter` (see below on how this writer is configured) writes them to a CSV file. The path of this file is provided as batch parameter (`outputFile`).

The `tasklet billsDeleter` deletes all processed bills (10.000 in one transaction).

The `chunkLoggingListener`, which is also used in the example above, can be utilized for all chunk steps to log exceptions together with the items where these exceptions occurred (see [listeners](#) for further details on listeners). Its implementation can be found in the module modules/batch. Note that classes used for items have to have an appropriate `toString()` method in order for this listener to be useful.

Restarts

A batch execution is considered a restart, if it was run already (with the same parameters) and there was a (non skippable) failure or the batch has been stopped.

There are basically two ways to do a restart:

- Undo all changes and restart from scratch.
- Restore the state of that batch at the time the error occurred and continue processing.

The first approach has two major disadvantages: One is that depending on what the batch does, reverting all of its changes can get quite complex. And you easily end up having implemented a batch that is restartable, but not if it fails in the wrong step.

The second disadvantage is that if a batch runs for several hours and then it fails it has to start all over again. And as the time for executing batches is usually quite limited, this can be problematic.

If reverting all changes is as easy as deleting all files in a given directory or something like that and the expected duration for an execution of the batch is rather short, you might consider the option of always starting at the beginning, otherwise you shouldn't.

Spring Batch supports implementing the second option. By default, if a batch is restarted with the same parameters as a previous execution of this batch, then this new execution continues processing at the step where the last execution was stopped or failed. If the last execution was already complete, an exception is raised.

The step itself has to be implemented in a way so that it can restore its internal state, which is the main drawback of this second option.

However, there are 'standard implementations' that are capable of doing so and these can easily be adapted to your needs. They are introduced in the section on [chunk processing](#).

For instructing Spring Batch to always restart a batch at the very beginning even though there has been an execution of this batch with the same parameters already, set the `restartable` attribute of the `Job` element to false.

By default, setting this attribute to false means that the batch is not restartable (i.e. it cannot be started with the same parameters once more). It would raise an error if there was attempt to do so, so that it cannot be restarted where it left off.

We use our own `JobLauncher` (`JobLauncherWithAdditionalRestartCapabilities`) as described in the section on the [general configuration](#) to modify this behavior so that those batches are always restarted from the first step on by adding an extra parameter (instead of raising an exception), so that you do not have to take care of that yourself. So don't think of a batch marked with `restartable="false"` as a batch that is not restartable (as most people would probably assume just looking at the attribute) but as a batch that restarts always from the first step on.

Note that if a batch is restartable by restoring its internal state, it might not work correctly if the batch is started with different parameters after it failed, which usually comes down to the same thing as restating it from scratch. So, the batch has to be restarted and completed successfully before executing the next regular 'run'. When scheduling batches, you should make that sure.

Chunk Processing

Chunk processing is item based processing. Items can be bills, persons or whatever needs to be processed. Those items are grouped into chunks of a fixed size and all items within such a chunk are processed in one transaction. There is not one transaction for every single (small) item because there would be too many commits which degrades performance.

All items of a chunk are read by an `ItemReader` (e.g. from a file or from database), processed by an

ItemProcessor (e.g. modified or converted) and written out as a whole by an **ItemWriter** (e.g. to a file or to database).

The size of a chunk is also called commit interval. One has to be careful , while choosing a large chunk size: When a skip or retry occurs for a single item (see [exception handling](#)), the current transaction has to be rolled back and all items of the chunk have to be reprocessed. This is especially a problem when skips and retries occur more often and results in long runtimes.

The most important advantages of chunk processing are:

- good trade-off between size and number of transactions (configurable via commit size)
- transaction timeouts that do not have to be adapted for larger amounts of data that needs to be processed (as there is always one transaction for a fixed number of items)
- an exception handling that is more fine-grained than aborting/restarting the whole batch (item based skipping and retrying, see [exception handling](#))
- logging items where exceptions occurred (which makes failure analysis much more easy)

Note that you could actually achieve similar results using [tasklets](#) as described below. However, you would have to write many lines of additional code whereas you get these advantages out of the box using chunk processing (logging exceptions and items where these exceptions occurred is an extension, see [example batch](#)).

Also note that items should not be too "big". For example, one might consider processing all bills within one month as one item. However, doing so you would not have those advantages any more. For instance, you would have larger transactions, as there are usually quite a lot of bills per month or payment method and if an exception occurs, you would not know which bill actually caused the exception. Additionally you would lose control of commit size, since one commit would process many bills hard coded and you cannot choose smaller chunks.

Nevertheless, there are sometimes, situations where you cannot further "divide" items, e.g. when these are needed for one single call to an external system (e.g. for creating a PDF of all bills within a certain month, if PDFs are created by an external system). In this case you should do as much of the processing as possible on the basis of "small" items and then add an extra step to do what cannot be done based on these "small" items.

ItemReader

A reader has to implement the **ItemReader** interface, which has the following method:

```
public T read() throws Exception;
```

T is a type parameter of the **ItemReader** interface to be replaced with the type of items to be read.

The method returns all items (one at a time) that need to be processed or null if there are no more items.

If an exception occurs during read, Spring Batch cannot tell which item caused the exception (as it has not been read yet). That is why a reader should contain as little processing logic as possible,

minimizing the potential for failures.

Caching

By default, all items read by an [ItemReader](#) are cached by Spring Batch. This is useful because when a skippable exception occurs during processing of a chunk, all items (or at least those, that did not cause the exception) have to be reprocessed. These items are not read twice but taken from the cache then.

This is often necessary, because if a reader saves its current state in member variables (e.g. the current position within a list of items) or uses some sort of cursor, these will be updated already and the next calls of the read method would deliver the next items ready and not those that have to be reprocessed.

However this also means that when the items read by an [ItemReader](#) are entities, these might be detached, because these might have been read in a different transaction. In some standard implementations Spring Batch even manually detaches entities in [ItemReaders](#).

In case these entities are to be modified it is a good practice that the [ItemReader](#) only reads IDs and the [ItemProcessor](#) loads the entities for these IDs to avoid the problem.

Reading from Transactional Queues

In case the reader reads from a transactional queue (e.g. using JMS), you must not use caching, because then an item might get processed twice: Once from cache and once from queue to where it has been returned after the rollback. To achieve this, set `reader-transactional-queue="true"` in the chunk element in the step definition.

Moreover the `equals` and `hashCode` methods of the class used for items have to be appropriately implemented for Spring Batch to be able to identify items that were processed before unsuccessfully (causing a rollback and thereby returning them to the queue). Otherwise the batch might be caught in an infinite loop trying to process the same item over and over again (e.g. when the item is about to be skipped, see [exception handling](#)).

Reading from the Database

When selecting data from a database, there is usually some sort of cursor used. One challenge is to make this cursor not participate in the chunk's transaction, because it would be closed after the first chunk.

We will show how to use JDBC based cursors for [ItemReader](#) implementations in later releases of this documentation.

For JPA/JPQL based queries, cursors cannot be used, because JPA does not know of the concept of a cursor. Instead it supports pagination as introduced in the chapter on the data access layer, which can be used for this purpose as well. Note that pagination requires the result set to be sorted in an unambiguous order to work reliably. The order itself is irrelevant as long as it does not change (you can e.g. sort the entities by their primary key).

An [ItemReader](#) using pagination should inherit from the [AbstractPagingItemReader](#), which already

provides most of the needed functionality. It manages the internal state, i.e. the current position, which can be correctly restored after a restart (when using an unambiguous order for the result set).

Classes inheriting from `AbstractPagingItemReader` must implement two methods.

The method `doReadPage()` performs the actual read of a page. The result is not returned (return type is void) but used to replace the content of the 'results' instance variable (type: List).

Due to our layering concept and the persistence layer being the only place where access to the database should take place, you should not directly execute a query in this method, but call a DAO, which itself executes the query (using pagination).

`AbstractPagingItemReader` provides methods for finding out the current position: use `getPage()` for the current page and `getPageSize()` for the (max.) page size. These values should be passed to the DAO as parameters. Note that the `AbstractPagingItemReader` starts counting pages from zero, whereas the `PaginationTo` used for pagination (retrieved by calling `SearchCriteriaTo.getPagination()`) starts counting from one, which is why you always have to increment the page number by one.

The second method is `doJumpToPage(int)`, which usually only requires an empty implementation.

Furthermore, you need to set the property `pageSize`, which specifies how many items should be read at once. A page size that is as big as the commit interval usually results in the best performance.

The approach of using pagination for `ItemReader` should not be used when items (usually entities) are added or removed or modified by the batch step itself or in parallel with the execution of the batch step so that the order changes, e.g. by other batches or due to operations started by clients (i.e. if the batch is executed in online mode). In this case there might be items processed twice or not processed at all. Be aware that due to hibernate's Hi/Lo-Algorithm newer entities could get lower IDs than existing IDs and you probably will not process all entities if you rely on strict ID monotony!

A simple solution for such scenarios would be to introduce a new flag 'processed' for the entities read if that is an option (as it is also done in the example batch). The query should be rewritten then so that only unprocessed items are read (additionally limiting the result set size to the number of items to be processed in the current chunk, but not more).

Note that most of the standard implementations provided by Spring Batch do not fit to the layering approach in devonfw applications, as these mostly require direct access to an `EntityManager` or a JDBC connection for example. You should think twice when using them and not break the layering concept.

Reading from Files

For reading simply structured files, e.g. for those in which every line corresponds to an item to be processed by the batch, the `FlatFileItemReader` can be used. It requires two properties to be set: The first one the `LineMapper` (property `lineMapper`), which is used to convert a line (i.e. a String) to an item. It is a very simple interface which will not be discussed in more detail here. The second one is

the resource, which is actually the file to be read. When set in the XML, it is sufficient to specify the path with a "file:" in front of it if it is a normal file from the file system.

In addition to that, the property `linesToSkip` (integer) can be set to skip headers for example. For reading more than one line before for creating an item, a `RecordSeparatorPolicy` can be used, which will not be discussed in more detail here, too. By default, all lines starting with a '#' will be considered to be a comment, which can be changed by changing the comment property (string array). The encoding property can be used to set the encoding. A `FlatFileItemReader` can restore its state after restarts.

For reading XML files, you can use the `StaxEventItemReader` (StAX is an alternative to DOM and SAX), which will not be discussed in further detail here.

In case the standard implementations introduced here do not fit your needs, you will need to implement your own `ItemReader`. If this `ItemReader` has some internal state (usually stored in member variables), which needs to be restored in case of restarts, see the section on [saving and restoring state](#) for information on how to do this.

ItemProcessor

A processor must implement the `ItemProcessor` interface, which has the following method:

```
public O process(I item) throws Exception;
```

As you can see, there are two type parameters involved: one for the type of items received from the `ItemReader` and one for the type of items passed to the `ItemWriter`. These can be the same.

If an item has been selected by the `ItemReader`, but there is no need to further process this item (i.e. it should not be passed to the `ItemWriter`), the `ItemProcessor` can return null instead of an item.

Strictly interpreting chunk processing, the `ItemProcessor` should not modify anything but should only give instructions to the `ItemWriter` on how to do modifications. For entities however this is not really practical and as it requires no special logic in case of rollbacks/restarts (as all modifications are transactional), it is usually OK to modify them directly.

In contrast to this, performing accesses to files or calling external systems should only be done in `ItemReader/ItemWriter` and the code needed for properly handling failures (restarts for example) should be encapsulated there.

It is usually a good practice to make `ItemProcessor` implementations stateless, as the process method might be called more than once for one item (see the section on `ItemReader` why). If your ItemProcessor really needs to have some internal state, see [saving and restoring state](#) on how to save and restore the state for restarts.

Do not forget to implement use cases instead of implementing everything directly in the ItemProcessor if the processing logic gets more complex.

ItemWriter

A writer has to implement the ItemWriter interface, which has the following method:

```
public void write(List<? extends T> items) Exception;
```

This method is called at the end of each chunk with a list of all (processed) items. It is not called once for every item, because it is often more efficient doing 'bulk writes', e.g. when writing to files.

Note that this method might also be called more than once for one item (see the section on ItemReader's why).

At the end of the write method, there should always be a flush.

When writing to files, this should be obvious, because when a chunks completes, it is expected that all changes are already there in case of restarts, which is not true if these changes were only buffered but have not been written out.

When modifying the database, the flush method on the EntityManager should be called, too (via a DAO), because there might be changes not written out yet and therefore constraints were not checked yet. This can be problematic, because Spring Batch considers all exceptions that occur during commit as critical, which is why these exceptions cannot be skipped. You should be careful using deferred constraints for the same reason.

Writing to Database or Transactional Queues

All changes made which are transactional can be conducted directly, there is no special logic needed for restarts, because these changes are applied if and only if the chunk succeeds.

Writing to Files

For writing simply structured files, the FlatFileItemWriter can be used. Similar to the FlatFileItemReader it requires the resource (i.e. the file) and a LineAggregator (property lineAggregator instead of the lineMapper) to be set.

There are various properties that can be used of which we will only present the most important ones here. As with the FlatFileItemReader, the encoding property is used to set the encoding. A FlatFileHeaderCallback (property headerCallback) can be used to write a header.

The FlatFileItemWriter can restore its state correctly after restarts. In case, the files contain too many lines (written out in chunks that did not complete successfully), these lines are removed before continuing execution.

For writing XML files, you can use the StaxEventItemWriter, which will not be discussed in further detail here.

Just as with ItemReader and ItemProcessor: In case your ItemWriter has some internal state this state is not managed by a standard implementation, see saving and restoring state on how to make your implementation restartable (restart by restoring the internal state).

Saving and Restoring State

For saving and restoring (in case of restarts) state, e.g. saving and restoring values of member variables, the `ItemStream` interface should be implemented by the `ItemReader`/`ItemProcessor` /`ItemWriter`, which has the following methods:

```
public void open(ExecutionContext executionContext) throws ItemStreamException;
public void update(ExecutionContext executionContext) throws ItemStreamException;
public void close() throws ItemStreamException;
```

The `open` method is always called before the actual processing starts for the current step and can be used to restore state when restarting.

The `ExecutionContext` passed in as parameter is basically a map to be used to retrieve values set before the failure. The method `containsKey(String)` can be used to check if a value for a given key is set. If it is not set, this might be because the current batch execution is no restart or no value has been set before the failure.

There are several getter methods for actually retrieving a value for a given key: `get(String)` for objects (must be serializable), `getInt(String)`, `getLong(String)`, `getDouble(String)` and `getString(String)`. These values will be the same as after the subsequent call to the `update` method after the last chunk that completed successfully. Note that if you update the `ExecutionContext` outside of the `update` method (e.g. in the `read` method of an `ItemReader`), it might contain values set in chunks that did not finish successfully after restarts, which is why you should not do that.

So the `update` method is the right place to update the current state. It is called after each chunk (and before and after each step).

For setting values, there are several put methods: `put(String, Object)`, `putInt(String, int)`, `putLong(String, long)`, `putDouble(String, double)` and `putString(String, String)`. You can choose keys (`String`) freely as long as these are unique within the current step.

Note that when a skip occurs, the `update` method is sometimes but not always called, so you should design your code in a way that it can deal with both situations.

The `close` method is usually not needed.

Do not misuse the `ItemStream` interface for purposes other than storing/restoring state. For instance, do not use the `update` method for flushing, because you will not have the chance to properly handle failure (e.g. skipping). For opening or closing a file handle, you should rather use a `StepExecutionListener` as introduced in the section on [listeners](#). The state can also be restored in the `beforeStep(ExecutionListener)` method (instead of the `open` method).

Note that when a batch that always starts from scratch (i.e. the `restartable` attribute has been set to `false` for the batch job) is restarted, the `ExecutionContext` will not contain any state from the previous (failed) execution, so there is no use in storing the state in this case and usually no need to, of course, because the batch will start all over again.

Tasklet based Processing

Tasklets are the alternative to chunk processing. In the section on [chunk processing](#) we already mentioned the advantages of chunk processing as compared to tasklets. However, if only very few data needs to be processed (within one transaction) or if you need to do some sort of bulk operation (e.g. deleting all records from a database table), where the currently processed item does not matter and it is unlikely that a 'fine grained' exception handling will be needed, tasklets might still be considered an option. Note that for the latter use case you should still use more than one transaction, which is possible when using tasklets, too.

Tasklets have to implement the interface with the same name, which has the following method:

```
public RepeatStatus execute(StepContribution contribution, ChunkContext chunkContext)
throws Exception;
```

This method might be called several times. Every call is executed inside a new transaction automatically. If processing is not finished yet and the execute method should be called once more, just use `RepeatStatus.CONTINUABLE` as return value and `RepeatStatus.FINISHED` otherwise.

The `StepContribution` parameter can be used to set how many items have been processed manually (which is done automatically using chunk processing), there is, however, usually no need to do so.

The `ChunkContext` is similar to the `ExecutionContext`, but is only used within one chunk. If there is a retry in chunk processing, the same context should be used (with the same state that this context had when the exception occurred).

Note that tasklets serve as the basis for chunk processing internally. For chunk processing there is a Spring Batch internal tasklet, which has an execute method that is called for every chunk and itself calls `ItemReader`, `ItemProcessor` and `ItemWriter`.

That is the reason why a `StepContribution` and a `ChunkContext` are passed to tasklets as parameters, even though they are more useful in chunk processing. Moreover this is also the reason why you have to use the tasklet element in the XML even though you want to specify a step that uses chunk processing (see [the example batch](#)).

Exception Handling

As already mentioned, in chunk processing you can configure a step so that items are skipped or retried when certain exceptions occur.

If retries are exhausted (by default, there is no retry) and the exception that occurred cannot be skipped (by default, no exception can be skipped), the batch will fail (i.e. stop executing).

In tasklet based processing this cannot be done, the only chance is to implement the needed logic yourself.

Skipping

Before skipping items you should think about what to do if a skip occurs. If a skip occurs, the exception will be logged in the server log. However if no one evaluates those logs on a regular basis

and informs those who are affected further actions need to take place when implementing the batch.

Implement the `SkipListener` interface to be informed when a skip occurs. For example, you could store a notification or send a message to someone. For skips that occurred in `ItemReader`'s there is no information available about the item that was skipped (as it has not been read yet) which is why there should be as little processing logic as possible in an `ItemReader`. It might also be a reason why you might want to forbid to skip exceptions that might occur in readers.

Do not try to catch skipped exceptions and write something into the database in a new transaction (e.g. a notification) instead of using a `SkipListener`, because a skipped item might be processed more than once before actually being skipped (for example, if a skippable exception is thrown during a call of an `ItemWriter`, Spring Batch does not know which item of the current chunk actually caused the exception and therefore has to retry each item separately in order to know which item actually caused the exception).

Skippable exception classes can be specified as shown below:

```
<batch:chunk ... skip-limit="10">
    <batch:skippable-exception-classes>
        <batch:include class="..."/>
        <batch:include class="..."/>
        ...
    </batch:skippable-exception-classes>
</batch:chunk>
```

The attribute `skip-limit`, which has to be set in case there is any skippable exception class configured, is used to set how many items should be skipped at most. It is useful to avoid situations where many items are skipped but the batch still completes successfully and no one notices this situation.

Skippable exception classes are specified by their fully qualified name (e.g. `java.lang.Exception`), each of such class set in its own `include` element as shown above. Subclasses of such classes are also skipped.

To programmatically decide whether to skip an exception or not, you can set a skip policy as shown below:

```
<batch:chunk ... skip-policy="mySkipPolicy">
```

The skip policy (here `mySkipPolicy`) has to be a bean that implements the interface `SkipPolicy` with the following method:

```
public boolean shouldSkip(java.lang.Throwable t,
                           int skipCount)
                           throws SkipLimitExceededException
```

To skip the exception and continue processing, just return true and otherwise false.

The parameter `skipCount` can be used for a skip limit. A `SkipLimitExceeded` should be thrown if there should be no more skips. Note that this method is sometimes called with a `skipCount` less than zero to test if an exception is skippable in general.

When a `SkipPolicy` is set, the attribute `skip-limit` and element `skippable-exception-classes` are ignored.

You could of course skip every exception (using `java.lang.Exception` as skippable exception class). This is, however, not a good practice as it might easily result in an error in the code that is ignored as the batch still completes successfully and everything seems to be fine. Instead, you should think about what kind of exceptions might actually occur, what to do if they occur and if it is OK to skip them. If an unexpected exception occurs, it is usually better to fail the batch execution and analyze the cause of the exception before restarting the batch.

Exceptions that can occur in instances of `ItemWriter` that write something to file should not be skipped unless the `ItemWriter` can properly deal with that. Otherwise there might be data written out even though the according item is skipped, because operations in the file systems are not transactional.

Another situation where skips can be problematic is when calls to external interfaces are being made and these calls change something "on the other side", as these calls are usually not transactional. So be careful using skips here, too.

Retrying

For some types of exceptions, processing should be retried independently of whether the exception can be skipped or would otherwise fail the batch execution.

For example, if there was a database timeout, this might be because there were too many requests at the time the chunk was processed. And it is not unlikely that retrying to successfully complete the chunk would succeed.

There are, of course, also exceptions where retrying does not make much sense. E.g. exceptions caused by the business logic should be deterministic and therefore retrying does not make much sense in this case.

Nevertheless, retrying every exception results in longer runtime but should in general be considered OK if you do not know which exceptions might occur or do not have the time to think about it.

Retryable exception classes can be set similarly to setting skippable exception classes:

```
<batch:chunk ... retry-limit="3">
    <batch:retryable-exception-classes>
        <batch:include class="..."/>
        <batch:include class="..."/>
        ...
    </batch:retryable-exception-classes>
</batch:chunk>
```

The `retry-limit` attribute specifies how many times one individual item can be retried, as long as the exception thrown is "retryable".

As with skippable exception classes, retryable exception classes are set in include elements and their subclasses are retried, too.

To programmatically decide, whether to retry an exception or not, you can use a [RetryPolicy](#), which is not covered in more detail here.

Note that even if no retry is configured, an item might nevertheless be processed more than once. This is because if a skippable exception occurs in a chunk, all items of the chunk that did not cause the exception have to reprocessed, which is done in a separate transaction for every item, as the transaction in which these items were processed in the first place was rolled back. And even if the exception is not skippable, there is no guarantee that Spring Batch will not attempt to reprocess each item separately.

Listeners

Spring Batch provides various listeners for various events to be notified about.

For every listener there is an interface which can either be implemented by an ItemReader, ItemProcessor, ItemWriter or Tasklet or by a separate listener class, which can be registered for a step like this:

```
<batch:tasklet>
    <batch:chunk .../>
    <batch:listeners>
        <batch:listener ref="listener1"/>
        <batch:listener ref="listener2"/>
        ...
    </batch:listeners>
</batch:tasklet>
<beans:bean id="listener1" class="..."/>
<beans:bean id="listener2" class="..."/>
...
```

The most commonly use listener is probably the [StepExecutionListener](#), which has methods that are called before and after the execution of the step. It can be utilized e.g. for opening and closing files.

The following example shows how to use the listener:

```
public class MyListener implements StepExecutionListener {  
  
    public void beforeStep(StepExecution stepExecution) {  
        // take actions before processing of the step starts  
    }  
  
    public ExitStatus afterStep(StepExecution stepExecution) {  
        try {  
            // take actions after processing is finished  
        } catch (Exception e) {  
            stepExecution.addFailureException(e);  
            stepExecution.setStatus(BatchStatus.FAILED);  
            return ExitStatus.FAILED.addExitDescription(e);  
        }  
        return null;  
    }  
}
```

In the `afterStep(StepExecution)` method, you can check the outcome of the batch execution (completed, failed, stopped etc.) checking the `ExitStatus`, which can be accessed via `StepExecution.getExitStatus()`. You can even modify the `ExitStatus` by returning a new `ExitStatus`, which is something we will not discuss in further detail here. If you do not want to modify the `ExitStatus`, just return null.

Throwing an exception in this method has no effect. If you want to fail the whole batch in case an exception occurs, you have to do an exception handling as shown above. This does not apply to the `beforeStep` method.

For other types of listeners (among others the `SkipListener` mentioned already) see [Spring Batch Reference Documentation - 5. Configuring a Step - Intercepting Step Execution](#).

Note that exception handling for listeners is often a problem, because exceptions are mostly ignored, which is not always documented very well. If an important part of a batch is implemented in listener methods, you should always test what happens when exceptions occur. Or you might think about not implementing important things in listeners ...

If you want an exception to fail the whole batch, you can always wrap it in a `FatalStepExecutionException`, which will stop the execution.

Parameters

The section on [starting and stopping batches](#) already showed how to start a batch with parameters.

One way to get access to the values set is using the `StepExecutionListener` introduced in the section on [listeners](#) like this:

```
public void beforeStep(StepExecution stepExecution) {  
  
    String parameterValue = stepExecution.getJobExecution().getJobParameters().  
        getString("parameterKey");  
}
```

There are getter methods for strings, doubles, longs and dates. Note that when set via the `CommandLineJobRunner` or `SpringBootBatchCommandLine`, all parameters will be of type string unless the type is specified in brackets after the parameter key, e.g. `processUntil(date)=2015/12/31`. The parameter key here is `processUntil`.

Another way is to inject values. In order for this to work, the bean has to have step scope, which means there is a new object created for every execution of a batch step. It works like this:

```
<bean id="myProcessor" class="...MyItemProcessor" scope="step">  
    <property name="parameter" value="#{jobParameters['parameterKey']}" />  
<bean>
```

There has to be an appropriate setter method for the parameter of course.

As already mentioned in the section on `restarts`, a batch that successfully completed with a certain set of parameters cannot be started once more with the same parameters as this would be considered a restart, which is not necessary, because the batch was already finished.

So using no parameters for a batch would mean that it can be started until it completes successfully once, which usually does not make much sense.

As batches are usually not executed more than once a day, we propose introducing a general `date` parameter (without time) for all batch executions.

It is advisable to add the date parameter automatically in the `JobLauncher` if it has not been set manually, which can be done as shown below:

```
private static final String DATE_PARAMETER = "date";  
  
...  
  
if (jobParameters.getDate("DATE_PARAMETER") == null) {  
  
    Date dateWithoutTime = new Date();  
    Calendar cal = Calendar.getInstance();  
    cal.setTime(dateWithoutTime);  
    cal.set(Calendar.HOUR_OF_DAY, 0);  
    cal.set(Calendar.MINUTE, 0);  
    cal.set(Calendar.SECOND, 0);  
    cal.set(Calendar.MILLISECOND, 0);  
    dateWithoutTime = cal.getTime();  
  
    jobParameters = new JobParametersBuilder(jobParameters).addDate(  
        DATE_PARAMETER, dateWithoutTime).toJobParameters();  
  
    ... // using the jobParametersIncrementer as shown above  
}
```

Keep in mind that you might need to set the date parameter explicitly for restarts. Also note that automatically setting the date parameter can be problematic if a batch is sometimes started before and sometimes after midnight, which might result in a batch not being executed (as it has already been executed with the same parameters), so at least for productive systems you should always set it explicitly.

The date parameters can also be useful for controlling the business logic, e.g. a batch can process all data that was created until the current date (as set in the date parameter), thereby giving a chance to control how much is actually processed.

If your batch has to run more than once a day you could easily adapt the concept of timestamps. If you are using an external batch scheduler, they often provide a counter for the execution and you might automatically pass this instead of the date parameter.

Performance Tuning

Most important for performance are of course the algorithms that you write and how fast (and scalable) these are, which is the same as for client processing. Apart from that, the performance of batches is usually closely related to the performance of the database system.

If you are retrieving information from the database, you can have one complex query executed in the `ItemReader` (via a DAO) retrieving all the information needed for the current set of items, or you can execute further queries in the `ItemProcessor` (or `ItemWriter`) on a per item basis to retrieve further information.

The first approach is usually by far more performant, because there is an overhead for every query being executed and this approach results in less queries being executed. Note that there is a tradeoff between performance and maintainability here. If you put everything into the query

executed by an [ItemReader](#), this query can get quite complex.

Using cursors instead of pagination as described in the section on [ItemReaders](#) can result in a better performance for the same reason: When using a cursor, the query is only executed once, when using pagination, the query is usually executed once per chunk. You could of course manually cache items, however this easily leads to a high memory consumption.

Further possibilities for optimizations are query (plan) optimization and adding missing database indexes.

Testing

The Section [Testing](#) covers how to unit and integration test in detail. Therefore we focus here on testing batches.

In order for the unit test to run a batch job the unit test class must extend the [AbstractSpringBatchIntegrationTest](#) class. Annotation used to load the job's [ApplicationContext](#):

```
@SpringBootTest(classes = {…}): Indicates which JavaConfig classes (attribute classes)  
@ImportResource("classpath:../sample_BatchContext.xml") : Indicates XML files that contain the  
'ApplicationContext'. Use @ContextConfiguration(…) if Spring Boot is not used.
```

```
public abstract class AbstractSpringBatchIntegrationTest extends AbstractComponentTest  
{…}
```

```
@SpringBootTest(classes = { SpringBootBatchApp.class }, webEnvironment =  
WebEnvironment.RANDOM_PORT)  
@ImportResource("classpath:config/app/batch/beans-productimport.xml")  
@EnableAutoConfiguration  
public class ProductImportJobTest extends AbstractSpringBatchIntegrationTest {..}
```

Testing Batch Jobs

For testing the complete run of a batch job from beginning to end involves following steps:

- set up a test condition
- execute the job
- verify the end result.

The test method below begins by setting up the database with test data. The test then launches the Job using the [launchJob\(\)](#) method. The [launchJob\(\)](#) method is provided by the [JobLauncherTestUtils](#) class.

Also provided by the utils class is [launchJob\(JobParameters\)](#), which allows the test to give particular parameters. The [launchJob\(\)](#) method returns the [JobExecution](#) object which is useful for asserting particular information about the Job run. In the case below, the test verifies that the Job ended with [ExitStatus COMPLETED](#).

```
@SpringBootTest(classes = { SpringBootBatchApp.class }, webEnvironment = WebEnvironment.RANDOM_PORT)
@ImportResource("classpath:config/app/batch/beans-productimport.xml")
@EnableAutoConfiguration
public class ProductImportJobTest extends AbstractSpringBatchIntegrationTest {

    @Inject
    private Job productImportJob;

    @Test
    public void testJob() throws Exception {
        .....
        .....
        JobExecution jobExecution = getJobLauncherTestUtils(this.productImportJob)
            .launchJob(jobParameters);
        assertThat(jobExecution.getStatus()).isEqualTo(BatchStatus.COMPLETED);
        .....
        .....
    }
}
```

Note that when using the `launchJob()` method, the batch execution will never be considered as a restart (i.e. it will always start from scratch). This is achieved by adding a unique (random) parameter.

This is not true for the method `launchJob(JobParameters)` however, which will result in an exception if the test is executed twice or a batch is executed in two different tests with the same parameters.

We will add methods for appropriately handling this situation in future releases of devonfw. Until then you can help yourself by using the method `getUniqueJobParameters()` and then add all required parameters to those parameters returned by the method (as shown in the section on [parameters](#)).

Also note that even if skips occurred, the BatchStatus is still COMPLETED. That is one reason why you should always check whether the batch did what it was supposed to do or not.

Testing Individual Steps

For complex batch jobs individual steps can be tested. For example to test a `createCsvFile`, run just that particular Step. This approach allows for more targeted tests by allowing the test to set up data for just that step and to validate its results directly.

```
JobExecution jobExecution = getJobLauncherTestUtils(this.billExportJob).launchStep(
    "createCsvFile");
```

Validating Output Files

When a batch job writes to the database, it is easy to query the database to verify the output. To facilitate the verification of output files Spring Batch provides the class [AssertFile](#). The method

`assertFileEquals` takes two File objects and asserts, line by line, that the two files have the same content. Therefore, it is possible to create a file with the expected output and to compare it to the actual result:

```
private static final String EXPECTED_FILE = "classpath:expected.csv";
private static final String OUTPUT_FILE = "file:./temp/output.csv";
AssertFile.assertFileEquals(new FileSystemResource(EXPECTED_FILE), new
FileSystemResource(OUTPUT_FILE));
```

Testing Restarts

Simulating an exception at an arbitrary method in the code can be done relatively easy using [AspectJ](#). Afterwards you should restart the batch and check if the outcome is still correct.

Note that when using the `launchJob()` method, the batch is always started from the beginning (as already mentioned). Use the `launchJob(JobParameters)` instead with the same parameters for the initial (failing) execution and for the restart.

Test your code thoroughly. There should be at least one restart test for every step of the batch job.

11. Guides

11.1. 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.

11.1.1. 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.

11.1.2. 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.

11.1.3. Bean configuration

Wiring and Bean configuration can be found in [configuration guide](#).

11.2. 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

11.2.1. 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.

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.

XML-based beans configuration

It is still possible and allowed to provide (bean-) configurations using XML, though not recommended. These configuration files are no more bundled via a main xml config file but loaded individually from their respective owners, e.g. for unit-tests:

```
@SpringApplicationConfiguration(classes = { SpringBootApp.class }, locations = {  
    "classpath:/config/app/batch/beans-productimport.xml" })  
public class ProductImportJobTest extends AbstractSpringBatchIntegrationTest {  
    ...  
}
```

Configuration XML-files reside in an adequately named subfolder of:

`src/main/resources/app`

Batch configuration

In the directory `src/main/resources/config/app/batch` we place the configuration for the batch jobs. Each file within this directory represents one batch job. See [batch-layer](#) for further details.

BeanMapper Configuration

In the directory `src/main/resources/config/app/common` we place the configuration for the bean-mapping. See [bean-mapper configuration](#) for further details.

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 2. 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`.

11.2.2. 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

The business configuration contains all business configuration values of the application, which can be edited by administrators through the GUI. The business configuration values are stored in the database in key/value pairs.

The database table `business_configuration` has the following columns:

- ID
- Property name
- Property type (Boolean, Integer, String)
- Property value
- Description

According to the entries in this table, the administrative GUI shows a generic form to change business configuration. The hierarchy of the properties determines the place in the GUI, so the GUI bundles properties from the same hierarchy level and name. Boolean values are shown as checkboxes, integer and string values as text fields. The properties are read and saved in a typed form, 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`

11.2.3. 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).

11.3. 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 artefacts.

11.3.1. 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 annoation (e.g. `MappedSuperclass` tough 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 refactorings (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`.

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).

11.3.2. 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 the 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 the devon 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`. IMPORTANT: 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`, `@ManyToMany`). 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.

11.3.3. 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.

11.3.4. 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.

11.3.5. 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 [GenericDao.forceIncrementModificationCounter](#).

- 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.

11.3.6. Database Auditing

See [auditing guide](#).

11.3.7. Testing Entities and DAOs

See [testing guide](#).

11.3.8. 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.

11.3.9. 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](#).

11.3.10. 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.

11.3.11. 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\(...\)](#)):

```

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.

11.3.12. Spring-Data

If you are using the springframework 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 artefacts like XML descriptors, [NamedQueries](#) 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 [NamedQueries](#) 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.

11.3.13. Data Access Object

The *Data Acccess 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.

11.3.14. 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's in order to pre-load the nested 'N child entities** into the first level cache of the JPA vendor implementation. This will behave as if the **@ManyToMany** relation to **IngredientEntity** was having **FetchType.EAGER** but only for that 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.

11.4. 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 =  
    GenericRevisionedRepositoryFactoryBean.class)  
...  
public class SpringBootApp {  
    ...  
}
```

Now let your [Entity]Repository extend from DefaultRevisionedRepository 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
);

```

11.5. 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`.

11.5.1. Batches

Transaction control for batches is a lot more complicated and is described in the [batch layer](#).

11.6. SQL

For general guides on dealing or avoiding SQL, preventing SQL-injection, etc. you should study [data-access layer](#).

11.6.1. 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 CamlCase (e.g. `RestaurantOrder`)
- Column names in camlCase (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](#).

11.7. 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.

11.7.1. 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 overweights this disadvantage. An even more fine-grained production directory structure which contains one subfolder 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.

11.7.2. 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.

11.7.3. 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_....

11.8. SAP HANA

This section contains hints for those who use [SAP HANA](#), a very powerful and fast RDBMS. If you have chosen a different persistence technology on purpose you can simply ignore this guide. Besides general hints about the driver there are tips for more tight integration with other SAP features or products.

11.8.1. Driver

The hana JDBC driver is available in Maven Central what makes your life very easy. 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>
```

The variable `hana.driver.version` may be [2.3.55](#), but check yourself at <http://central.maven.org/maven2/com/sap/cloud/db/jdbc/ngdbc/> for the proper or most recent version.

11.8.2. 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](#).

11.8.3. Pooling

TODO

11.8.4. Fuzzy Search

See <https://blogs.sap.com/2015/08/28/dynamism-of-fuzzy-search-in-sap-hana/> or the [SAP HANA Search Developer Guide](#)

11.9. Oracle RDBMS

This section contains hints for those who use Oracle RDBMS. If you use a different persistence technology you can simply ignore it. 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.

11.9.1. 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](#). (In case the build of the docker-image [fails reproducably](#) and you want to give up with the Dockerfiles from Oracle you can also try this unofficial [docker-oracle-xe](#) solution).

To connect to your local XE database you need to use `xe` as the [SID](#) of your main database that can not be changed. The `hostname` should be `localhost` and the `port` is by default `1521` if you did not remap it with docker to something else. However, 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 with [SQL Developer](#) switch `Connection Type` from `Basic` to `Advanced` and enter the `Custom JDBC URL` like e.g.

```
jdbc:oracle:thin:@//localhost:1521/xepdb1
```

The same way you can also connect from your devon4j app via JDBC.

11.9.2. Driver

The oracle JDBC driver is not available in maven central. Depending on the Oracle DB version and the Java version, you can use either the 11g/ojdbc6, 12c/ojdbc7, or 12c/ojdbc8 version of the driver. Oracle JDBC drivers usually are backward and forward compatible so you should be able to use the 12c/ojdbc8 driver with an 11g DB etc. As a rule of thumb, use the 12c/ojdbc8 driver unless you must use Java7. All JDBC drivers can be downloaded without registration: [11g/ojdbc6](#), [12c/ojdbc7](#), and [12c/ojdbc8](#). Your project should use a maven repository server such as [nexus](#) or [artifactory](#). Your dependency for the oracle driver should look as follows (use artifactId "ojdbc6" or "ojdbc7" for the older drivers):

```
<dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ojdbc8</artifactId>
  <version>${oracle.driver.version}</version>
</dependency>
```

oracle.driver.version being 11.2.0.4 for 11g/ojdbc6, or 12.1.0.1 for 12c/ojdbc7, or 12.2.0.1 for 12c/ojdbc8 or newer

11.9.3. 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](#). The 11g version of UCP can be downloaded without registration [here](#), the 12c version of UCP is available at the same download locations as the 12c JDBC driver (see above). As a rule of thumb, use the version that is the same as the JDBC driver version. Again, you have to upload the artefact manually to your maven repository. The dependency should look like this:

```
<dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ucp</artifactId>
  <version>${oracle.ucp.version}</version>
</dependency>
```

with oracle.ucp.version being 11.2.0.4 or 12.2.0.1 or newer.

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.

11.9.4. 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.

11.9.5. 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.

11.10. 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.

11.10.1. 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.

11.11. 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.

11.11.1. 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 3. 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 6. 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

Log-level	Description	Impact	Active Environments
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.

11.11.2. 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 7. 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...]
```

11.11.3. Security

In order to prevent [log forging](#) attacks we provide a special appender for logback in [devonfw-logging](#). If you use it (see [configuration](#)) you are safe from such attacks.

11.11.4. 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.

11.11.5. 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](#).

11.12. Security

Security is todays 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.

11.12.1. 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 8. Security threats and protection-mechanisms

Threat	Protection	Link to details
A1 Injection	validate input, escape output, use proper frameworks	SQL Injection
A2 Broken Authentication and Session Management	encrypt all channels, use a central identity management with strong password-policy	Authentication
A3 XSS	prevent injection (see A1) for HTML, JavaScript and CSS and understand same-origin-policy	client-layer
A4 Insecure Direct Object References	Using direct object references (IDs) only with appropriate authorization	logic-layer
A5 Security Misconfiguration	Use devonfw application template and guides to avoid	application template and sensitive configuration
A6 Sensitive Data Exposure	Use secured exception facade, design your data model accordingly	REST exception handling

Threat	Protection	Link to details
A7 Missing Function Level Access Control	Ensure proper authorization for all use-cases, use <code>@DenyAll</code> as default to enforce	Method authorization
A8 CSRF	secure mutable service operations with an explicit CSRF security token sent in HTTP header and verified on the server	service-layer security
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 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.
Log-Forging	Escape newlines in log messages	logging security

11.12.2. Tools

Dependency Check

To address [A9 Using Components with Known Vulnerabilities](#) we integrated [OWASP dependency check](#) into the devonfw maven build. If you build an devonfw 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 always be generated when the site is build (`mvn site`).

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 marvellous presentation [Toolbox of a security professional](#) from [Christian Schneider](#).

11.13. 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?)

11.13.1. 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.

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](#).

11.13.2. 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 9. 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 10. 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 devonfw provides a ready to use module `devon4j-security` that is based on `spring-security` and makes your life a lot easier.

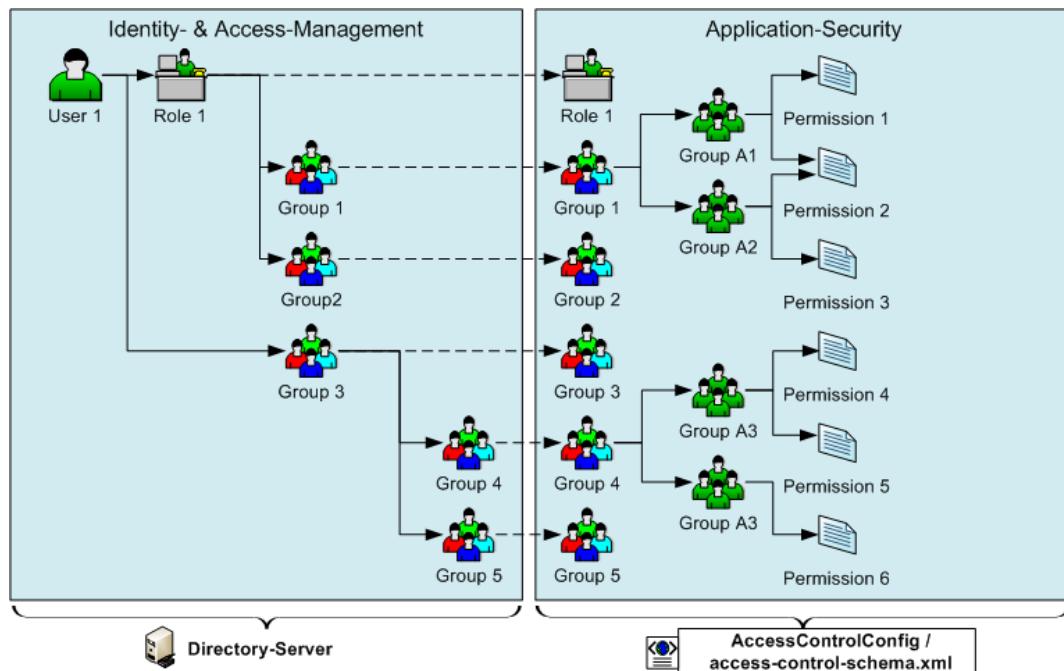


Figure 2. devon4j Security Model

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](#).

11.14. 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.

11.14.1. 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.

11.14.2. 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.

11.14.3. 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);
}
```

11.14.4. 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?

11.15. 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.

11.15.1. 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 4. 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. Therefor 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() ...
}
```

11.15.2. 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.

11.16. 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.

11.16.1. 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).

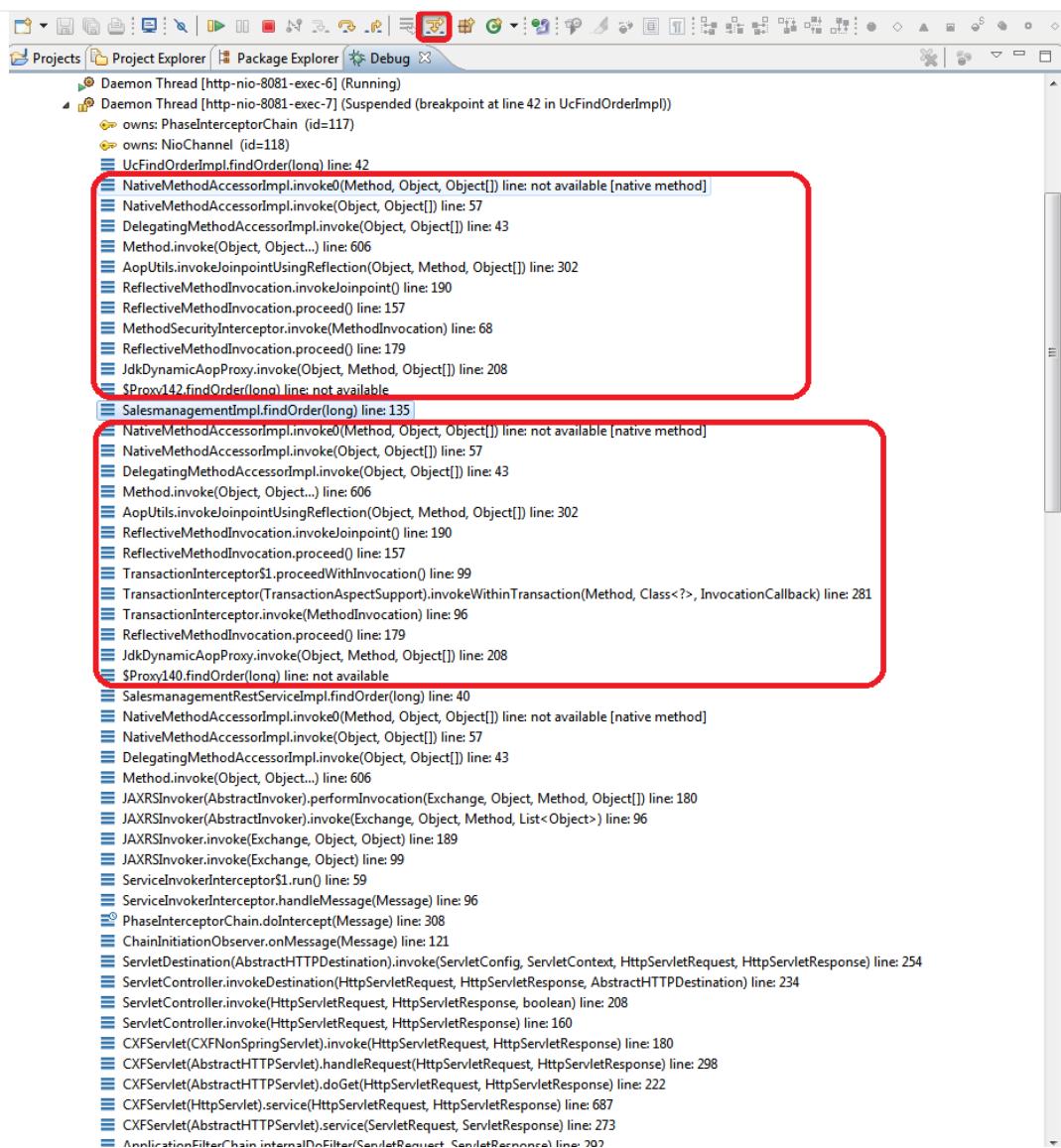
11.16.2. AOP Usage

We recommend to use AOP with care but we consider it established for the following cross cutting concerns:

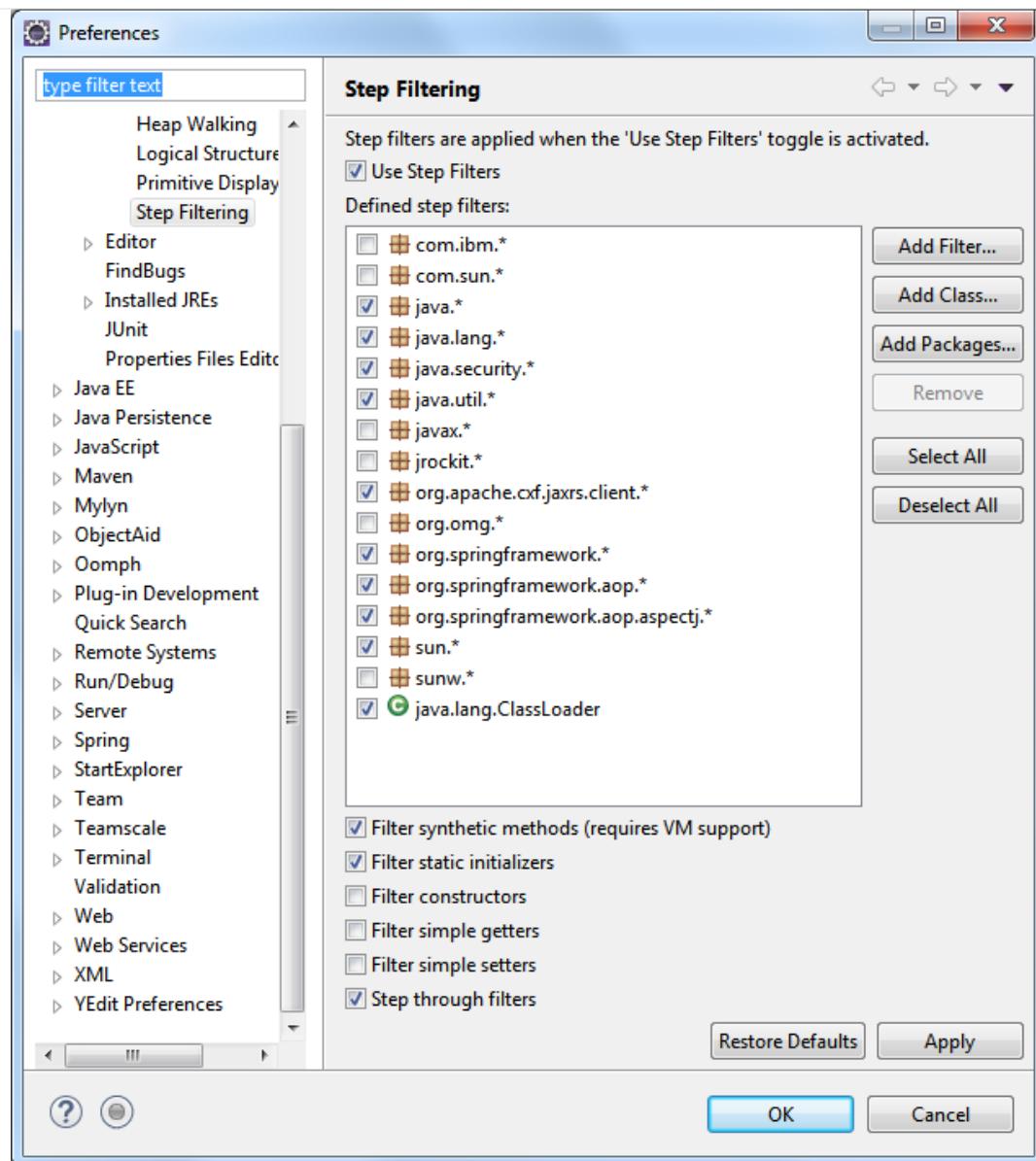
- [Transaction-Handling](#)
- [Authorization](#)
- [Validation](#)
- [Trace-Logging](#) (for testing and debugging)
- Exception facades for [services](#) but only if no other solution is possible (use alternatives such as [JAX-RS provider](#) instead).

11.16.3. 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.*
```

11.17. Exception Handling

11.17.1. 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`)
- 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.

11.17.2. 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();
}

```

11.17.3. 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 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)
 - 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.

11.17.4. Common Errors

The following errors may occur in any devon application:

Table 11. 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

11.18. Internationalization

Internationalization (I18N) is about writing code independent from locale-specific informations. 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:

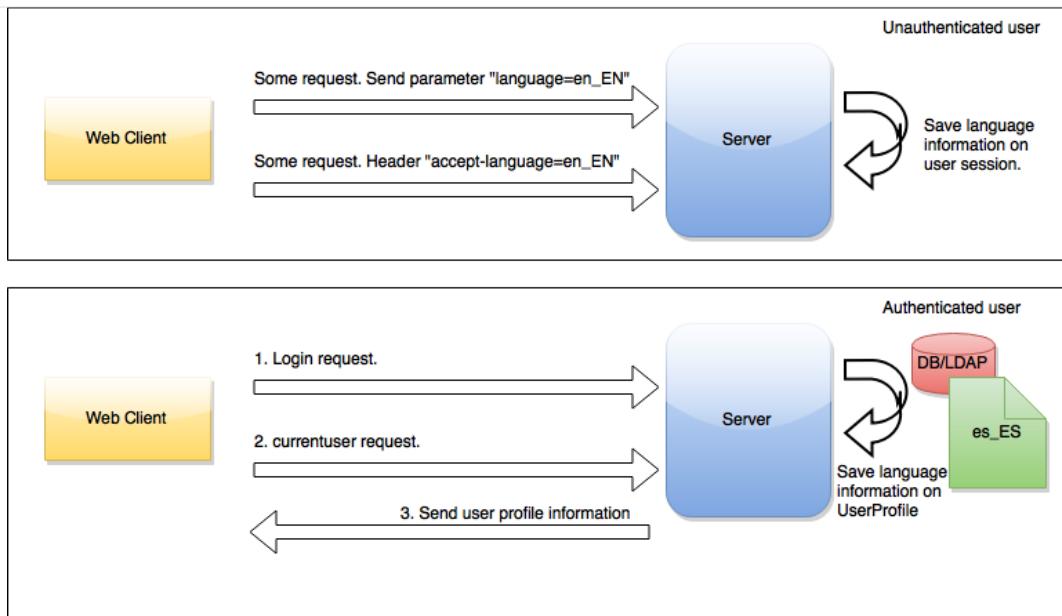
- Bind locale information to the user.
- Get the messages in the current user locale.

11.18.1. 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:



11.18.2. 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

```
waiter=Waiter
chief=Chief
cook=Cook
barkeeper=Barkeeper
```

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");
```

11.19. 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.

11.19.1. JAXB

We use [JAXB](#) to serialize Java objects to XML or vice-versa.

JAXB and Inheritance

TODO @XmlSeeAlso <http://stackoverflow.com/questions/7499735/jaxb-how-to-create-xml-from-polymorphic-classes>

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 dedicated for the XML mapping you can easily avoid such situations. When this is not suitable follow these instructions to define the mapping: TODO

https://weblogs.java.net/blog/kohsuke/archive/2005/09/using_jaxb_20s.html

11.20. 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](#).

11.20.1. 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 ObjectMapper {

    public RestaurantObjectMapperFactory() {
        super();
        // JSON configuration code goes here
    }
}
```

11.20.2. 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:

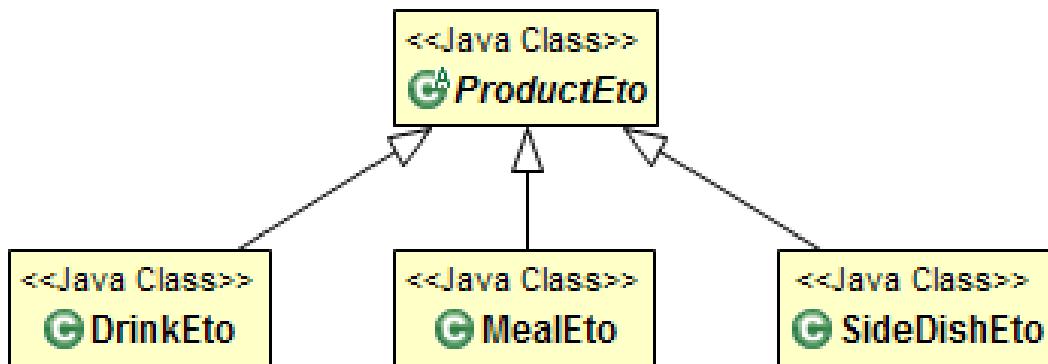


Figure 3. Transfer-Objects using Inheritance

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`.

11.20.3. 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:

1. 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`.

2. 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.

- 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.

11.21. 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](#)).

11.21.1. 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/`).

11.21.2. 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 12. Usage of HTTP methods

HTTP Method	RESTful Meaning	devonfw
<code>GET</code>	Read single element. Search on an entity (with parametrized url)	Read a single element.
<code>PUT</code>	Replace entity data. Replace entire collection (typically not supported)	Not used

HTTP Method	RESTful Meaning	devonfw
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 use **POST** instead of **DELETE** as according to the HTTP standard **DELETE** must not have payload and URLs are limited in length.

11.21.3. 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 13. 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

HTTP Code	Meaning	Response	Comment
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)

11.21.4. 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	Server-side validation is documented in the Validation guide . The protocol to communicate these validation errors to the client is worked on at https://github.com/oasp/oasp4j/issues/218
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 .

11.21.5. 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:

```
com.devonfw.application.mtsj.dishmanagement.service.impl.rest
```

```
@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.

11.21.6. 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.

11.21.7. 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:** [/products/123](#)

For loading of a single resource, embed the [identifier](#) of the resource in the URL (for example [/products/123](#)).

The response contains the resource in JSON format, using a JSON object at the top-level, for example:

```
{  
  "name": "Steak",  
  "color": "brown"  
}
```

Unparameterized loading of a collection of resources

- **HTTP Method:** [GET](#)
- **URL example:** [/products](#)

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": [  
    {  
      "name": "Steak",  
      "color": "brown"  
    },  
    {  
      "name": "Broccoli",  
      "color": "green"  
    }  
  ]  
}
```

Saving a resource

- **HTTP Method:** `POST`
- **URL example:** `/products`

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, an empty HTTP 204 response is generated.

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:** `/products/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": [
    {
      "name": "Steak",
      "color": "brown"
    },
    {
      "name": "Broccoli",
      "color": "green"
    }
  ]
}
```

Compare the code needed on server side to accept this request:
 com.devonfw.application.mtsj.dishmanagement.service.api.rest

```
@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> findCategoriesByPost( @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:** `/products/123`

For deletion of a single resource, embed the `identifier` of the resource in the URL (for example `/products/123`).

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"  
        ],  
        // ....  
    }  
}
```

11.21.8. 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.

11.21.9. REST Testing

For testing REST services in general consult the [testing guide](#).

For manual testing REST services there are browser plugins:

- Firefox: [httprequester](#) (or [poster](#))
- Chrome: [postman](#) ([advanced-rest-client](#))

11.21.10. 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. In devon4j we are using spring-security that already solves CSRF token generation and verification. The integration is part of the application template as well as the sample-application.

For testing in development environment the CSRF protection can be disabled using the JVM option [-DCsrfDisabled=true](#) when starting the application.

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).

11.22. 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).

11.22.1. 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 TablemanagmentWebService {

    private Tablemanagement tableManagement;

    @Override
    public TableEto getTable(String id) {

        return this.tableManagement.findTable(id);
    }
}
```

11.22.2. 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](#)).

11.22.3. SOAP Testing

For testing SOAP services in general consult the [testing guide](#).

For testing SOAP services manually we strongly recommend [SoapUI](#).

11.23. 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.

11.23.1. 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. Have a look at the following features to get an idea why you want to use the solution offered here.

11.23.2. Requirements

You need to add (at least one of) these dependencies to your application:

```
<!-- Starter for consuming REST services -->
<dependency>
    <groupId>com.devonfw.java.starters</groupId>
    <artifactId>devon4j-starter-cxf-client-rest</artifactId>
</dependency>
<!-- Starter for consuming SOAP services -->
<dependency>
    <groupId>com.devonfw.java.starters</groupId>
    <artifactId>devon4j-starter-cxf-client-ws</artifactId>
</dependency>
```

11.23.3. 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);  
}
```

Then all you need to do is this:

```
@Named  
public class UcMyUseCaseImpl extends MyUseCaseBase implements UcMyUseCase {  
    @Inject  
    private ServiceClientFactory serviceClientFactory;  
  
    ...  
    private MyResult callMyServiceMethod(MyArgs myArgs) {  
        MyService myService = this.serviceClientFactory.create(MyService.class);  
        MyResult myResult = myService.myMethod(myArgs); // synchronous call of service  
        over the wire  
        return myResult;  
    }  
}
```

As you can see the synchronous invocation of a service is very simple. 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`).

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();
```



TODO add configuration properties for external/mock service from Sneha

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 (see above). So you can simply add this to your `application.properties`:

```
service.client.app.foo.url=https://foo.company.com:8443/services/rest  
service.client.app.bar.url=http://bar.company.com:8080/services/rest  
service.client.default.url=https://api.company.com/services/rest
```

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.

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 (mainly for testing).
- [ServiceHeaderCustomizerOAuth](#) for OAuth (passes a security token from security context such as a [JWT](#) via OAuth).
- [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.

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.

Asynchronous Support

An important aspect is also asynchronous (and reactive) support. So far we only propose this as an enhancement for a future release with an API like this:

```

public interface ServiceClientAsyncFactory extends ServiceClientFactory {
    AsyncClient<S> createAsync(Class<S> serviceInterface);
}

public interface AsyncClient<S> {
    S getClient();
    <R> Mono<R> call(R result);
    <T> Flux<T> call(Collection<? extends T> result);
    <R> void call(R result, Consumer<R> callback);
    <R> CompletableFuture<R> callFuture(R result);
}

```

This API would allow typesafe usage like this:

```

@Named
public class UcMyUseCaseImpl extends MyUseCaseBase implements UcMyUseCase {
    @Inject private ServiceClientFactoryAsync clientFactory;

    @Override @RolesAllowed(...)
    public void doSomething(Bar bar) {
        AsyncClient<MyExternalServiceApi> client = this.clientFactory.createAsync
        (MyExternalServiceApi.class);
        Mono<Some> result = client.call(client.getService().doSomething(convert(bar)));
        // client.call(client.getService().doSomething(convert(bar)), x ->
        processSync(x));
        return process(result);
    }
}

```

How can this work? The ServiceClientAsyncFactory implementation would create its own dynamic proxy for the given service interface. That proxy would only track the last call that was invoked internally and always return a dummy result (`null` for Object types, `false` for boolean, `0` for primitive numbers). The actual implementation of the `call` methods can access the internal invocation that has been recorded from the last service call. It will then trigger the actual service call internally according to the desired style (using a `Consumer` callback, `Mono`, `Flux`, `Future`...).

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.).

Various implementations of such sidecar proxy apps are available as free open source software.
TODO: Decide for the best available solution and suggest here as default.

- 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](#)

11.24. Testing

11.24.1. 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.

11.24.2. 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](#).

11.24.3. 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)));
    ...
}

```

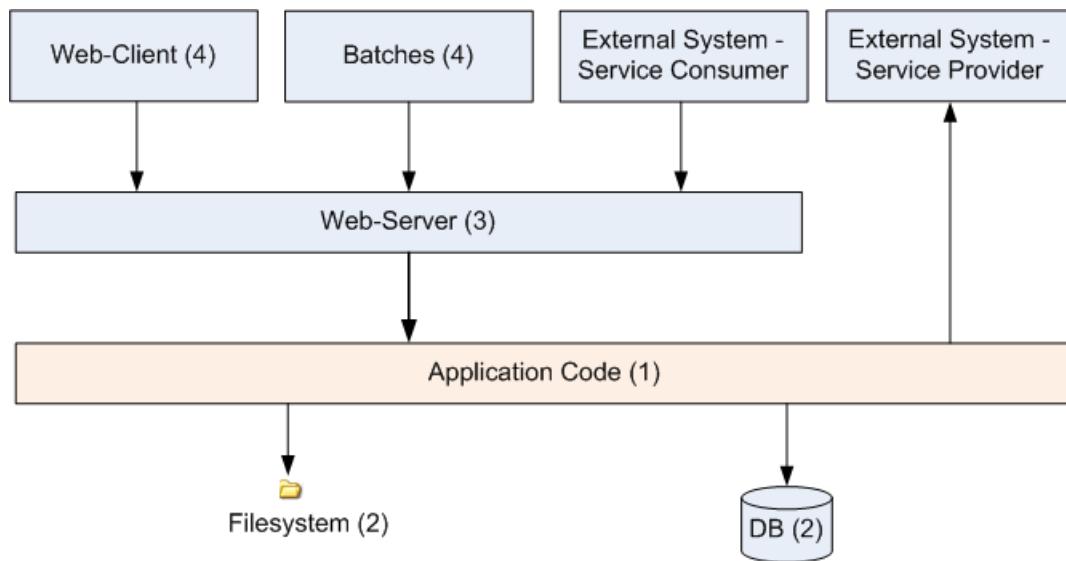
A GET request to `../fault` returns an OK status header, then garbage, and then closes the connection.

11.24.4. 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.

11.24.5. 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.

TODO

As an example let us go to the class `Tablemanagement`. When testing the method `deleteTable()` there are several scenarios that can happen and thus should be covered by tests.

First let us see the valid conditions to delete a table:

- One needs permission to delete a table `PermissionConstants.DELETE_TABLE`
- The table to delete needs to exist (the table with the given id has to be in the database) and
- The table to delete is required to be `TableState.FREE`

Invalid conditions are: No credentials, table does not exist or table is not free. If you combine one invalid condition with valid conditions this yields the following test cases. Note that not working actions yield exceptions that can be expected in a test method.

- The caller of the method does not have the required credentials

```
@Test(expected = AccessDeniedException.class)
public void testDeleteTableWithoutCredentials() {...}
```

- The caller has the required credentials but the table to be deleted is occupied

```
@Test(expected = IllegalEntityStateException.class)
public void testDeleteTableWithCredentialsButNotDeletable() {...}
```

- The caller has the required credentials but the table to be deleted does not exist

```
@Test(expected = ObjectNotFoundException.class)
public void testDeleteTableWithCredentialsNotExisting() {...}
```

- The caller has the required credentials and the table to be deleted exists and is free

```
@Test
public void testDeleteTableWithCredentials() {...}
```

This type of testing is known as [equivalence class analysis](#). Note that this is a general practice and can be applied to every level of tests.

11.24.6. 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.

11.24.7. 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.

11.24.8. 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.

11.24.9. 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.

11.25. 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](#)

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.

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.

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.

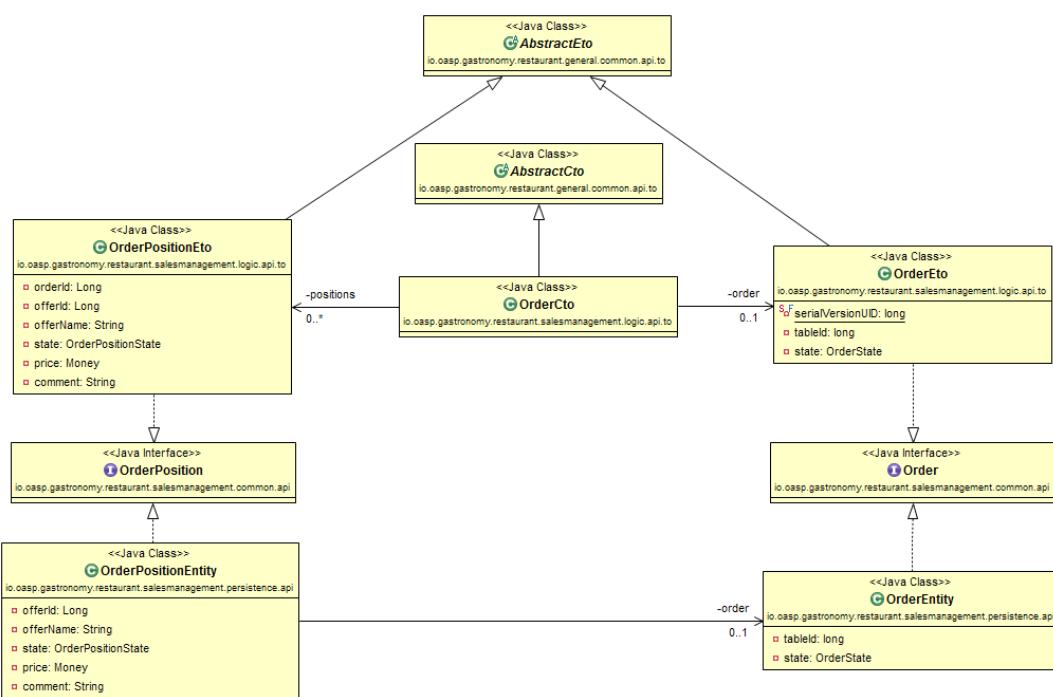


Figure 4. ETOs and CTOs

11.25.4. TO

Finally, there are typically transfer-objects for data that is never persistent. For very generic cases these just carry the suffix **To**.

11.25.5. SearchCriteriaTo

For searching we create or generate a '«BusinessObject»SearchCriteriaTo' representing a query to find instances of '«BusinessObject»'.

11.25.6. 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. In such case you also need `beanmapping` between STOs and ETOs what means extra effort and complexity that should be avoided.

11.26. 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.

11.26.1. Bean-Mapper Dependency

To get access to the `BeanMapper` we use this dependency in our POM:

```
<dependency>
    <groupId>com.devonfw.java</groupId>
    <artifactId>devon4j-beanmapping</artifactId>
</dependency>
```

11.26.2. Bean-Mapper Configuration

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>
```

11.26.3. 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.

11.27. 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](#).

11.27.1. 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.

Segment	Value	Explanation
<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.

11.27.2. 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.

11.27.3. Datatypes in Entities

The usage of custom datatypes in entities is explained in the [persistence layer guide](#).

11.27.4. 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](#).

11.28. 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>

11.29. 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?

11.29.1. Local Cache

11.29.2. Shared Cache

Distributed Cache

11.29.3. Products

- <http://ehcache.org/>
- <http://hazelcast.org/>
- <http://terracotta.org/>
- <http://memcached.org/>

11.29.4. 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

11.30. 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.

11.30.1. Configuring CORS support

On the server side we have defined a new filter in Spring security chain filters to support CORS and we have configured devonfw security chain filter to use it.

You only have to change `CORSDisabled` property value in `application-default.properties` properties file.

```
#CORS support
security.cors.enabled=false
```

11.31. BLOB support

11.31.1. Introduction

BLOB stands for **Binary Large Object**. A BLOB may be an image, an office document, ZIP archive or any other multimedia object. devon4j supports BLOB via its `BinaryObject` data type. The devonfw Maven archetype generates the following Java files for dealing with BLOBS:

<code>general.common.api.BinaryObject</code>	Interface for a <code>BinaryObject</code>
<code>general.dataaccess.api.BinaryObjectEntity</code>	Instance of <code>BinaryObject</code> entity, contains the actual BLOB
<code>general.dataaccess.api.dao.BinaryObjectDao.java</code>	DAO for <code>BinaryObject</code> entity
<code>general.dataaccess.base.dao.BinaryObjectDaoImpl</code>	Implementation of the <code>BinaryObjectDao</code>
<code>general.logic.api.to.BinaryObjectEto</code>	ETo for <code>BinaryObject</code>
<code>general.logic.base.UcManageBinaryObject</code>	Use case for managing <code>BinaryObject</code> . This use case contains methods for finding, getting, deleting and saving a BLOB.
<code>general.logic.impl.UcManageBinaryObjectImpl</code>	Implementation of the <code>UcManageBinaryObject</code>

11.31.2. Implementing BLOB support: an example

In the sample application the business component Offermanagement uses BLOBS for product pictures. Feel free to use the following approach as starting point for BLOB support in your application.

Logic Layer

Use the methods declared in `general.logic.base.UcManageBinaryObject` in the implementation of your business component. Let's take a look at an example from the sample application.

The method

```
OffermanagementImpl.updateProductPicture(Long productId, Blob blob, BinaryObjectEto  
binaryObjectEto)
```

saves a new picture for a given product.

This is done by calling an appropriate method, declared in the BinaryObject use case.

```
@Override  
 @RolesAllowed(PermissionConstants.SAVE_PRODUCT_PICTURE)  
 public void updateProductPicture(Long productId, Blob blob, BinaryObjectEto  
 binaryObjectEto) {  
  
    ...  
    binaryObjectEto = getUcManageBinaryObject().saveBinaryObject(blob,  
    binaryObjectEto);  
    ...  
}
```

Service Layer

Following the devonfw conventions, you must implement a REST service for each business component. There you define, how BLOBs are uploaded/downloaded. According to that, the REST service for the business component Offermanagement is implemented in a class named OffermanagementRestServiceImpl.

The coding examples below are taken from the afore mentioned class.

The sample application uses the content-type "multipart/mixed" to transfer pictures plus additional header data.

Upload

```

@Consumes("multipart/mixed")
@POST
@Path("/product/{id}/picture")
public void updateProductPicture(@PathParam("id") long productId,
    @Multipart(value = "binaryObjectEto", type = MediaType.APPLICATION_JSON)
BinaryObjectEto binaryObjectEto,
    @Multipart(value = "blob", type = MediaType.APPLICATION_OCTET_STREAM)
InputStream picture)
    throws SerialException, SQLException, IOException {

    Blob blob = new SerialBlob(IOUtils.readBytesFromStream(picture));
    this.offerManagement.updateProductPicture(productId, blob, binaryObjectEto);

}

```

A new Blob object is being created by reading the data (`IOUtils.readBytesFromStream(picture)`).

Download

```

@Produces("multipart/mixed")
@GET
@Path("/product/{id}/picture")
public MultipartBody getProductPicture(@PathParam("id") long productId) throws
SQLException, IOException {

    Blob blob = this.offerManagement.findProductPictureBlob(productId);
    byte[] data = IOUtils.readBytesFromStream(blob.getBinaryStream());

    List<Attachment> atts = new LinkedList<>();
    atts.add(new Attachment("binaryObjectEto", MediaType.APPLICATION_JSON, this
.offerManagement
    .findProductPicture(productId)));
    atts.add(new Attachment("blob", MediaType.APPLICATION_OCTET_STREAM, new
ByteArrayInputStream(data)));
    return new MultipartBody(atts, true);
}

```

As you may have noticed, the data is loaded into the heap before it is added as an Attachment to the MultiPart body.

Caution!

Using a byte array will cause problems, when dealing with large BLOBs.

Why is the sample application using a byte array then?

As of now, there is no universal solid way of streaming a BLOB directly from a database to the client without reading the BLOB's content to memory, when streaming over a RESTful service based on JDBC and JAX RS. Following this approach means: whenever a file is uploaded or downloaded as

BLOB it is loaded completely to memory before it is written to the database.

11.31.3. Further Reading

- [The multipart content type](#)
- [JAX-RS : Support for Multiparts](#)
- [Component Implementation](#)
- [BLOBs and the Data Access Layer](#)
- [Security Vulnerability Unrestricted File Upload](#)

11.32. 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).

11.32.1. 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 bugfixes 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.

11.32.2. 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 14. Dependencies for classes removed from Java 8 since 9+

Class	GroupId	ArtifactId	Version
<code>javax.xml.bind.*</code>	<code>javax.xml.bind</code>	<code>jaxb-api</code>	<code>2.3.1</code>
<code>com.sun.xml.bind.*</code>	<code>org.glassfish.jaxb</code>	<code>jaxb-runtime</code>	<code>2.3.1</code>
<code>java.activation.*</code>	<code>javax.activation</code>	<code>javax.activation-api</code>	<code>1.2.0</code>
<code>java.transaction.*</code>	<code>javax.transaction</code>	<code>javax.transaction-api</code>	<code>1.2</code>
<code>java.xml.ws.*</code>	<code>javax.xml.ws</code>	<code>jaxws-api</code>	<code>2.3.1</code>
<code>javax.jws.*</code>	<code>javax.jws</code>	<code>javax.jws-api</code>	<code>1.1</code>
<code>javax.annotation.*</code>	<code>javax.annotation</code>	<code>javax.annotation-api</code>	<code>1.3.2</code>

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 15. 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
org.javassist	javassist	3.25.0-GA	194 , 228 , 246 , 171

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 16. 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 17. 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

GroupId	ArtifactId	(min.) Version	Issue
org.apache.maven.plugins	maven-surefire-plugin	2.22.2	SUREFIRE-1439
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

11.32.3. 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](#)

11.33. Application Performance Management

This guide gives hints how to manage, monitor and analyse performance of Java applications.

11.33.1. 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.

11.33.2. 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](#)

11.33.3. Alternatives

- [PinPoint](#)
- [OpenAPM](#)
- [AppDynamics](#)
- [Zabbix](#)

[1] "Stammdaten" in German.

12. devonfw Development

12.1. IDE Setup

This Tutorial explains how to setup the development environment to work on and contribute to devonfw4j with your Windows computer.

We are using a pre-configured [devon-ide](#) for development. To get started follow these steps:

1. Get a Git client. For Windows use:

- <https://gitforwindows.org/>

Important: install with option Use Git from the Windows Command Prompts but without Windows Explorer integration.



- Download TortoiseGit from <https://tortoisegit.org/>

2. Download the IDE

- If you are a member of Capgemini: download [devonfw ide package](#) or the higher integrated [devonfw distribution](#) (for devonfw please find the setup guide within the devon-dist).
- If you are not member of Capgemini: We cannot distribute the package. Please consult [devon-ide](#) to setup and configure the IDE manually. If you need help, please get in touch.

3. Choose a project location for your project (e.g. `C:\projects\devonfw`, referred to with `$projectLoc` in this setup guides following steps). Avoid long paths and white spaces to prevent trouble. Extract the downloaded ZIP files via [Extract Here](#) (e.g. using [7-Zip](#)). Do not use the Windows native ZIP tool to extract as this is not working properly on long paths and filenames.

4. Run the script `update-all-workspaces.bat` in `$projectLoc`.

```

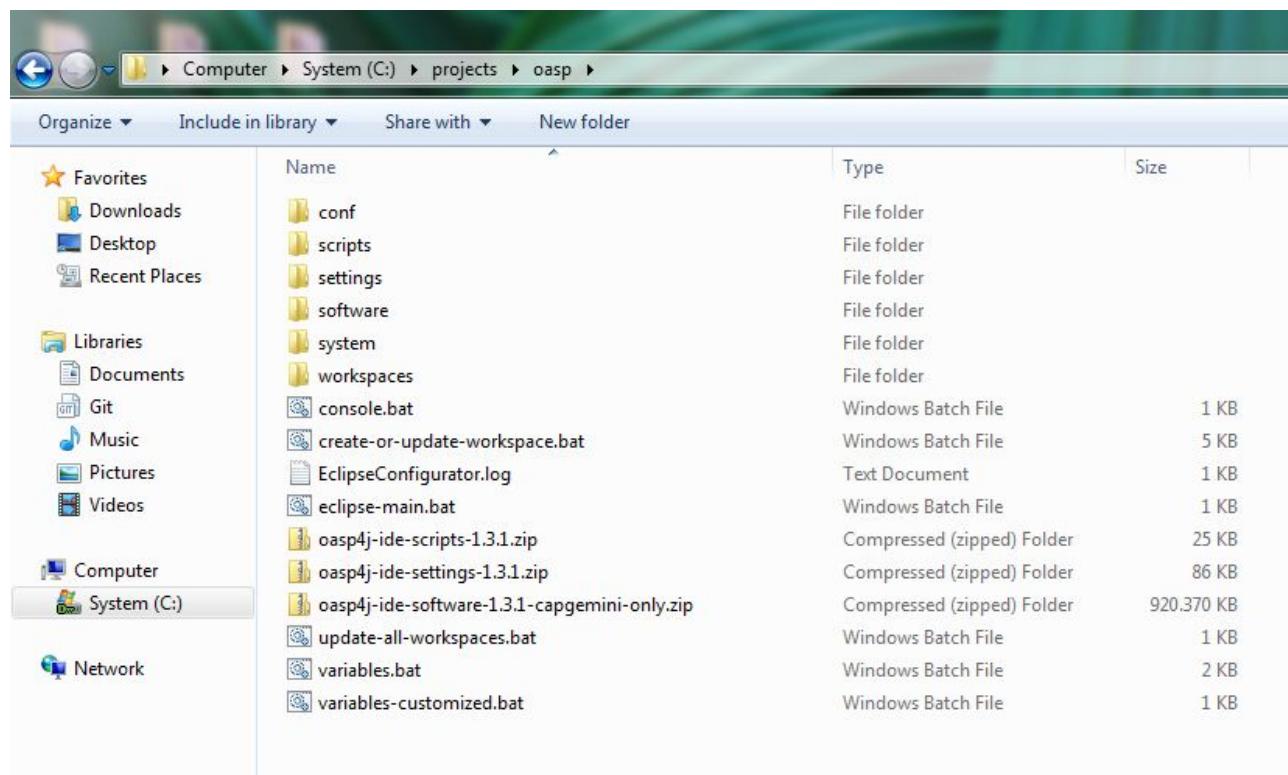
C:\WINDOWS\system32\cmd.exe
Copied workspaces\main\oasp4j\oasp4j-ide\oasp4j-ide-settings\src\main\settings\maven\settings.xml to conf\.m2\settings.xml
Apr 29, 2015 11:24:03 AM io.oasp.ide.eclipse.configurator.core.Configurator logC all
INFO: io.oasp.ide.eclipse.configurator.core.Configurator -u
Apr 29, 2015 11:24:03 AM io.oasp.ide.eclipse.configurator.core.Configurator main
INFO: Updating workspace
Apr 29, 2015 11:24:03 AM io.oasp.ide.eclipse.configurator.core.Configurator collectWorkspaceFiles
INFO: Collected 54 configuration files.
Apr 29, 2015 11:24:03 AM 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"

Finished updating all workspaces
Press any key to continue . .

```

Hint: You can use update-all-workspaces.bat whenever you created a new folder in **workspaces** to separate different workspaces. This update will create new Eclipse start batches allowing to run a number of Eclipse instances using different workspaces in parallel.

You should end up having a structure like this in **\$projectLoc**



5. Open **console.bat** and check out the git repositories you need to work on into **workspaces\main**, with the following commands:

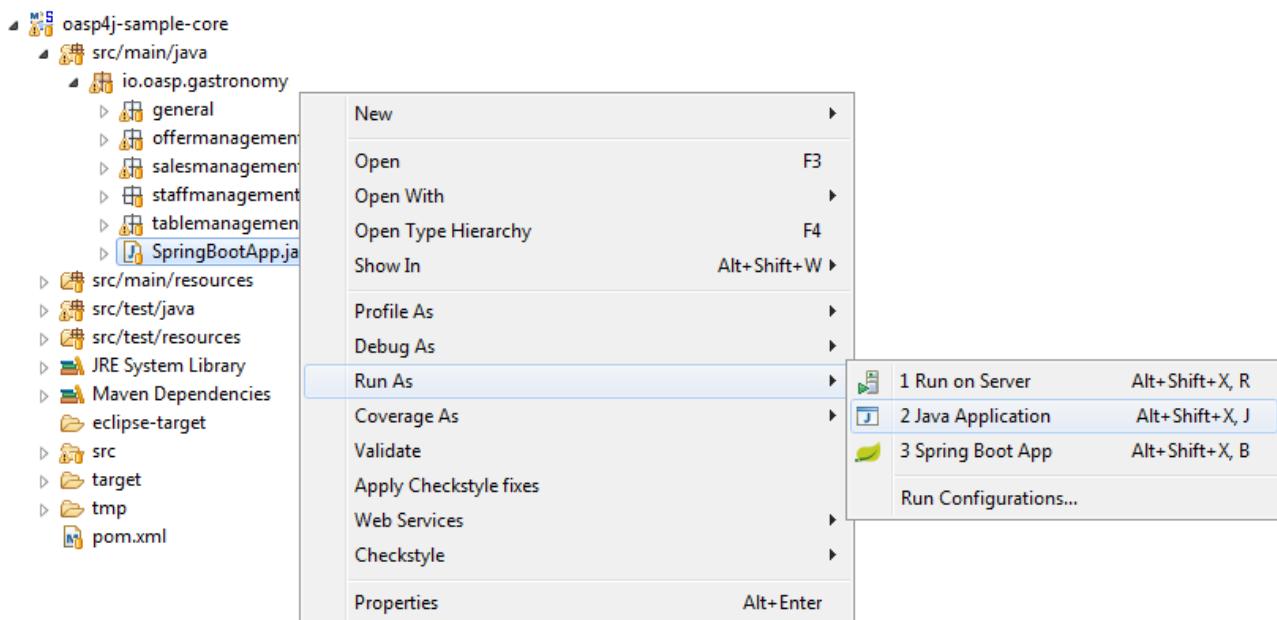
```

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

```

Do another check whether there are files in folder **workspaces\main\my-thai-star\!**

6. Run the script `eclipse-main.bat` to start the Eclipse IDE.
7. In Eclipse select `File > Import > Maven > Existing Maven Projects` and then choose the cloned projects from your workspace by clicking the `Browse` button and select the folder structure (`workspaces\main\my-thai-star\java\MTSJ`).
8. Execute the application by starting the 'SpringBootApp'. Select the class and click the right mouse button. In the context menu select the entry `Run as ⇒ Java Application` (or `Debug as ...`). The application starts up and creates log entries in the Eclipse Console Tab.



Once started, the sample application runs on <http://localhost:8081/mythaistar>, login with waiter/waiter and have a look at the services list provided.

12.2. Issue creation and resolution

12.2.1. Issue creation

You can create an issue [here](#). Please consider the following points:

- If your issue is related to a specific building block (like e.g. devon4ng), open an issue on that specific issue tracker. If you're unsure which building block is causing your problem open an issue on this repository.
- Put a label on the issue to mark whether you suggest an enhancement, report an error or something else.

When reporting errors:

- Include the version of devon4j you are using.
- Include screenshots, stack traces.
- Include the behavior you expected.
- Using a debugger you might be able to find the cause of the problem and you could be the one to contribute a bug-fix.

12.2.2. Preparation for issue resolution

Before you get started working on an issue, check out the following points:

- try to complete all other issues you are working on before. Only postpone issues where you are stuck and consider giving them back in the queue (backlog).
- check that no-one else is already assigned or working on the issue
- read through the issue and check that you understand the task completely. Collect any remaining questions and clarify them with the one responsible for the topic.
- ensure that you are aware on which branch the issue shall be fixed and start your work in the corresponding workspace.
- if you are using git perform your changes on a feature branch.

12.2.3. Definition of Done

- actual issue is implemented (bug fixed, new feature implemented, etc.)
- new situation is covered by tests (according to test strategy of the project e.g. for bugs create a unit test first proving the bug and running red, then fix the bug and check that the test gets green, for new essential features create new tests, for GUI features do manual testing)
- check the code-style with sonar-qube in eclipse. If there are anomalies in the new or modified code, please rework.
- check out the latest code from the branch you are working on (svn update, git pull after git commit)
- test that all builds and tests are working (mvn clean install)
- commit your code (svn commit, git push) - for all your commits ensure you stick to the conventions for code contributions (see [code contribution](#)) and provide proper comments (see [coding conventions](#)).
- if no milestone was assigned please assign suitable milestone
- set the issue as done

12.3. Code contribution

We are looking forward to your contribution to devon4j. This page describes the few conventions to follow. Please note that this is an open and international project and all content has to be in (American) English language.

For contributions to the code please consider:

- We are working issue-based so check if there is already an issue in our tracker for the task you want to work on or create a new issue for it.
- In case of more complex issues please get involved with the community and ensure that there is a common understanding of what and how to do it. You do not want to invest into something that will later be rejected by the community.

- Before you get started ensure that you comment the issue accordingly and you are the person assigned to the issue. If there is already someone else assigned get in contact with him if you still want to contribute to the same issue. You do not want to invest into something that is already done by someone else.
- Create a [fork](#) of the repository on github to your private github space.
- Clone this fork.
- Before doing any change choose the branch you want to add your feature to. In most cases this will be the [develop](#) branch to add new features. However, if you want to fix a bug, check if an according maintenance branch [develop-x.y](#) already exists and switch to that one before.
- Then the first step is to create a local feature branch (named by the feature you are planning so `feature/«issue-id»-«keyword») and checkout this branch.
- Start your modifications.
- Ensure to stick to our [coding-conventions](#).
- Check in features or fixes as individual commits associated with an [issue](#) using the commit message format:

```
#<issueId>: <describe your change>
```

Then github will automatically link the commit in the issue. In case you worked on an issue from a different repository (e.g. change in devon4j-sample due to issue in devon4j) we use this commit message format:

```
devonfw/<repository>#<issueId>: <describe your change>
```

So as an example:

```
devonfw/devon4j#1: added REST service for tablemanagement
```

- If you completed your feature (bugfix, improvement, etc.) use a [pull request](#) to give it back to the community.
- Your pull request will automatically be checked if it builds correctly (no compile or test errors), can be merged without conflicts, and [CLA](#) has been signed. Please ensure to do the required tasks and reworks unless all checks are satisfied.
- From here a reviewer should take over and give feedback. In the best case, your contribution gets merged and everything is completed.
- In case you should not get feedback for weeks, do not hesitate to ask the community.
- If one (typically the reviewer) has to change the base branch (because the wrong develop branch was used, see above) onto which the changes will be merged, one can do the same by following the instructions at [here](#).
- see also the [documentation](#) guidelines.

12.4. devonfw Documentation

We are using the github wiki feature to create and maintain the documentation of the devonfw. The WIKI is source for generation of pdf-versions to generate the platform guide and tutorials.

There are two editions of this wiki:

- [devon4j \(official wiki\)](#) for stable states of this wiki that is read-only for regular users.
- [devon4j \(community wiki\)](#) for development and contributions to this wiki. Here you can edit and contribute.

12.4.1. Contribution to devon4j documentation

Contributions and improvements to the documentation are welcome. However, you should be aware of the following aspects:

- Your contributions will become part of the devon4j documentation and is licensed under creative commons (see footer).
- If you want to contribute larger changes (beyond fixing a typo or a link) please consider to get in contact with the community (by creating an [issue](#)) before getting started. You do not want to write complete chapters and then get your work rejected afterwards.
- Please consult the [DocGen manual](#) as we are using DocGen to generate the documentation starting from [devon4j-doc](#).

If you consider all the aspects above you can just edit the community wiki (in [devonfw-wiki](#), see above) if you have a [github-account](#).

13. Tutorials

13.1. Introduction

This is an step by step tutorial for starting an devonfw server application from setting up the environment to packaging for production.

The tutorial starts by setting up the programmer environment with the aid of the devon-ide project and verifies everything is correct by running the my-thai-start restaurant sample application of the devonfw project.

Afterwards a new blank application is created by using the provided archetypes and all generated files are reviewed to explain what devonfw is providing.

A classical CRUD use case is developed for creating, retrieving updating and deleting an entity. With this entity we introduce cross cutting concerns such as exception handling, validation and securing the access from the web.

Finally the sample will be ready for deployment to a web server so we will package it on a WAR (or EAR) file.

13.2. Creating a new application

13.2.1. 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. 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=3.1.0 -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
```

Further providing additional properties (using `-D` parameter) you can customize the generated app:

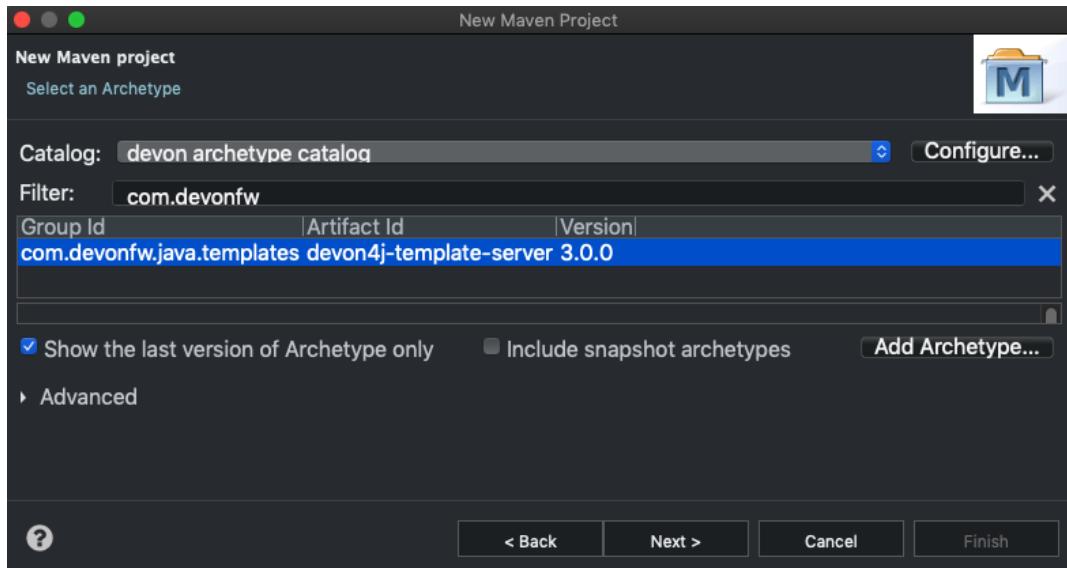
Table 18. Options for app template

property	comment	example
dbType	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.)	-DdbTpye=postgresql
batch	Option to add an <code>batch</code> module	-Dbatch=batch
earProjectName	Option to add an EAR module with the given name	-DearProjectName=ear

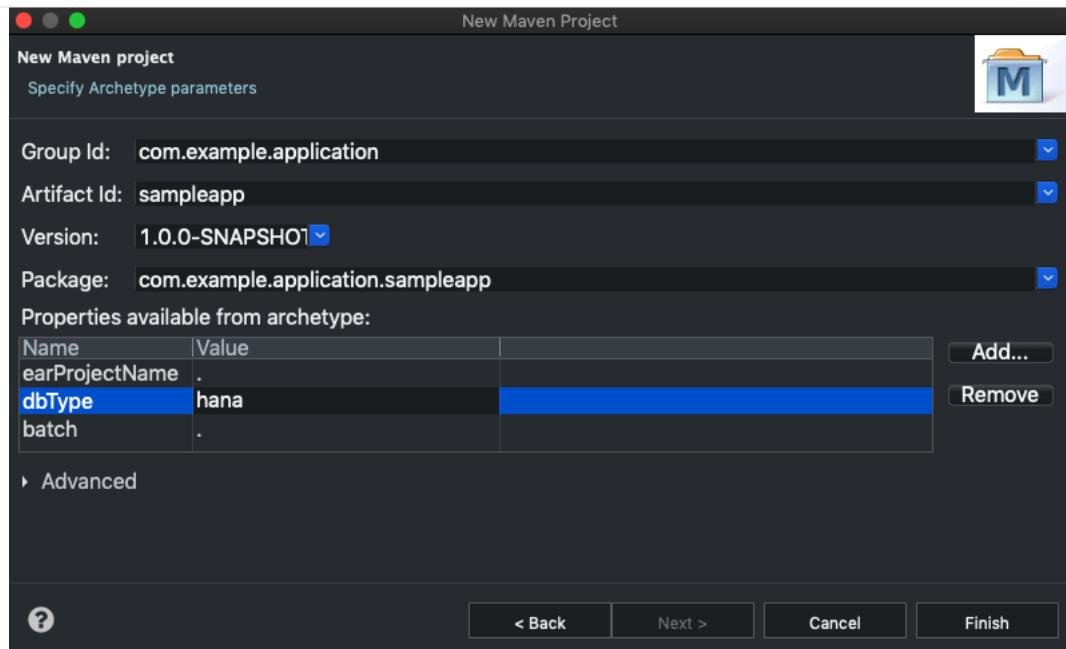
From Eclipse

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.

13.2.2. 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

Core Module

Core module contains the base classes and the base configuration for the application. We are going to describe each Java file and each XML configuration file that archetype has generated.

Java

Those are the different Java files contained in each package:

- general.common

File	Descripcion
api.ApplicationEntity.java	Abstract interface for a MutableGenericEntity of this application.
api.BinaryObject.java	Interface for a BinaryObject.
api.NlsBundleApplicationRoot.java	NlsBundle for this application.
api.Usermanagement.java	Interface to get a user from its login.
api.UserProfile.java	Interface for the profile of a logged user.
api.constants.PermissionConstants.java	Constants for AccessControlPermission's keys.
api.datatype.OrderBy.java	Enum for sort order.
api.datatype.Role.java	Enum for roles.
api.exception.ApplicationBusinessException.java	Abstract business main exception.
api.exception.ApplicationException.java	Abstract main exception.
api.exception.ApplicationTechnicalException.java	Abstract technical main exception.
api.exception.IllegalEntityStateException.java	Manage entities illegal state exceptions.
api.exception.IllegalPropertyChangeException.java	Manage entities illegal property changes exceptions.
api.exception.NoActiveUserException.java	Manage exceptions when user require to be logged in.
api.security(userData).java	Container class for the profile of a user.
api.to.AbstractCto.java	Abstract class for Composite Transfer Object.
api.to.AbstractEto.java	Abstract class for Entity Transfer Object.
api.to.AbstractTo.java	Abstract class for a plain Transfer Object.
api.to.SearchCriteriaTo.java	Abstract class for a Transfer Object with the criteria for a search query.
api.to.UserDetailsClientTo.java	.
base.AbstractBeanMapperSupport.java	Provides access to the BeanMapper.
impl.security.ApplicationAuthenticationProvider.java	Responsible for the security aspects of authentication.

File	Descripcion
impl.security. PrincipalAccessControlProviderImpl.java	Implementation of PrincipalAccessControlProvider.

- general.dataaccess

File	Descripcion
api.ApplicationPersistenceEntity.java	Abstract Entity for all Entities with an id and a version field.
api.BinaryObjectEntity.java	BinaryObject entity.
api.dao.ApplicationDao.java	Interface for all DAOs of the application.
api.dao.ApplicationRevisionedDao.java	Interface for all revised DAOs of the application.
api.dao.BinaryObjectDao.java	DAO for BinaryObject entity.

- general.gui.api

File	Descripcion
LoginController.java	Controller for login page.

- general.logic

File	Descripcion
api.UseCase.java	Annotation to mark all use-cases.
api.to.BinaryObjectEto.java	ETO for a BinaryObject.
base.AbstractUc.java	Abstract base class for any use case in the application.
base.UcManageBinaryObject.java	Use case for managing BinaryObject.
impl.UcManageBinaryObjectImpl.java	Implementation of the UcManageBinaryObject interface.
impl.UsermanagementDummyImpl.java	Implementation of Usermanagement.

- general.service.impl.rest

File	Descripcion
ApplicationAccessDeniedHandler.java	Class to manage denied access.
ApplicationObjectMapperFactory.java	MappingFactory class to resolve polymorphic conflicts within the application.
SecurityRestServiceImpl.java	Class that represents REST service for security.

Resources

Those are the different XML files contained in resources folder:

- config

File	Descripcion
app.common.beans-common.xml	Contains beans definition for application common beans like propertyConfigurer bean.
app.common.beans-dozer.xml	Beans relationated with Dozer Mappers.
app.common.dozer-mapping.xml	Dozer mapping configuration.
app.dataaccess.beans-dataaccess.xml	Parent from the other data access files.
app.dataaccess.beans-db-plain.xml	Data source configuration for profile db-plain (testing).
app.dataaccess.beans-db-server.xml	Data source configuration for profile distinct to db-plain .
app.dataaccess.beans-jpa.xml	Contains neccessary beans to configure JPA.
app.dataaccess.NamedQueries.xml	
app.gui.dispatcher-servlet.xml	
app.logic.beans-logic.xml	Component scan configuration for classes in logic path.
app.security.access-control-schema.xml	
app.security.beans-security-filters.xml	Security filters definition.
app.security.beans-security.xml	Application security configuration.
app.service.beans-monitoring.xml	
app.service.beans-service.xml	Importing configuration files, REST beans definition and configuration.
app.websocket.websocket-context.xml	Scan component package definition for websockets.
app.application.default.properties	Default application properties values.
app	beans-application
Root file configuration. It starts the chain and imports other configuration files.	env
application	Specific application properties values.

- db

File	Descripcion
migration.V0001__Create_schema.sql	Script template to create the database schema and tables definition.

Test

Those are different Java files to serve as base classes in testing:

- general.common

File	Descripcion
AbstractSpringIntegrationTest.java	.
AccessControlSchemaXmlValidationTest.java	Tests if the access-control-schema.xml is valid.
PermissionCheckTest.java	Test to check if all relevant methods in use case implementations have permission checks.

Server Module

This module contains two files:

- lockback.xml: This file is in the resources folder and it is responsible to configure the log.
- pom.xml: This file has Maven configuration for packaging the application as a WAR. Also, this file has a profile to package the Javascript client ZIP file into the WAR.

EAR Module

This module only contains a pom.xml file to packaging the application as EAR from the WAR generated.

13.2.3. Database configuration and creation

Including driver installation if oracle or other db is required.

13.2.4. Editing the pom.xml

How to edit the pom.xml file for the project to add dependencies and modules for the application.

13.2.5. Known Issues

- Could not resolve archetype com.devonfw.java.templates:devon4j-template-server:... from any of the configured repositories. In Eclipse: Open Window > Preferences Open Maven > Archetypes Click 'Add Remote Catalog' and add the following: Catalog File: <http://repo1.maven.org/maven2/archetype-catalog.xml> Description: maven catalog

14. For Core-Developers

14.1. Creating a Release

This page documents how to create and publish a release of devon4j.

For each release there is a [milestone](#) that contains an issue for creating the release itself (the github issue of that issue is referred as «[issue](#)»). The release version is referred as «x.y.z».

14.1.1. Releasing the code

To release the code follow these steps.

- Create a clean clone of the repository:

```
git clone https://github.com/devonfw/devon4j.git
```

- In case you want to build a (bug fix) release from a different branch, switch to that branch:

```
git checkout -b develop-<>x.y<> origin/develop-<>x.y<>
```

- Ensure your branch is up-to-date:

```
git pull
```

- Ensure that the result is what you want to release ([mvn clean install](#)).
- Bump the release version by removing the [-SNAPSHOT](#) from [devon4j.version](#) property in top-level [pom.xml](#).
- Create an annotated tag for your release:

```
git tag -a release/x.y.z -m "#<>issue<>: tagged x.y.z"
```

e.g For release 2.5.0 the command would look like

```
git tag -a release/2.5.0 -m "#618: tagged 2.5.0"
```

where #618 is the issue number created for release itself under release milestone. You can confirm if the tag is created by listing out the tags with the following command

```
git tag
```

Configure OSSRH

For publishing artifacts to OSSRH, we need an OSSRH account with necessary rights for publishing and managing staging repositories. And configure this account in devonfw distribution to create connection and deploy to OSSRH.

- If you do not already have an account on OSSRH, create an account on the link below <https://issues.sonatype.org/secure/Signup!default.jspa>
- You need manager access to deploy artifacts to OSSRH. For same contact devonfw administrators for OSSRH.
- Open file `conf/.m2/setting.xml` in your devon distribution (devon-ide) and add a new server with following details

```
<server>
  <id>osssrh</id>
  <username><>osssrh_username</></username>
  <password><>osssrh_password</></password>
</server>
```

Here `<>osssrh_username</>` and `<>osssrh_password</>` are the account details used to login into OSSRH and should have rights to publish artifacts to OSSRH for `groupId` name `com.devonfw` (and its children). Please use [password encryption](#) and prevent storing passwords in plain text. The id `osssrh` points to the OSSRH repository for snapshot and release declared in the `<distributionManagement>` section of the `devon4j/pom.xml`.

- Optionally you may want to explicitly define PGP key via the associated email-address:

```
<profile>
  <id>devon.osssrh</id>
  <activation>
    <activeByDefault>true</activeByDefault>
  </activation>
  <properties>
    <gpg.keyname>your.email@address.com</gpg.keyname>
  </properties>
</profile>
```

Configure PGP

Artifacts should be PGP signed before they can be deployed to OSSRH. Artifacts can be signed either by using command line tool GnuPG or GUI based tool Gpg4win Kleopatra (preferred). Follow the steps below to sign artifacts using either of the two tools.

- Download tools GnuPg - <https://www.gnupg.org/download/> gpg4win - <https://www.gpg4win.org/download.html>
- Installation Installation is self explanatory for GnuPG and gpg4win. To verify installation of

GnuPg, open windows command line and run "gpg --version or gpg2 --version"

- Generate PGP key pair for signing artifacts.



Remember the passphrase set for PGP keys as it will be used later for authentication during signing of artifacts by maven.

Using GnuPg follow either of the link below

<http://central.sonatype.org/pages/working-with-pgp-signatures.html#generating-a-key-pair>

<https://www.youtube.com/watch?v=DE3FVty3NgE&feature=youtu.be>

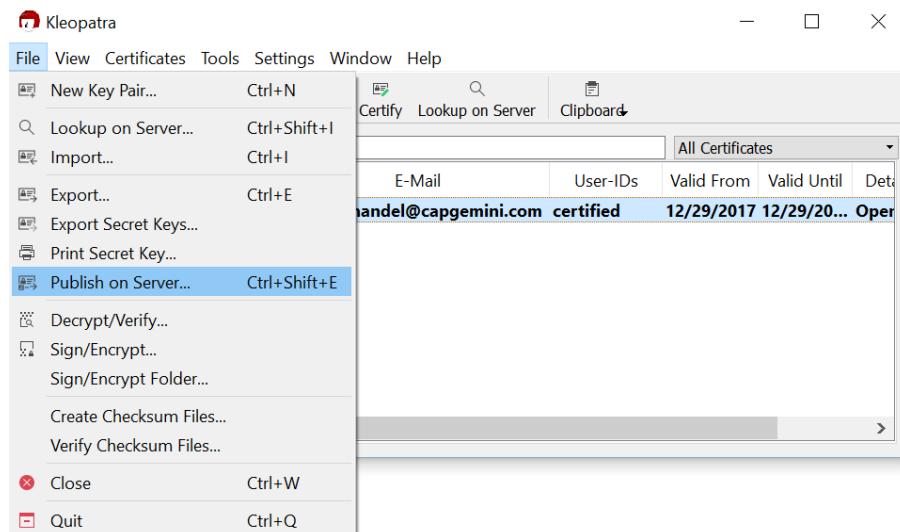
Using Kleopatra follow link below

<https://www.deepdotweb.com/2015/02/21/pgp-tutorial-for-windows-kleopatra-gpg4win/>

Exporting PGP key to public key-server

Using GnuPg - <http://central.sonatype.org/pages/working-with-pgp-signatures.html#distributing-your-public-key>

Using Kleopatra, click on the certificate entry you want to publish to OpenPGP certificate servers and select File > Publish on Server as shown below. These instructions are as per Kleopatra 3.0.1-gpg4win-3.0.2, for latest versions there might be some variation.



Deploy to OSSRH

- Go to the root of devon4j project and run following command. Make sure there are no spaces between comma separated profiles.

```
mvn clean deploy -P deploy
```

- A pop will appear asking for passphrase for PGP key. Enter the passphrase and press "OK".

```
C:\> C:\WINDOWS\system32\cmd.exe - mvn clean deploy -P deploy,all
Downloaded: https://repo.maven.apache.org/maven2/log4j/log4j/1.2.14/log4j-1.2.14.pom (3 KB at 2.5 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/com/thoughtworks/qdox/qdox/1.12.1/qdox-1.12.1.pom
Downloaded: https://repo.maven.apache.org/maven2/com/thoughtworks/qdox/qdox/1.12.1/qdox-1.12.1.pom (18 KB at 18.1 KB/sec)
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/maven-toolchain/2.2.1/maven-toolchain-2.2.1.jar
Downloading: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-invoker/2.0.9/maven-invoker-2.0.9.jar
Downloading: https://repo.maven.apache.org/maven2/commons-collections/commons-collections/3.2/commons-collections-3.2.jar
Downloading: https://repo.maven.apache.org/maven2/commons-lang/commons-lang/2.4/commons-lang-2.4.jar
Downloading: https://repo.maven.apache.org/maven2/org/apache/httpcomponents/httpclient/4.2.3/httpclient-4.2.3.jar
Downloading: https://repo.maven.apache.org/maven2/org/apache/httpcomponents/httpclient/4.2.3/httpclient-4.2.3.jar
Downloaded: https://repo.maven.apache.org/maven2/org/apache/httpcomponents/httpclient/4.2.3/httpclient-4.2.3.jar (37 KB at 44.4 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/apache/httpcomponents/httpclient/4.2.2/httpclient-4.2.2.jar (219 KB at 98.4 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/com/thoughtworks/qdox/qdox/1.11.1/qdox-1.11.1.jar
Downloaded: https://repo.maven.apache.org/maven2/commons-collections/commons-collections/3.2/commons-collections-3.2.jar (219 KB at 98.4 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/maven-toolchain/2.2.1/maven-toolchain-2.2.1.jar (60 KB at 18.7 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-invoker/2.0.9/maven-invoker-2.0.9.jar (28 KB at 8.6 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/commons-collections/commons-collections/3.2/commons-collections-3.2.jar (558 KB at 121.4 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/com/thoughtworks/qdox/qdox/1.12.1/qdox-1.12.1.jar (176 KB at 36.6 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/log4j/log4j/1.2.14/log4j-1.2.14.jar (359 KB at 73.9 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-container-default/1.0-alpha-9/plexus-container-default-1.0-alpha-9.jar (191 KB at 37.3 KB/sec)
Downloaded: https://repo.maven.apache.org/maven2/org/apache/httpcomponents/httpclient/4.2.3/httpclient-4.2.3.jar (423 KB at 76.9 KB/sec)
[INFO] Not executing Javadoc as the project is not a Java classpath-capable package
[INFO]
[INFO] --- maven-gpg-plugin:1.5:sign (sign-artifacts) @ oasp4j ---
```



If you face the error below, contact one of the people who have access to the repository for access rights.

```
[INFO] -----
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-deploy-plugin:2.8.2:deploy (default-deploy) on project oasp4j: Failed to deploy artifacts: Could not transfer artifact io.oasp.java.dev:oasp4j:pom:dev-20180111.090940-29 from/to ossrh (https://oss.sonatype.org/content/repositories/snapshots): Access denied to: https://oss.sonatype.org/content/repositories/snapshots/io/oasp/java/dev/oasp4j/dev-SNAPSHOT/oasp4j-dev-20180111.090940-29.pom, ReasonPhrase: Forbidden. -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoExecutionException
```

- Open [OSSRH](#), login and open staging repositories.
- Find your deployment repository as [comdevonfw-NNNN](#) and check its [Content](#).
- Then click on [Close](#) to close the repository and wait a minute.
- Refresh the repository and copy the URL.
- Create a vote for the release and paste the URL of the staging repository.
- After the vote has passed with success go back to OSSRH and click on [Release](#) to publish the release and stage to maven central.
- Edit the top-level `pom.xml` and change `devon4j.version` property to the next planned release version including the `-SNAPSHOT` suffix.
- Commit and push the changes:

```
git commit -m "#<<issue>>: open next snapshot version"
git push
```

- In case you build the release from a branch other than [develop](#) ensure to follow the next steps. Otherwise you are done here and can continue to the next section. To merge the changes (bug fixes) onto develop do:

```
git checkout develop  
git merge develop-<>x.y><
```

- You most probably will have a conflict in the top-level **pom.xml**. Then resolve this conflict. In any case edit this **pom.xml** and ensure that it is still pointing to the latest planned **SNAPSHOT** for the **develop** branch.
- If there are local changes to the top-level **pom.xml**, commit them:

```
git commit -m "#<>issue>: open next snapshot version"
```

- Push the changes of your **develop** branch:

```
git push
```

14.1.2. Releasing the documentation

- Initially and only once you have to create a local checkout of the github pages and of the wiki repository connected to the [community wiki](#):

```
git clone https://github.com/devonfw/devonfw.github.io.git  
git clone https://github.com/devonfw/devon4j.wiki.git  
cd devon4j.wiki  
git remote add source https://github.com/devonfw-wiki/devon4j.wiki.git
```

- Pull from **origin** as well as from **source**:

```
git pull origin  
git pull source
```

- Carefully review all changes that have been done on the forge wiki. Potentially reject changes if necessary.
- When you are complete push your changes:

```
git push origin
```

- Edit the **pom.xml** and change **version** to bump it to the release version (remove the **-SNAPSHOT** suffix).
- Commit the changes:

```
git commit -m "#<>issue>: bumped release version <>x.y.z><"
```

- Create an annotated tag for your release:

```
git tag -a release/x.y.z -m "#<<issue>>: tagged <x.y.z>"
```

- Build and deploy the documentation PDF from the cloned wiki repository via

```
mvn clean deploy -P oss
```

- Open [OSSRH](#), login and open staging repositories.
- Find your deployment repository as `comdevonfw-NNNN` and check its [Content](#)
- Only proceed if the PDF is sane (otherwise check what failed and restart)
- Then click on [Close](#) to close the repository and wait a minute.
- Refresh the repository.
- Click on [Release](#) to publish the release and stage to maven central.
- Edit the top-level `pom.xml` and change `version` property to the next planned release version including the `-SNAPSHOT` suffix.
- Commit and push the changes:

```
git commit -m "#<<issue>>: open next snapshot version"
git push --tags
```

- Also push the changes back to the community wiki:

```
git push source master
```

- Create a new folder for your version in your checkout of [devonfw.github.io/devon4j](#) (as `<x.y.z>`).
- Copy the just generated `devon4j-doc.pdf` into the new release version folder.
- Copy the `index.html` from the previous release to the new release version folder.
- Edit the new copy of `index.html` and replace all occurrences of the version to the new release as well as the release date.
- Generate the maven site from the `devon4j` release checkout (see [code release](#)):

```
mvn site
mvn site:deploy
```

- Review that the maven site is intact and copy it to the new release version folder (from `devon4j/target/devon4j/maven` to `devonfw.github.io/devon4j/<x.y.z>/maven`).
- Update the link in the `devon4j/index.html` to the latest stable documentation.

- Add, commit and push the new release version folder.

```
git add <>x.y.z>>
git commit -m "devonfw/devon4j#<>issue>>: released documentation"
git push
```

14.1.3. Finalize the Release

- Close the issue of the release.
- Close the milestone of the release (if necessary correct the release date).
- Ensure that the new release is available in maven central.
- Write an announcement for the new release.

Part IV: devon4ng

15. 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

15.1. 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.

15.1.1. Layers

The architecture describes two layers. The terminology is based on common language in web development.



Figure 5. 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.

15.1.2. 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*.

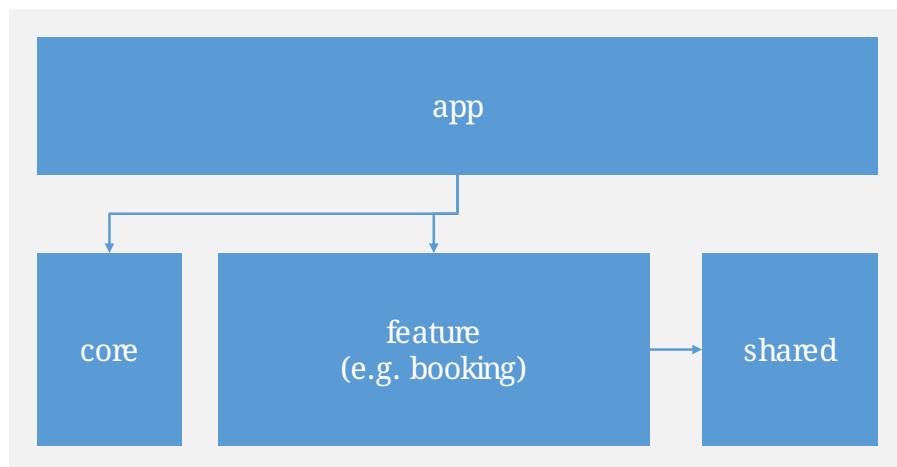


Figure 6. Modules

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.

16. Meta Architecture

16.1. Introduction

16.1.1. 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).

16.1.2. 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.

16.1.3. 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.

16.2. 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.

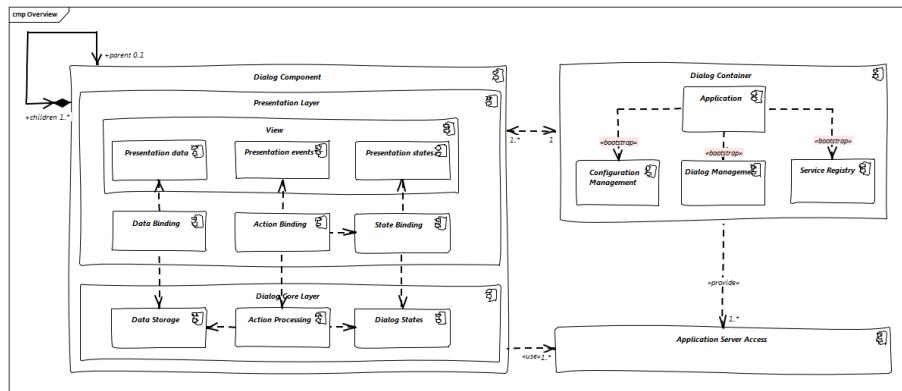


Figure 1 Overview

16.2.1. 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

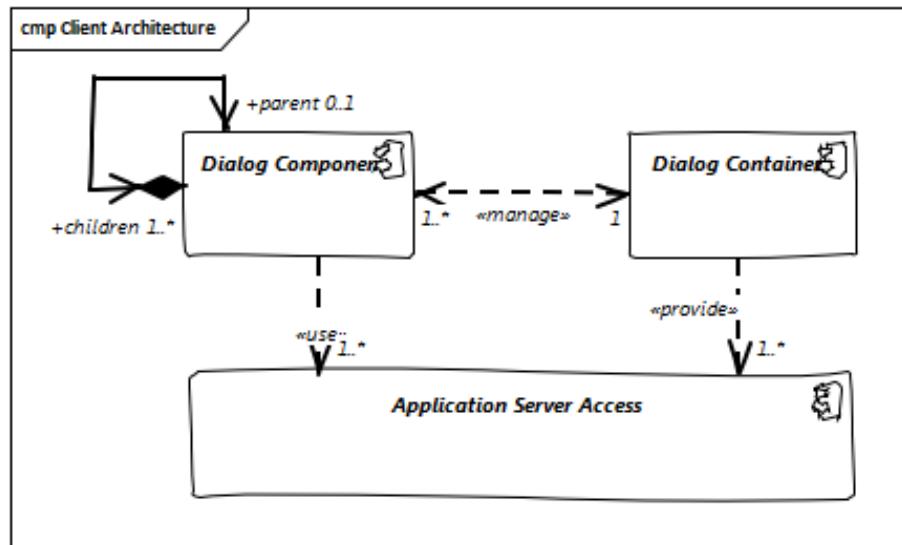


Figure 2 Overview of Client Architecture

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.

16.2.2. Dialog Container Architecture

The dialog container can be further structured into the following components with their respective tasks described in own sections:

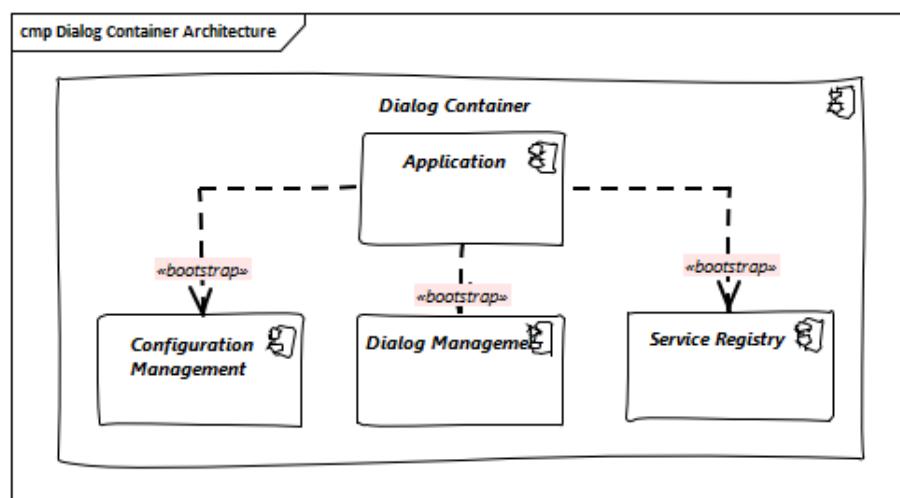


Figure 3 Dialog Container Architecture

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.

16.2.3. 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:

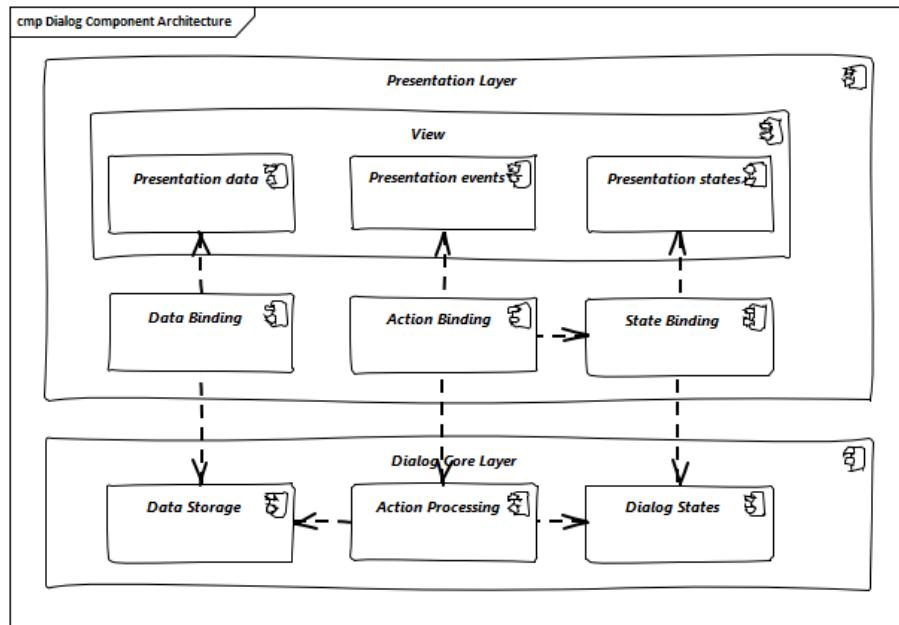


Figure 4 Overview of dialog component architecture

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.

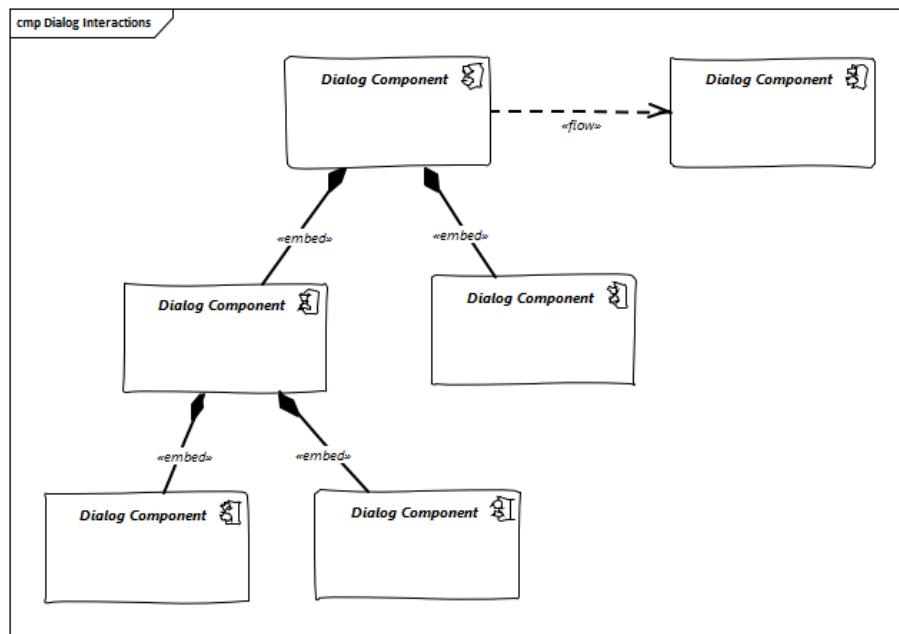
- **Communication with the application server**

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.

16.3. Appendix

16.3.1. 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.

16.4. 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

17. Layers

17.1. 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

17.1.1. Smart and Dumb Components

The architecture applies the concept of *Smart* and *Dumb Components* (syn. *Containers* and *Presenters*). The concept means that components are divided 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*.

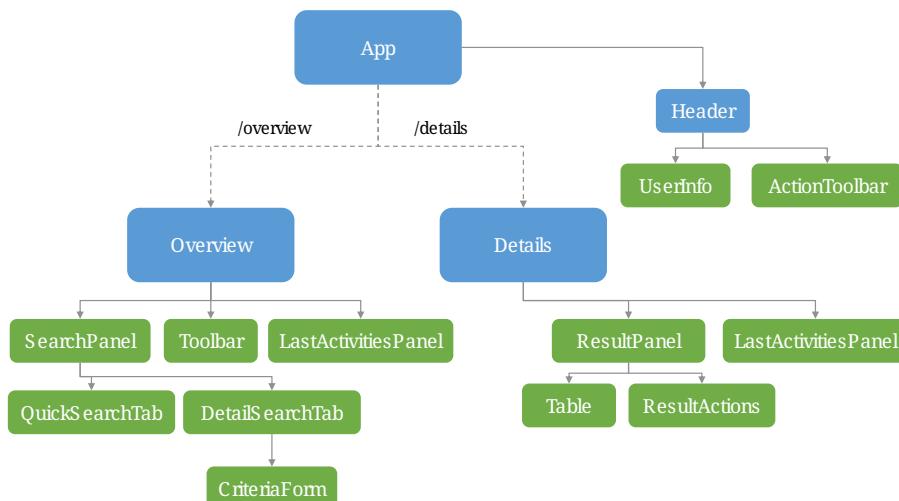


Figure 7. Component tree example

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 19. 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>

17.1.2. 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*.

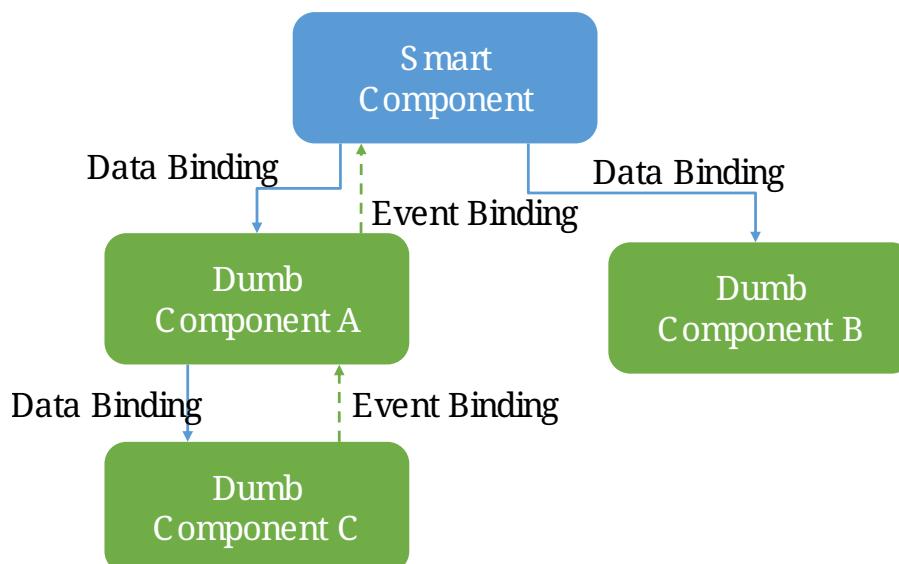


Figure 8. Smart and Dumb Component Interaction

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 5. 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 6. 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()`).

17.2. 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](#).

17.2.1. 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.

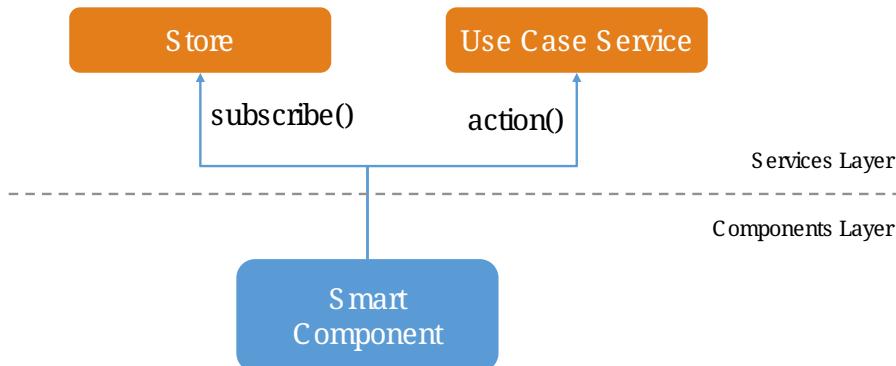


Figure 9. Boundaries to components layer

17.2.2. 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 7. 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.

17.2.3. 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`).

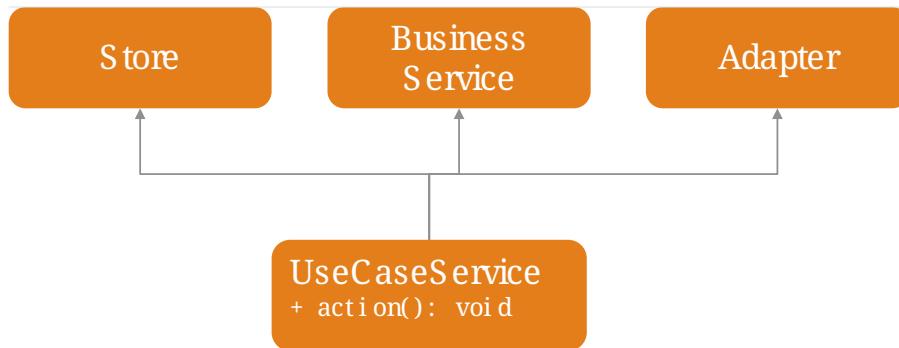


Figure 10. Use case services are the main API to trigger state transitions

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](#).

17.2.4. 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 8. 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');
  }

}
  
```

17.2.5. 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.

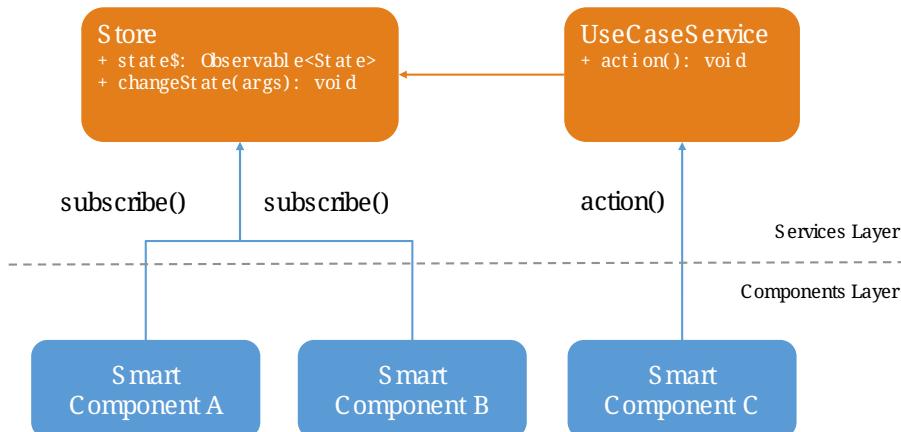


Figure 11. Smart Component interaction

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

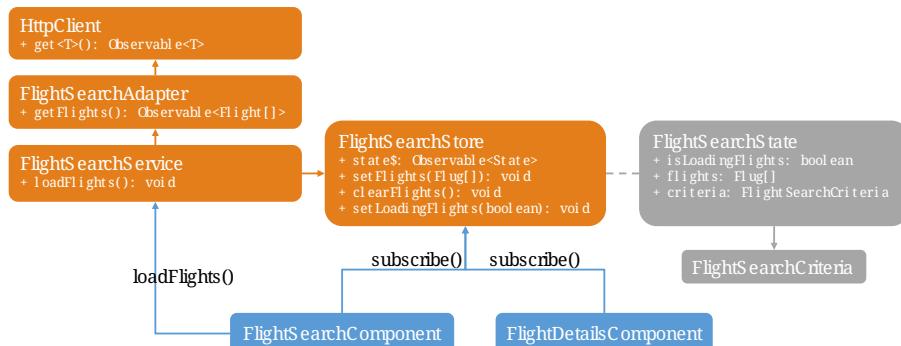


Figure 12. Smart Components interaction example

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.

18. Guides

18.1. 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.

18.1.1. Semantic Versioning

When working with package managers it is very important to understand the concept of [semantic versioning](#).

Table 20. 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.

18.1.2. 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
```

18.1.3. 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 9. *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.

18.1.4. devon4ng NPM-Yarn Workflow

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 ^[2]. 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 10. 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.

[2] A package is a file or directory that is described by a package.json. .

19. Angular

19.1. 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.

19.1.1. 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);
      }
    })
  }
}
```

19.1.2. 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.

19.1.3. 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](#)

19.1.4. 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.

19.1.5. 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)

19.1.6. 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.

19.1.7. 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 {
    console.log('SUBMIT');
    console.log(dishForm.value);
    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 AutoPolyfill. 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 is 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 11. 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--&gt;
      &lt;!--&lt;plate-form platename="test" platedescription="test"&gt;&lt;/plate-form&gt;--&gt;
      &lt;script src="./devon4ngAngularElements.js"&gt; &lt;/script&gt;
      &lt;script&gt;
        var elContainer = document.getElementById('container');
        var el= document.createElement('dish-form');
        el.setAttribute('platename','test');
        el.setAttribute('platedescription','test');
        el.addEventListener('submitDishEvent',(ev)=&gt;{
          var elView= document.createElement('dish-view');
          elContainer.innerHTML = '';
          elContainer.appendChild(elView);
        });
        elContainer.appendChild(el);
      &lt;/script&gt;
    &lt;/body&gt;
&lt;/html&gt;</pre>
```

Listing 12. 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.

19.1.8. 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>
```

19.2. 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.

19.3. 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 redirects to other windows in a "second level". It is a simple example, but useful to understand the relation between angular modules and lazy loading.

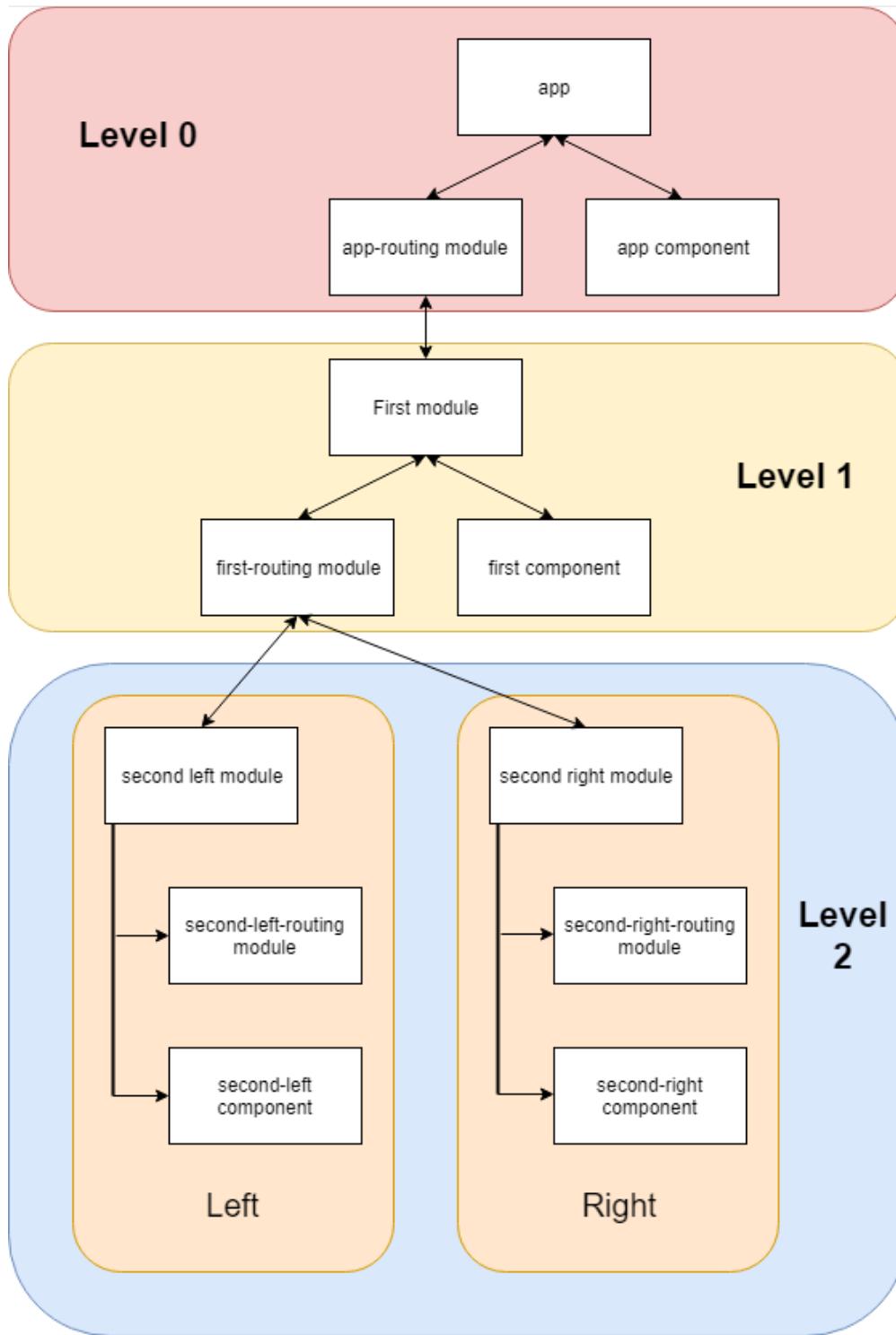


Figure 13. Levels app structure.

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](#).

19.3.1. 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 13. 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 14. 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 15. 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 16. 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 17. 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]
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 14. 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.

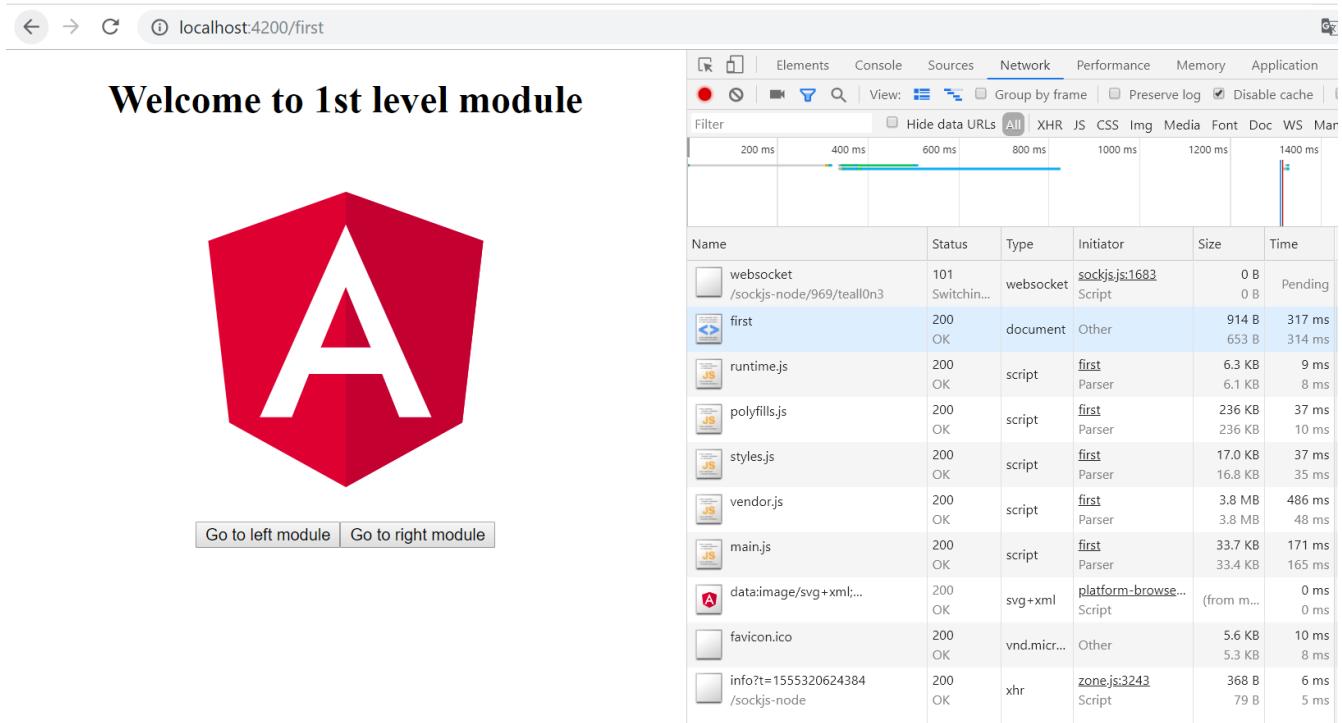


Figure 15. First level eager.

If we click on **[Go to right module]** a second level module opens, but there is no 'second-right' document.

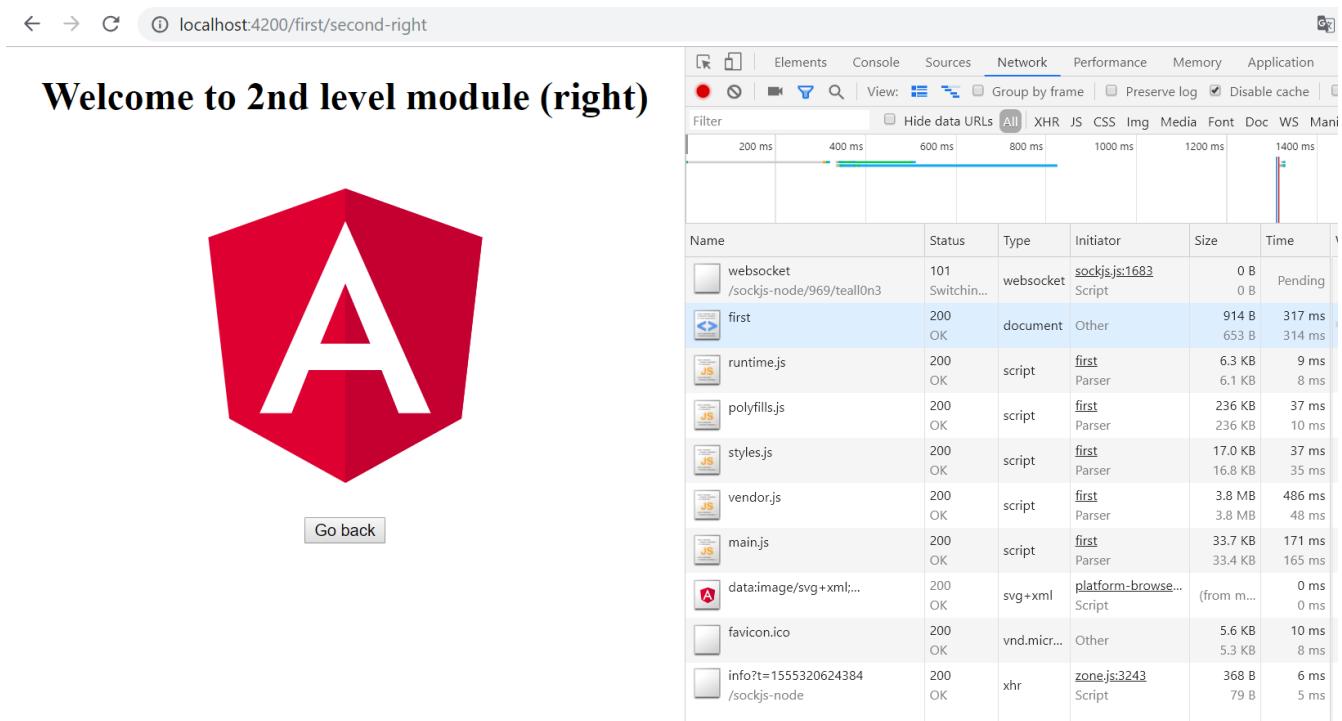


Figure 16. Second level right eager.

But, typing the url directly will load 'second-right' but no 'first', even if we click on **[Go back]**

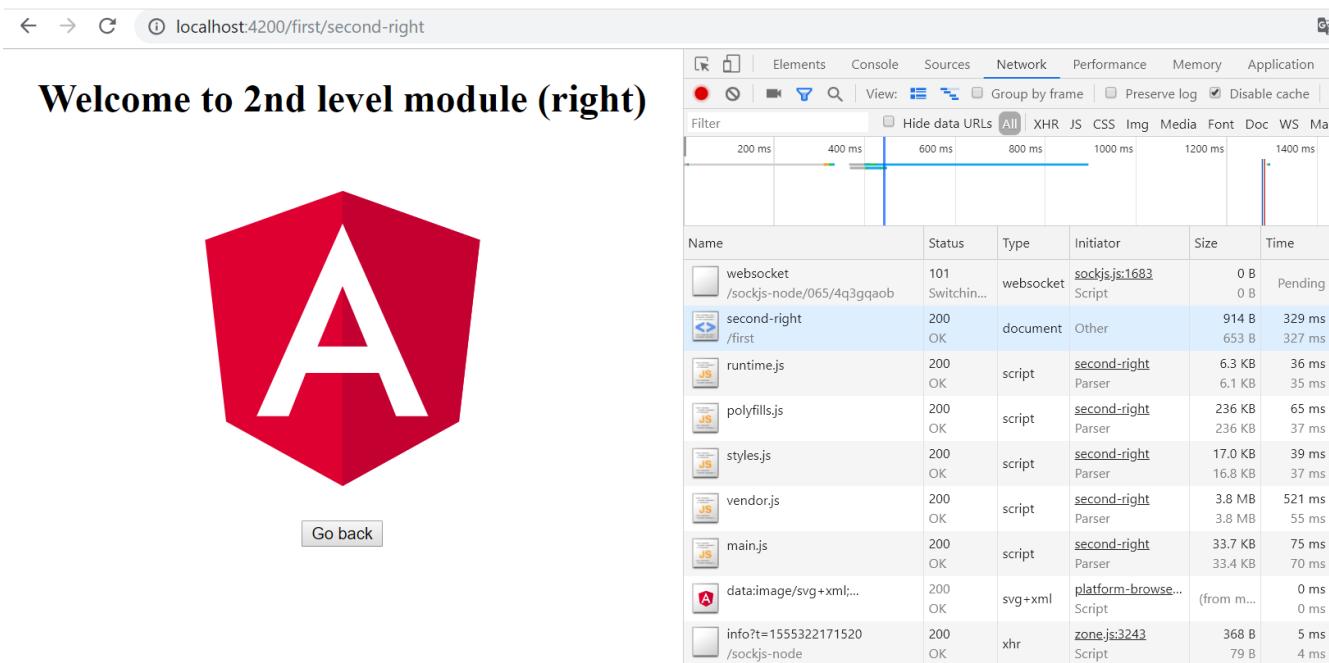


Figure 17. Second level right eager direct url.

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 18. 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 19. 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`'.

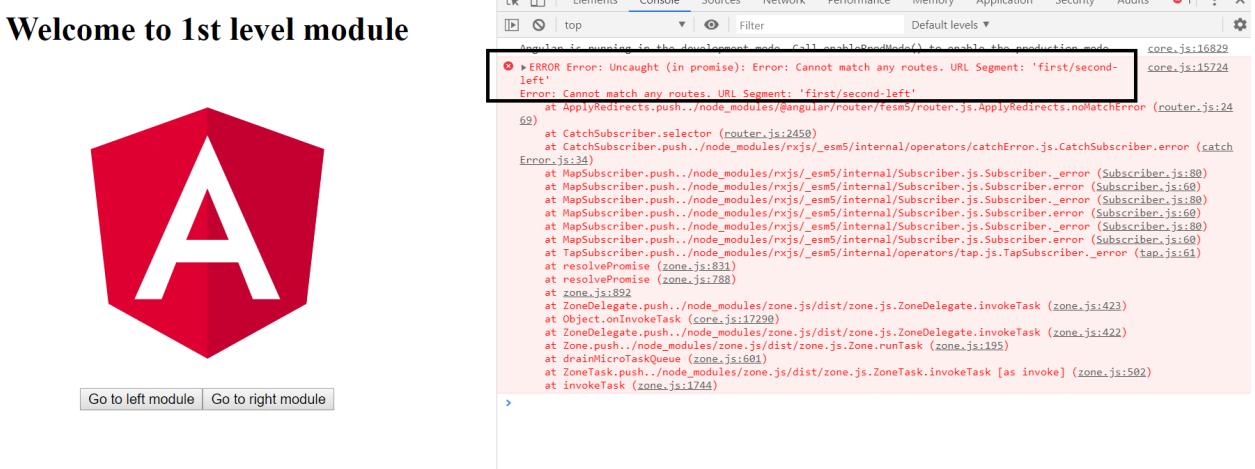


Figure 18. First level lazy wrong path.

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 19. 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' and the text "Welcome to 1st level module". Below the logo are two buttons: "Go to left module" and "Go to right module". To the right of the browser is the developer tools Network tab. The Network tab shows a list of resources loaded by the page. The "first" resource is highlighted, indicating it is currently being viewed. Other listed resources include "websocket", "runtime.js", "polyfills.js", "styles.js", "vendor.js", "main.js", and "first-first-module.js". The "Headers" and "Response" sections are also visible on the right side of the Network tab.

Figure 20. 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 20. 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 21. 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:4
200/ **

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.j
s.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 21. Second level lazy loading compilation.

Clicking on **[Go to left module]** triggers the load of SecondLeftModule.

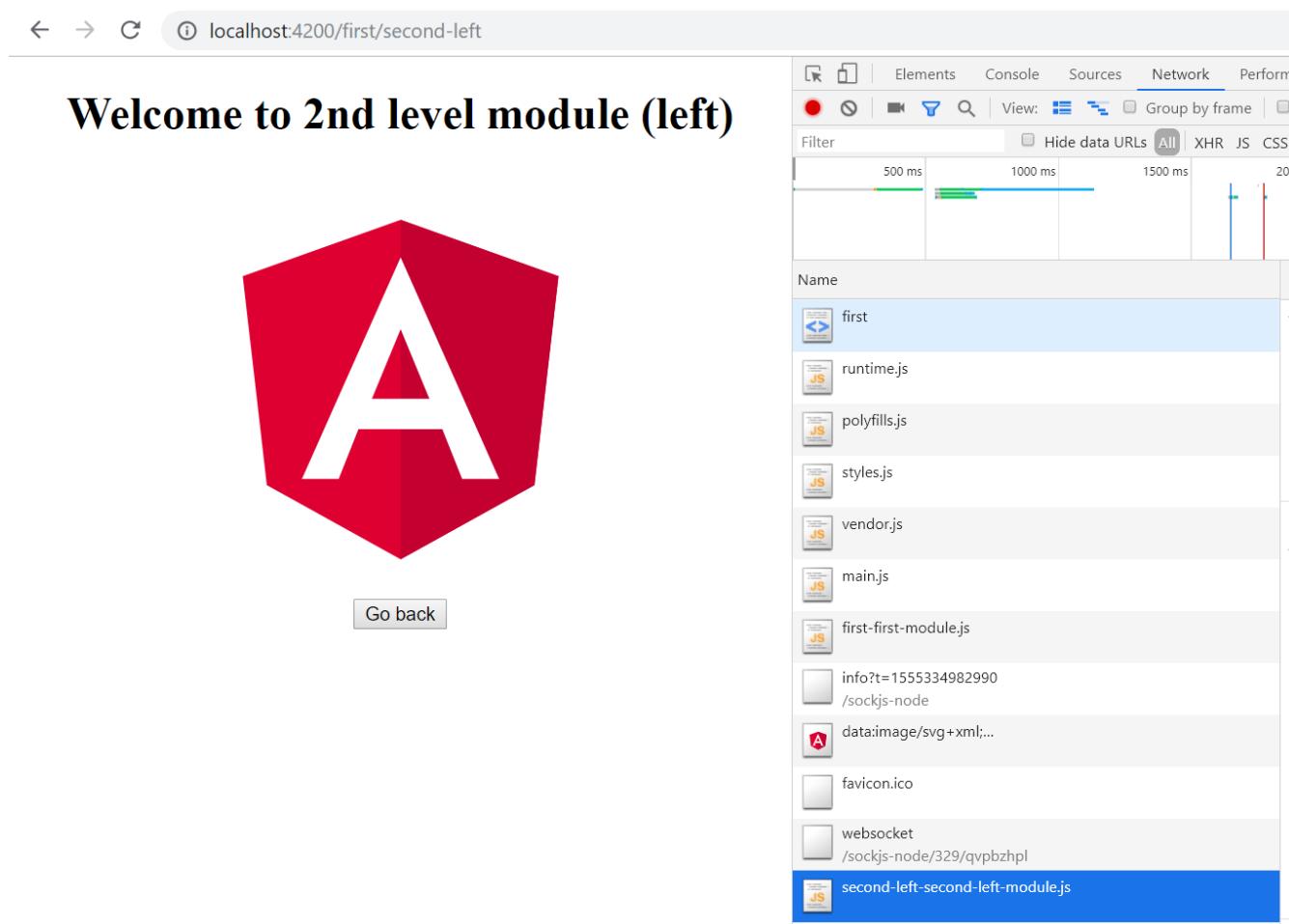


Figure 22. Second level lazy loading network.

19.4. 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.

19.4.1. 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.

19.4.2. 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.

19.4.3. 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 22. Packing using npm

```
npm pack
```

Listing 23. 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.

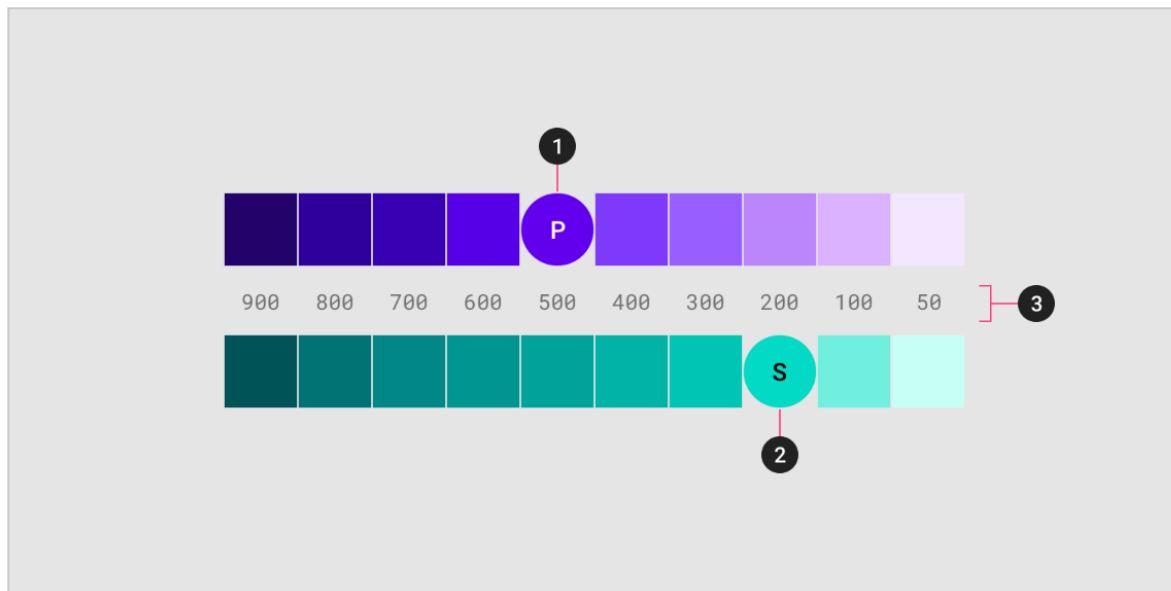
19.5. Angular Material Theming

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, ...

19.6. 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 23. Palettes and variants.

In angular material, a palette is represented as a scss map.

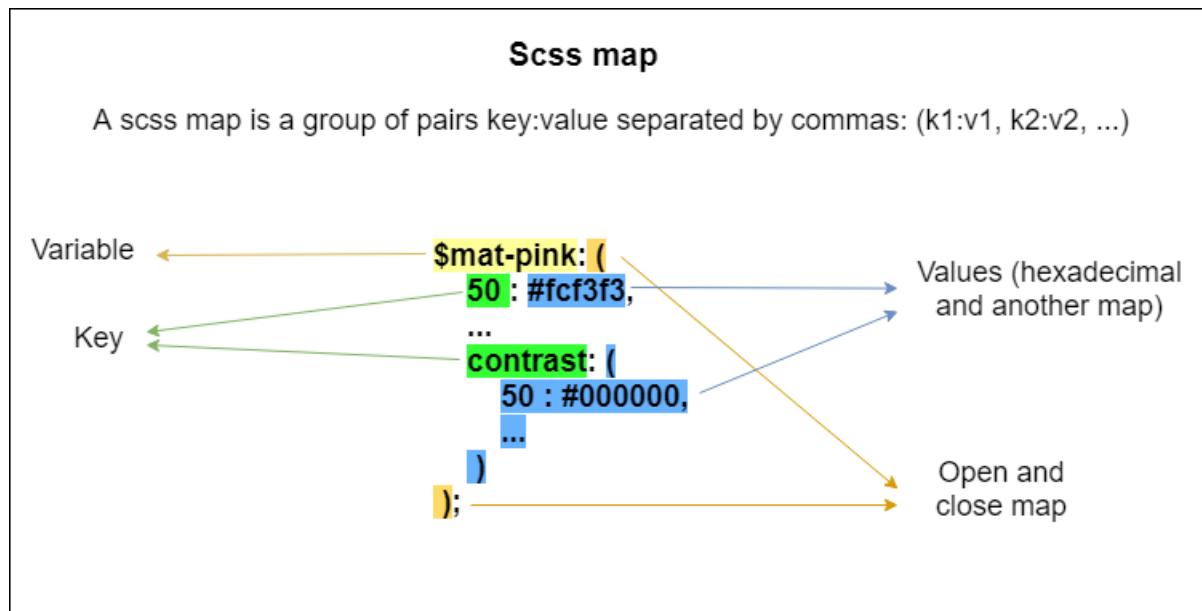


Figure 24. Scss map and palettes.



Some components can be forced to use primary, accent or warn palettes using the attribute **color**, for example: <mat-toolbar color="primary">.

19.7. Prebuilt themes

Available prebuilt themes:

- deeppurple-amber.css

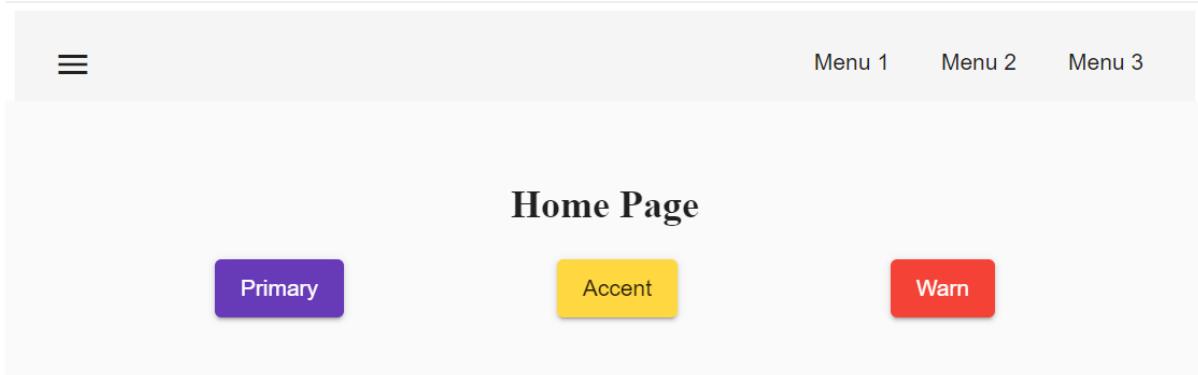


Figure 25. *deeppurple-amber* theme.

- `indigo-pink.css`

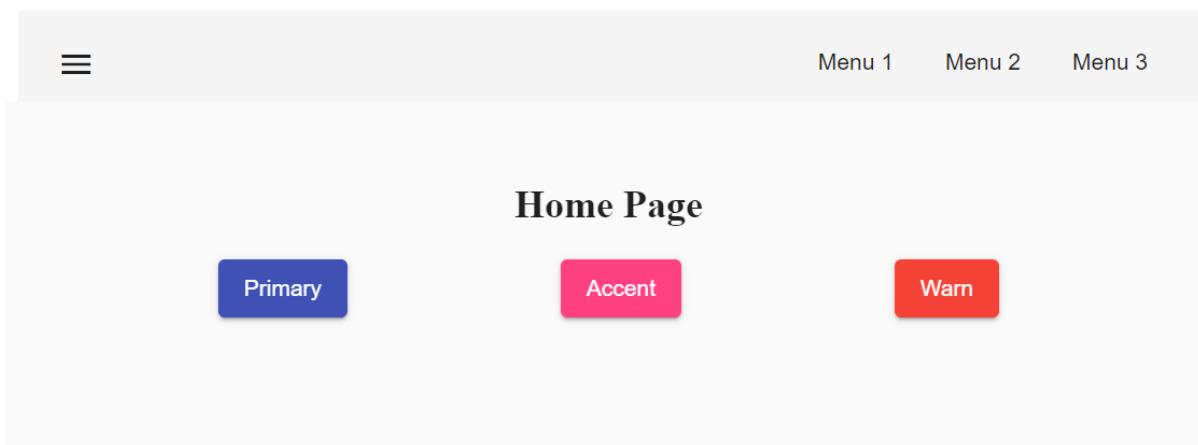


Figure 26. *indigo-pink* theme.

- `pink-bluegrey.css`

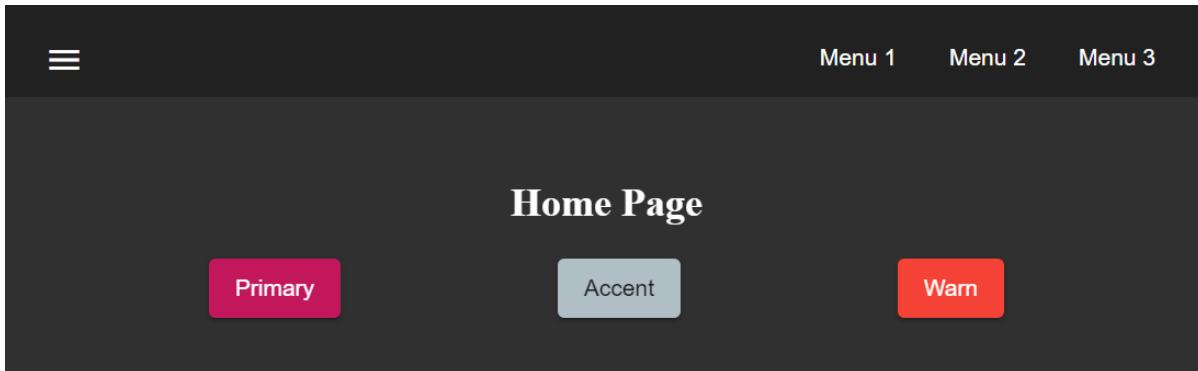


Figure 27. *ink-bluegrey* theme.

- `purple-green.css`

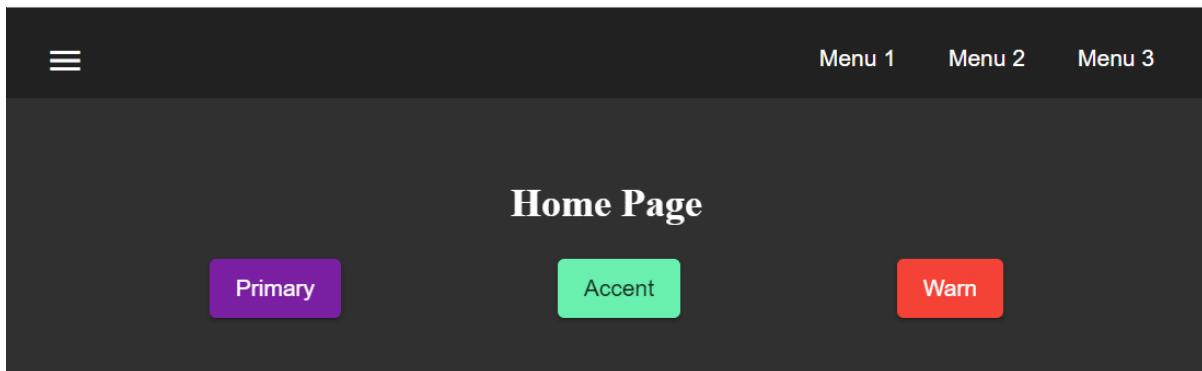


Figure 28. purple-green theme.

The prebuilt themes can be added using `@import`.

```
@import '@angular/material/prebuilt-themes/deeppurple-amber.css';
```

19.8. Custom themes

Sometimes prebuilt 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](#).

19.8.1. 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 beginning 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.

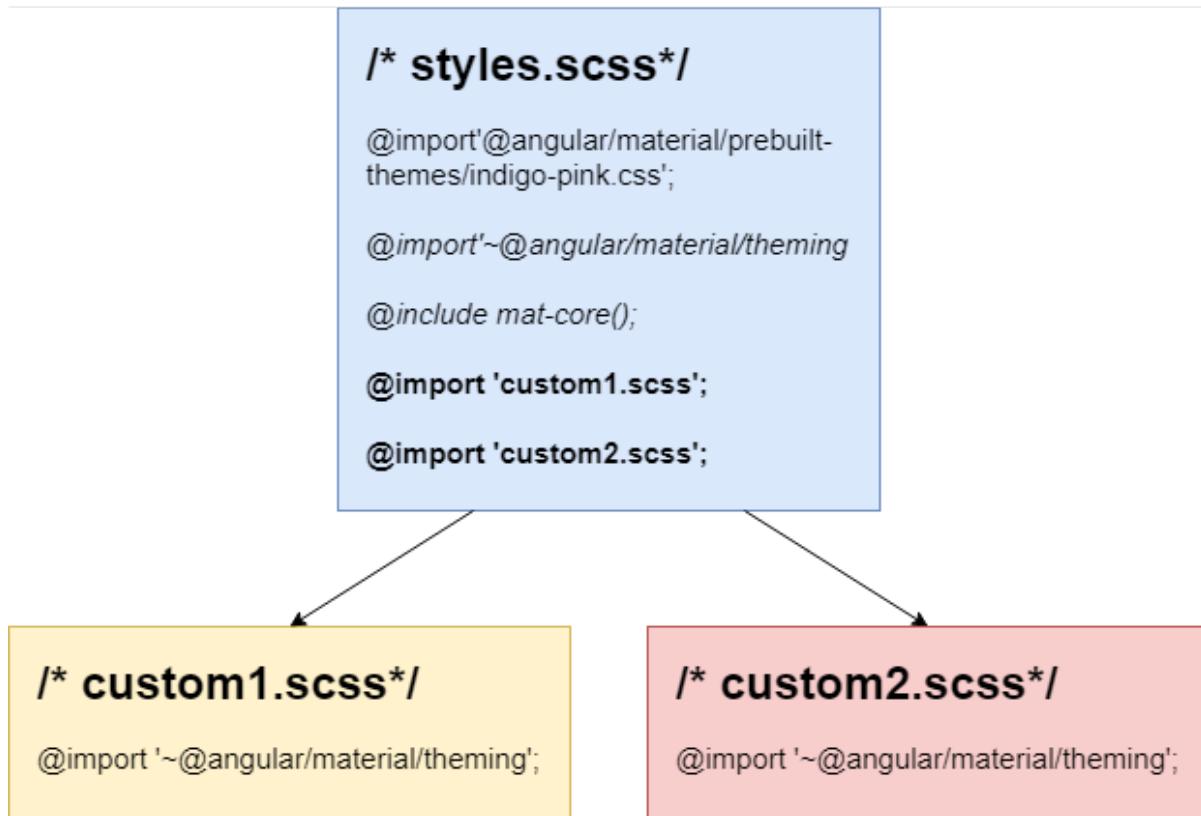


Figure 29. Theme files structure.

19.8.2. 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 24. 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 25. 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.

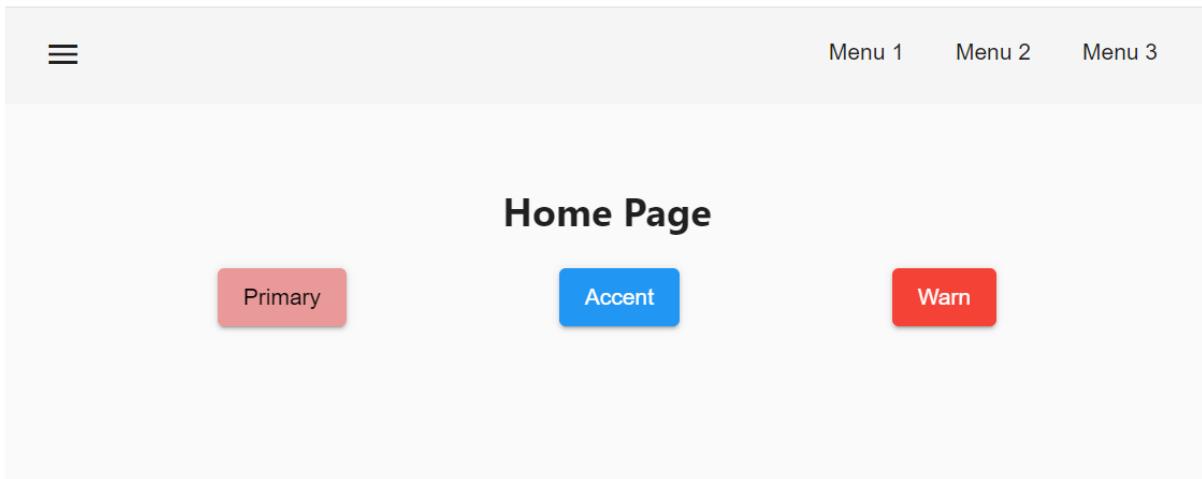


Figure 30. Custom light theme.

- **mat-dark-theme:** Similar to mat-light-theme but returns a theme whose foreground is basically white and background black.

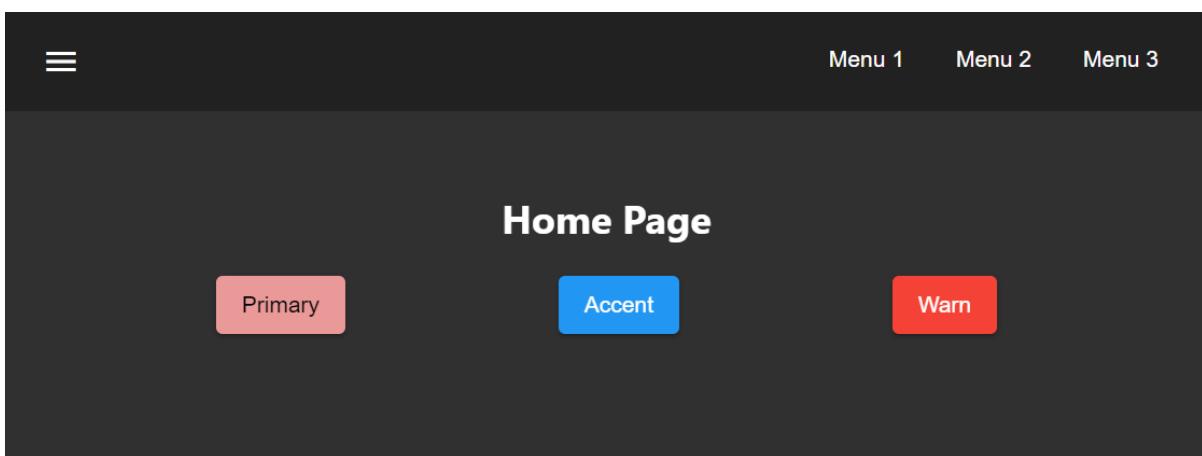


Figure 31. Custom dark theme.

For this example we are going to use mat-dark-theme and save its result in \$custom-dark-theme.

Listing 26. 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 27. 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 28. 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 29. File src/index.html.

```
...
<app-root class="custom-dark-theme"></app-root>
...
```

This will apply **\$custom-dark-theme** theme for the entire application.

19.8.3. 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 30. 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 31. 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 32. 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 33. 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 34. File styles.scss updated with custom-cap-theme class.

```
...
@import 'styles-custom-cap.scss';
.custom-cap-theme {
  @include angular-material-theme($custom-cap-theme);
}
```

19.8.4. 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 35. File app.module.ts

```
import {OverlayContainer} from '@angular/cdk/overlay';

@NgModule({
  // ...
})
export class AppModule {
  constructor(overlayContainer: OverlayContainer) {
    overlayContainer.getContainerElement().classList.add('custom-cap-theme');
  }
}
```

19.9. Useful resources

- [Angular Material's oficial theming guide](#)
- [Material Desing: Color theme creation](#)
- [Palette generator](#)
- [SCSS tutorial](#)

19.10. Angular Progressive Web App

Progressive 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 immediately (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.

19.11. Assumptions

This guide assumes that you already have installed:

- Node.js
- npm package manager
- Angular CLI

19.12. Sample Application

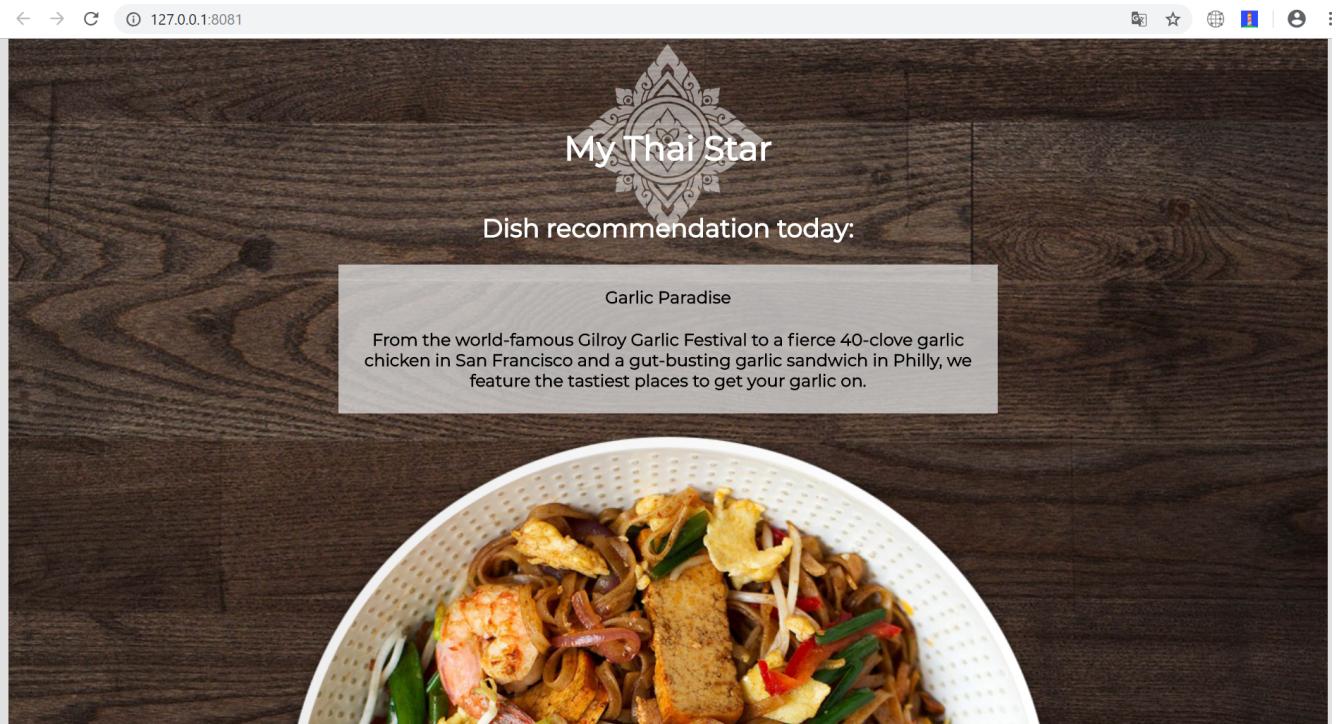


Figure 32. Basic angular PWA.

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.

19.12.1. Step 1: Create a new project

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**.

19.12.2. 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';
...
```

19.12.3. 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,
        };
      },
    );
}
}
```

19.12.4. 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());
    });
  }
  ...
}
```

19.12.5. Step 5: Make it Progressive.

Turining an angular app into a PWA is pretty easy, just one module has to be added. To do so, run: `ng add @angular/pwa`. This command also adds two important files, explained below.

- 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 tghis 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](#).

- ngs-w-config.json

nsgw-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](#).

19.12.6. 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"
      }
    }
  ]
}
```

19.12.7. 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.

```

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+02:00 (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+02:00 (GMT+02:00)] "GET /nsw-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+02:00 (GMT+02:00)] "GET /nsw.json?nsw-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+02:00 (GMT+02:00)] "GET /nsw.json?nsw-cache-bust=0.667599063360541" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36"
|
```

Figure 33. 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 es2015-polyfills.js.map (es2015-polyfills) 284 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 34. 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.

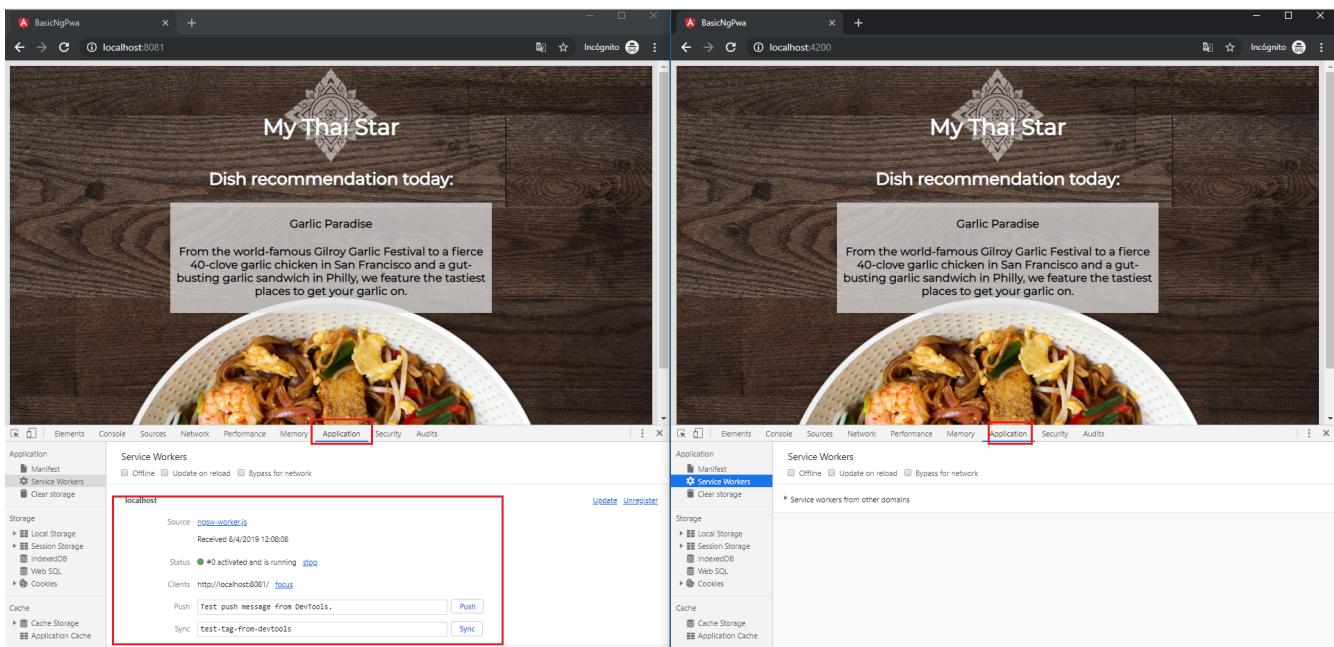


Figure 35. Application service worker comparison.

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.

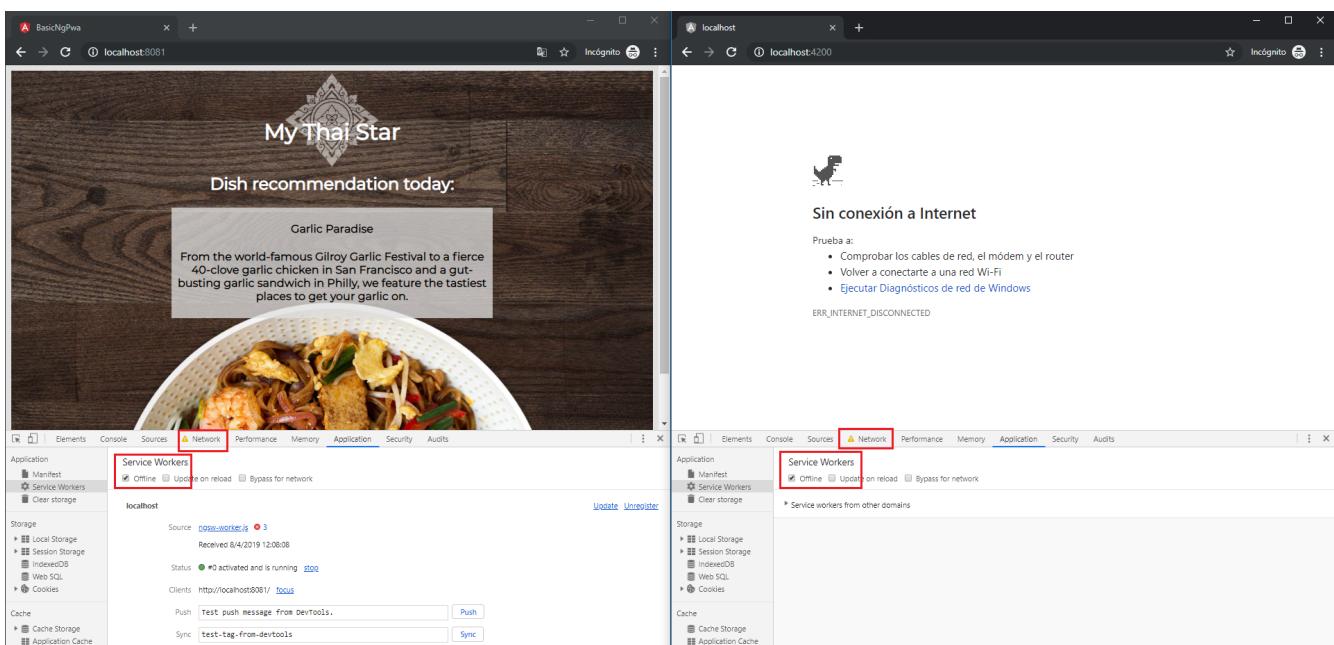


Figure 36. Offline application.

Finally, browser extensions like [Lighthouse](#) can be used to test whether an application is progressive or not.

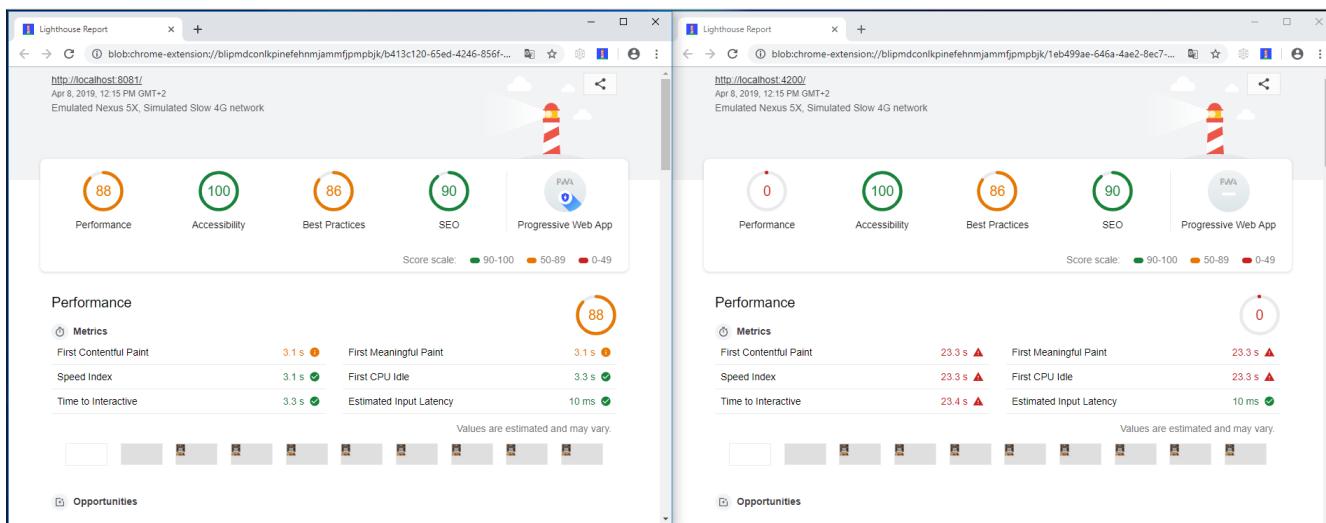


Figure 37. Lighthouse report.

19.13. APP_INITIALIZER

19.13.1. 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.

19.13.2. 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: 'Хълебарски', 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 };
```

19.13.3. 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.

19.13.4. 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"  
    ]  
    ....  
  }  
}....
```

19.13.5. 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": {  
        "^\/config": "/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",  
....
```

19.13.6. 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.

19.13.7. 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,
};
```

19.14. 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.

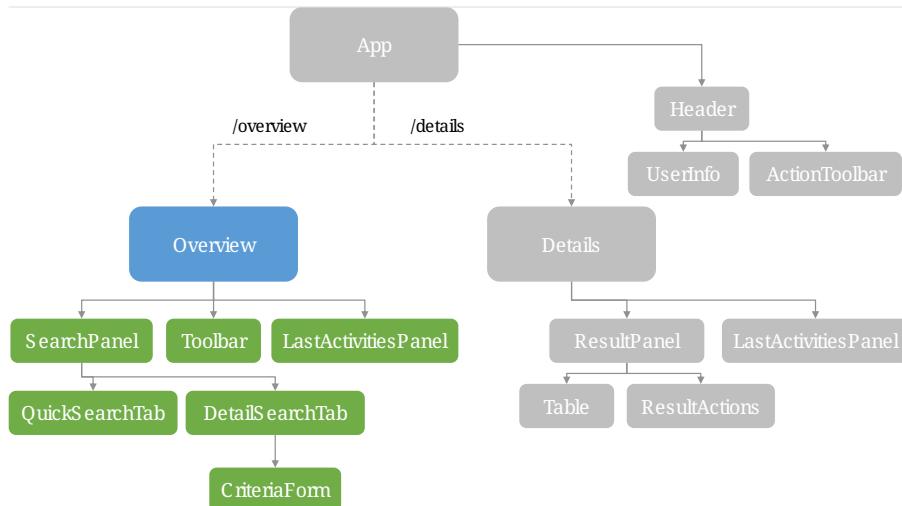


Figure 38. Component Tree with highlighted subtree

19.14.1. 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.

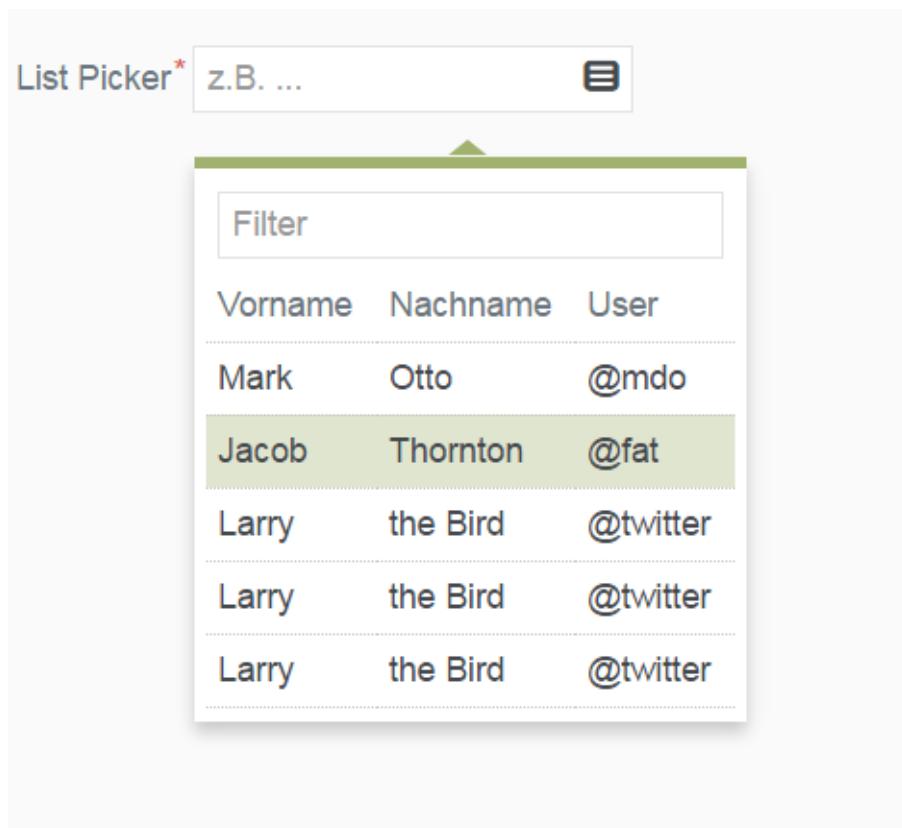


Figure 39. Component decomposition example before

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*.

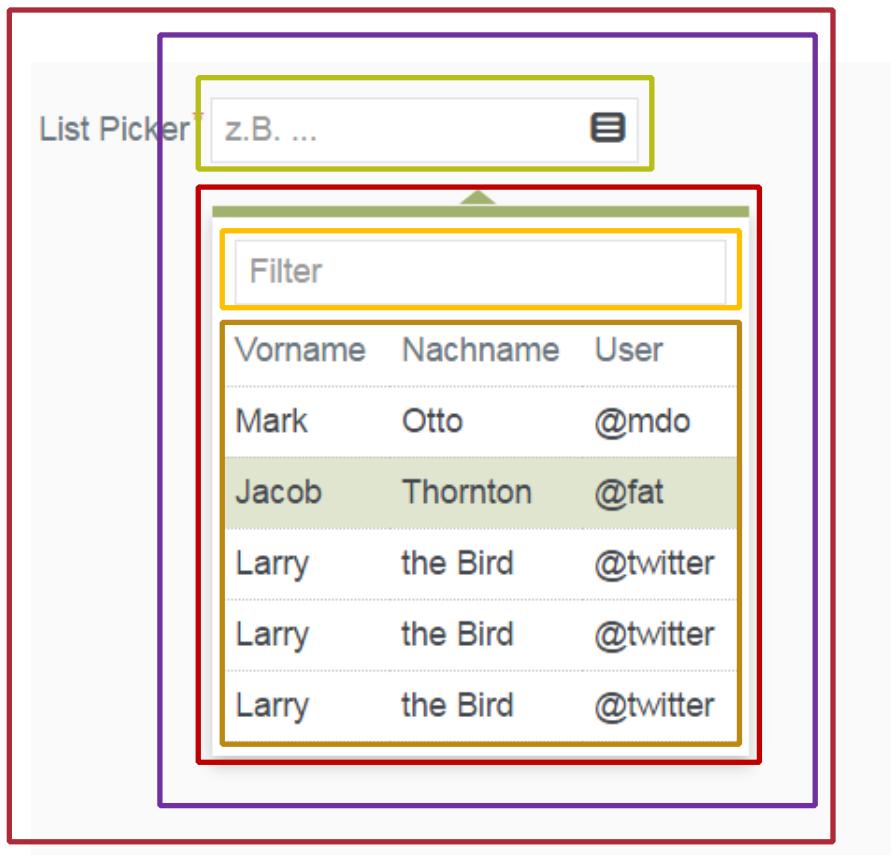


Figure 40. Component decomposition example after

This leads to the following component tree.

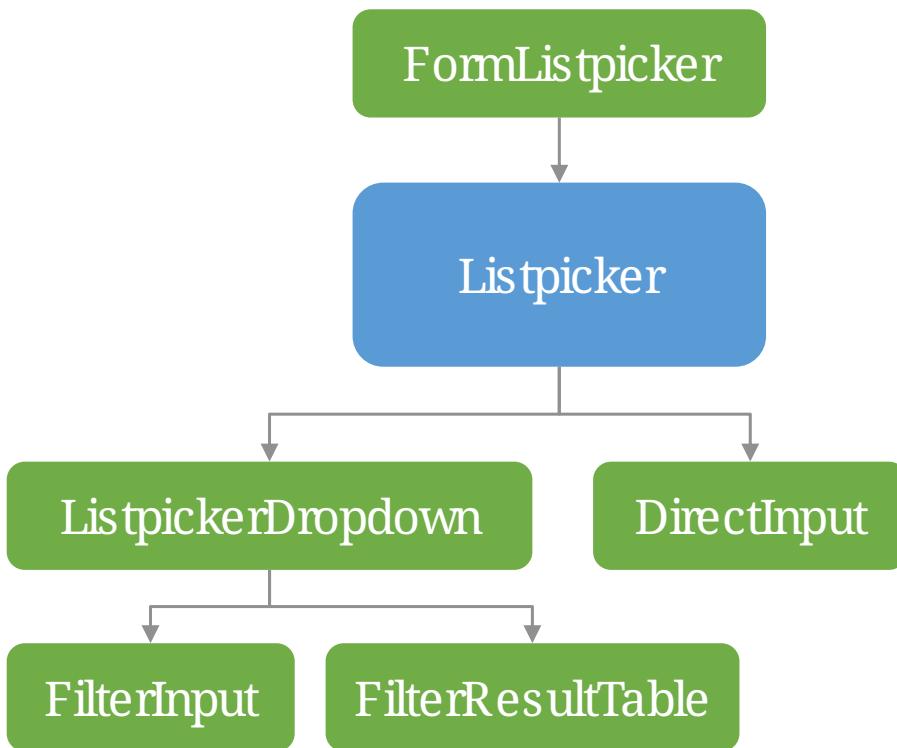


Figure 41. Component decomposition example component tree

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.

19.14.2. 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 36. 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 37. 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.

19.14.3. 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.

19.15. 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.

19.15.1. 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.

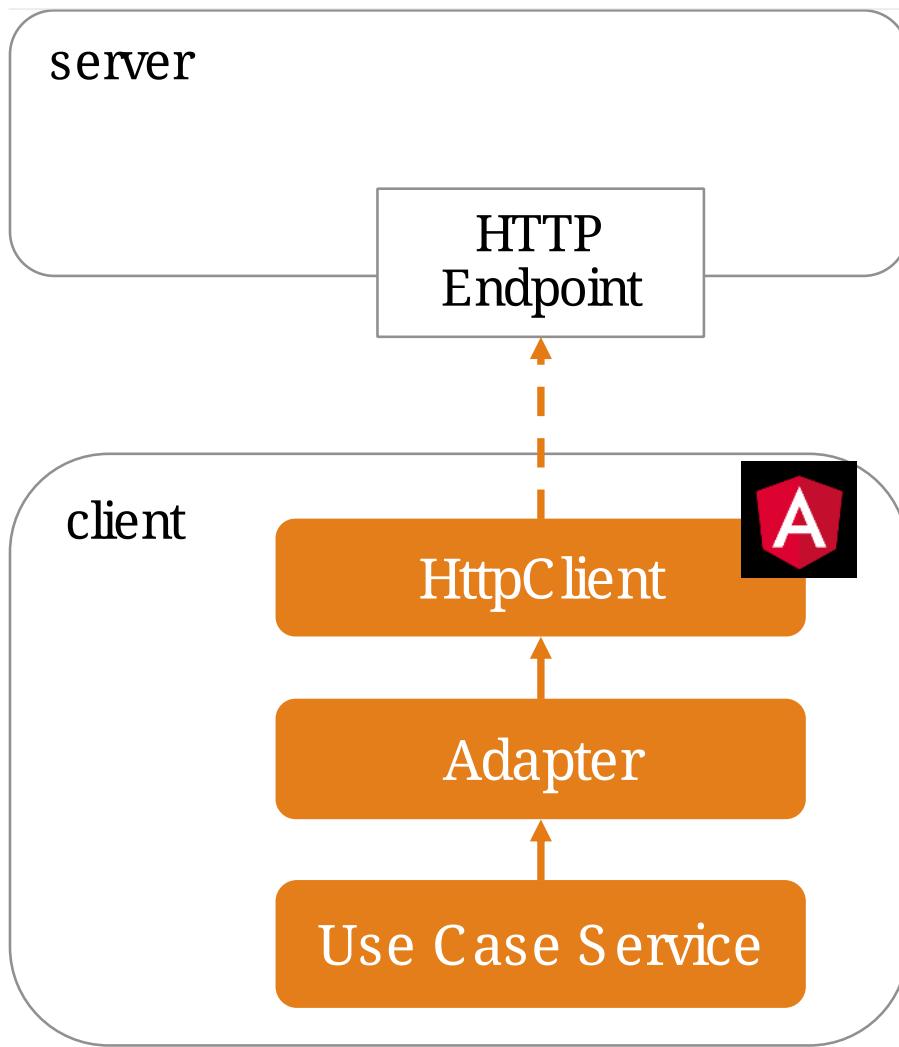


Figure 42. Adapters handle backend communication

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 38. Example adapter

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';

import { FlightTo } from './flight-to';

@Injectable()
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 39. 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()
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
  }
}
```

19.15.2. Token management

19.15.3. Global 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.

19.15.4. 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.

19.15.5. 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,  
  ....
```

19.15.6. 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.

19.16. File Structure

19.16.1. 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 40. 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.

19.16.2. Feature Modules

A feature module contains the modules definition and two folders representing both layers.

Listing 41. 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](#)).

19.16.3. Components Layer

The component layer reflects the distinction between *Smart Components* and *Dumb Components*.

Listing 42. 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 43. 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 44. 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!

19.17. 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](#)

19.17.1. devon4ng i18n approach

The official approach could be a bit complicated, therefore the recommended one is to use the recommended library **NGX Translate** from <http://www.ngx-translate.com/>.

Install NGX Translate

In order to include this library in your devon4ng **Angular >= 4.3** project you will need to execute in a terminal:

```
$ npm install @ngx-translate/core @ngx-translate/http-loader --save
# or if you use yarn
$ yarn add @ngx-translate/core @ngx-translate/http-loader
```

- **@ngx-translate/core** is the core library to provide i18n capabilities.
- **@ngx-translate/http-loader** is a loader for ngx-translate that loads translations using http.

Configure NGX Translate

Depending on the volume of the devon4ng application we will include the NGX Translate library in the `app.module.ts` or in the `core.module.ts` transversal to the application.

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule, HttpClient } from '@angular/common/http';
import { TranslateModule, TranslateLoader } from '@ngx-translate/core';
import { TranslateHttpLoader } from '@ngx-translate/http-loader';
```

Next, an exported function for factories has to be created:

```
// AoT requires an exported function for factories
export function HttpLoaderFactory(http: HttpClient) {
    return new TranslateHttpLoader(http);
}

@NgModule({
    imports: [
        BrowserModule,
        HttpClientModule,
        TranslateModule.forRoot({
            loader: {
                provide: TranslateLoader,
                useFactory: HttpLoaderFactory,
                deps: [HttpClient]
            }
        })
    ],
    bootstrap: [AppComponent]
})
export class AppModule {} // or CoreModule
```

The `TranslateHttpLoader` also has two optional parameters:

- `prefix: string = "/assets/i18n/"`
- `suffix: string = ".json"`

By using those default parameters, it will load the translations files for the lang "en" from: `/assets/i18n/en.json`. In general, any translation file will loaded from the `/assets/i18n/` folder.

Those parameters can be changed in the `HttpLoaderFactory` method just defined. For example if you want to load the "en" translations from `/public/lang-files/en-lang.json` you would use:

```
export function HttpLoaderFactory(http: HttpClient) {
    return new TranslateHttpLoader(http, "/public/lang-files/", "-lang.json");
}
```

For now this loader only support the json format.



If you're still on Angular < 4.3, please use `Http` from `@angular/http` with `http-loader@0.1.0`.

Usage

In order to translate any label in any HTML template you will need to use the `translate` pipe available:

```
 {{ 'HELLO' | translate }}
```

An **optional** parameter from the component TypeScript class could be included as follows:

```
 {{ 'HELLO' | translate:param }}
```

So, `param` has to be defined in the class. The default language used is defined as follows:

```
// imports

@Component({
  selector: 'app',
  template: `
    <div>{{ 'HELLO' | translate }}</div>          // Without param
    <div>{{ 'HELLO' | translate:param }}</div>      // With param
  `
})
export class AppComponent {
  // This param will be used in the translation
  param = { value: 'world' };

  constructor(translate: TranslateService) {
    // this language will be used as a fallback when a translation isn't found in
    the current language
    translate.setDefaultLang('en');

    // the lang to use, if the lang isn't available, it will use the current
    loader to get them
    translate.use('en');
  }
}
```

In order to change the language used you will need to create a button or selector that calls the `this.translate.use(language: string)` method from `TranslateService`. For example:

```
toggleLanguage(option) {
    this.translate.use(option);
}
```

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}}"
}
```

The `TranslateParser` 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

Service

If you need to access translations in any component or service you can do it injecting the `Translateservice` into them:

```
translate.get('HELLO', {value: 'world'}).subscribe((res: string) => {
    console.log(res);
    //=> 'hello world'
});
```

Pipe

The use of pipes can be possible too:

template:

```
<div>{{ 'HELLO' | translate:param }}</div>
```

component:

```
param = {value: 'world'};
```

Directives

Finally, it can also be used with directives:

```
<div [translate]="'HELLO'" [translateParams]="{value: 'world'}"></div>
```

or, using the content of your element as a key

```
<div translate [translateParams]="{value: 'world'}">HELLO</div>
```



You can find a complete example at <https://github.com/devonfw/devon4ng-application-template>.

Please, visit <https://github.com/ngx-translate/core> for more info.

19.18. Routing

A basic introduction to the Angular Router can be found in [Angular Docs](#).

This guide will show common tasks and best practices.

19.18.1. 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.

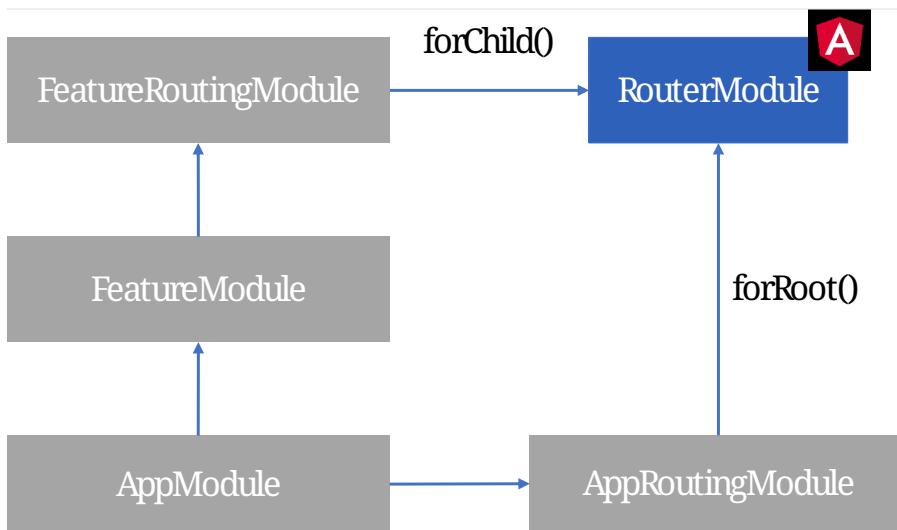


Figure 43. Routing module declaration

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 45. `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 46. 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 47. 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 48. 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 { }
```

19.18.2. 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*.

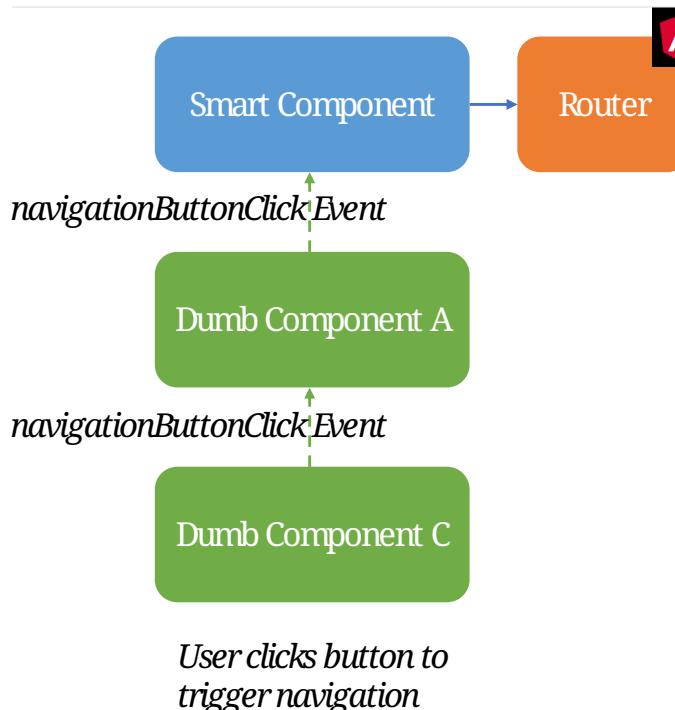


Figure 44. Triggering navigation

19.18.3. 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 49. 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 50. 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 51. 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 52. 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 53. 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 54. Implementing the guard

```

...
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'admin', loadChildren: 'app/admin/admin.module#AdminModule', canLoad:
  [ExampleCanLoadGuard] }
];

```

19.19. 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.

19.19.1. Testing Concept

The following figure shows you an overview of the application architecture devided in testing areas.

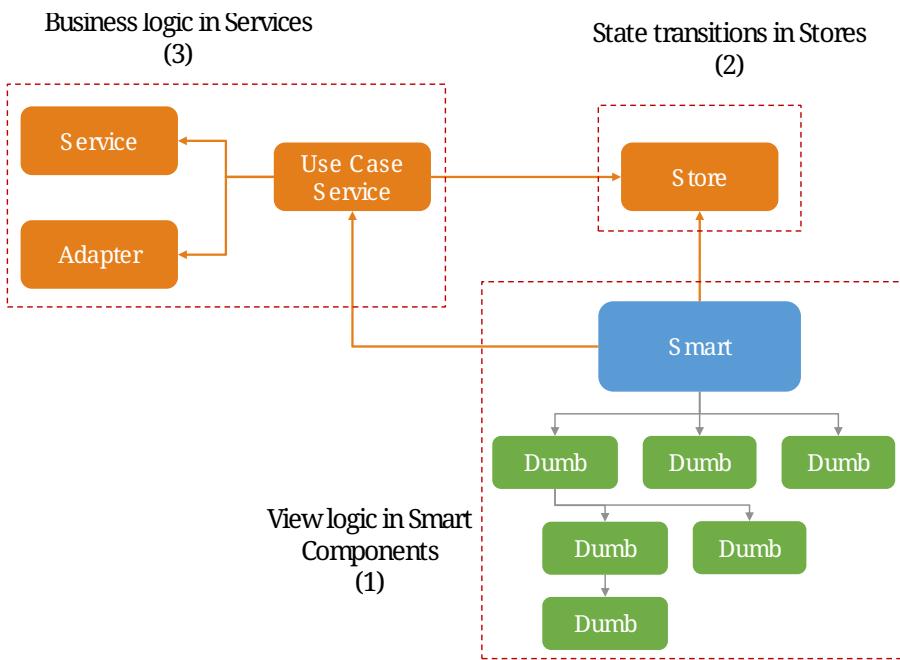


Figure 45. Testing Areas

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.

19.19.2. 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 55. Example test setup for Smart Components

```

describe('PrintFlightComponent', () => {

  let fixture: ComponentFixture<PrintCPrintFlightComponent>;
  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 56. 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.

19.19.3. 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 57. 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();
  });
});
```

19.19.4. 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 58. 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 59. 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 60. 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 61. 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)');
}));
```

19.19.5. 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.

19.19.6. 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:

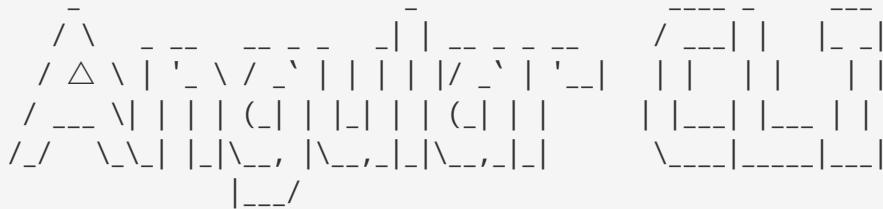
```
$ cd [devonfw_dist_folder]  
$ cd software/nodejs  
$ npm uninstall @angular/cli --no-save  
$ npm install @angular/cli --no-save
```

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:

...

19.20. 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.

19.20.1. 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 46. Printing Angular CLI Version

19.20.2. 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
```

19.20.3. 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.

19.20.4. 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.

19.20.5. 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 47. 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 48. 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.

19.20.6. 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
```

20. Ionic

20.1. Ionic: 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.

20.2. 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.

20.3. Basic environment set up

20.3.1. 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
```

20.3.2. Update Ionic CLI

To update the installed version of the CLI run:

```
npm install -g ionic@latest
```

If the devonfw's ionic CLI has to be updated, the steps are a little bit different:

- open the devonfw bundled console.
- `cd [devonfw_dist_folder]`
- `cd software/nodejs`
- `npm uninstall ionic --no-save`
- `npm install ionic@latest --no-save`

```
C:\Devon-dist-current\Devon-dist_3.0.0\software\nodejs
λ npm install ionic@latest --no-save
C:\Devon-dist-current\Devon-dist_3.0.0\software\nodejs\ionic -> C:\Devon-dist-current\Devon-dist_3.0.0\software\nodejs\node_modules\ionic\bin\ionic
+ ionic@4.10.3
added 261 packages from 174 contributors in 29.639s

C:\Devon-dist-current\Devon-dist_3.0.0\software\nodejs
λ ionic --version
4.10.3
```

20.4. Basic project set up

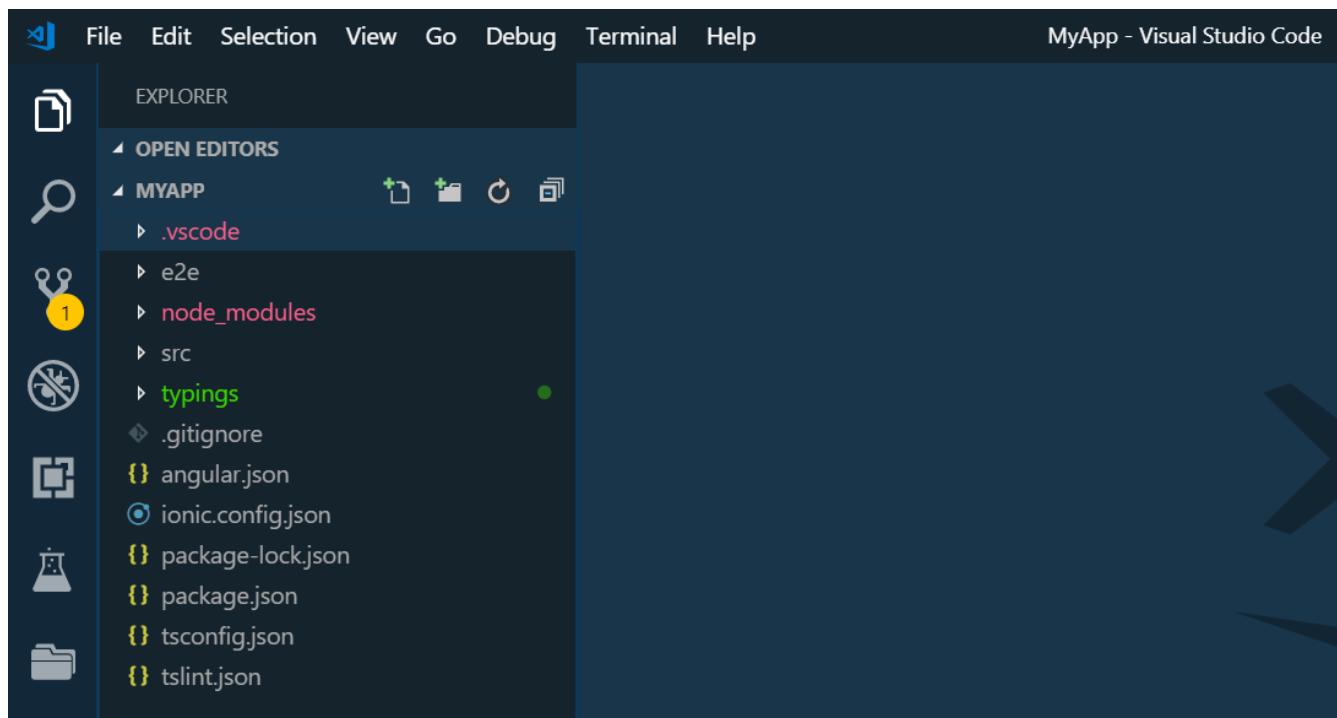
The set up of an ionic application is pretty immediate and can be done in one line:

```
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	mrchecker-template	update-ionic-cli.P	ionic-blank-proje
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 v4, while the

others belong to earlier versions (before v4).

20.5. Ionic: From code 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.

20.6. Assumptions

This document assumes that the reader has already:

- Source code of an ionic 4 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.

20.7. From ionic 4 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`

20.7.1. 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 4 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' field in 'angular.json'.
2 // ng build -prod replaces 'environment.ts' with 'environment.prod.ts'
3 // The list of file replacements can be found in 'angular.json'
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 `zone.run` package
13  * to ignore zone related error stack frames such as `zone.run`.
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 export const environment = {
2   production: false,
3 };
4
5 export const SERVER_URL = 'http://10.0.2.2:8081/';

```



```

angular.json
-----
5 "projects": {
6   "app": {
7     "root": "",
8     "sourceRoot": "src",
9     "projectType": "application",
10    "prefix": "app",
11    "schematics": {},
12    "architect": {
13      "build": {
14        "builder": "@angular-devkit/build-angular:browser",
15        "options": {
16          "outputPath": "dist/app",
17          "index": "src/index.html",
18          "main": "src/main.ts",
19          "polyfills": "src/polyfills.ts",
20          "tsConfig": "tsconfig.app.json",
21          "assets": [
22            "src/favicon.ico",
23            "src/assets"
24          ],
25          "styles": [
26            "src/styles.css"
27          ],
28          "scripts": []
29        },
30        "configurations": {
31          "production": {
32            "outputPath": "dist/app",
33            "index": "src/index.html",
34            "main": "src/main.ts",
35            "polyfills": "src/polyfills.ts",
36            "tsConfig": "tsconfig.app.json",
37            "assets": [
38              "src/favicon.ico",
39              "src/assets"
40            ],
41            "styles": [
42              "src/styles.css"
43            ],
44            "scripts": []
45          }
46        }
47      },
48      "android": {
49        "fileReplacements": [
50          {
51            "replace": "src/environments/environment.ts",
52            "with": "src/environments/environment.android.ts"
53          }
54        ]
55      },
56      "ci": {
57        "progress": false
58      }
59    }
60  },
61  "serve": {
62    "builder": "@angular-devkit/build-angular:dev-server",
63    "options": {
64      "browserTarget": "app:build"
65    },
66    "configurations": {
67      "production": {
68        "browserTarget": "app:build:production"
69      },
70      "ci": {
71        "progress": false
72      }
73    }
74  }
75}

```

```

"build": {
  ...
  "configurations": {
    ...
    "android": {
      "fileReplacements": [
        {
          "replace": "src/environments/environment.ts",
          "with": "src/environments/environment.android.ts"
        }
      ]
    },
  }
}

```

20.7.2. Build

Once configured, it is necessary to build the Ionic 4 app using this new configuration:

- `ionic build --configuration=android`

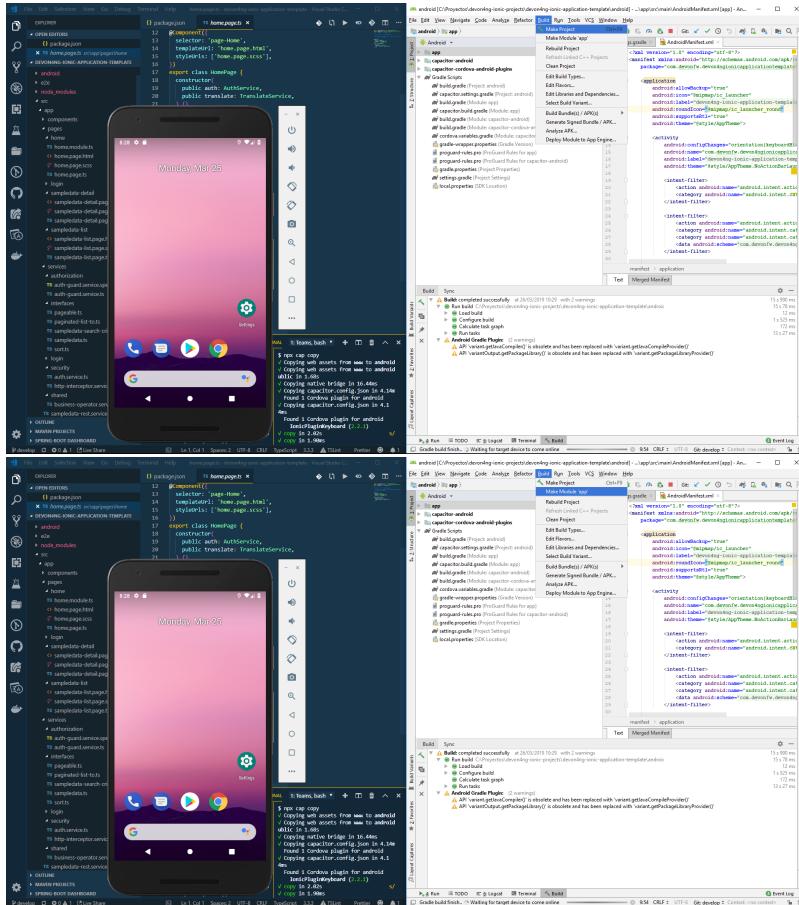
The next commands copy the build application on a folder named android and open android studio.

- `npx cap add android`
- `npx cap copy`
- `npx cap open android`

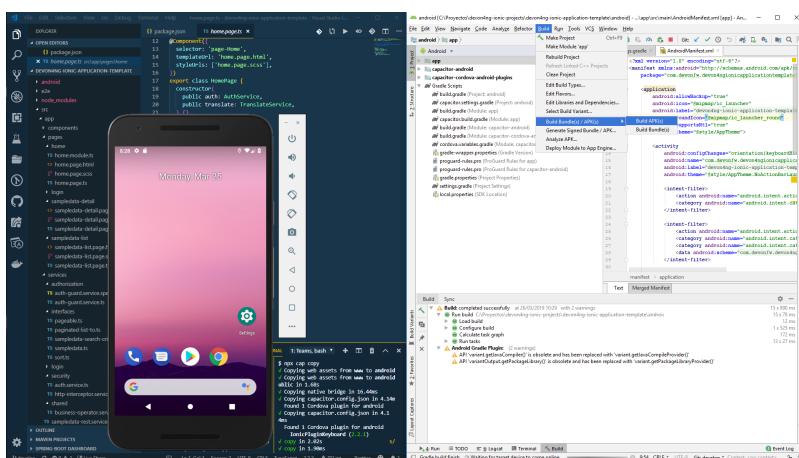
20.8. From Android project to emulated device

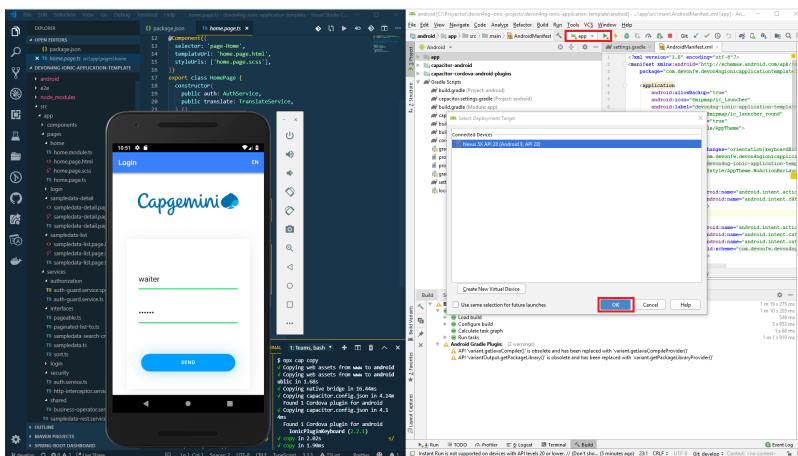
Once Android Studio is opened, follow these steps:

1. Click on "Build" → Make project.
2. Click on "Build" → Make Module 'app' (default name).



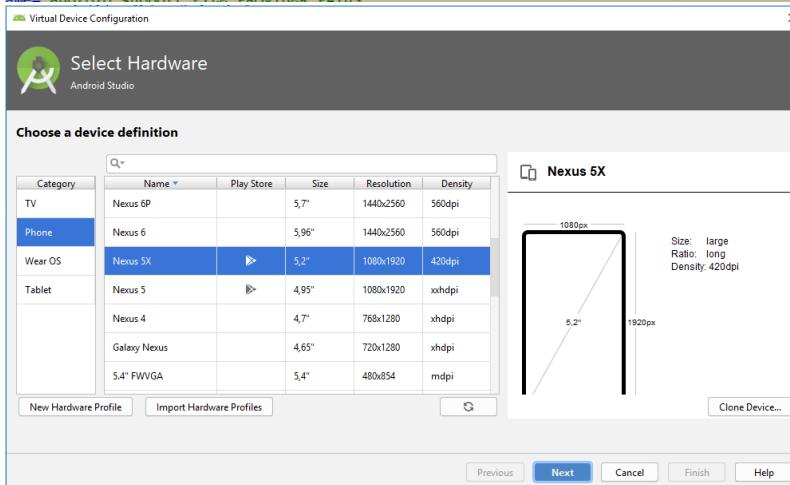
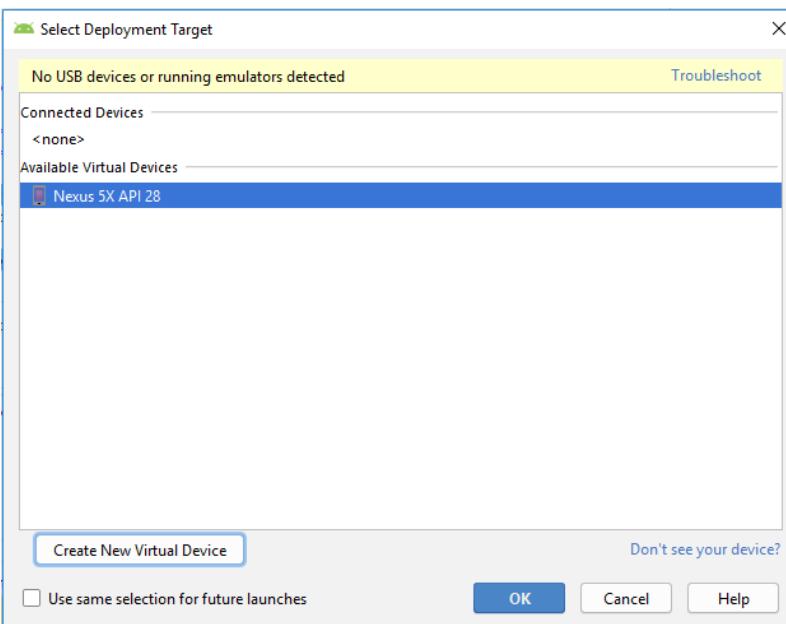
3. Click on "Build" → Build Bundle(s) / APK(s) → Build APK(s).
4. Click on run and choose a device.





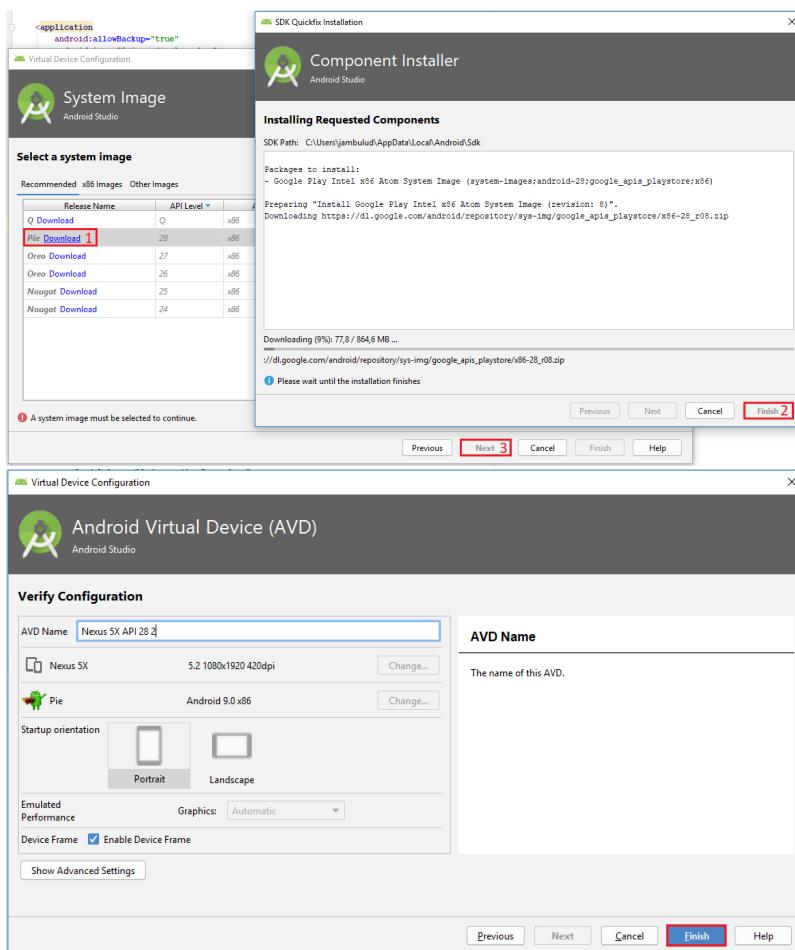
If there are no devices available, a new one can be created:

1. Click on "Create new device"
2. Select hardware and click "Next". For example: Phone → Nexus 5X.

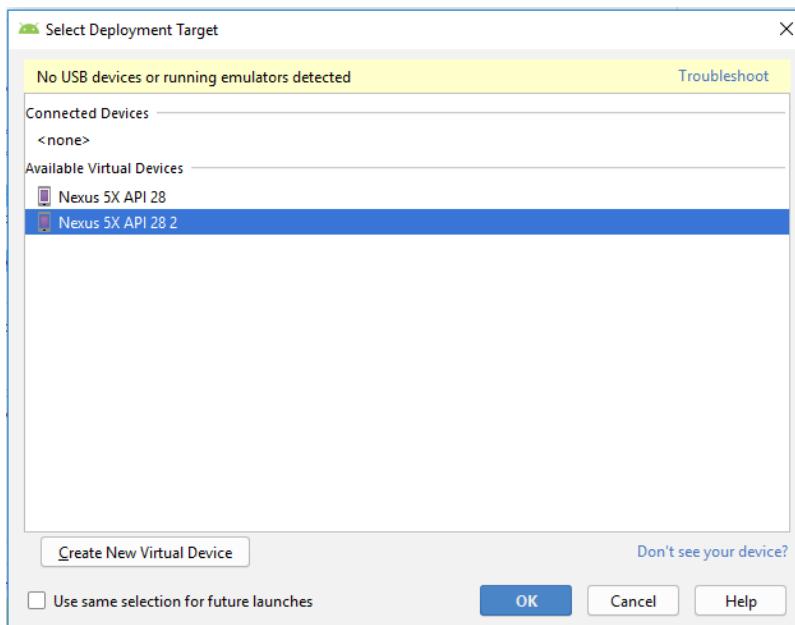


3. Download a system image.
 1. Click on download.
 2. Wait until the installation finished and then click "Finish".

3. Click "Next".
4. Verify configuration (default configuration should be enough) and click "Next".



5. Check that the new device is created correctly.



20.9. From Android project to real device

To test on a real android device, an easy aproach to comunicate 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.



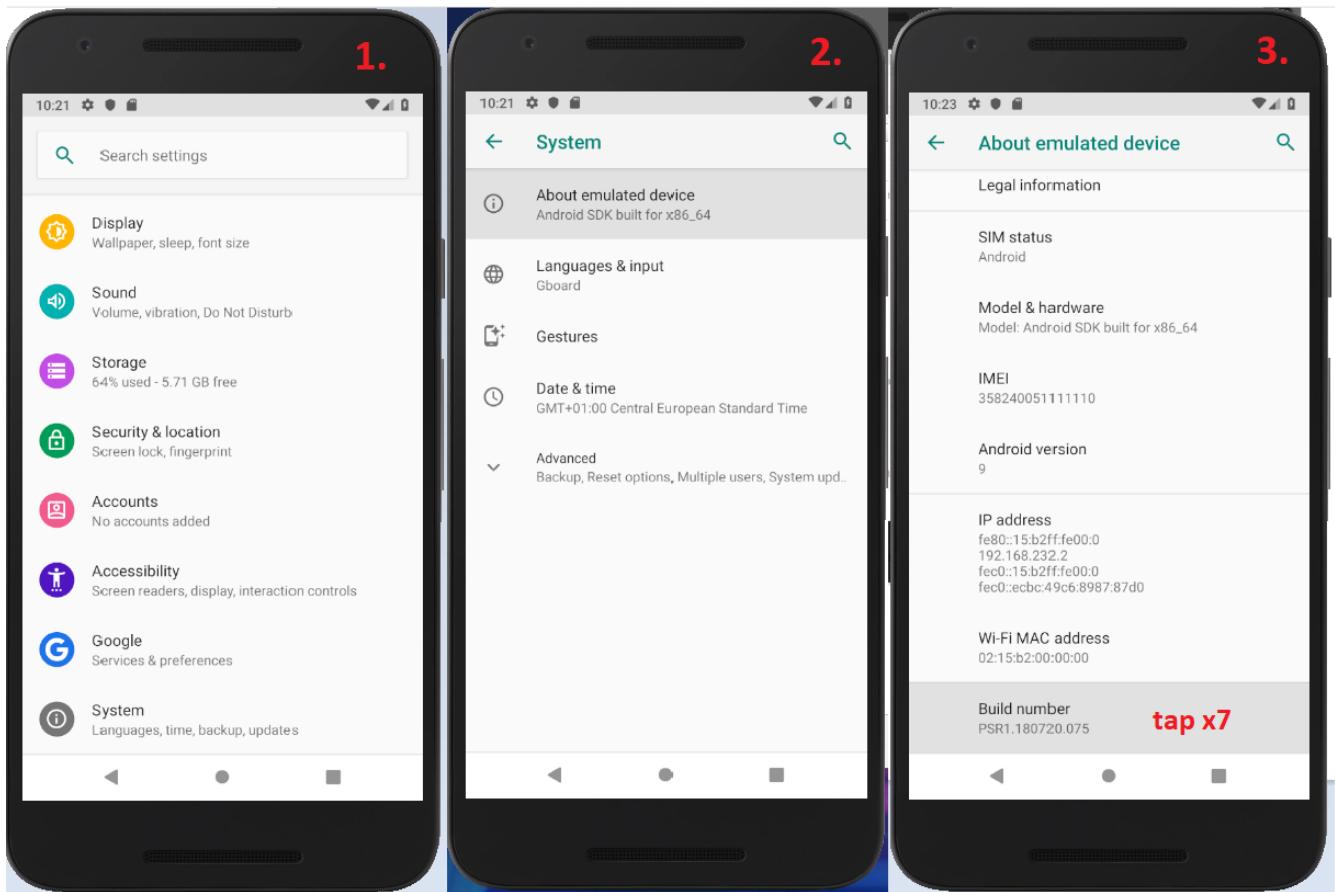
```
TS environment.android.ts x {} capacitor.config.json  
1 export const environment = {  
2   production: false,  
3 };  
4  
5 export const SERVER_URL = 'http://192.168.43.17:8081/';  
6 |
```

After this configuration, follow the build steps in "From ionic 4 to Android project" and the first three steps in "From Android project to emulated device".

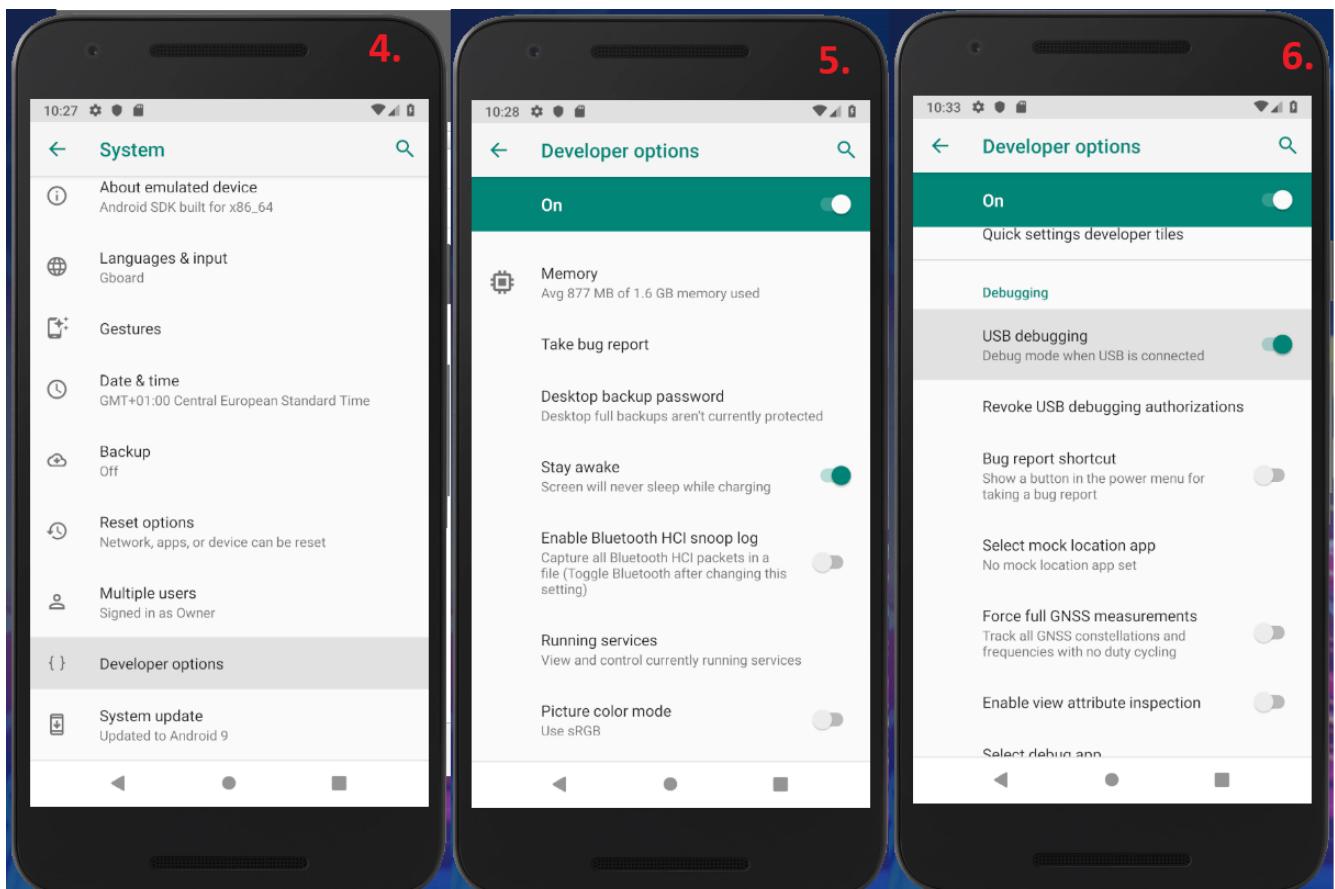
20.9.1. Send APK to Android through USB

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.



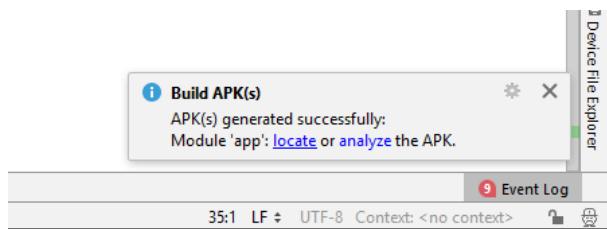
4. Go to "System" again an then to "Developer options"
5. Check that the options are "On".
6. Check that "USB debugging" is activated.



After this, do the step four in "From Android project to emulated device" and choose the connected smartphone.

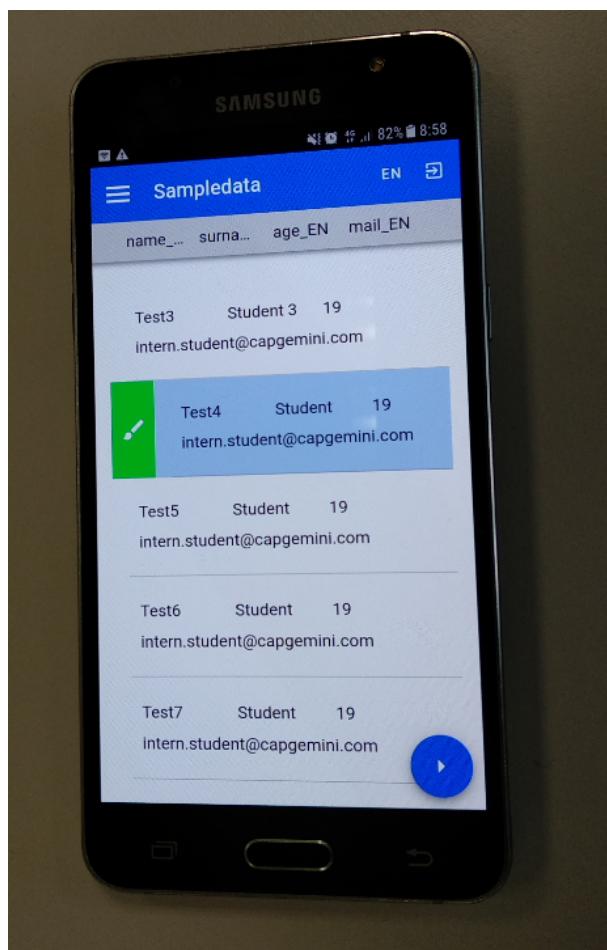
20.9.2. Send APK to Android through email

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 send using email. Download the APK on your phone and click it to install.



20.10. Result

If everything goes correctly, the Ionic 4 application will be ready to be tested.



20.11. 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.

20.12. Assumptions

This guide assumes that you already have installed:

- Node.js
- npm package manager
- Angular CLI
- Ionic 4 CLI
- Capacitor

Also, it is a good idea to read the document about PWA using Angular.

20.13. Sample Application

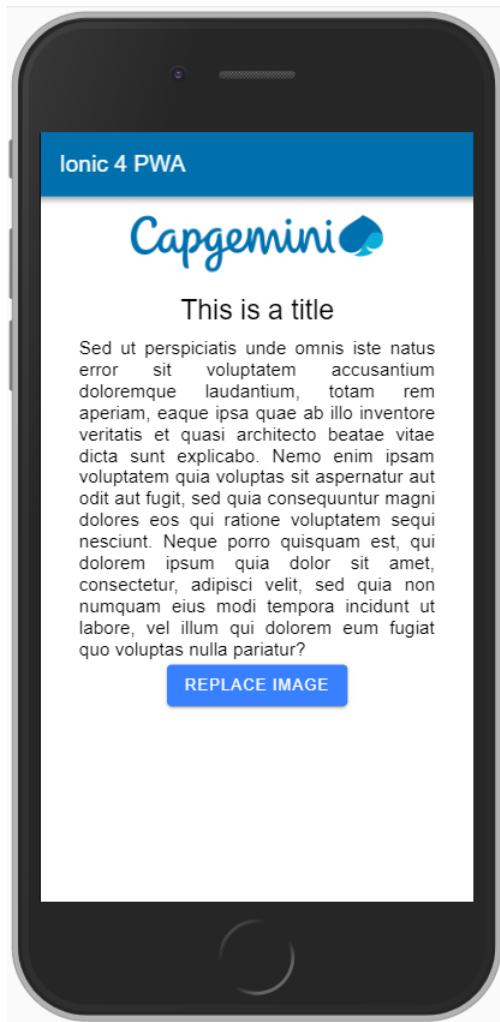


Figure 49. Basic ionic PWA.

To explain how to build progressive web apps (PWA) using Ionic 4, a basic application is going to be built. This app will be able to take photos even without network using PWA elements.

20.13.1. 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**.

20.13.2. 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](#).

20.13.3. 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,
    });
  }
}
```

20.13.4. 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);
```

20.13.5. Step 5: Make it Progressive.

Turining an ionic 4 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.

20.13.6. 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/**"]
```

20.13.7. Step 7: Check that your app is a PWA

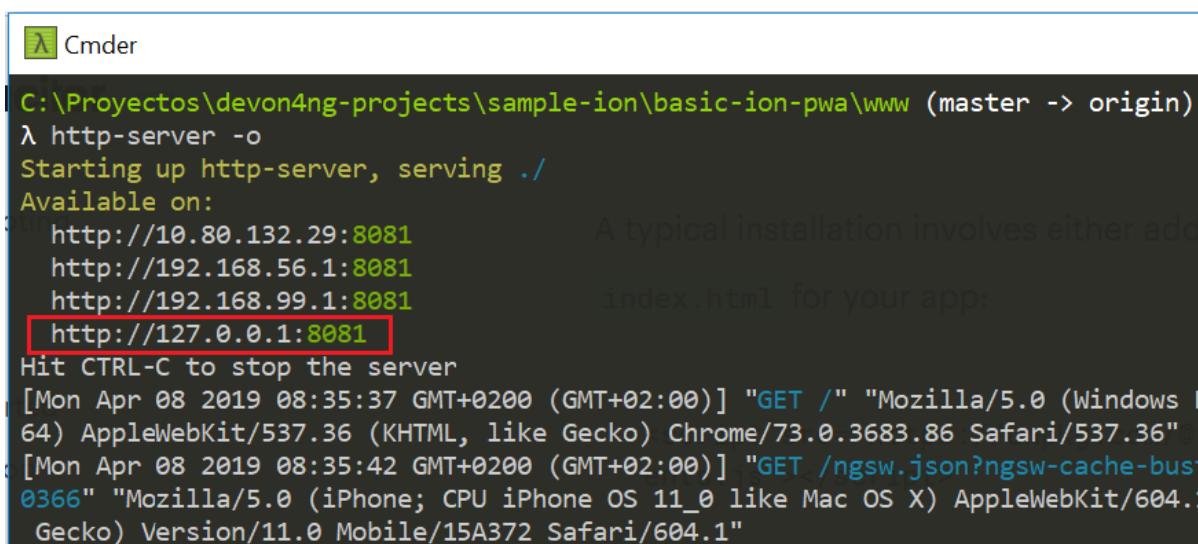
To check if an app is a PWA lets compare its normal behaviour against itself but built for production. Run in the project's root folder the commands below:

`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](#).

Go to the `www` folder running `cd www`.

`http-server -o` to serve your built app.



```
Cmder
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+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+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 50. 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 51. 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.

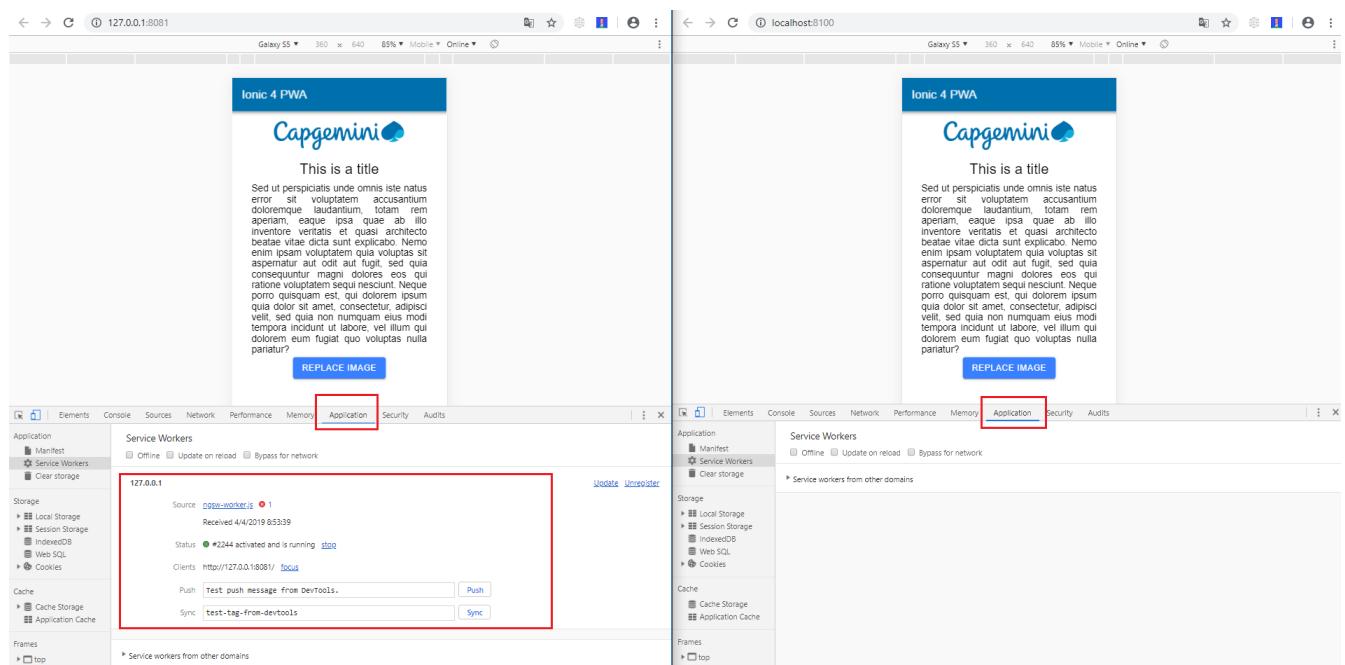


Figure 52. Application service worker comparison.

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.

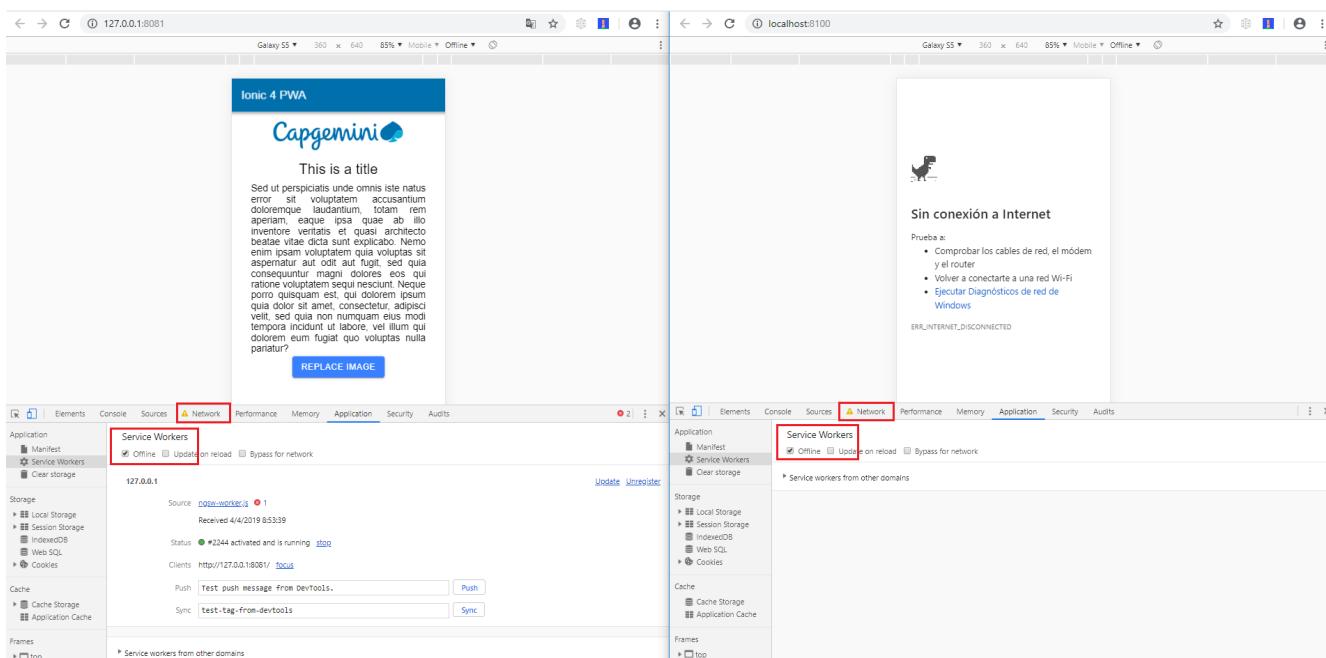


Figure 53. Offline application.

Finally, plugins like [Lighthouse](#) can be used to test whether an application is progressive or not.

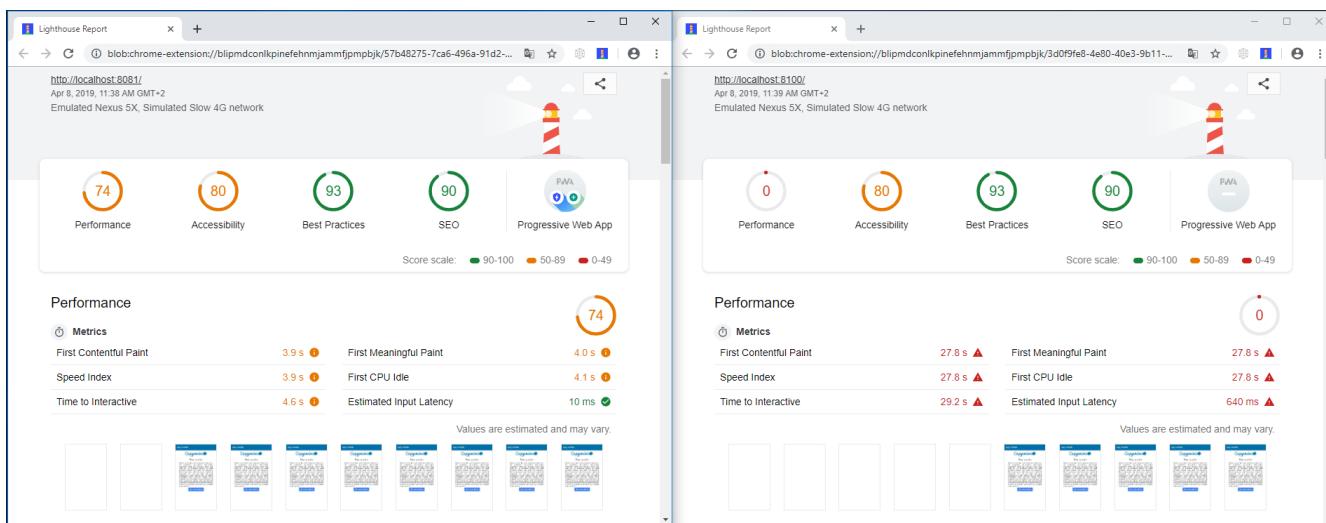


Figure 54. Lighthouse report.

21. Layouts

21.1. 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.

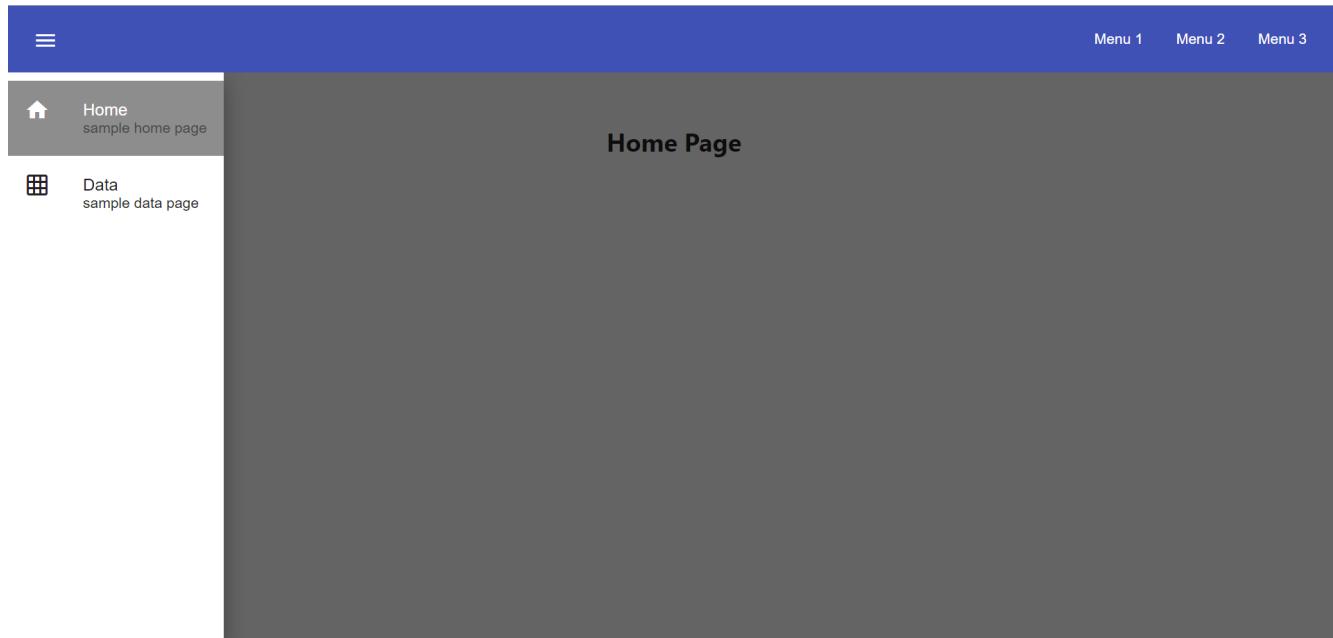


Figure 55. This is what the finished application will look like

21.2. 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 Devcon 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 56. Blank application

21.3. 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 62. 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 63. 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 64. 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 65. In src/index.html:

```
<link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
```

21.4. 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 66. 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 57. 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 67. 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 58. 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 68. `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 69. `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 59. 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 70. `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 71. 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 72. home.component.html:

```
<h2>Home Page</h2>
```

Listing 73. home.component.scss:

```
h2 {
  text-align: center;
  margin-top: 50px;
}
```

Listing 74. data.component.html:

```
<h2>Data Page</h2>
```

Listing 75. data.component.scss:

```
h2 {
  text-align: center;
  margin-top: 50px;
}
```

The pages look somewhat better now:



Home Page

Figure 60. Home page



Data Page

Figure 61. 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 76. 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 77. app.component.html:

```
...
<button mat-icon-button aria-label="menu" class="menu" (click)="sidenav.toggle()">
  <mat-icon>menu</mat-icon>
</button>
...
```

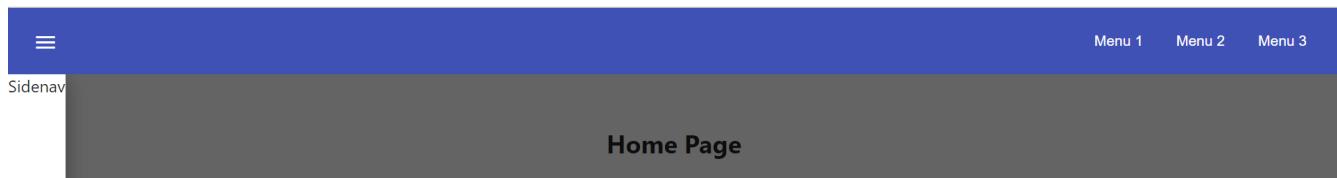


Figure 62. Sidenav is implemented

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 78. 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 79. 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 80. 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.

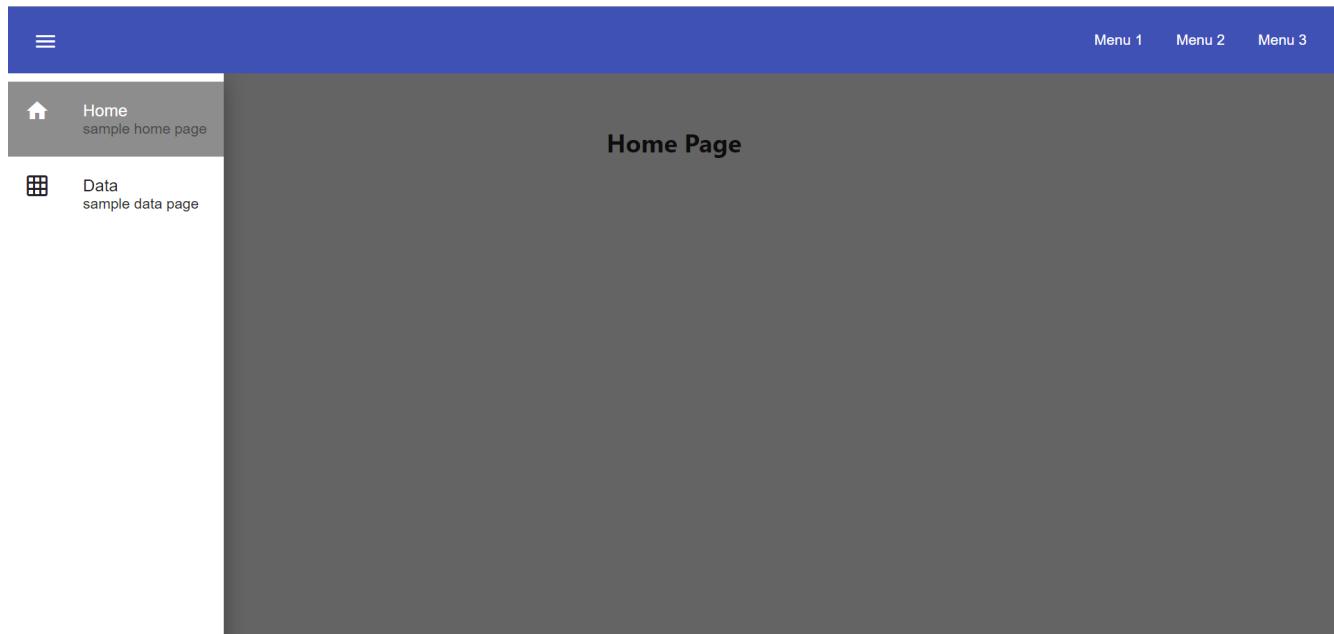


Figure 63. Finished application

21.5. 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.

22. NgRx

22.1. NgRx

NgRx is a state management framework for Angular based on the [Redux](#) pattern.

22.1.1. 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.

22.1.2. 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

22.1.3. 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](#).

22.1.4. Concept

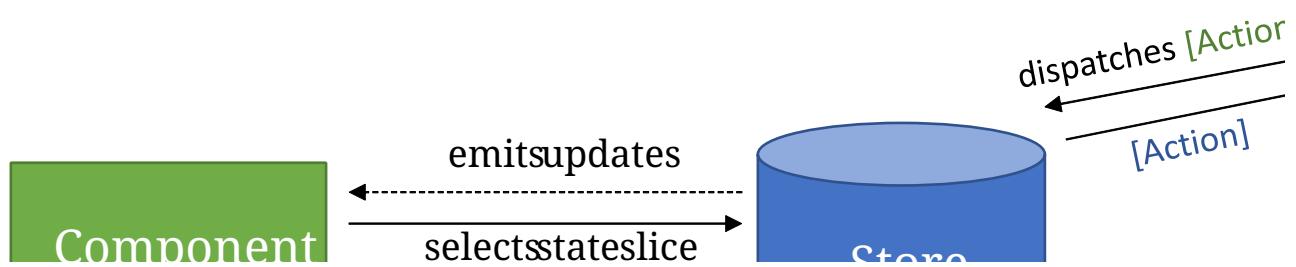


Figure 64. NgRx architecture overview

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](#).

22.2. 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.

22.2.1. 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 { }
```

22.2.2. 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(),
    }
  ]
};
```

22.2.3. 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>
```

22.2.4. 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.

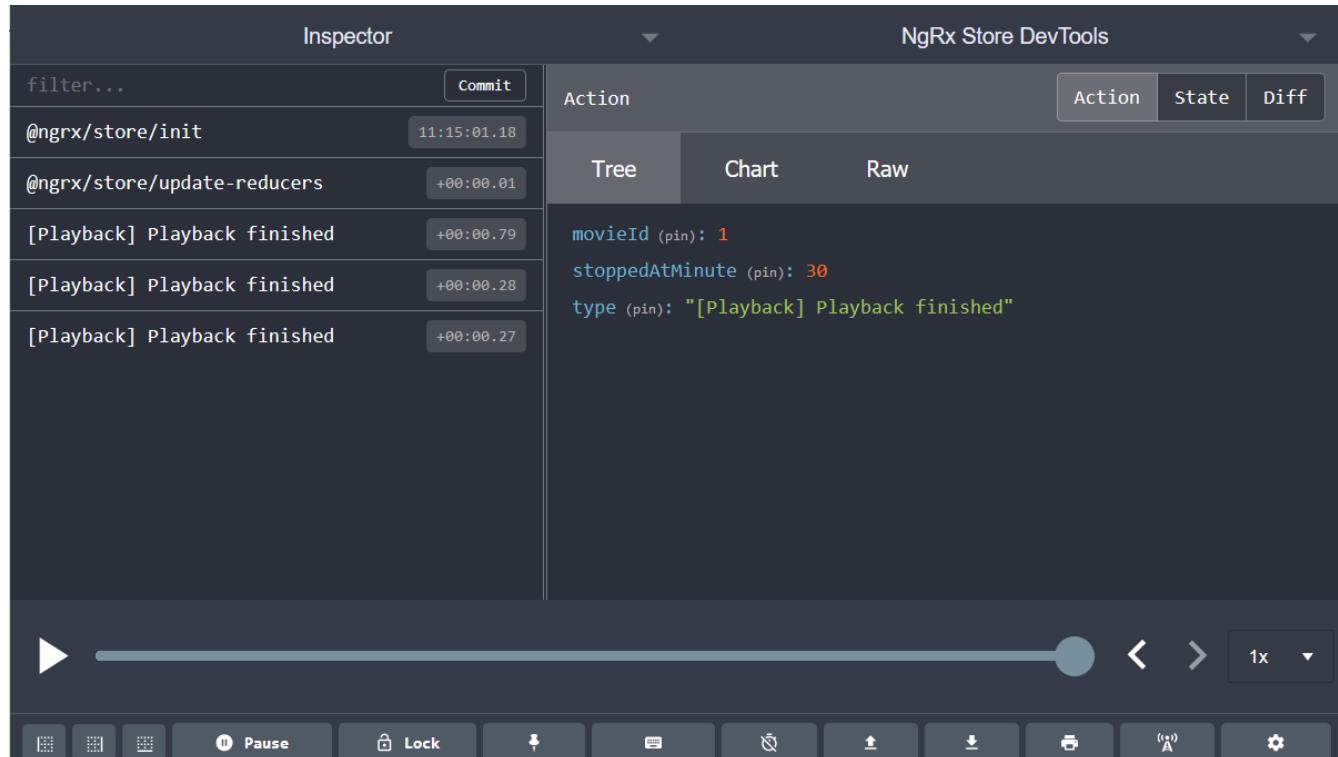


Figure 65. Redux devtools

Continue with [learning about effects](#)

22.2.5. 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.

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.](#)

22.2.6. 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](#).

23. Cookbook

23.1. 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 81. 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 82. 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();
}

}
```

23.1.1. Add Electron to an Angular application

This cookbook recipe explains how to integrate Electron in an Angular 6+ 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 7.2.3 and Electron 4.0.2 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": "^4.0.2",  
  "electron-builder": "^20.38.5",  
  "electron-reload": "^1.4.0",  
  "npm-run-all": "^4.1.5",  
  "npx": "^10.2.0",  
  "wait-on": "^3.2.0",  
  "webdriver-manager": "^12.1.1"  
  ...  
},
```

As indicated above, instead of this `npm install` can be used:

```
$ npm install -D electron electron-builder electron-reload npm-run-all npx wait-on  
webdriver-manager
```

Or with `yarn`:

```
$ yarn add -D electron electron-builder electron-reload npm-run-all npx wait-on  
webdriver-manager
```

Add Electron build configuration

In order to configure electron builds properly a `electron-builder.json` must be included in the root folder of the application. 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": "app-name",  
  "directories": {  
    "output": "release/"  
  },  
  "files": [  
    "**/*",  
    "!**/*.ts",  
    "!*.code-workspace",  
    "!LICENSE.md",  
    "!package.json",  
    "!package-lock.json",  
    "!src/",  
    "!e2e/",  
    "!hooks/",  
    "!angular.json",  
    "!_config.yml",  
    "!karma.conf.js",  
    "!tsconfig.json",  
    "!tslint.json"  
  ],  
  "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 file:

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.

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 new one named `tsconfig-serve.json` in the root folder.

tsconfig.json

This file needs to be modified to add the `main.ts` and `src/**/*` folders excluding the `node_modules`:

```
{  
  ....  
},  
"include": [  
  "main.ts",  
  "src/**/*"  
],  
"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": {  
    "sourceMap": true,  
    "declaration": false,  
    "moduleResolution": "node",  
    "emitDecoratorMetadata": true,  
    "experimentalDecorators": true,  
    "target": "es5",  
    "typeRoots": [  
      "node_modules/@types"  
    ],  
    "lib": [  
      "es2017",  
      "es2016",  
      "es2015",  
      "dom"  
    ]  
  },  
  "include": [  
    "main.ts"  
  ],  
  "exclude": [  
    "node_modules",  
    "**/*.spec.ts"  
  ]  
}
```

Modify angular.json

`angular.json` has to be modified so the project is build inside `/dist` without an intermediate folder.

```
{  
  ....  
  "architect": {  
    ....  
    "build": {  
      outputPath: "dist",  
      ....  
    }  
  }  
}
```

Add Angular Electron directives

In order to use Electron's webview tag and its methods inside an Angular application our project needs the directive `webview.directive.ts` file. We recommend to create this file inside a **shared** module folder, although it has to be declared inside the main module `app.module.ts`.

Listing 83. File `webview.directive.ts`

```
import { Directive } from '@angular/core';  
  
@Directive({  
  selector: '[webview]',  
})  
export class WebviewDirective {}
```

Add access Electron APIs

To call Electron APIs from the Renderer process, install `ngx-electron` module.

With `npm`:

```
$ npm install ngx-electron --save
```

Or with `yarn`:

```
$ yarn add ngx-electron --save
```

This package contains a module named **NgxElectronModule** which exposes Electron APIs through a service called **ElectronService**

Update `app.module.ts` and `app-routing.module.ts`

As an example, the `webview.directive.ts` file is located inside a **shared** module:

Listing 84. File app.module.ts

```
// imports
import { NgxElectronModule } from 'ngx-electron';
import { WebviewDirective } from './shared/directives/webview.directive';

@NgModule({
  declarations: [AppComponent, WebviewDirective],
  imports: [
    ...
    NgxElectronModule
    ...
  ],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

Here NgxElectronModule is also added so ElectronService can be injected wherever is needed.

After that is done, the use of hash has to be allowed so electron can reload content properly. On the [app-routing.module.ts](#):

```
...
  imports: [RouterModule.forRoot(routes,
    {
      ...
      useHash: true,
    },
  )],
```

Usage

In order to use Electron in any component class the ElectronService must be injected:

```
import { ElectronService } from 'ngx-electron';

...

constructor(
  // other injected services
  public electronService: ElectronService,
) {
  // previous code...

  if (electronService.isElectronApp) {
    // Do electron stuff
  } else {
    // Do other web stuff
  }
}
```



A list of all accessible APIs can be found at [Thorsten Hans' ngx-electron repository](#).

Create the electron window in `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, screen } from 'electron';
import * as path from 'path';
import * as url from 'url';

let win: any;
let serve: any;
const args: any = process.argv.slice(1);
serve = args.some((val) => val === '--serve');

function createWindow(): void {
  const electronScreen: any = screen;
  const size: any = electronScreen.getPrimaryDisplay().workAreaSize;

  // Create the browser window.
  win = new BrowserWindow({
    x: 0,
    y: 0,
    width: size.width,
    height: size.height,

    // Needed if you are using service workers
    webPreferences: {
      nodeIntegration: true,
      nodeIntegrationInWorker: true,
```

```
}

});

if (serve) {
  // tslint:disable-next-line:no-require-imports
  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,
    }),
  );
}

// Uncomment the following line if you want to open the DevTools by default
// 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 `main.js` when building needs to be added.

```
{
  ...
  "main": "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.

```
"scripts": {
    "postinstall": "npx electron-builder install-app-deps",
    ".": "sh .angular-gui/.runner.sh",
    "ng": "ng",
    "start": "ng serve --proxy-config proxy.conf.json -o",
    "start:electron": "npm-run-all -p serve electron:serve",
    "compodoc": "compodoc -p src/tsconfig.app.json -s",
    "test": "ng test --browsers Chrome",
    "test:ci": "ng test --browsers ChromeHeadless --watch=false",
    "test:firefox": "ng test --browsers Firefox",
    "test:ci:firefox": "ng test --browsers FirefoxHeadless --watch=false",
    "test:firefox-dev": "ng test --browsers FirefoxDeveloper",
    "test:ci:firefox-dev": "ng test --browsers FirefoxDeveloperHeadless --watch=false"
},
    "test:electron": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "ngsw-config": "npx ngsw-config dist ngsw-config.json",
    "ngsw-copy": "cp node_modules/@angular/service-worker/ngsw-worker.js dist/",
    "serve": "ng serve",
    "serve:open": "npm run start",
    "serve:pwa": "npm run build:pwa && http-server dist -p 8080",
    "serve:prod": "ng serve --open --prod",
    "serve:prodcompose": "ng serve --open --configuration=prodcompose",
    "serve:node": "ng serve --open --configuration=node",
    "build": "ng build",
    "build:pwa": "ng build --configuration=pwa --prod --build-optimizer && npm run ngsw-config && npm run ngsw-copy",
    "build:prod": "ng build --prod --build-optimizer",
    "build:prodcompose": "ng build --configuration=prodcompose",
    "build:electron": "npm run electron:serve-tsc && ng build --base-href ./",
    "build:electron:dev": "npm run build:electron -- -c dev",
    "build:electron:prod": "npm run build:electron -- -c production",
    "electron:start": "npm-run-all -p serve electron:serve",
    "electron:serve-tsc": "tsc -p tsconfig-serve.json",
    "electron:serve": "wait-on http-get://localhost:4200/ && npm run electron:serve-tsc && electron . --serve",
    "electron:local": "npm run build:electron:prod && electron .",
    "electron:linux": "npm run build:electron:prod && npx electron-builder build --linux",
    "electron:windows": "npm run build:electron:prod && npx electron-builder build --windows",
    "electron:mac": "npm run build:electron:prod && npx electron-builder build --mac"
},
```

Here the important thing to look out for is that the base href when building electron can be changed as needed. In our case:

```
"build:electron": "npm run postinstall:electron && npm run electron:serve-tsc && ng build --base-href \\\"\\\" ",
```



Some of these lines are intended to be shortcuts used in other scripts. Do not hesitate to modify them depending on your needs.

Some usage examples:

```
$ npm run electron:start          # Serve Angular app and run it inside electron  
$ npm run electron:local          # Serve Angular app for production and run it  
inside electron  
$ npm run electron:windows         # Build Angular app for production and package  
it for Windows OS
```

```
$ yarn run electron:start          # Serve Angular app and run it inside  
electron  
$ yarn run electron:local          # Serve Angular app for production and run it  
inside electron  
$ yarn run electron:windows         # Build Angular app for production and  
package it for Windows OS
```

Part V: devon4net

24. Arquitecture basics

[[architecture_guide.asciidoc_navy#introduction#]] == Introduction The [Open Application Standard Platform \(OASP\)](#) 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.

[[architecture_guide.asciidoc_navy#overview-onion-design#]] == 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.

[[architecture_guide.asciidoc_-navy#layer-specification#]] == 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:

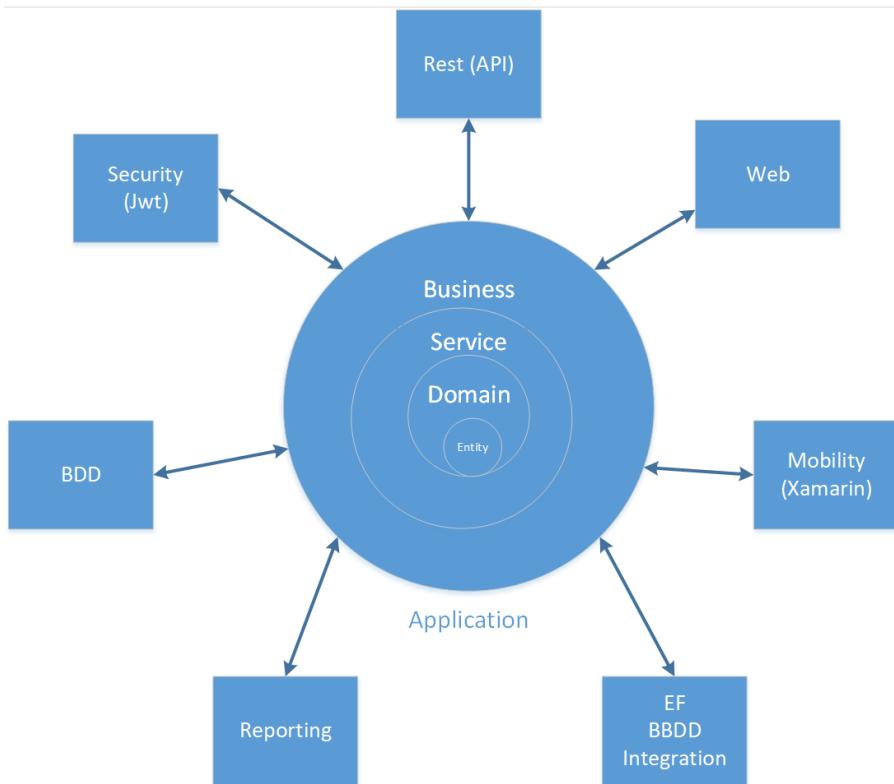


Figure 66. High level architecture representation

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.

[[architecture_guide.asciidoc_navy#application-layer#]] ===== 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.

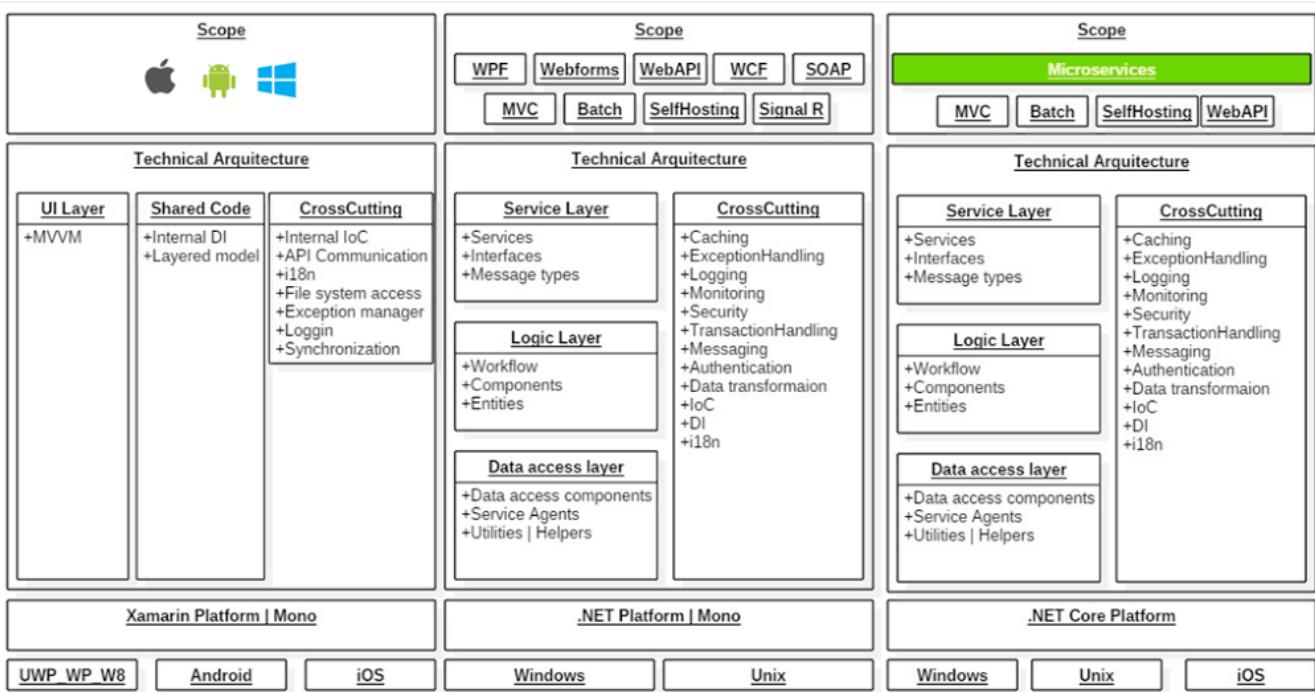


Figure 67. Net application stack

The given application template integrates Swagger contract automatic generation. This provides the possibility to external applications (angular, mobile apps, external services...) to consume the data from a well defined exposed contract.

[[architecture_guide.asciidoc_navy#business-layer]] ===== 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 exposure. This allow the data to be optimized and ready for different data consumers.

[[architecture_guide.asciidoc_navy#service-layer]] ===== Service layer The service 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.

In order to be as flexible as the implementation of *Repository Pattern* in the *Data Layer*, each service implementation inherits from EntityService class:

```
public class Service<TContext> : IService where TContext: DbContext
```



Once more <T> is the mapped class which reference the entity from the database context. This abstraction allows to write services implementation with different database contexts

[[architecture_guide.asciidoc_navy#domain-layer]] ===== Domain layer

The data layer provides access to data directly exposed from other systems. The main source use 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.

[[architecture_guide.asciidoc_navy#cross-cutting-concerns#]] ===== 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.

[[architecture_guide.asciidoc_navy#communication-between-layers-interfaces#]] ==

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.

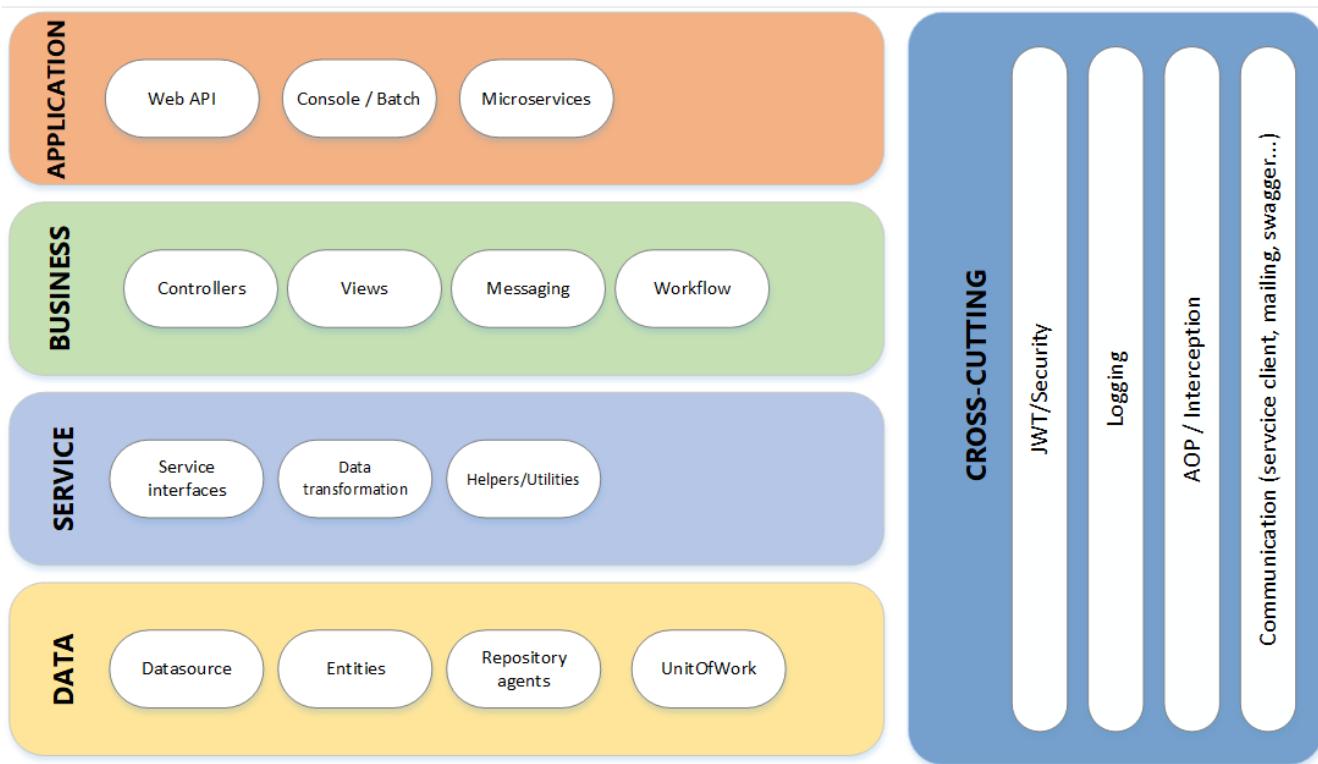


Figure 68. Architecture representation in deep

[[architecture_guide.asciidoc_navy#templates#]] == Templates
 [[architecture_guide.asciidoc_navy#state-of-the-art#]] === State of the art

The provided bundle contains two .Net templates (Classic .Net Framework 4.5+ and .Net Core Framework).

Both templates share the same architecture. the current version contains the next functionalities implemented:

	.NET Framework	.NET Core
WebAPI Console	X	X
Controller abstraction	X	X
Data access repository pattern (Generics & ASync)	X	X
Unit of work implementation		X
DI (Microsoft.Extensions.DependencyInjection)		X
Aspect oriented programming support for controllers (ActionFilterAttribute)		X
Log to file support (Serilog)		X
Swagger API doc auto generation from summary comments (Swashbuckle)	X	X
Json web token support		X
Cross-origin resource sharing (CORS) support		X
Controller/Service/Repository Tobago patterns templates		X
Dockerizable solution		X
Nuget Package sample	X	X
Nuget Server instance	X	X

Figure 69. Current available functionality

[[architecture_guide.asciidoc_navy#software-stack#]] === Software stack

Table 21. Technology Stack of OASP

Topic	Detail	Implementation
runtime	language & VM	Microsoft .Net 4.6 oder .Net Core Version
persistence	OR-mapper	Entity Framework Core / Entity Framework 6 - Code TBD
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

[[architecture_guide.asciidoc_navy#target-platforms#]] === Target platforms

Thanks to the new .Net Core platform from Microsoft, the developed software can be published Windows, Linux, OS X and Android platforms.

The compete RID (Runtime Identifier) catalog is this:

- Windows
 - Portable
 - win-x86
 - win-x64
 - Windows 7 / Windows Server 2008 R2
 - win7-x64
 - win7-x86
 - Windows 8 / Windows Server 2012
 - win8-x64
 - win8-x86
 - win8-arm
 - Windows 8.1 / Windows Server 2012 R2
 - win81-x64
 - win81-x86
 - win81-arm
 - Windows 10 / Windows Server 2016

- win10-x64
- win10-x86
- win10-arm
- win10-arm64
- Linux
 - Portable
 - linux-x64
 - CentOS
 - centos-x64
 - centos.7-x64
 - Debian
 - debian-x64
 - debian.8-x64
 - Fedora
 - fedora-x64
 - fedora.24-x64
 - fedora.25-x64 (.NET Core 2.0 or later versions)
 - fedora.26-x64 (.NET Core 2.0 or later versions)
 - Gentoo (.NET Core 2.0 or later versions)
 - gentoo-x64
 - openSUSE
 - opensuse-x64
 - opensuse.42.1-x64
 - Oracle Linux
 - ol-x64
 - ol.7-x64
 - ol.7.0-x64
 - ol.7.1-x64
 - ol.7.2-x64
 - Red Hat Enterprise Linux
 - rhel-x64
 - rhel.6-x64 (.NET Core 2.0 or later versions)
 - rhel.7-x64
 - rhel.7.1-x64
 - rhel.7.2-x64

- rhel.7.3-x64 (.NET Core 2.0 or later versions)
- rhel.7.4-x64 (.NET Core 2.0 or later versions)
- Tizen (.NET Core 2.0 or later versions)
 - tizen
- Ubuntu
 - ubuntu-x64
 - ubuntu.14.04-x64
 - ubuntu.14.10-x64
 - ubuntu.15.04-x64
 - ubuntu.15.10-x64
 - ubuntu.16.04-x64
 - ubuntu.16.10-x64
- Ubuntu derivatives
 - linuxmint.17-x64
 - linuxmint.17.1-x64
 - linuxmint.17.2-x64
 - linuxmint.17.3-x64
 - linuxmint.18-x64
 - linuxmint.18.1-x64 (.NET Core 2.0 or later versions)
- OS X
 - osx-x64 (.NET Core 2.0 or later versions)
 - osx.10.10-x64
 - osx.10.11-x64
 - osx.10.12-x64 (.NET Core 1.1 or later versions)
- Android
 - android
 - android.21

[[architecture_guide.asciidoc_navy#external-links#]] == 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

25. Coding conventions

[[codeconvention.asciidoc_navy#introduction#]] == 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)

[[codeconvention.asciidoc_navy#capitalization-conventions#]] == Capitalization Conventions
[[codeconvention.asciidoc_navy#terminology#]] === 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.

[[codeconvention.asciidoc_navy#general-naming-conventions#]] == 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 nonalphanumeric 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

[[codeconvention.asciidoc_navy#names-of-assemblies-and-dlls#]] == 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 o 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.

[[codeconvention.asciidoc_navy#general-coding-style#]] == 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 flowerbox.
- 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
Porperty	Pascal case. Try to use get and set convention {get;set;}
Constant	Pascal case
Enum	Pascal case
Variable (inline)	Camel case
Param	Camel case

[[codeconvention.asciidoc_navy#use-of-region-guideline#]] == 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.

[[codeconvention.asciidoc_navy#use-of-comment-guideline#]] == Use of Comment guideline Code is the only completely reliable documentation: write “good code” first!

[[codeconvention.asciidoc_navy#avoid-unnecessary-comments#]] === 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.

[[codeconvention.asciidoc_navy#effective-comments#]] === 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

[[codeconvention.asciidoc_navy#external-links#]] == External links [Naming guidelines](#)

[General naming conventions](#)

[Capitalization conventions](#)

[Assembly and Name Spaces conventions](#)

26. Environment

[[environment.asciidoc_navy#overview#]] == Overview

[[environment.asciidoc_navy#required-software#]] == Required software [Visual Studio Code](#)

[C# Extension for VS Code](#)

[.Net Core SDK](#)

[[environment.asciidoc_navy#setting-up-the-environment#]] == Setting up the environment .
Download and install [Visual Studio Code](#)

1. Download and install [.Net Core SDK](#)
2. [Intall the extension Omnisharp](#) in Visual Studio Code

[[environment.asciidoc_navy#hello-world#]] === Hello world . 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'.
 1. 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**.
 1. 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.
 1. Run the "Hello World" program:
- Type **dotnet run**.

[[environment.asciidoc_navy#debug#]] === 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](#).

[[environment.asciidoc_navy#external-links#]] == External links

[.Net Core](#)

[Using .NET Core in Visual Studio Code](#)

[.Net Core in Visual Studio Code tutorial](#)

27. User guide



27.1. OASP4NET Guide

[[userguide.asciidoc_navy#introduction#]] == Introduction

Welcome to OASP4Net 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.

[[userguide.asciidoc_navy#the-package#]] == The package

Devon4Net package solution contains:

File / Folder	Content
Documentation	User documentation in HTML format
Samples	Different samples implemented in .NET and .NET Core. Also includes My Thai Star Devon flagship restaurant
Template	Main .net and .NET Core templates to start developing from scratch
License	License agreement
README.md	Github main page

[[userguide.asciidoc_navy#application-templates#]] === Application templates

The application templates given in the bundle are ready to use.

At the moment .NET Core and .NET templates are supported. Each template is ready to be used as a simple console application or being deployed in a web server like IIS.

[[userguide.asciidoc_navy#samples#]] === Samples

[[userguide.asciidoc_navy#my-thai-star#]] ===== My Thai Star

You can find My Thai Star .NET port application at [Github](#).

[[userguide.asciidoc_navy#gmailapiconsumer#]] === GMailAPIConsumer The GMailAPIConsumer sample contains both implementations (.NET and :ET Core) of a microservice able to connect to Google API services in order to send emails. The Microservice uses the My Thai Star email address to show how to communicate with Google API.

[[userguide.asciidoc_navy#multiplatform#]] === Multiplatform The main purpose of this sample is how to deploy the .NET Core template to different platforms. The sample shows how to build a micro webserver able to be deployed to Linux and iOS.

The main purpose of this sample is how to deploy the .NET Core template to different platforms. The sample shows how to build a micro webserver able to be deployed to Linux and iOS.

Please take a look to the .csproj file in order to see how it works. The next lines in the .csproj show how achieve this:

```
<PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
    <RuntimeIdentifiers>win10-x64;osx.10.11-x64;ubuntu.14.04-x64</RuntimeIdentifiers>
</PropertyGroup>
```

[[userguide.asciidoc_navy#cookbook#]] == Cookbook [[userguide.asciidoc_navy#data-management#]] === Data management To use EF Core, install the package for the database provider(s) you want to target. This walkthrough 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**

[[userguide.asciidoc_navy#ef-code-first#]] === EF Code first

In order to design your database model from scratch, we encourage to follow the Microsoft guidelines described [here](#).

[[userguide.asciidoc_navy#ef-database-first#]] === 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.

[[userguide.asciidoc_navy#register-your-context-with-dependency-injection#]] ===== 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:

```
services.AddDbContext<YourModelContext>(  
    options =>  
options.UseSqlServer(Configuration.GetConnectionString("Connection name at  
appsettings.json")));
```



You should put your Model and Entities in the template's OASP4Net.Domain.Entities project.

[[userguide.asciidoc_navy#repositories-and-services#]] === Repositories and Services

Services and *Repositories* are an important part of OASP4NET proposal. To make them work properly, first of all must be declared and injected at Startup.cs at *DI Region*.

Services are declared in OASP4Net.Business.Common and injected in Controller classes when needed. Use services to build your application logic.

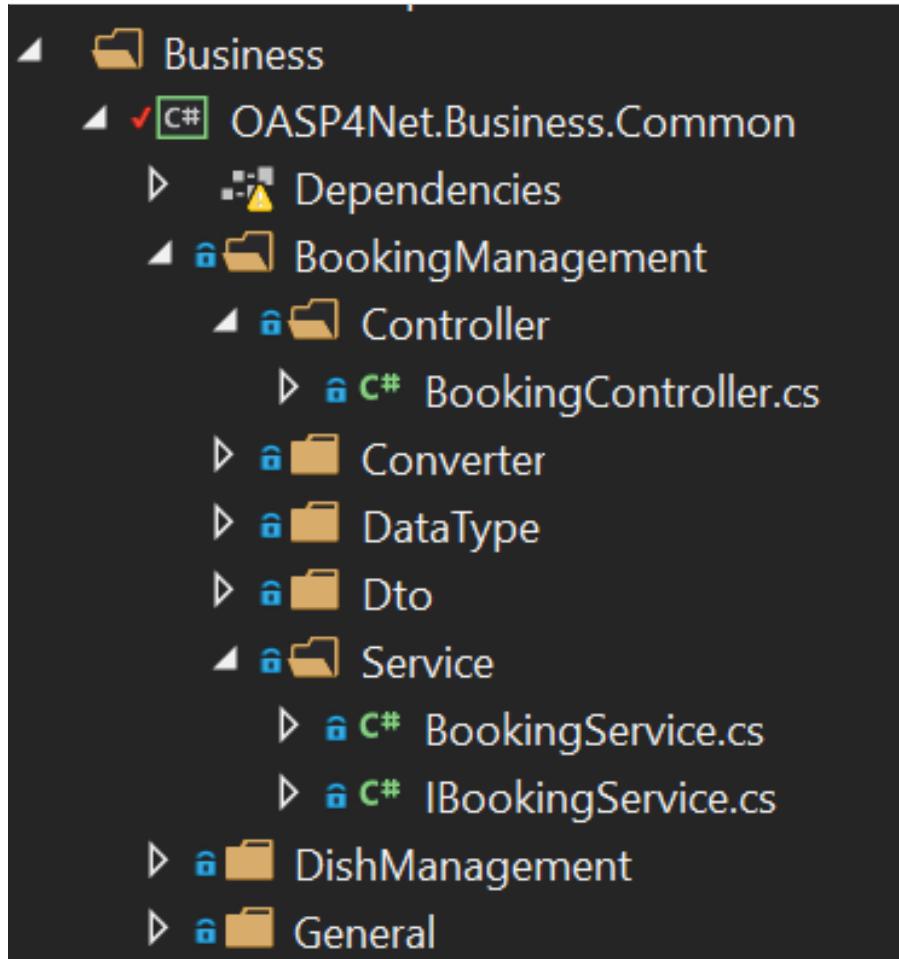


Figure 70. Screenshot of OASP4Net.Business.Common project in depth

For example, My Thai Star Booking controller constructor looks like this:

```
public BookingController(IBookingService bookingService, IMapper mapper)
{
    BookingService = bookingService;
    Mapper = mapper;

}
```

Currently OASP4NET 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 *OASP4Net.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.

[[userguide.asciidoc_navy#swagger-integration#]] === 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. Autentication 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 efective 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:

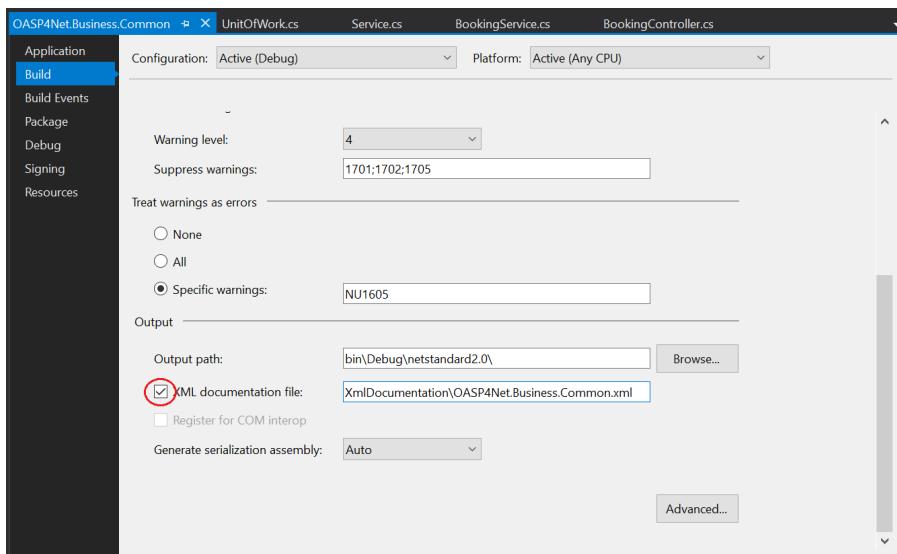


Figure 71. Project settings tab

We propose to generate the file under the XmlDocument folder. For example in OASP4Net.Domain.Entities project in My Thai Star .NET implementation the ootput folder is:

XmlDocumentation\OASP4Net.Business.Common.xml

The file *OASP4Net.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*.

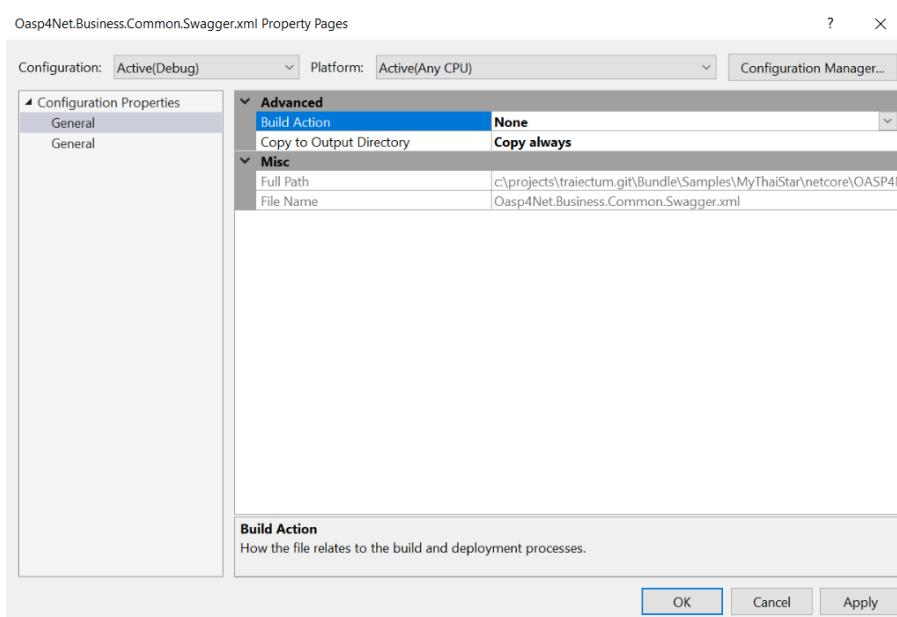


Figure 72. Swagger XML document file properties

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.

[[userguide.asciidoc_navy#logging-module#]] === Logging module

An important part of life software is the need of using log and traces. OASP4NET has a log module preconfigured 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. OASP4NET uses the `serilog` implementation. This implementation provides a huge quantity information about events and traces.

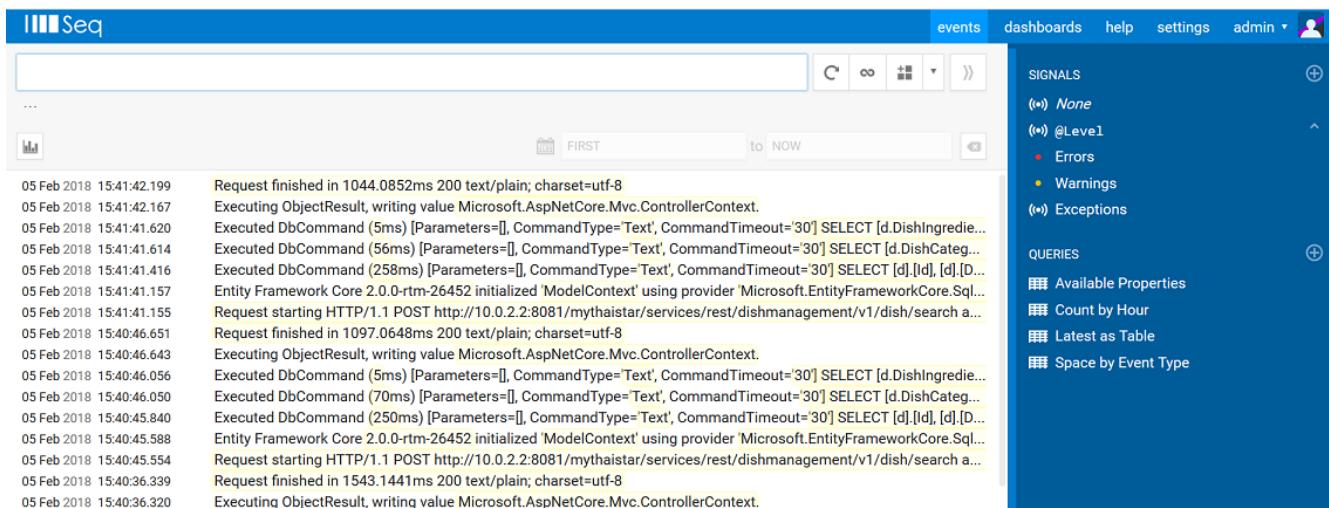
[[userguide.asciidoc_navy#log-file#]] ===== Log file OASP4NET can write the log information to a simple text file. You can configure the file name and folder at appsettings.json file (LogFile attribute) at OASP4Net.Application.WebApi project.

[[userguide.asciidoc_navy#database-log#]] ===== Database log OASP4NET can write the log information to a SQLite database. You can configure the file name and folder at appsettings.json file (LogDatabase attribute) at OASP4Net.Application.WebApi project.

With this method you can launch queries in order to search the information you are looking for.

[[userguide.asciidoc_navy#seq-log#]] ===== Seq log OASP4NET can write the log information to a serilog server. You can configure the serilog URL at appsettings.json file (SeqLogServerUrl attribute) at OASP4Net.Application.WebApi project.

With this method you can make queries via HTTP.



By default you can find the log information at *Logs* folder.

[[userguide.asciidoc_navy#jwt-module#]] === JWT module

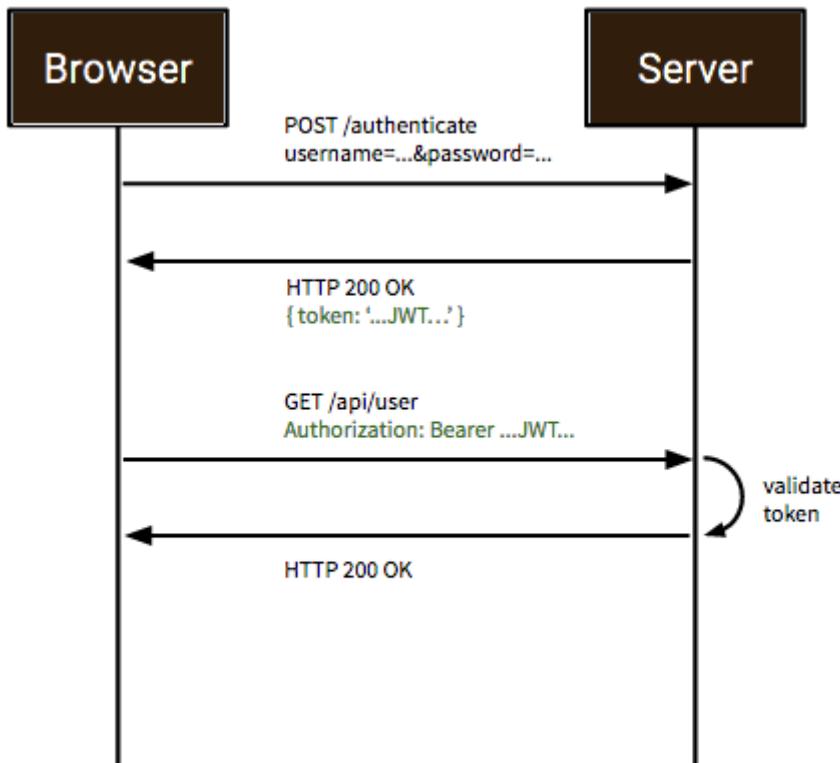
JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties allowing you to decode, verify and generate JWT.

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 OASP4Net.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

`[[userguide.asciidoc_navy#aop-module#]]` == 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 OASP4NET has AOP module preconfigured and activated for controllers at Startup.cs file at OASP4Net.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.

`[[userguide.asciidoc_navy#docker-support#]]` == Docker support

OASP4NET 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.

[[userguide.asciidoc_navy#testing-with-xunit#]] == 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.

[[userguide.asciidoc_navy#the-first-test#]] === 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;
        }
    }
}
```

[[userguide.asciidoc_navy#the-first-test-with-theory#]] === 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	<p>[Fact]</p> <pre>public void Test() { }</pre>
Setup	<pre>public class TestFixture { public TestFixture() { ... } }</pre>
Teardown	<pre>public class TestFixture : IDisposable { ... public void Dispose() { ... } }</pre>

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).

[[userguide.asciidoc_navy#publishing#]] == Publishing [[userguide.asciidoc_navy#nginx#]] ====
 Nginx In order to deploy your application to a Nginx server on Linux platform you can follow the instructions from *Microsoft* [here](#).

[[userguide.asciidoc_navy#iis#]] ===== 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*

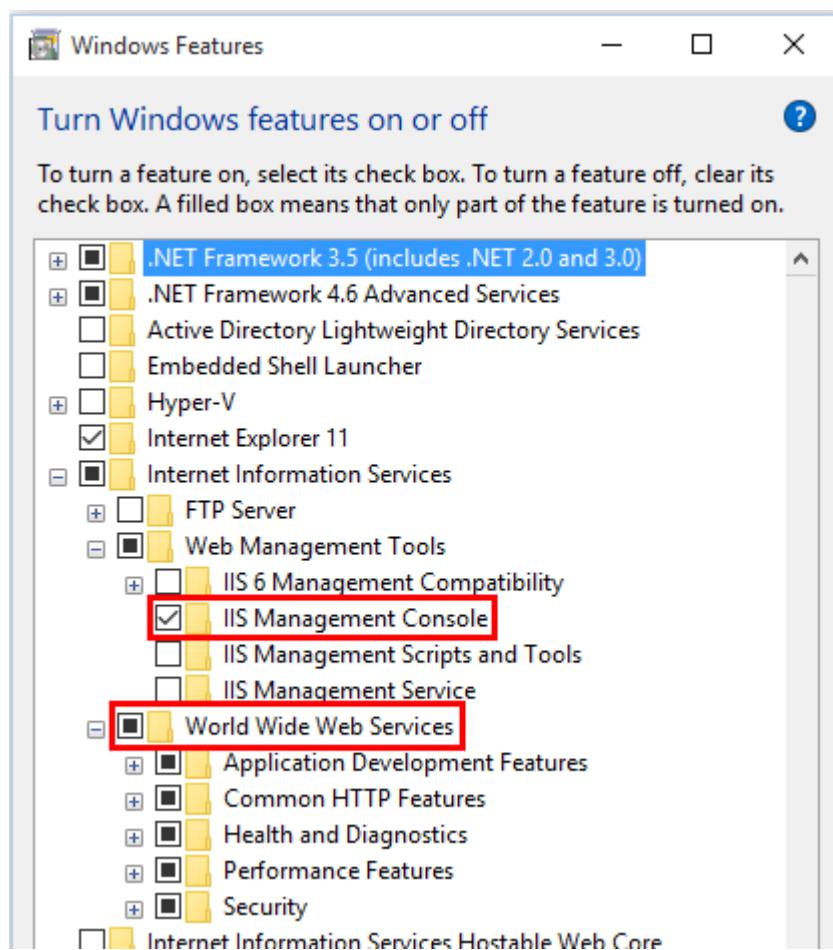
WebListener server will not work in a reverse-proxy configuration with IIS. You must use the [Kestrel server](#).

IIS configuration

Enable the Web Server (IIS) role and establish role services.

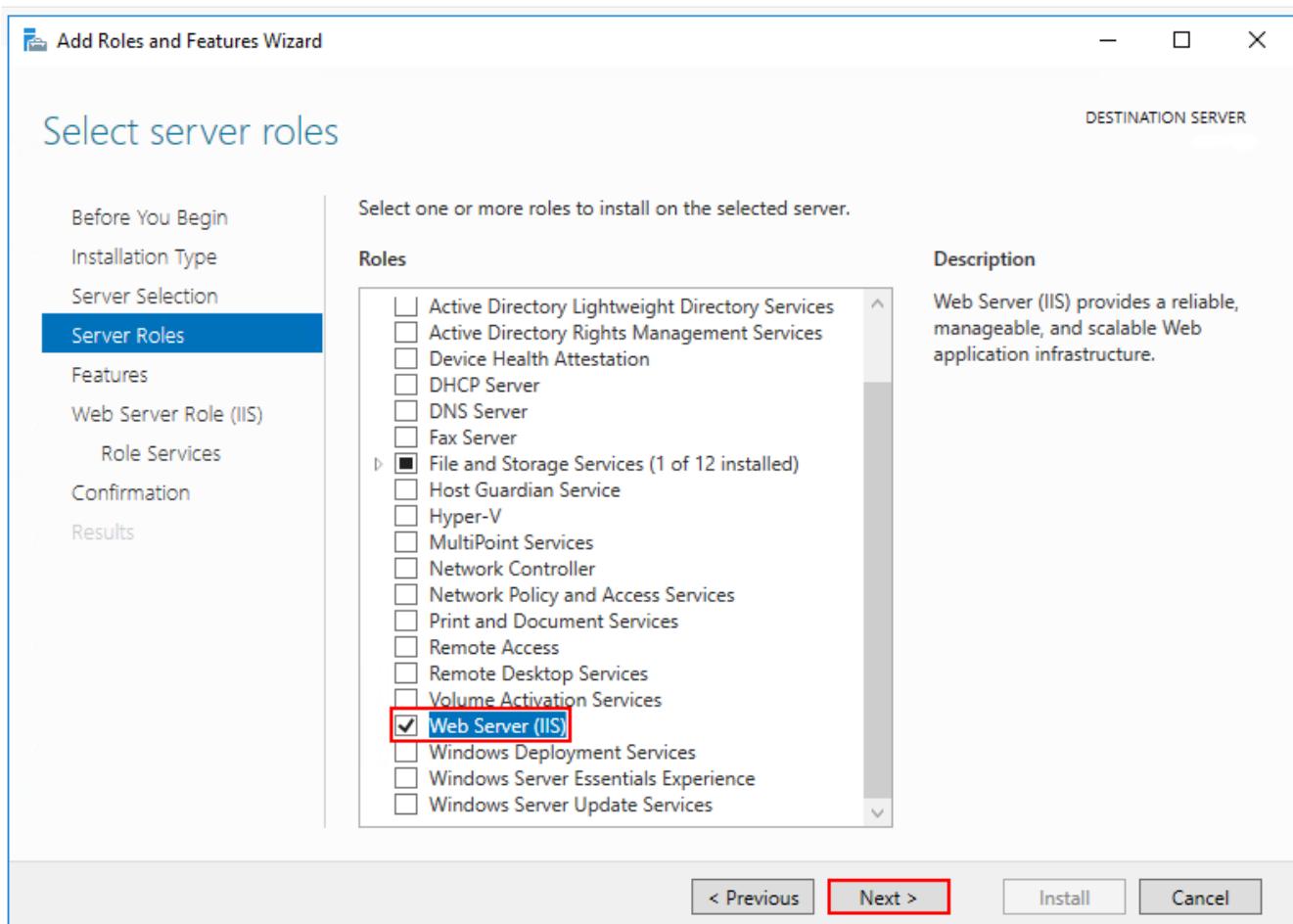
Windows desktop operating systems

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.

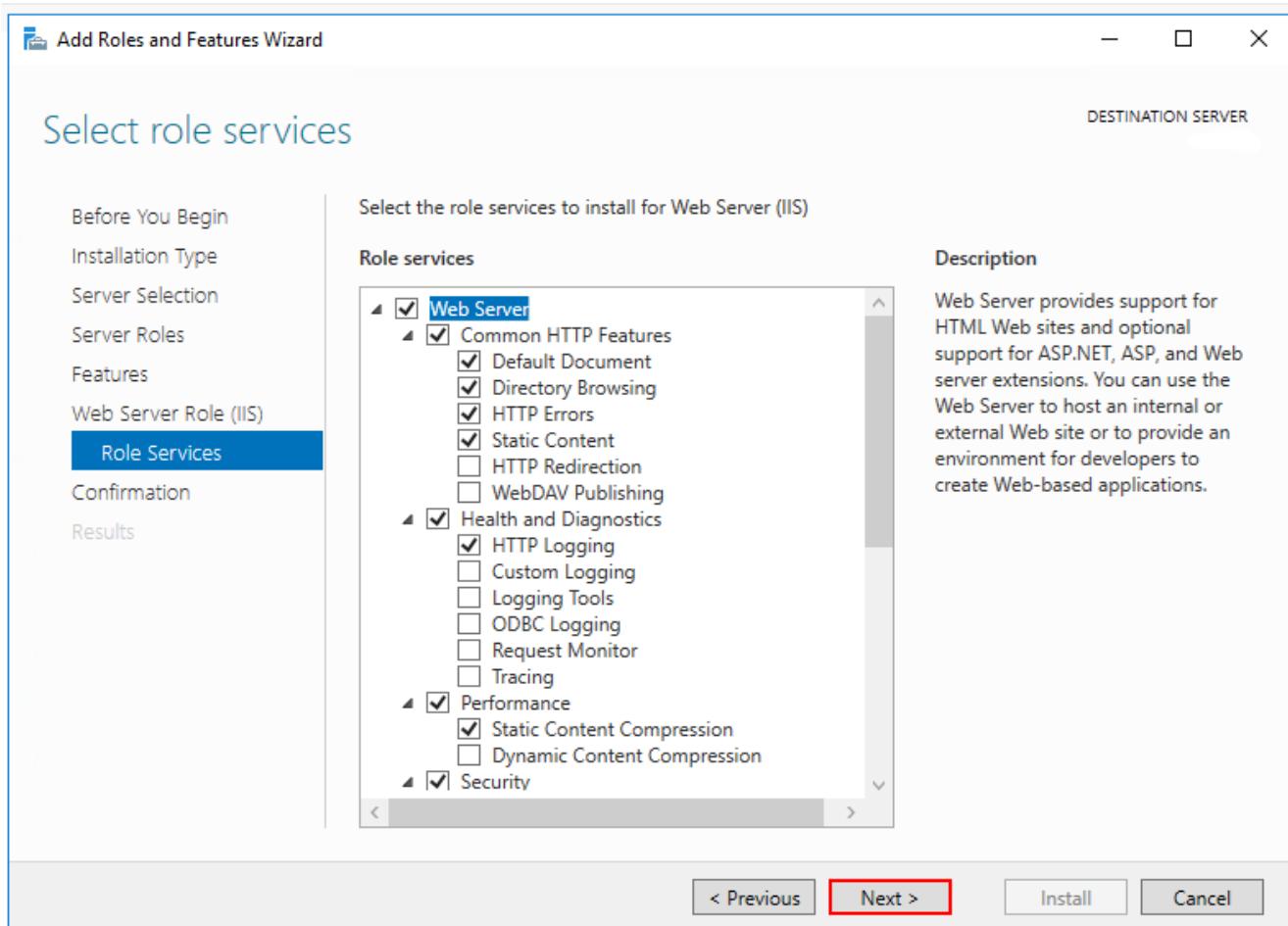


*Conceptually, the IIS configuration described in this document also applies to hosting ASP.NET Core applications on Nano Server IIS, but refer to [ASP.NET Core with IIS on Nano Server](#) for specific instructions.

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>
```

[[userguide.asciidoc_navy#azure#]] ===== 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.

[[userguide.asciidoc_navy#external-links#]] == 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](#)

28. Packages

28.1. Packages overview



OASP4Net is composed by a number of packages that increases the functionality and boosts time development. Each package has its own configuration to make them work properly. In *appsettings.json* set up your environment. On *appsettings.{environment}.json* you can configure each component.

28.2. The packages

You can get the OASP packages on [nuget.org](https://www.nuget.org).

```
[[packages.asciidoc_navy#oasp4net.domain.context#]] === OASP4Net.Domain.Context
[[packages.asciidoc_navy#description#]] === Description OASP4Net.Domain.Context contains the extended class OASP4NetBaseContext 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.
```

```
[[packages.asciidoc_navy#configuration#]] === Configuration
```

- Install package on your solution:

```
PM> Install-Package OASP4Net.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 OASP4Net.Application.Configuration.Startup/DataBaseConfiguration/ConfigureDataBase configure your connections.

[[packages.asciidoc_navy#oasp4net.domain.unitofwork#]] === OASP4Net.Domain.UnitOfWork
[[packages.asciidoc_navy#description#]] ===== Description Unit of work implementation for OASP 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.

[[packages.asciidoc_navy#configuration#]] ===== Configuration

- Install package on your solution:

```
PM> Install-Package OASP4Net.Domain.UnitOfWork
```

- Add this line of code:

```
services.AddUnitOfWorkDependencyInjection();
```

On

```
Startup.cs/ConfigureServices(IServiceCollection services)
```

or on:

```
OASP4Net.Application.Configuration.Startup/DependencyInjectionConfiguration/ConfigureDependencyInjectionService method.
```

[[packages.asciidoc_navy#notes#]] ===== Notes Now you can use the unit of work via dependency injection on your classes:

Figure 73. 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.

[[packages.asciidoc_navy#oasp4net.infrastructure.aop#]] === OASP4Net.Infrastructure.AOP
[[packages.asciidoc_navy#description#]] ===== Description Simple AOP Exception handler for .Net Controller classes integrated with Serilog.

[[packages.asciidoc_navy#configuration#]] ===== Configuration - Install package on your solution:

```
PM> Install-Package OASP4Net.Domain.AOP
```

Add this line of code on ConfigureServices method on Startup.cs

```
services.AddAopAttributeService();
```

[[packages.asciidoc_navy#notes#]] ===== 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.

[[packages.asciidoc_navy#oasp4net.infrastructure.applicationuser#]] =====
OASP4Net.Infrastructure ApplicationUser [[packages.asciidoc_navy#description#]] ===== Description
OASP4NET Application user classes to implement basic Microsoft's basic authentication in order to be used on authentication methodologies such Jason Web Token (JWT).

[[packages.asciidoc_navy#configuration#]] ===== Configuration

- Install package on your solution:

```
PM> OASP4Net.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:

```
OASP4Net.Application.Configuration.Startup/DependencyInjectionConfiguration/ConfigureDependencyInjectionService 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();
    ...
}
```

[[packages.asciidoc_navy#notes#]] ===== 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 *OASP4Net.Infrastructure.JWT* and *OASP4Net.Infrastructure.JWT.MVC*.

[[packages.asciidoc_navy#oasp4net.infrastructure.communication#]] =====

OASP4Net.Infrastructure.Communication [[packages.asciidoc_navy#description#]] ===== 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*.

[[packages.asciidoc_navy#configuration#]] ===== Configuration

- Install package on your solution:

```
PM> OASP4Net.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);
```

[[packages.asciidoc_navy#notes#]] ===== Notes - Example:

```
private async Task RestManagementServiceSample(EmailDto dataToSend)
{
    var url = Configuration["EmailServiceUrl"];
    var restManagementService = new RestManagementService();
    await restManagementService.CallPostMethod(url, dataToSend);
}
```

[[packages.asciidoc_navy#oasp4net.infrastructure.cors#]] === OASP4Net.Infrastructure.Cors
[[packages.asciidoc_navy#description#]] ===== Description Enables CORS configuration for OASP4Net 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.

[[packages.asciidoc_navy#configuration#]] ===== Configuration

- Install package on your solution:

```
PM> OASP4Net.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 Corspolicy will be enabled 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
  }
]
```

[[packages.asciidoc_navy#notes#]] ===== 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();
}
```

[[packages.asciidoc_navy#oasp4net.infrastructure.extensions#]] ===
OASP4Net.Infrastructure.Extensions [[packages.asciidoc_navy#description#]] ===== Description
Miscellaneous extension library which contains : - Predicate expression builder - DateTime
formatter - HttpClient - HttpContext (Middleware support)

[[packages.asciidoc_navy#configuration#]] ===== Configuration - Install package on your solution:

```
PM> OASP4Net.Infrastructure.Extensions
```

[[packages.asciidoc_navy#notes#]] ===== Notes

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 an 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 criterias:
    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);
}

```

HttpContext

- TryAddHeader method is used on *OASP4Net.Infrastructure.Middleware* component to add automatically response header options such authorization.

Cryptography

- Adds to *string* class the following conversion methods:

ToSHA256
ToSHA512
ToMD5

Datetime

- Adds the `ConvertDateTimeToMilliseconds` method to `DateTime` class. It is very helpfull to get aligned with frontend frameworks.

Http Client

- Contains synchronous and asyncrhonous methods to perform Http method calls such:

Post
Put
Patch

[[packages.asciidoc_navy#oasp4net.infrastructure.jwt#]] === OASP4Net.Infrastructure.JWT
[[packages.asciidoc_navy#description#]] ===== 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/>
- OASP component to manage JWT standard to provide security to .Net API applications.

[[packages.asciidoc_navy#configuration#]] ===== Configuration

- Install package on your solution:

PM> OASP4Net.Infrastructure.JWT

- You can configure your JWT configuration on `appsettings.{environment}.json`:

```
"JWT": {
    "Audience": "MyThaiStar",
    "Issuer": "MyThaiStar",
    "TokenExpirationTime": 60,
    "ValidateIssuerSigningKey": true,
    "ValidateLifetime": true,
    "ClockSkew": 5,
    "Certificate": "oasp4net.pfx",
    "CertificatePassword": "oasp4net"
}
```

- *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 exist an exception will be raised.
- Add this line of code:

```
services.AddBusinessCommonJwtPolicy();
```

On

```
Startup.cs/ConfigureServices(IServiceCollection services)
```

or on:

```
OASP4Net.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");
```

[[packages.asciidoc_navy#notes#]] ===== Notes

- The certificate will be used to generate the symmetric key to encrypt the json web token.

[[packages.asciidoc_navy#oasp4net.infrastructure.jwt.mvc#]] =====
OASP4Net.Infrastructure.JWT.MVC [[packages.asciidoc_navy#description#]] ===== Description -
OASP Extended controller to interact with JWT features

[[packages.asciidoc_navy#configuration#]] ===== Configuration

- Extend your `_ Microsoft.AspNetCore.Mvc.Controller_` class with *OASP4NetJWTController* class:

```
public class LoginController : OASP4NetJWTController
{
    private readonly ILoginService _loginService;

    public LoginController(ILoginService loginService, SignInManager<ApplicationUser>
signInManager, UserManager<ApplicationUser> userManager, ILogger<LoginController>
logger, IMapper mapper) : base(logger,mapper)
    {
        _loginService = loginService;
    }

    ...
}
```

[[packages.asciidoc_navy#notes#]] ===== 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 loged = await _loginService.LoginAsync(loginDto.UserName,
loginDto.Password);

        if (loged)
        {
            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 *OASP4NetJWTController* you will have available these methods to simplify user management:

```

public interface IOASP4NetJWTController
{
    // 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);
}

```

[[packages.asciidoc_navy#oasp4net.infrastructure.middleware#]] =====
OASP4Net.Infrastructure.Middleware [[packages.asciidoc_navy#description#]] ===== Description -
OASP4Net 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.
- OASP4Net supports the following automatic response headers:
 - AccessControlExposeHeader
 - StrictTransportSecurityHeader
 - XFrameOptionsHeader
 - XssProtectionHeader
 - XContentTypeOptionsHeader
 - ContentSecurityPolicyHeader
 - PermittedCrossDomainPoliciesHeader
 - ReferrerPolicyHeader:toc: macro

[[packages.asciidoc_navy#configuration#]] ===== Configuration - Install package on your solution:

PM> OASP4Net.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.

[[packages.asciidoc_navy#oasp4net.infrastructure.mvc#]] === OASP4Net.Infrastructure.MVC
 [[packages.asciidoc_navy#description#]] ===== Description Common classes to extend controller functionality on API. Also provides support for paged results in OASP applications and autommaper injected class.

[[packages.asciidoc_navy#configuration#]] ===== Configuration - Install package on your solution:

PM> OASP4Net.Infrastructure.MVC

[[packages.asciidoc_navy#notes#]] ===== 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}");
    }
}
```

[[packages.asciidoc_navy#oasp4net.infrastructure.swagger#]] ===
OASP4Net.Infrastructure.Swagger [[packages.asciidoc_navy#description#]] ===== Description - OASP
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).

[[packages.asciidoc_navy#configuration#]] ===== Configuration

- Install package on your solution:

```
PM> OASP4Net.Infrastructure.Swagger
```

- You can configure your Swagger configuration on *appsettings.{environment}.json*:

```
"Swagger": {  
    "Version": "v1",  
    "Title": "OASP4Net API",  
    "Description": "A simple ASP.NET Core Web API capable project",  
    "Terms": "OASP",  
    "Contact": {  
        "Name": "OASP4Net",  
        "Email": "",  
        "Url": ""  
    },  
    "License": {  
        "Name": "OASP4Net",  
        "Url": ""  
    },  
    "Endpoint": {  
        "Name": "V1 Docs",  
        "Url": "/swagger/v1/swagger.json"  
    }  
}
```

- Add this line of code:

```
services.ConfigureSwaggerService();
```

On

```
Startup.cs/ConfigureServices(IServiceCollection services)
```

- Also add this line of code:

```
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.

[[packages.asciidoc_navy#notes#]] ===== 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 74. Swagger documentation

- Ensure that your XML files have the attribute copy always to true:

Figure 75. Swagger documentation

```

[[packages.asciidoc_navy#oasp4net.infrastructure.test#]]      ===      OASP4Net.Infrastructure.Test
[[packages.asciidoc_navy#description#]] ===== Description OASP Base classes to create unit tests and
integration tests with Moq and xUnit.
```

```

[[packages.asciidoc_navy#configuration#]] ===== Configuration - Load the template: > dotnet new -i
OASP4Net.Test.Template > dotnet new OASP4NetTest
```

```

[[packages.asciidoc_navy#notes#]] ===== Notes - At this point you can find these classes: *
BaseManagementTest * DatabaseManagementTest<T> (Where T is a OASP4NetBaseContext 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.

28.3. Required software

[Visual Studio Code](#)

[C# Extension for VS Code](#)

[.Net Core SDK](#)

[CORS in .Net Core](#)

29. Templates

29.1. Templates

[[templates.asciidoc_navy#overview#]] == Overview

The .Net Core and .Net Framework given templates allows to start coding an application with the following functionality ready to use:

	.NET Framework	.NET Core
WebAPI Console	X	X
Controller abstraction	X	X
Data access repository pattern (Generics & ASync)	X	X
Unit of work implementation		X
DI (Microsoft.Extensions.DependencyInjection)		X
Aspect oriented programming support for controllers (ActionFilterAttribute)		X
Log to file support (Serilog)		X
Swagger API doc auto generation from summary comments (Swashbuckle)	X	X
Json web token support		X
Cross-origin resource sharing (CORS) support		X
Controller/Service/Repository Tobago patterns templates		X
Dockerizable solution		X
Nuget Package sample	X	X
Nuget Server instance	X	X

Figure 76. Current available functionality

Please refer to [User guide](#) in order to start developing.

[[templates.asciidoc_navy#.net-core-2.1.x#]] == .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 OASP4Net template, please follow this instructions:

[[templates.asciidoc_navy#using-oasp4net-template#]] === Using OASP4Net template . Open your favourite terminal (Win/Linux/iOS) . Go to future project's path . Type `dotnet new -i OASP4Net.WebAPI.Template` . Type `dotnet new OASP4NetAPI` . Go to project's path . You are ready to start developing with OASP4Net

[[templates.asciidoc_navy#links#]] == Links

[= fa floppy o] [.Net templates](#)

30. Samples

30.1. Samples

[[samples.asciidoc_navy#my-thai-star-restaurant]] == 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)

[[samples.asciidoc_navy#angular-requeriments]] === Angular requeriments

- [Node](#)
- [Angular CLI](#)
- [Yarn](#)

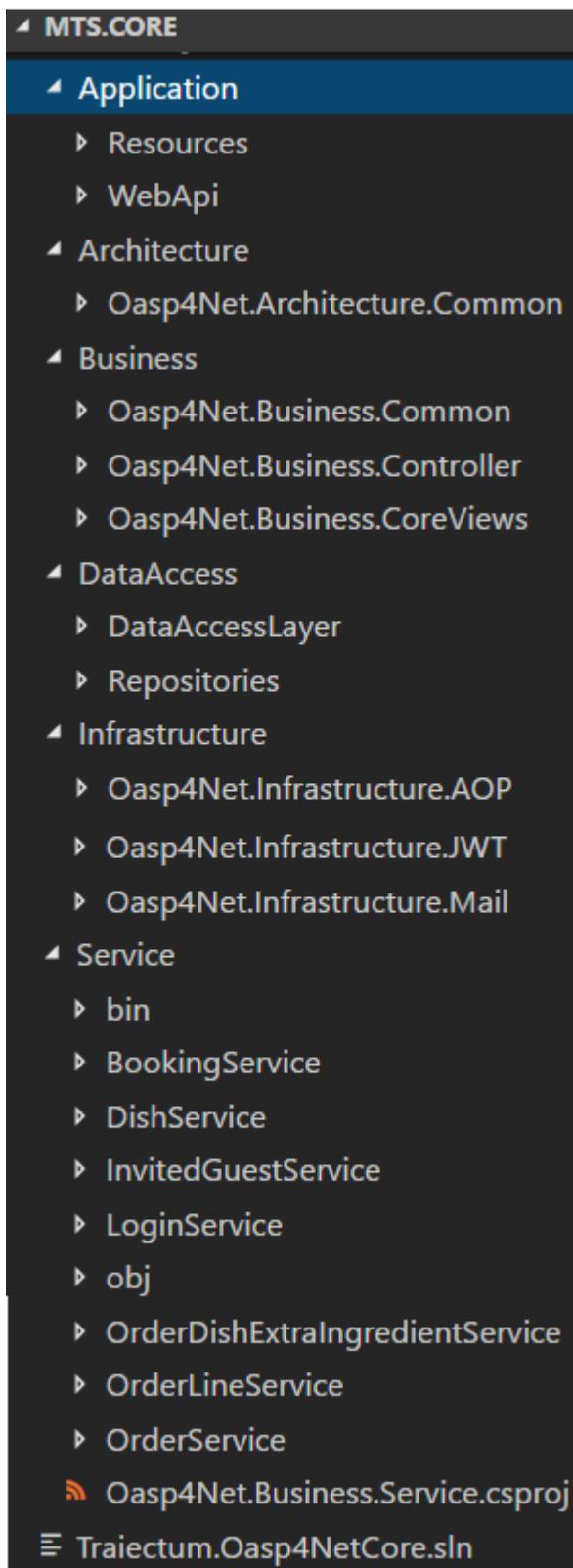
[[samples.asciidoc_navy#angular-client]] === Angular client

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

[[samples.asciidoc_navy#.net-core-server]] === .Net Core server

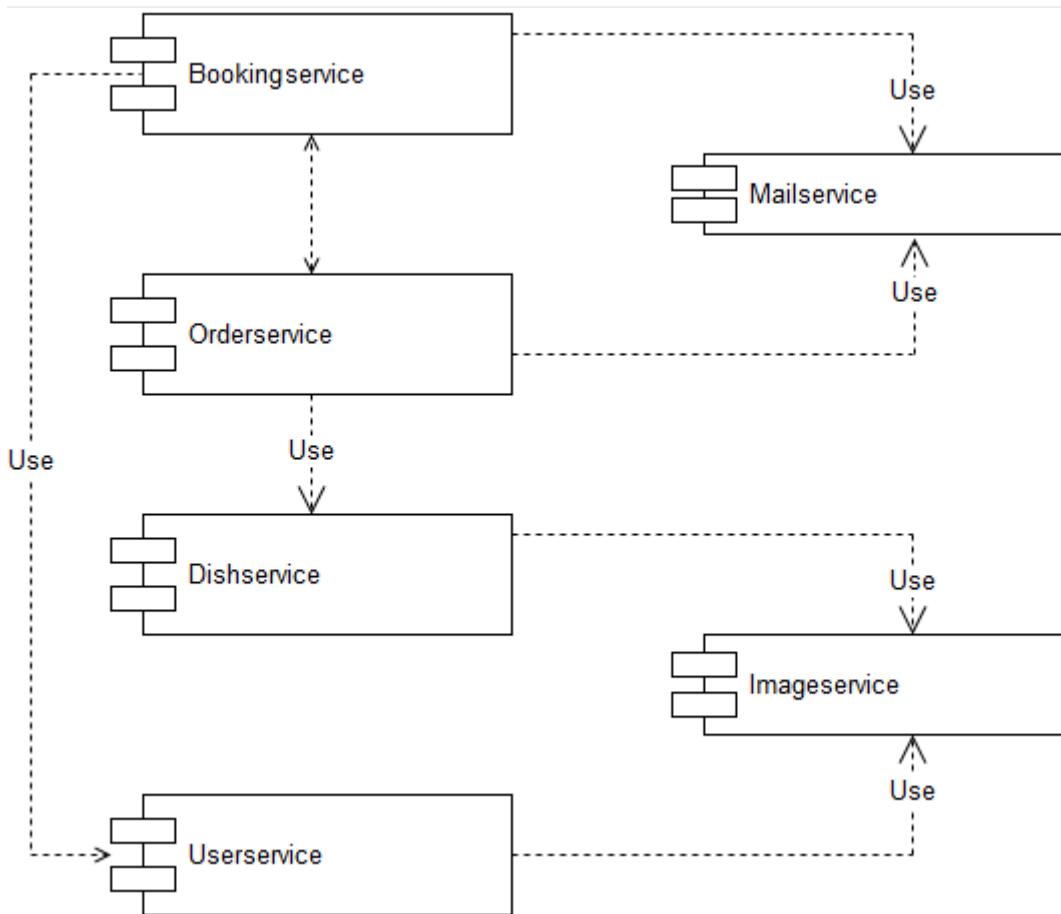
[[samples.asciidoc_navy#basic-architecture-details]] === Basic architecture details

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.



[[samples.asciidoc_navy#components#]] ===== 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.

[[samples.asciidoc_navy#layers#]] === Layers [[samples.asciidoc_navy#introduction#]] ====
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

[[samples.asciidoc_navy#application-layer#]] === 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 lauches as a self host console application (microservice approach) and as a Web Api application hosted on IIS/IIS Express.

[[samples.asciidoc_navy#business-layer#]] === 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 framwork attributes.

This layer also includes the object response classes in order to interact with external clients.

[[samples.asciidoc_navy#service-layer#]] === 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*.

[[samples.asciidoc_navy#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 (DbContext).

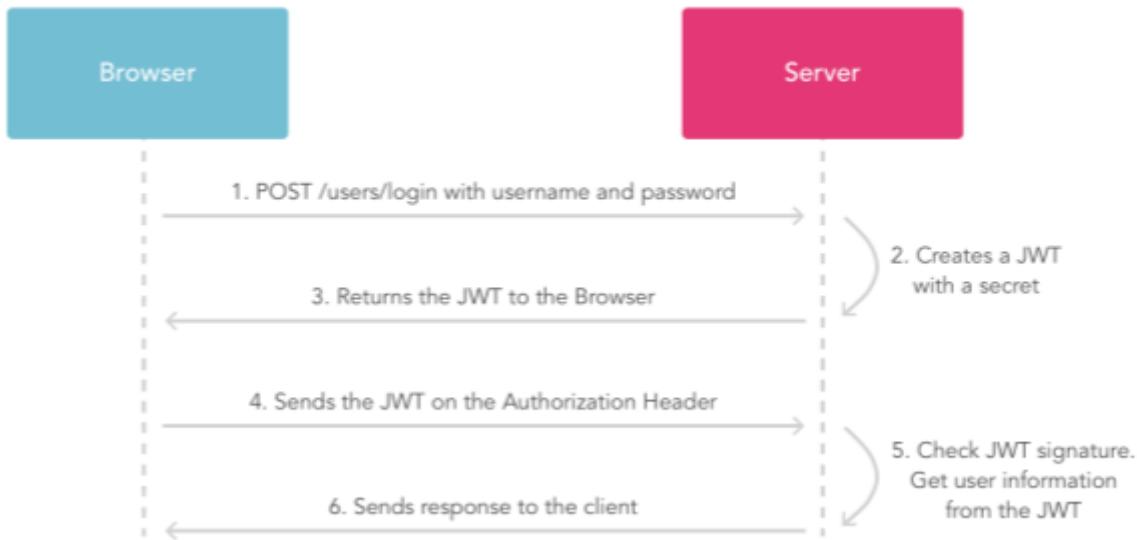
In this layer will make use of the *Repository patter* and *Unit of work* in order to encapsulte the complexibility. Making use of this combined patters we ensure an organized a common and easy work model.

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*.

[[samples.asciidoc_navy#cross-cutting-concerns#]] === Cross-Cutting concerns the layer to make use of transversal components such JWT and mailing.

[[samples.asciidoc_navy#jwt-basics#]] === 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.



[[samples.asciidoc_navy#jwt-implementation-details#]] === 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.

[[samples.asciidoc_navy#authentication#]] === 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 *Microsfot's authorization schema* Database. It is started up the first time the application launches.

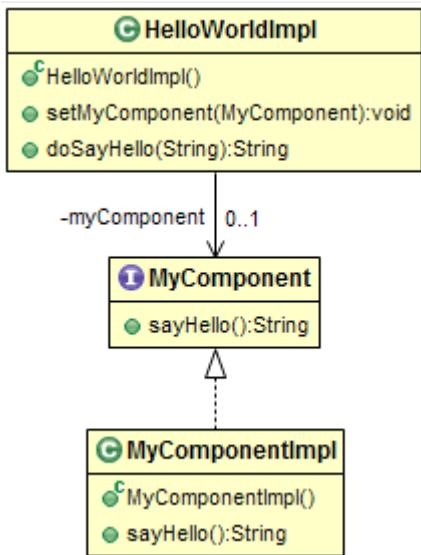
YYou can read more about _Authorization on:

[Authorization in ASP.NET Core](#)

[Claim based authorization](#)

[[samples.asciidoc_navy#dependency-injection#]] === Dependency injection

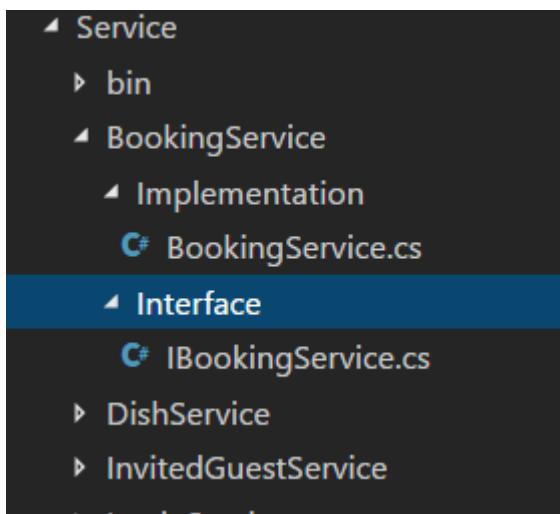
As it is explained in the [Microsoft documentation](#) we are going to implement the *dependency injection* pattern basing our solution on *.Net Core*.



- 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*.

[[samples.asciidoc_navy#layer-communication-method#]] === 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.

[[samples.asciidoc_navy#api-exposed#]] === 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 potentiel 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. Autentication 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.

[[samples.asciidoc_navy#google-mail-api-consumer#]] == Google Mail API Consumer [= fa floppy o] [Google Mail API Consumer](#)

Application	MyThaiStarEmailService.exe
Config file	MyThaiStarEmailService.exe.Config
Default port	8080

[[samples.asciidoc_navy#overview#]] === Overview . Execute MyThaiStarEmailService.exe. . The first time google will ask you for credentials (just one time) in your default browser:

- Account: mythaistarrestaurant@gmail.com
- Password: mythaistarrestaurant2501
 1. Visit the url: <http://localhost:8080/swagger>
 2. Your server is ready!

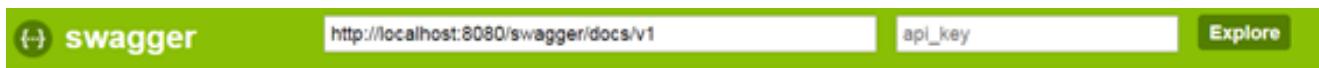


Figure 77. GMail Server Swager contract page

[[samples.asciidoc_navy#json-example#]] === 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"
  }
}
```

[[samples.asciidoc_navy#configure-the-service-port#]] === 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>
```

[[samples.asciidoc_navy#external-links#]] ===== External links

[Google API Account Configuration](#)

[About Scopes](#)

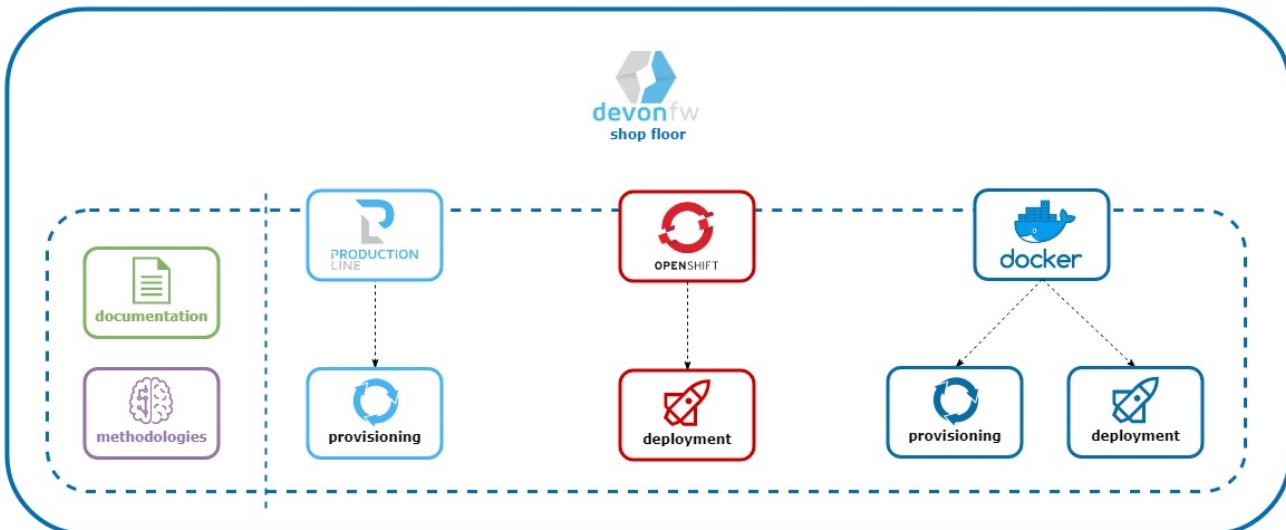
[[samples.asciidoc_navy#downloads#]] == Downloads

[= fa floppy o] [My Thai Star \(.Net Core Server + Angular client\)](#)

[= fa floppy o] [Google Mail API Consumer](#)

Part VI: devonfw shop floor

31. 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.



Let's start.

32. 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.

32.1. 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](#).

32.2. 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](#).

32.3. Step 2 - Create the project

32.3.1. 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.

32.3.2. 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](#)

32.3.3. 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.

32.4. Step 3 - Deployment

The third is configure our deployment environment. Here is the list for the different options:

- [dsf4openshift](#).

32.5. 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.

33. Provisioning environments

33.1. 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](#).

33.1.1. How to obtain your Production Line

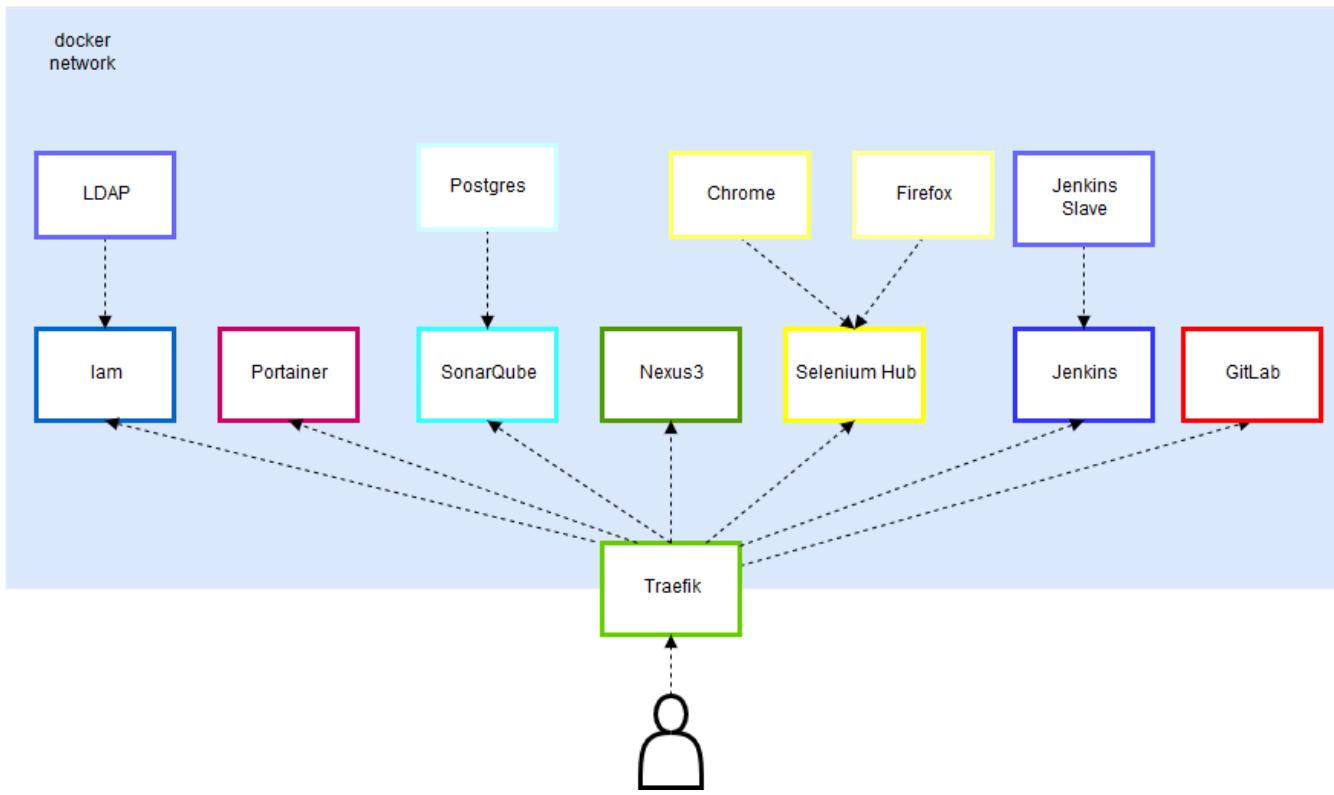
You can order your Production Line environment instance following the [official guide](#). Remember that you need to order at least the next tools: * Jenkins * GitLab * SonarQube * Nexus

[Back.](#)

33.2. dsf4docker provisioning environment



33.2.1. Architecture overview



33.2.2. Prerequisite

To use dsf4docker provisioning environment you need a remote server and you must clone or download devonfw shop floor.

33.2.3. 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
```

33.2.4. 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](#).

34. Configuration and services integration

34.1. Nexus Configuration

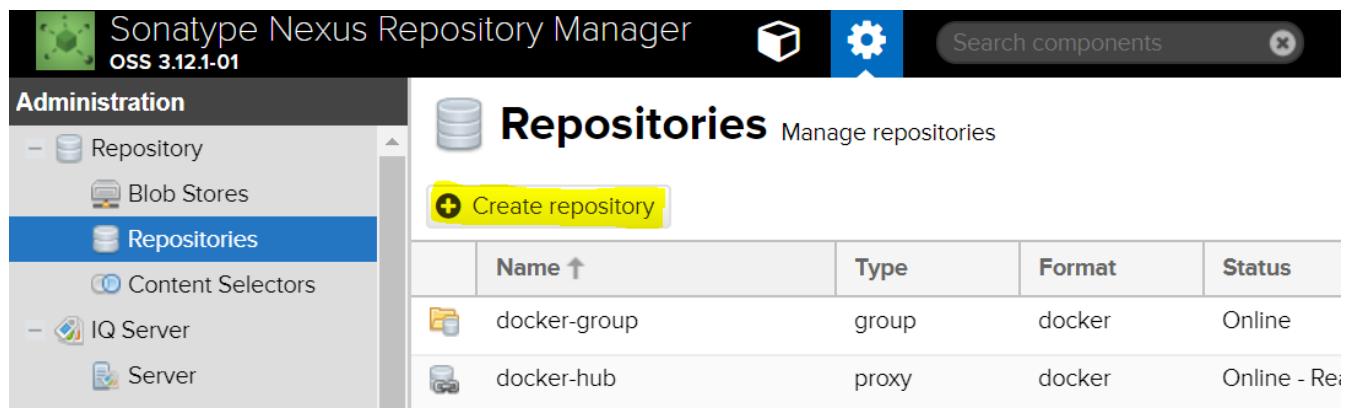
In this document you will see how you can configure Nexus repository and how to integrate it with Jenkins.

34.1.1. 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.



The screenshot shows the Sonatype Nexus Repository Manager interface version OSS 3.12.1-01. The top navigation bar includes the logo, the title "Sonatype Nexus Repository Manager", and a "Search components" bar. Below the title are icons for a cube and a gear. The left sidebar, titled "Administration", has a tree structure with "Repository" expanded, showing "Blob Stores", "Repositories" (which is selected and highlighted in blue), and "Content Selectors". Under "Repository", there are "docker-group" and "docker-hub". The main content area is titled "Repositories" and "Manage repositories". It features a "Create repository" button with a plus sign. A table lists existing repositories:

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
snapshots

Online: If checked, the repository accepts incoming requests

Maven 2

Version policy:

What type of artifacts does this repository store?

Snapshot

Layout policy:

Validate that all paths are Maven artifact or metadata paths

Permissive

Storage

Blob store:

Blob store used to store asset contents

default

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

Allow redeploy

Create repository

Cancel

34.1.2. 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

Filter X

- nx-anonymous
- Others
- Users

Admins

nx-admin

Create local user

Cancel

34.1.3. Jenkins integration

To use Nexus in our pipelines you need to configure Jenkins.

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 'Domain' column contains an icon of a castle and the text 'Global credentials (unrestricted)'. The 'Credentials' column contains the text 'Credentials'. 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 as follows:

- Kind: Username with password
- Scope: Global (Jenkins, nodes, items, all child items, etc)
- Username: nexus-api
- Password: (redacted)
- ID: nexus
- Description: Nexus credentials.

At the bottom right of the form is a blue 'OK' button.

Add the nexus user to maven global settings

Now you need to go to Manage Jenkins clicking on left menu and enter in **Managed files**.

You can edit or remove your configuration files

The screenshot shows the 'MyGlobalSettings' configuration file in the 'Global Maven settings.xml' section. The file content is as follows:

```

<!-- MyGlobalSettings -->
<!-- global settings -->

```

Edit the Global Maven settings.xml to add your nexus repositories credentials as you could see in the next image:

The configuration

ID	MavenSettings
Name	MyGlobalSettings
Comment	global settings
Replace All	<input checked="" type="checkbox"/>
Server Credentials	ServerId: pl-nexus Credentials: nexus-api/******** (nexus-api) <input type="button" value="Add"/>
<input type="button" value="Delete"/> <input type="button" value="?"/>	

And you are done.

34.2. 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.

34.2.1. Generate user token

To generate the user token, go to your account clicking in the left icon on the top menu bar.

The screenshot shows the SonarQube administration interface. At the top, there's a navigation bar with links: sonarqube, Projects, Issues, Rules, Quality Profiles, Quality Gates, Administration. The Administration link is currently selected. On the right side, there's a user dropdown menu with options: jodacalc, jodacalc@capgemini.com, My Account (which is highlighted with a yellow box), and Log out. Below the navigation bar, there's a search bar and some general settings information.

Go to security tab and generate the token.

The screenshot shows the SonarQube user profile page for 'jodacalc'. It features a red square icon with a white letter 'J', the name 'jodacalc', and tabs for Profile, Security (which is highlighted with a blue underline), Notifications, and Projects.

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.

The screenshot shows the SonarQube Tokens page. It lists a single token: 'jenkins' (Created: October 10, 2018) with a 'Revoke' button. At the bottom, there are buttons for 'Generate New Token' (highlighted with a yellow box), 'Enter Token Name', and 'Generate'.

34.2.2. 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. The Administration tab is selected. On the left, a sidebar lists various configuration sections: Analysis Scope, Dependency-Check, Flex, General, Java, JavaScript, PHP, and Python. The 'Webhooks' section under Analysis Scope is currently selected. The main content area displays the 'Webhooks' configuration page. It contains a table with two columns: 'Name' and 'URL'. A new entry is being added, with 'jenkins' in the Name field and 'http://jenkins-core:8080/jenkins/sonarqube-webhook/' in the URL field. A red 'X' icon is visible next to the URL field, likely for deletion. Below the table is a 'Reset' button and a note stating 'Default: <no value>'.

34.2.3. 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.

SonarQube Scanner

SonarQube Scanner installations

SonarQube Scanner
Name <input type="text" value="SonarQube-scanner"/>

Install automatically ?

 **Install from Maven Central**

Version

Delete Installer

Add Installer ▾

Delete SonarQube Scanner

Add SonarQube Scanner

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

Server URL

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.

34.2.4. SonarQube configuration

Now is time to configure your sonar in order to check correctly the quality of the project. To do it, please follow the official documentation about our plugins and Quality Gates and Profiles [here](#).

How to ignore files

Normaly the developers needs to ignore some files from Sonar analysis. To do it, 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'
```

35. Create project

35.1. Create and integrate git repository

35.1.1. 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. On the far left is the GitLab logo. To its right are several menu items: 'Projects' (with a dropdown arrow), 'Groups', 'Activity', 'Milestones', and 'Snippets'. In the center of the bar is a search field with a magnifying glass icon. To the right of the search field are icons for 'Issues', 'Merge Requests', 'Commits', and 'Dashboard'. At the far right is a user profile icon. Below the main navigation, there's a secondary navigation bar with tabs: 'Your projects' (which is selected and highlighted in blue), 'Starred projects', and 'Explore projects'. On the right side of this bar is a 'Last updated' dropdown 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' creation form. On the left, there's a sidebar with information about what a project is and how features are enabled. It also lists the project path as `https://shared-services.pls2-eu.capgemini.com/gitlab/jodacalc/`. The main form has three tabs at the top: 'Blank project' (selected), 'Create from template', and 'Import project'. The 'Project path' field contains the path from the sidebar. The 'Project name' field is set to 'my-awesome-project'. Below that is a 'Project description (optional)' field with a placeholder 'Description format'. To the right of the description field is a dropdown menu for 'Groups' containing 'jodacalc'. Under 'Visibility Level', the 'Internal' option is selected. 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](#)

35.2. start new devonfw project

It is time to create your devonfw project:

35.2.1. How to create new devonfw project

Here you can find the official guides to start new devonfw projects:

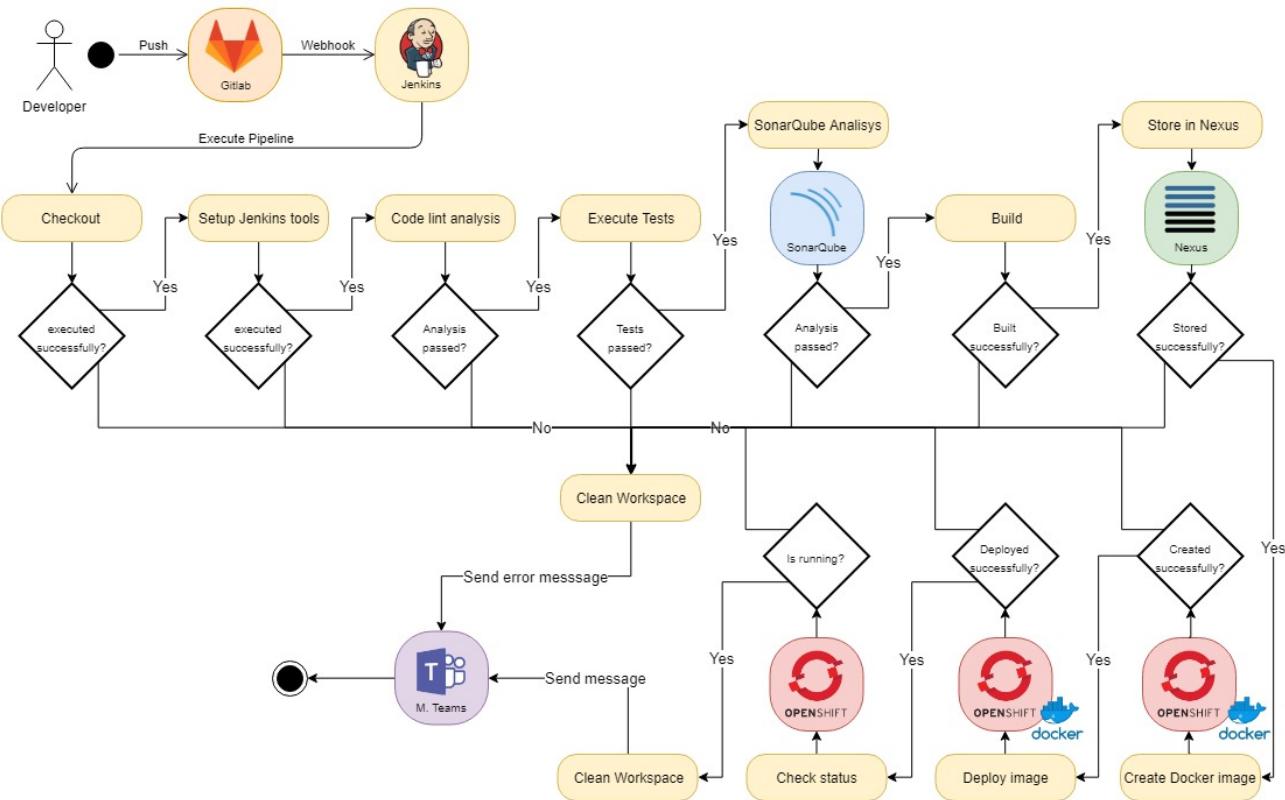
- visit our [devon4ng](#) guide.
 - visit our [devon4j](#) guide.

35.3. cicd configuration

35.3.1. 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:

- devon4j
 - 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.

35.3.2. 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.

36. Deployment environments

36.1. OpenShift

36.1.1. 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			
	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			

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.

36.1.2. 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:

Summary	
Repository	snapshots
Format	Maven2
Component Group	com.capgemini.inser
Component Name	Insereng-api
Component Version	0.6.0-20190115.12253.32
Path	https://nexus.capgemini.com/nexus/content/repositories/com/capgemini/insereng/api/0.6.0-SNAPSHOT/Insereng-api-0.6.0-20190115.12253.32/resources.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	datafile@2730FF6F-24B3AD1B-2D3E2D6-F476FD0-27FD7EFC-4efabccf-7ca1-47e8-b157-ac025a86a425
Containing repo	snapshots
Uploader	datplayer
Uploader's IP Address	172.16.201.1
Most popular version	0
Age	0
Popularity	0

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 sidebar of the OpenShift web interface. The sidebar has several sections: Overview, Applications (selected), Builds, Resources, Storage, Monitoring, and Catalog. The Applications section is expanded, showing sub-options: Deployments (selected), Stateful Sets, Pods, Services, and Routes. At the top right of the sidebar, there is a dropdown menu with options like Deploy, Actions, Edit, Pause Rollouts, Add Storage, Add Autoscaler, Edit Resource Limits, Edit Health Checks, Edit YAML, and Delete.

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 for 'maildev'. The top navigation bar shows 'Deployments > maildev'. Below it, the deployment name 'maildev' is listed with its creation date 'created 2 months ago'. A tab bar includes 'app' (selected) and 'maildev'. Below the tabs are 'History', 'Configuration', 'Environment', and 'Events'. A note says 'Deployment #1 is active. View Log' with a creation date '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, and Delete.

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\":...\"}{}"`

36.1.3. 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

 OpenShift Client Tools	Name <input type="text" value="OpenShiftv3.11.0"/>	
<input checked="" type="checkbox"/> Install automatically 		
 Run Shell Command	Label <input type="text"/>	 
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 <input type="text" value="/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

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 three sub-options: "Quota", "Membership", and "Config Maps". Below these is a button labeled "Secrets" which is also highlighted with a yellow background. Under "Secrets", there is another 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.



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.



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 Container Platform interface. The top navigation bar includes the 'OpenShift Web Console' logo, a search bar, and a URL: <https://ocp.itaas.s2-eu.capgemini.com/console/project/s2portaldev/overview>. The main title is '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 'APPLICATION' section, with a deployment config named 'devon4ng-dev, #1' listed below it. A search bar at the top right says 'Name' and 'Filter by name'.

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](#)

	Name	OpenShiftv3.11.0	?
<input checked="" type="checkbox"/> Install automatically ?			
	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 ?		
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.

	Name	Yarn	?
Custom Tool Configuration...			
<input checked="" type="checkbox"/> Install automatically ?			
	Label	?	
Command	<pre>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</pre>		
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

37. Monitoring

37.1. Build monitor view

This tool you will be able to see in real time what is the state of your Jenkins pipelines.

37.1.1. 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.

37.1.2. 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.



38. Annexes

38.1. 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:

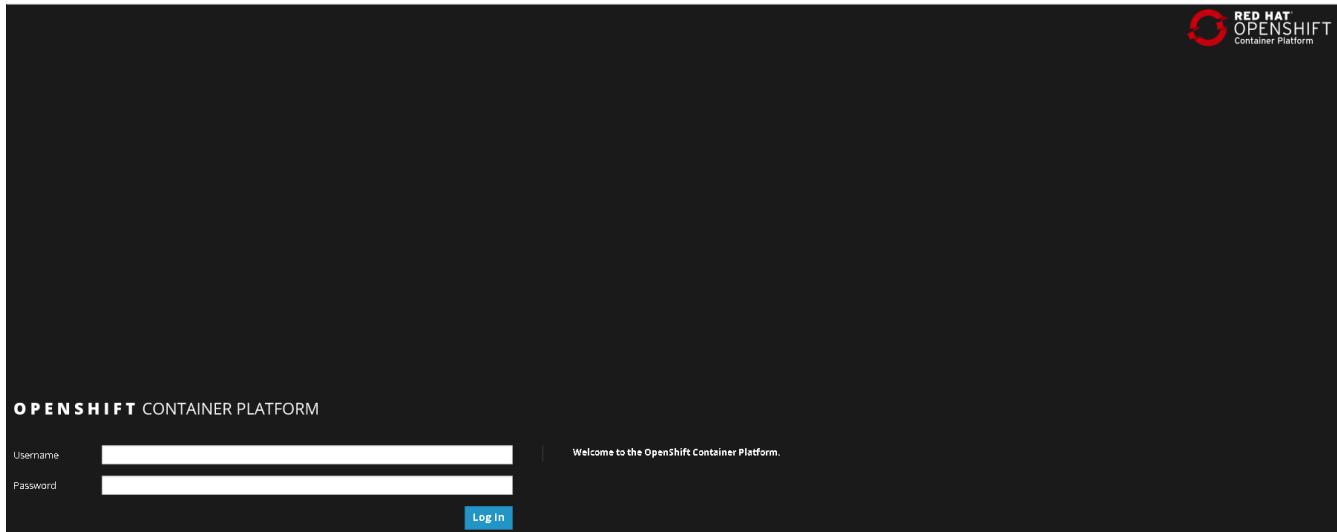
1. Deploy the BitBucket server image (from OpenShift web console)
2. Add a route for the external traffic (from OpenShift web console)
3. Configure the BitBucket server (from BitBucket server web console)

Prerequisites

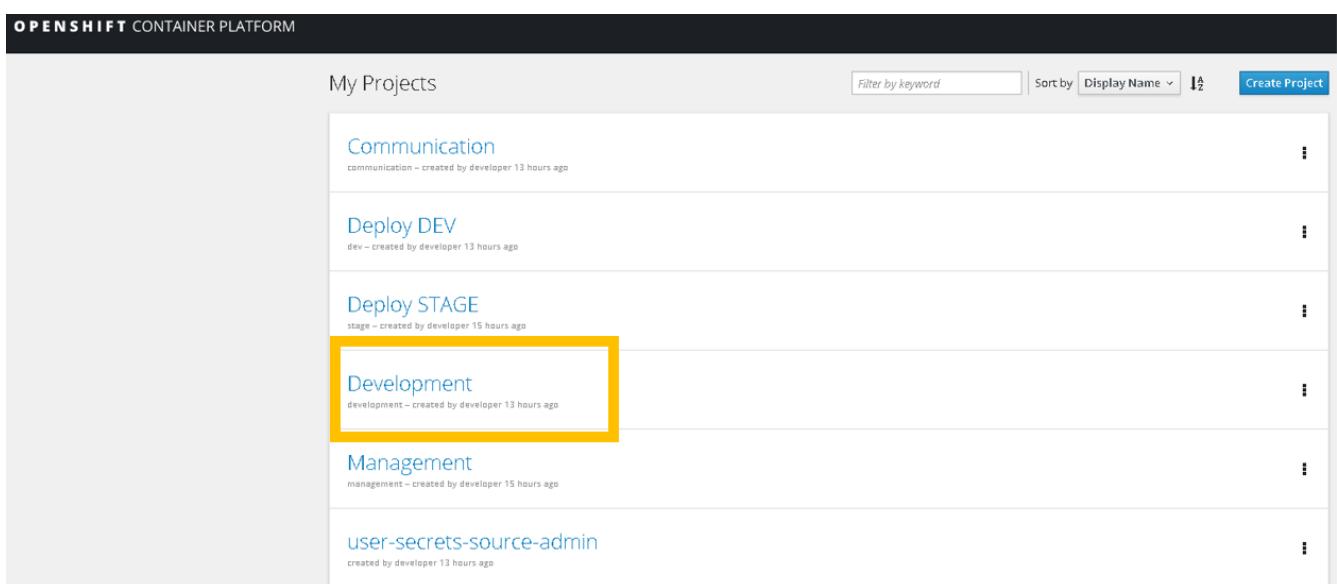
- OpenShift up & running
- Atlassian account (with personal account key). Not required for OpenShift, but for the initial BitBucket server configuration.

Procedure

] === Step 0: Log into our [OpenShift Web console](#)



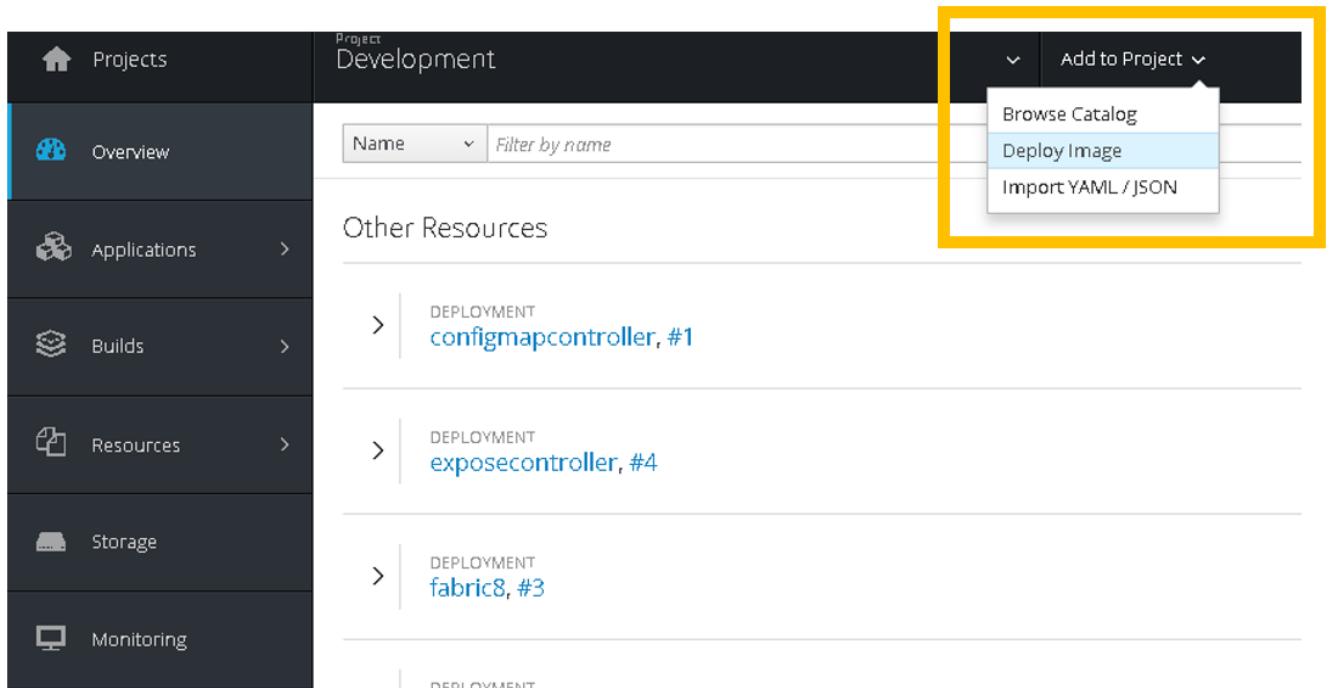
Step 1: Get into Development project



My Projects

- Communication
- Deploy DEV
- Deploy STAGE
- Development**
- Management
- user-secrets-source-admin

Step 2.1: Deploy a new image to the project



Project Development

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 'OPENSHIFT CONTAINER PLATFORM' interface. In the 'Development' section, under 'Deploy Image', the 'Image Stream Tag' tab is selected. The 'Image Name' input field contains 'atlassian/bitbucket-server'. A yellow box highlights both this input field and the magnifying glass search icon.

Step 2.3: Leave by the moment the default config. since it is enough for the basic setup. Press Create

The screenshot shows the configuration for a new application. It includes fields for 'Name' (bitbucket-server), 'Pull Secret' (Secret name), and environment variables. Under 'Labels', there is an 'app' label with value 'bitbucket-server'. At the bottom left of the configuration modal, a yellow box highlights the 'Create' button.

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 completion message 'Completed' and a blue 'Go to overview' button, which is highlighted with a yellow box. Below this, there is information about managing the app via the web console or command line tools, and a code block for the 'oc' command line tool.

```
oc login https://10.68.26.163:8443
oc project development
oc status
```

For more information about the command line tools, check the [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.

The screenshot shows the devonfw application interface. At the top, it says "APPLICATION bitbucket-server". Below that, under "DEPLOYMENT", it lists "bitbucket-server, #1". It details the container as "BITBUCKET-SERVER" with the image "atlassian/bitbucket-server 7f1a96b" and 318.0 MiB of memory. It shows ports: 7990/TCP and 1 other. To the right, there's a circular icon with the number "1" and arrows for scaling. In the "Networking" section, it shows "Internal Traffic" for "bitbucket-server" with port 7990/TCP mapping to 7990 and 1 other. On the right, a yellow box highlights the "External Traffic" section which shows the route "http://bitbucket-server-development.apps.10.68.26.163.nip.io" and the route configuration "Route bitbucket-server, target part 7990-tcp".

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

The screenshot shows the BitBucket setup wizard. The title is "Bitbucket setup". The "Welcome" screen has a "Language" dropdown set to "English (United States)". The "Database" section has two options: "Internal" (which is selected) and "External". A note says "For evaluation and demo purposes only." Below the "Internal" option, another note says "Recommended for production use. See our documentation for more information." At the bottom is a "Next" button.

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)

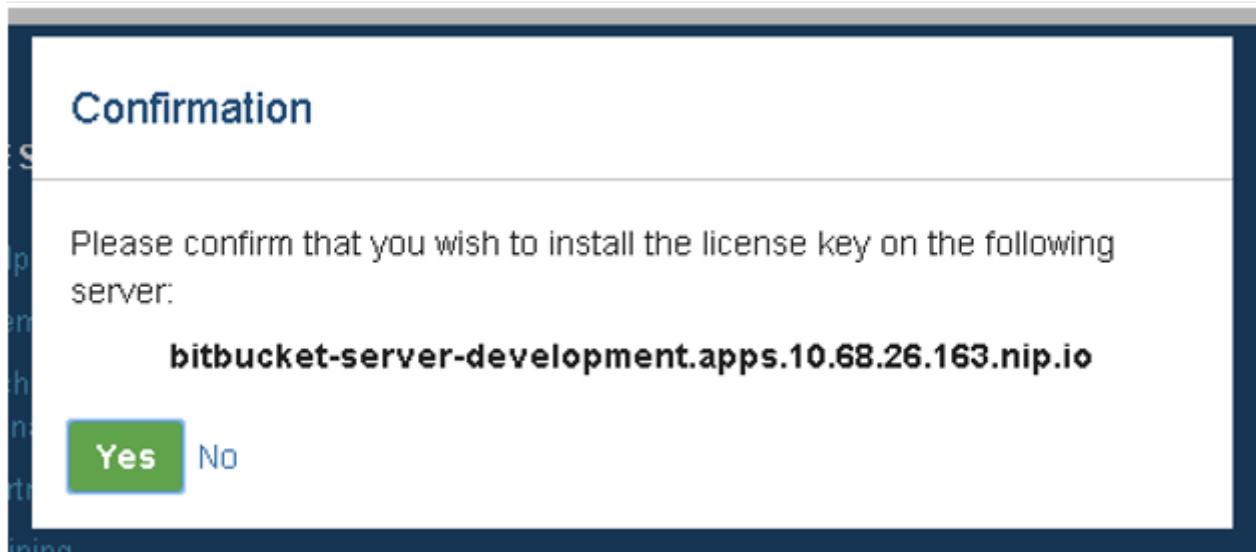
My Atlassian

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. The title is "Bitbucket setup". Under "Administrator account setup", there are fields for Username*, Full name*, Email address*, Password*, and Confirm password*. At the bottom are two buttons: "Integrate with JIRA" and "Go to Bitbucket", with "Go to Bitbucket" 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. The main message is "Welcome to Bitbucket Let's get started". Below it, it says "Familiar with Bitbucket? [Get back to it.](#)". On the right, there is a "Pull request" card for "Bugfix/TIS-57 buttons need to be red" with status "1 build green" and "TIS-57". A sidebar on the right says "User feedback is enabled" and "Ok, got it".

[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 the 'bitbucket-server' application. At the top, there are tabs for 'app' and 'bitbucket-server'. Below the tabs, there are buttons for 'Deploy' and 'Actions'. Underneath, there are tabs for 'History', 'Configuration' (which is selected), 'Environment', and 'Events'. The 'Configuration' tab has several sections: 'Selectors' (with 'app=bitbucket-server' and 'deploymentconfig=bitbucket-server'), 'Replicas' (set to 1), 'Strategy' (set to 'Rolling'), 'Timeout' (set to 600 sec), 'Update Period' (set to 1 sec), 'Interval' (set to 1 sec), 'Max Unavailable' (set to 25%), and 'Max Surge' (set to 25%). There is also a 'Template' section with a note about health checks. The 'Containers' section lists a single container named 'BITBUCKET-SERVER' with image 'atlassian/bitbucket-server:7f1a98b', ports 7990/TCP and 7999/TCP, and a mount point '/var/atlassian/application-data/bitbucket' (read-write). The 'Volumes' section is highlighted with a yellow box and contains a table with one row: 'bitbucket-server-1' (Remove), Type: 'empty dir (temporary directory destroyed with the pod)', and Medium: 'node's default'. To the right of the 'Volumes' section, there are sections for 'Autoscaling' (Add Autoscaler), 'Hooks' (Learn More), 'Triggers' (Manual (CLI) with 'oc rollout latest dc/bitbucket-server -n myproj'), 'Config' (Config), and 'New Image For' (myproject/bitbucket-server:latest).

Type:	empty dir (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

Volumes

bitbucket-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

[/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

[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:

New Image For:

`oc rollout latest dc/bitbucket-server -n mypi`

Config

myproject/bitbucket-server:latest

38.2. 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.

38.2.1. 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>

38.2.2. Prerequisites

- OpenShift up & running

38.2.3. 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 'My Project' interface. At the top, there's a navigation bar with 'Project My Project' on the left and 'Add to Project' with a dropdown menu on the right. The dropdown menu has options: 'Browse Catalog', 'Deploy Image' (which is highlighted with a yellow box), and 'Import YAML / JSON'. Below the navigation bar, there's a search bar with 'Name' and 'Filter by name' and a 'List by Resource Type' dropdown. The main area is titled 'Deployments' and lists two entries: 'bitbucket-server, #11' and 'crud, #1'. Each entry has a small icon and a 'DEPLOYMENT' label.

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)

This screenshot shows the 'Deploy Image' configuration page. At the top, there are tabs: 'My Project > Add to Project', 'Browse Catalog', 'Deploy Image' (which is underlined and highlighted with a yellow box), and 'Import YAML / JSON'. Below the tabs, there's a section for deploying an existing image from an image stream tag or docker pull spec. It includes fields for 'Namespace' (dropdown), 'Image Stream' (dropdown), and 'Tag' (dropdown). A radio button group is shown with 'Image Stream Tag' (unchecked) and 'Image Name' (checked). The 'Image Name' field contains 'selenium/hub' and has a magnifying glass icon to its right. The entire 'Image Name' input field is 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 Nameselenium/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

selenium-hub

A unique name for the route within the project.

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

/

Path that the router watches to route traffic to the service.

* Service

selenium-hub

Service to route to.

Target Port

4444 → 4444 (TCP)

Target port for traffic.

DONE !!

The Selenium Server is now accessible from outside the cluster. Click on the link of the route and you will be able to see the server home page.

APPLICATION
selenium-hub

DEPLOYMENT
selenium-hub, #1

CONTAINER: SELENIUM-HUB
Image: selenium/hub 61d075c 120.7 MB
Ports: 4444/TCP

Networking
SERVICE Internal Traffic
selenium-hub
4444/TCP (4444-tcp) → 4444

ROUTES External Traffic
<http://selenium-hub-development.apps.10.68.26.163.nip.io>
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.

Whoops! The URL specified routes to this help page.

For more information about Selenium Grid Hub please see the [docs](#) and/or visit the [wiki](#). Or perhaps you are looking for the Selenium Grid Hub [console](#).

Happy Testing!

Selenium is made possible through the efforts of our open source community, contributions from these [people](#), and our [sponsors](#).

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)

Project My Project Add to Project ▾

Name Filter by name Deploy Image

Browse Catalog Import YAML/JSON

Deployments

DEPLOYMENT bitbucket-server, #11

DEPLOYMENT crud, #1

Step 2: Introduce the appropriate resource name

(selenium-node-firefox in this case)

[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/node-firefox



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

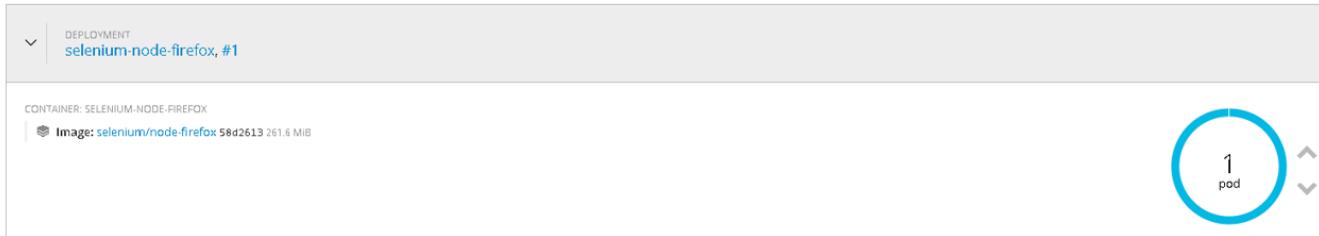
[About Environment Variables](#)

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.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:osjsh.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}{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



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](#)

38.3. 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

38.3.1. 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
```

38.3.2. 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
```

38.3.3. 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
```

38.3.4. 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
```

38.3.5. 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](#),

38.3.6. 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 'User Settings > Access Tokens'. It has a sub-section titled 'Personal Access Tokens' with the following text: 'You can generate a personal access token for each application you use that needs access to the GitLab API.' Below this, another section says: '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.'

To the right of these sections, there is a form for generating a new token. It includes fields for 'Name' (with placeholder 'My App'), 'Expires at' (with placeholder '2018-01-01'), and 'Scopes' (checkboxes for 'api', 'read_user', and 'read_registry'). A green button labeled 'Create personal access token' is located below the scopes.

Below the token creation form, a table titled 'Active Personal Access Tokens (2)' lists two existing tokens:

Name	Created	Expires	Scopes	Action
Jenkins CI server	Nov 2, 2017	In 6 months	api, read_user, read_registry	<button>Revoke</button>
devonfw-ci-token	Oct 30, 2017	In over 2 years	api, read_user, read_registry	<button>Revoke</button>

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 is set to 'GitLab API token'. Below it, the 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'API token' field contains a redacted string. The 'ID' and 'Description' fields are empty. At the bottom left is an 'OK' button.

Kind: GitLab API token

Scope: Global (Jenkins, nodes, items, all child items, etc)

API token:

ID:

Description:

OK

Then, we add the generated token in the **API token** field:

The screenshot shows the Jenkins 'Create New Credential' dialog. The 'Kind' dropdown is set to 'GitLab API token'. Below it, the 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'API token' field now contains a visible redacted string. The 'ID' and 'Description' fields are empty. A cursor arrow points towards the 'OK' button at the bottom left.

Kind: GitLab API token

Scope: Global (Jenkins, nodes, items, all child items, etc)

API token:

ID:

Description:

OK

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.

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/hooks/123456789](#) 1

Enabled GitLab triggers

Push Events

Opened Merge Request Events

Accepted Merge Request Events

Closed Merge Request Events

Rebuild open Merge Requests Never

Comments

Comment (regex) for triggering a build .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

Allow all branches to trigger this job

Filter branches by name 3

Include Cannot connect to GitLab to check whether selected branches exist. (show details)

Exclude

Filter branches by regex

Filter merge request by label

Secret token 4

Generate

Clear

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:



Build History trend —

#	Date	Started by
#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! :)

38.3.7. 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 user icon and the text "Back to credential domains" and "Add Credentials". The main area has a form for adding a new credential. The "Kind" dropdown is set to "Username with password". The "Scope" dropdown is set to "Global (Jenkins, nodes, items, all child items, etc)". The form fields are: "Username" (empty), "Password" (empty), "ID" (empty), and "Description" (empty). At the bottom right of the form is a blue "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***** (jenkins user credentials)	Username with password	"jenkins user credentials"
jenkins*****	Username with password	
Secret Token	Secret Token	Secret Token
user_jboss_realm_dig_id_directory	Username with password	"user_jboss_realm_dig_id_directory"
jenkins*****	Username with password	
jenkins***** (jenkins user credentials)	Username with password	"jenkins user credentials"
jenkins***** (jenkins user credentials)	Username with password	"jenkins user credentials for the realm *jenkins"
jenkins***** (jenkins user credentials)	Username with password	"jenkins user credentials for the realm *jenkins"
jenkins*****	Username with password	
jenkins***** (jenkins user credentials)	Username with password	"jenkins user for Resource access"
jenkins***** (jenkins user credentials)	Username with password	"jenkins user for Resource access"
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 --&gt;
&lt;activeProfiles&gt;
  &lt;activeProfile&gt;my.nexus&lt;/activeProfile&gt;
&lt;/activeProfiles&gt;
&lt;/settings&gt;</pre>
```

3. Use it in Jenkins Pipelines

38.3.8. 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:

The screenshot shows the Jenkins Plugins page with the search bar set to "sonarqu". The "Installed" tab is selected. A table lists three installed plugins:

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:

The screenshot shows the SonarQube "Log In to SonarQube" page. It has two input fields: one for "Login" containing "admin" and another for "Password" containing "*****". Below the fields are "Log in" and "Cancel" buttons.

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

devonfw-ci-token

Generate

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.

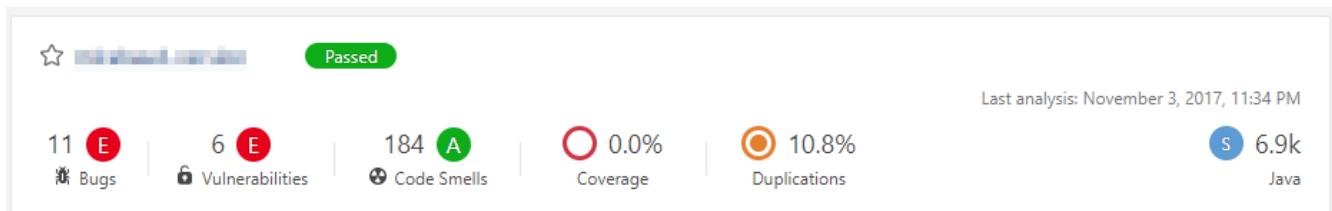
```
stage 'SonarQube Analysis'
def scannerHome = tool 'SonarQube scanner';
sh "${scannerHome}/bin/sonar-scanner \
-Dsonar.host.url=http://url-to-your-sq-server:9000/ \
-Dsonar.login=[SONAR_USER] -Dsonar.password=[SONAR_PASS] \
-Dsonar.projectKey=[PROJECT_KEY] \
-Dsonar.projectName=[PROJECT_NAME] \
-Dsonar.projectVersion=[PROJECT_VERSION] \
-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]"
}
```

38.4. OKD (*OpenShift Origin*)

38.4.1. 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](#)

38.4.2. 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

38.4.3. 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
```

38.4.4. 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.

38.4.5. 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](#)

38.4.6. 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
```

38.4.7. 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/icons.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.

Part VII: cicdgen

39. 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.

39.1. 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.

39.2. cicdgen CLI

For know more about how to use the cicdgen CLI, you can check the [CLI page](#)

39.3. cicdgen Schematics

For know more about how to use the cicdgen schematics, you can check the [schematics page](#)

39.4. Usage example

A [specific page](#) about how to use cicdgen is also available.

40. cicdgen CLI

40.1. 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](#)
- [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 --plurl devon.s2-eu.capgemini.com
```

- Generate all CICD files related to a devon4node project with OpenShift deployment.

```
$ cicdgen generate devon4ng --groupid com.devonfw --openshift --plurl devon.s2-eu.capgemini.com --ocurl ocp.itaas.s2-eu.capgemini.com --ocn devonfw
```

40.2. cicdgen usage example

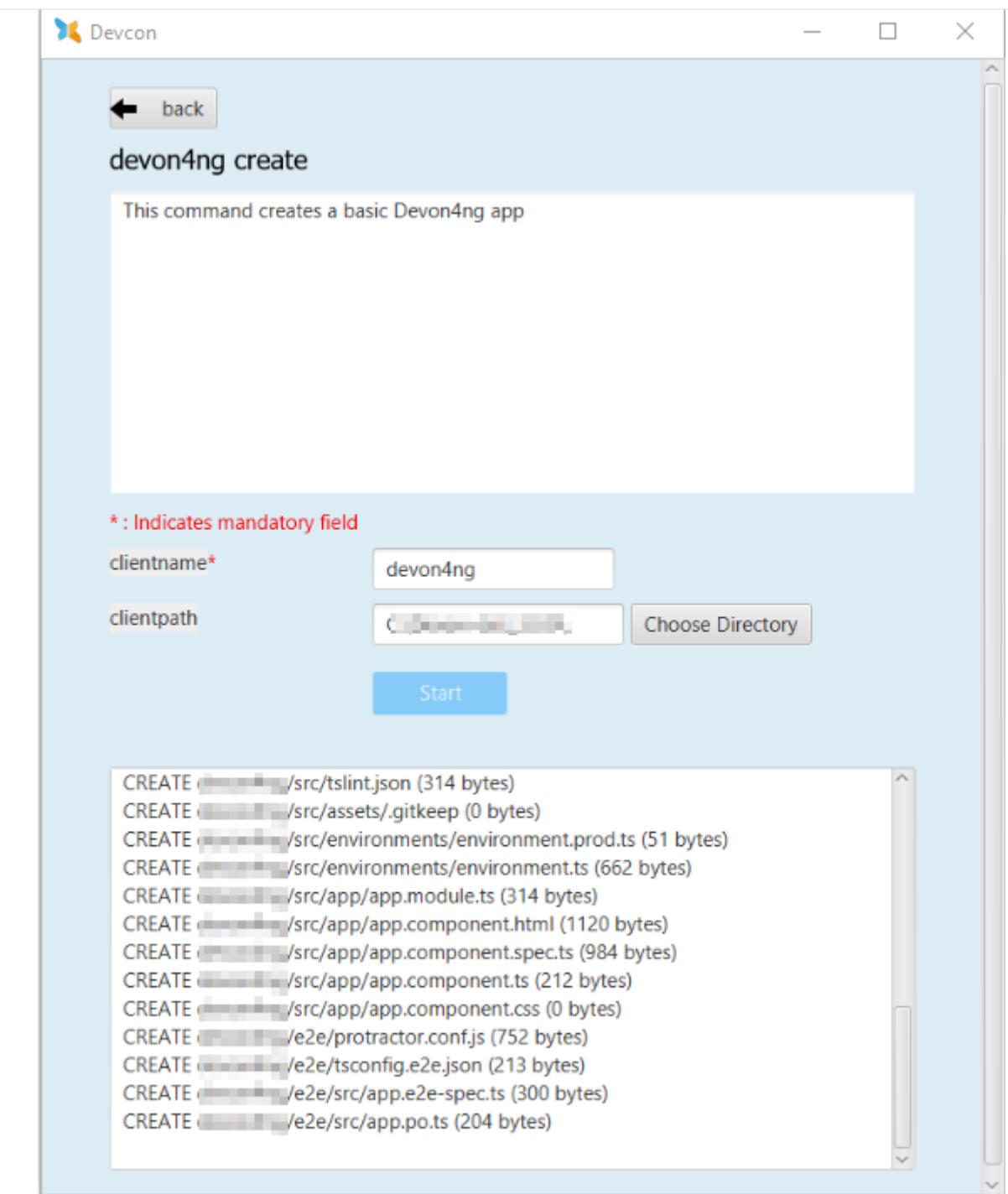
In this example we are going to show how to use cicdgen step by step in a devon4ng project.

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 devcon.



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

Keep your group current with content and updates from other services.

Search  All Sort by: Popularity ▾

MANAGE

- Configured
- My Accounts

CATEGORY

- All
- Analytics
- CRM
- Customer Support
- Developer Tools
- HR
- Marketing
- News & Social
- Project Management
- Others

Connectors for your team

	Yammer <small>Updated</small>	Receive updates from your Yammer network	Configure
	Trello	Manage Trello cards and tasks all in one place.	Configure
	GitHub	Manage and collaborate on code projects.	Configure
	Jenkins	Continuous Integration and Continuous Delivery	Configure
	Forms	Easily create surveys, quizzes, and polls.	Configure

All connectors

	Azure DevOps	Collaborate on and manage software projects online.	Add
	RSS	Get RSS feeds for your group.	Add
	Incoming Webhook		Add

- 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.

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

```
λ cicdgen --help
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]

Now we get in the previous steps, we will generate CICD files for a devonfw technology stack. We will use the command cicdgen generate <technology>. We will use the command cicdgen generate devon4ng. This command will generate all changes related to CICD for the devon4ng technology stack. The command cicdgen generate devon4node will generate all changes related to CICD for the devon4node technology stack. The command cicdgen generate devon4j will generate all changes related to CICD for the devon4j technology stack.
```

Options:
--help Show help [boolean]

Examples:
cicdgen devonfw-cicd generate devon4ng Generate all files for devon4ng

λ cicdgen generate devon4ng --help

λ

cmd

```
λ cicdgen generate devon4ng --help
Usage: cicdgen devonfw-cicd generate devon4ng [Options]
```

Options:
--help Show help [boolean]
--docker Should generate code for docker deployment? [boolean]
--plurl The production line url [string]
--openshift Should generate code for openshift deployment? [boolean]
--ocurl Openshift cluster url [string]
--ocn Openshift cluster namespace [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"]

Examples:
cicdgen devonfw-cicd generate devon4ng Generate all files for devon4ng

```
$ cicdgen generate devon4ng --groupid com.devonfw --docker --plurl devon.s2-eu.capgemini.com --teams --teamsname devon4ng --teamsurl https://outlook.office.com/webhook/...
```

```
cmd
{144,47} size, {144,1000} buffer
C:\Users\darrodrri\Documents\Proyectos\test\devon4ng (master)
λ cicdgen generate devon4ng --groupid com.devonfw --plurl devon.s2-eu.capgemini.com --teams --teamsname devon4ng --team
surl https://outlook.office.com/webhook/
CREATE /Jenkinsfile (11036 bytes)
CREATE /.dockerrcignore (15 bytes)
CREATE /Dockerfile (306 bytes)
CREATE /Dockerfile.ci (138 bytes)
CREATE /nginx.conf (102 bytes)
UPDATE /karma.conf.js (1339 bytes)
UPDATE /package.json (1346 bytes)
UPDATE /angular.json (3523 bytes)
[master 10f5893] Added CICD files to the project
 8 files changed, 361 insertions(+), 7 deletions(-)
create mode 100644 .dockerrcignore
create mode 100644 Dockerfile
create mode 100644 Dockerfile.ci
create mode 100644 Jenkinsfile
create mode 100644 nginx.conf
```

4. Create a git repository and upload the code

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 Create from template Import project

Project name: devon4ng

Project URL: https://devon.s2-eu.capgemini.com/darrodrri **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**

Project **Details** **Activity** **Milestones** **Snippets**

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 "darrodrri@capgemini.com"
```

Create a new repository

```
git clone https://devon.s2-eu.capgemini.com/gitlab/darrodrri/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/darrodrri/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/darrodrri/devon4ng.git
git push -u origin --all
git push -u origin --tags
```

Collapse sidebar

```
$ git remote add origin https://devon.s2-eu.capgemini.com/gitlab/darrodri/devon4ng.git
$ git push -u origin master
```

cmd

```
C:\Users\darrodri\Documents\Proyectos\test\devon4ng (master)
λ git remote add origin https://devon.s2-eu.capgemini.com/gitlab/darrodri/devon4ng.git

C:\Users\darrodri\Documents\Proyectos\test\devon4ng (master)
λ 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'.

C:\Users\darrodri\Documents\Proyectos\test\devon4ng (master)
λ |
```

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

The screenshot shows the Jenkins Multibranch Pipeline configuration for the project 'devon4ng'. The 'Branch Sources' tab is active, displaying a 'Git' configuration. The 'Project Repository' field contains the URL 'http://gitlab-core:80/gitlab/darrodri/devon4ng', with a red arrow pointing to the 'Internal GitLab URL' label. The 'Credentials' field is set to 'darrodri***** (Credentials for gitlab-val)'. The 'Behaviours' section includes a 'Discover branches' button. The 'Property strategy' is set to 'All branches get the same properties'. The 'Build Configuration' tab shows 'Mode' set to 'by Jenkinsfile' and 'Script Path' set to 'Jenkinsfile'.

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

Jenkins > configuration

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

Jenkins > Global Tool Configuration

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
```

Tool Home: /usr/bin

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.

Dario Rodriguez Gonzalez > devon4ng > Integrations Settings

Integrations

Webhooks can be used for binding events when something is happening within the project.

URL

http://jenkins-core:8080/jenkins/project/devon4ng

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
This URL will be triggered by a push to the repository
Branch name or wildcard pattern to trigger on (leave blank for all)

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

Confidential Comments
This URL will be triggered when someone adds a comment on a confidential issue

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

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.

41. cicdgen Schematics

41.1. 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
- Nexus 3
- SonarQube

The list of available schematics are:

- [devon4j](#)
- [devon4ng](#)
- [devon4node](#)

How to run the schematics

You can run the schematics using the schematic CLI provided by the angular team, but the easiest way to run it is using the [cicdgen CLI](#) which is a wrapper for the angular 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.

41.2. 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 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](#)

- --plurl

Url of Production Line. 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 docker](#) (same as `--docker`)

- --ocurl

OpenShift cluster url where the application will be builded and deployed.

- --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.

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.

- pom.xml

The pom.xml is modified in order to add the distributionManagement.

- Jenkinsfile

The Jenkinsfile is the file which defines 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 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 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/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.

- 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:
 - Setup pipeline: We 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 `--plurl` is also required. It will be used to upload the images to the Nexus3 inside Production Line. Example: if your PL url is `test.s2-eu.capgemini.com` you should execute the command in this way: `cicdgen generate devon4ng --groupid com.devonfw --docker --plurl test.s2-eu.capgemini.com`, and it will use `docker-registry-test.s2-eu.capgemini.com` as docker registry.

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.

41.3. 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](#)

- `--plurl`

Url of Production Line. 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`)

- `--ocurl`

OpenShift cluster url where the application will be builded and deployed.

- `--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 longer 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.

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 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.
- 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/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 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 `--plurl` is also required. It will be used to upload the images to the Nexus3 inside Production Line. Example: if your PL url is `test.s2-eu.capgemini.com` you should execute the command in this way: `cicdgen generate devon4ng --groupid com.devonfw --docker --plurl test.s2-eu.capgemini.com`, and it will use `docker-registry-test.s2-eu.capgemini.com` as docker registry.

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 use 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 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 and the nginx.conf from Jenkins to the image.

- nginx.conf

Configuration file for our nginx server. It defines the root folder of our application where docker copy the files to. Also it defines a fallback route to the index as described in the [angular deployment guide](#) in order to enable the angular routes.

41.4. 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](#)

- `--plurl`

Url of Production Line. 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`)

- `--ocurl`

OpenShift cluster url where the application will be builded and deployed.

- `--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.

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 to run the application in a docker container. It is necessary because we change a little bit the folder structure when we put all files in a docker image, so the script `start:prod` does not work.

- .gitignore

Defines all files that git will ignore. e.g: compiled files, IDE configurations.

- 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.
- 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/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 `--plurl` is also required. It will be used to upload the images to the Nexus3 inside Production Line. Example: if your PL url is `test.s2-eu.capgemini.com` you should execute the command in this way: `cicdgen generate devon4node --groupid com.devonfw --docker --plurl test.s2-eu.capgemini.com`, and it will use `docker-registry-test.s2-eu.capgemini.com` as docker registry.

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.

- `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 VIII: CobiGen — Code-based incremental Generator

42. Document Description

This document contains the documentation of the CobiGen core module as well as all CobiGen plug-ins and the CobiGen eclipse integration.

Current versions:

- CobiGen - Eclipse Plug-in v4.4.0
- CobiGen - Maven Build Plug-in v4.1.0
- CobiGen v5.3.0
- CobiGen - Java Plug-in v2.1.0
- CobiGen - XML Plug-in v4.1.0
- CobiGen - TypeScript Plug-in v2.1.1
- CobiGen - Property Plug-in v2.0.0
- CobiGen - Text Merger v2.0.0
- CobiGen - JSON Plug-in v2.0.0
- CobiGen - HTML Plug-in v2.0.1
- CobiGen - Open API Plug-in v2.3.0
- CobiGen - FreeMarker Template Engine v2.0.0
- CobiGen - Velocity Template Engine v2.0.0

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

43. 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 more 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.

44. CobiGen - Code-based incremental Generator

44.1. 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

44.2. Architecture

CobiGen is build as an extensible framework for incremental code generation. It provides extension points for new input readers allowing to read 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:

44.3. 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.

44.4. Selection of current and past CobiGen applications

General applications:

- Generation of a **Java CRUD application based on deonfw architecture** including all software-layers on the server plus code for js-clients (AngularJs). 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 testdata** 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 dependending 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 an good overview about the currently available tests in the software system, which enables further human analysis.

45. 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](#).

45.1. 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.

45.1.1. 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](#).

45.1.2. 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!

45.1.3. 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 85. 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 86. 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 87. 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.

45.2. Register Factory

45.2.1. 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.

45.2.2. Testdata Builder

Analogous to [Testdata Builder for devon4J](#)

45.2.3. 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!

46. CobiGen

46.1. 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
```

46.1.1. Context Configuration

The context configuration (`context.xml`) always has the following root structure:

Listing 88. 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 89. 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:

$$!(\text{NOT} \mid\mid \cdots \mid\mid \text{NOT}) \And \cdots \And \text{NOT} \mid\mid (\text{OR} \mid\mid \cdots \mid\mid \text{OR})$$

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 satisfyable, to be fully functional. A `<matcher>` node declares a specific characteristics a valid input should have.

Listing 90. 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 91. 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.

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 92. 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

46.1.2. 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 intially describe the first (more extensive) configuration style. Such an configuration root structure looks as follows:

Listing 93. 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 94. Example Template Configuration

```
<templates>
    <template id="..." destinationPath="..." templateFile="..." mergeStrategy="..."
              targetCharset="..." />
    ...
</templates>
```

A template declaration consist of multiple information:

- The attribute `id` 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 mergable)
- 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 95. 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-ending by the following file naming convention: `<filename>.<fileending>.ftl` Thus the file-ending `.ftl` will be removed after generation.
- `templateFile`: relative path to the file found
- `mergeStrategy`: (*optional*) not set means not mergable
- `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 96. 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.

46.1.3. 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 indeterministically. 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.

46.1.4. 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.

46.1.5. 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`

46.1.6. 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 bugfixes in any of the plug-ins as

they can be released completely independently.

46.2. Plug-ins

46.2.1. 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 97. 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 98. 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 99. 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 100. 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 101. 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.capgemini.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 fqdn)** (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)

46.2.2. 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.

46.2.3. 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 102. Example of a xml trigger definition.

```
<trigger id="..." type="xml" templateFolder="..."><!-- ... -->
  ...
</trigger>
```

This trigger type enables xml documents as inputs.

- type 'xpath'

Listing 103. 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 104. 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 105. 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 106. 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's 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 following four merge strategies are implemented and can be configured in the [templates.xml](#):

- [xmlmerge](#): In case of a conflict the base value is preferred
- [xmlmerge_override](#): In case of a conflict the patch value is preferred
- [xmlmerge_attachTexts](#): In case of a conflict the base value is preferred. Attributes and text nodes will be merged where possible
- [xmlmerge_override_attachTexts](#): In case of a conflict the patch value is preferred. Attributes and text nodes will be merged where possible

Currently only the document types included in LeXeMe are supported. On how the merger works consult the [LeXeMe Wiki](#).

46.2.4. 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 comparision on what files look like before and after merging

override

Listing 107. Before

```
// anchor:part:override:anchorend
Lorem Ipsum
```

Listing 108. Patch

```
// anchor:part:override:anchorend
Dolor Sit
```

Listing 109. After

```
// anchor:part:override:anchorend
Dolor Sit
```

Appending

Listing 110. Before

```
// anchor:part:append:anchorend  
Lorem Ipsum  
// anchor:part2:appendafter:anchorend  
Lorem Ipsum  
// anchor:part3:appendbefore:anchorend  
Lorem Ipsum
```

Listing 111. Patch

```
// anchor:part:append:anchorend  
Dolor Sit  
// anchor:part2:appendafter:anchorend  
Dolor Sit  
// anchor:part3:appendbefore:anchorend  
Dolor Sit
```

Listing 112. 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 113. Before*

```
// anchor:part:newline_append:anchorend  
Lorem Ipsum  
// anchor:part:append_newline:anchorend  
Lorem Ipsum  
(end of file)
```

Listing 114. Patch

```
// anchor:part:newline_append:anchorend  
Dolor Sit  
// anchor:part:append_newline:anchorend  
Dolor Sit  
(end of file)
```

Listing 115. 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.

46.2.5. 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.

46.2.6. 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 startegy, 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.

46.2.7. HTML Plug-in

The HTML Plug-in enables merging result HTML files to existing ones. This plug-in is used at the moment for generating 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.

47. Maven Build Integration

47.1. Maven Build Integration

For maven integration of CobiGen you can include the following build plugin into your build:

Listing 116. Build integration of CobiGen

```
<build>
  <plugins>
    <plugin>
      <groupId>com.devonfw.cobigen</groupId>
      <artifactId>maven-plugin</artifactId>
      <version>VERSION-YOU-LIKE</version>
      <executions>
        <execution>
          <id>cobigen-generate</id>
          <phase>site</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.

47.1.1. 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 117. 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 118. 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>`).

47.1.2. 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 \${basedir}*
- `<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 119. 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>
```

47.1.3. 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 120. Example of a full configuration including plugins

```
<build>
  <plugins>
    <plugin>
      <groupId>com.devonfw.cobigen</groupId>
      <artifactId>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>
```

47.1.4. A full example

1. A complete maven configuration example

```
<build>
  <plugins>
    <plugin>
      <groupId>com.devonfw.cobigen</groupId>
      <artifactId>maven-plugin</artifactId>
      <version>3.0.0</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>1.0.0</version>
        </dependency>
        <dependency>
          <groupId>com.devonfw.cobigen</groupId>
          <artifactId>javaplugin</artifactId>
          <version>1.6.0</version>
        </dependency>
      </dependencies>
    </plugin>
  </plugins>
</build>
```

48. Eclipse Integration

48.1. Installation

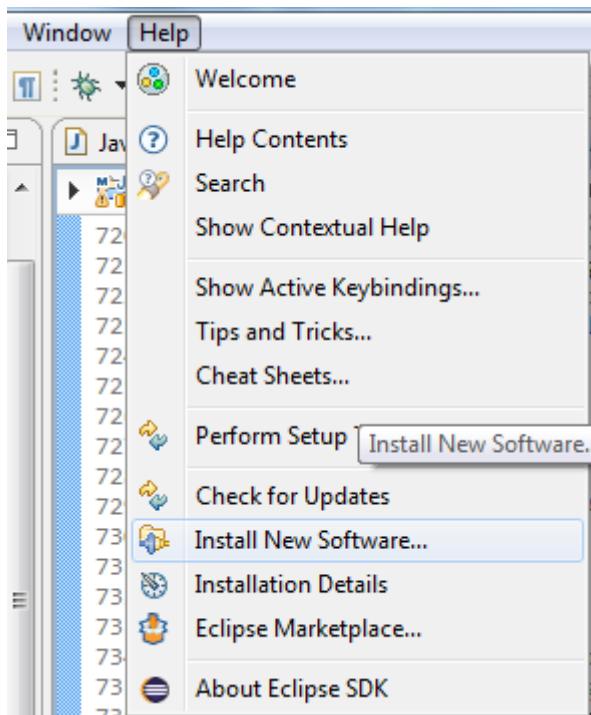
Remark: CobiGen is preinstalled in the [devonfw/devon-ide](#).

48.1.1. Preconditions

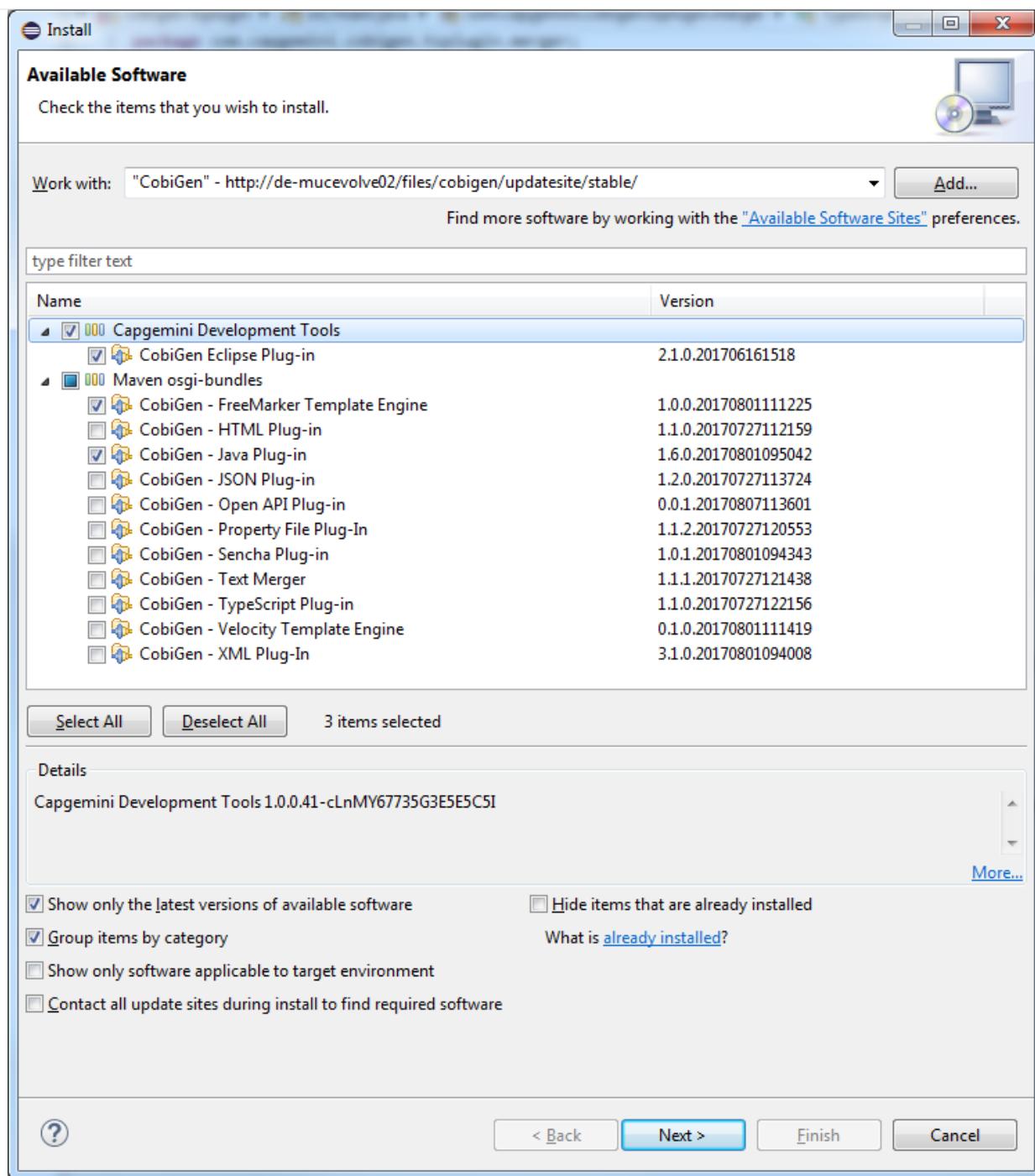
- Eclipse 4.x
- Java 7 Runtime (for starting eclipse with CobiGen). This is independent from the target version of your developed code.

48.1.2. 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: <http://de-mucevolve02/files/cobigen/updatesite/stable/>
 - Beta/RC releases: <http://de-mucevolve02/files/cobigen/updatesite/experimental/>



3. Follow the installation wizard

Select *CobiGen Eclipse Plug-in* → *Next* → *Next* → accept the license → *Finish* → *OK* → *Yes*

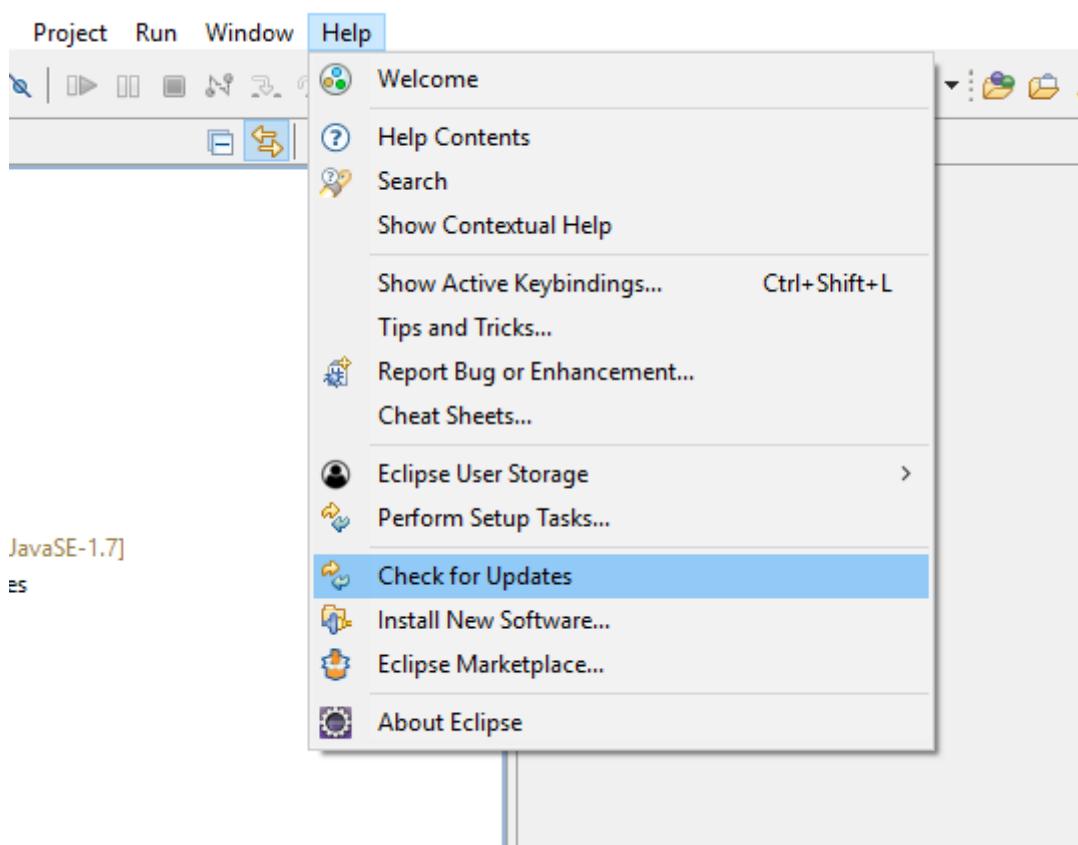
4. Once installed, a new menu entry named "CobiGen" will show up in the *Package Explorer*'s context menu. In the sub menu there will be the *Generate...* command, which may ask you to update the templates, and then you can start the generation wizard of CobiGen. You can adapt the templates by clicking 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/tools-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.

48.1.3. 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.

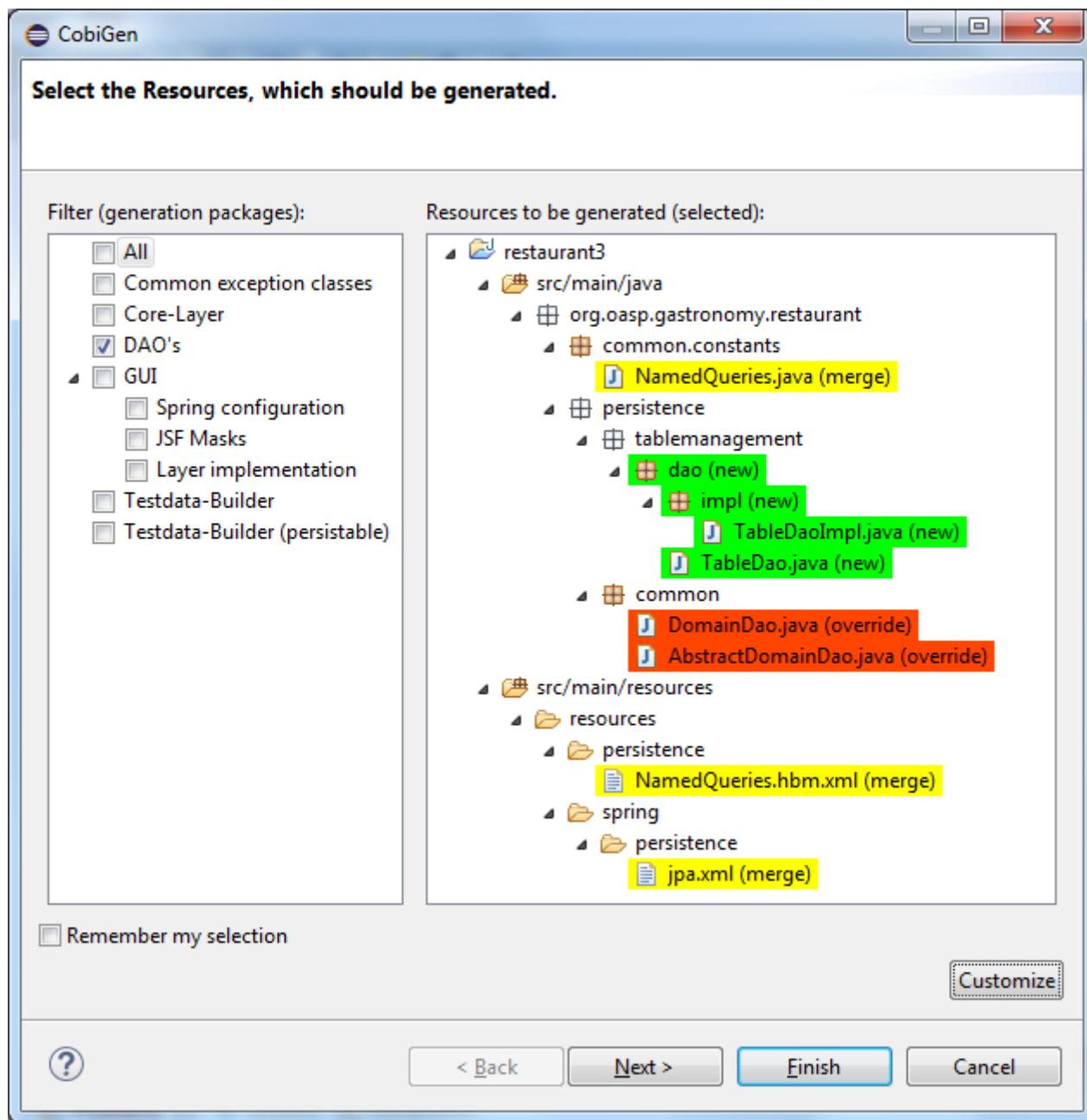
48.2. Usage

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.

48.2.1. 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, which might be effected by the generation. Dependent on the increment selection on the left side, potentially effected files will be shown on the right 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.

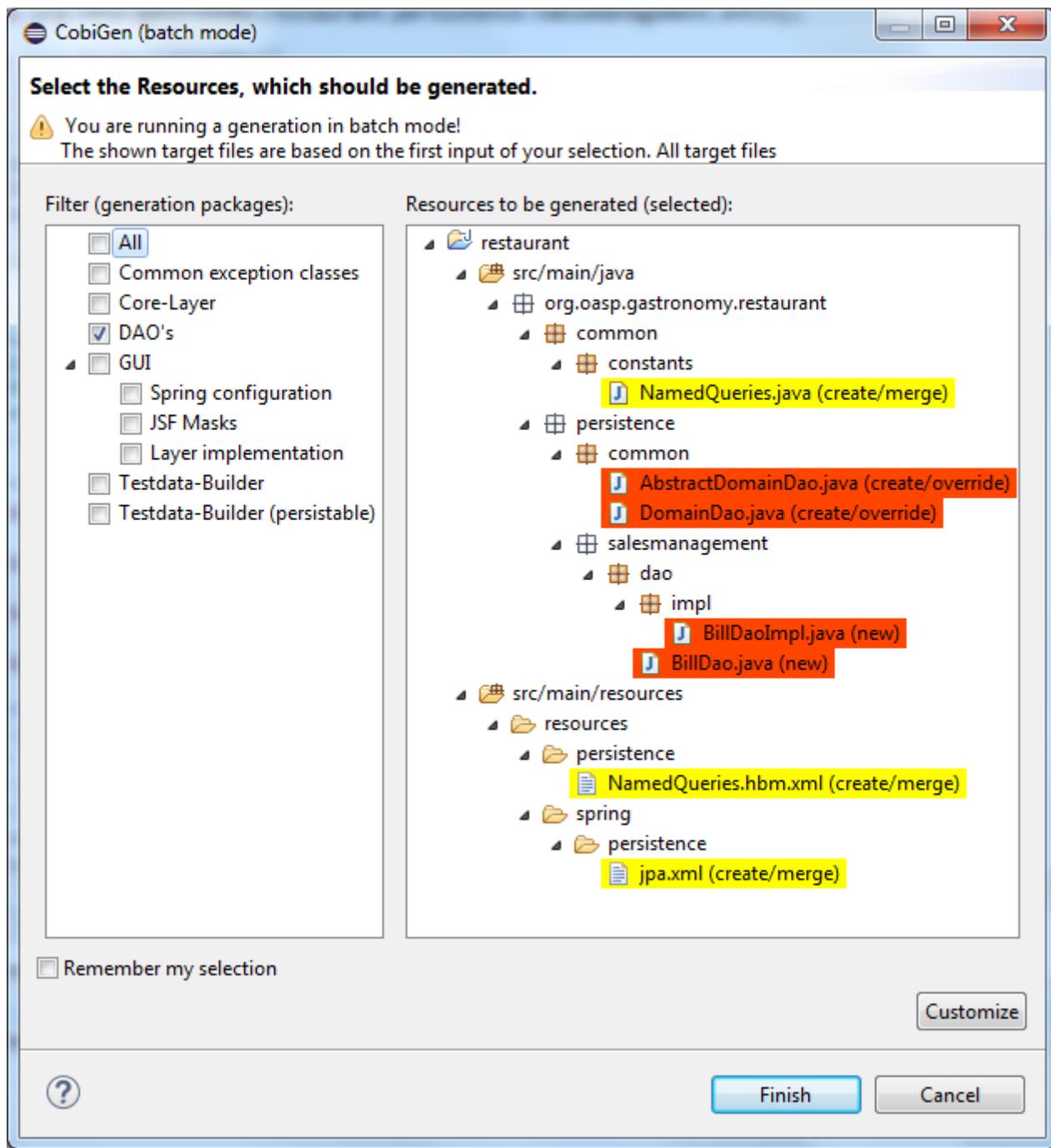
Selecting an increment on the left side will initialize the selection of all shown files to be generated on the right side, whereas green and yellow categorized files will be selected initially. A manual modification of the pre-selection can be performed by switching to the customization tree using the *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.

48.2.2. Batch mode

Are there 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.



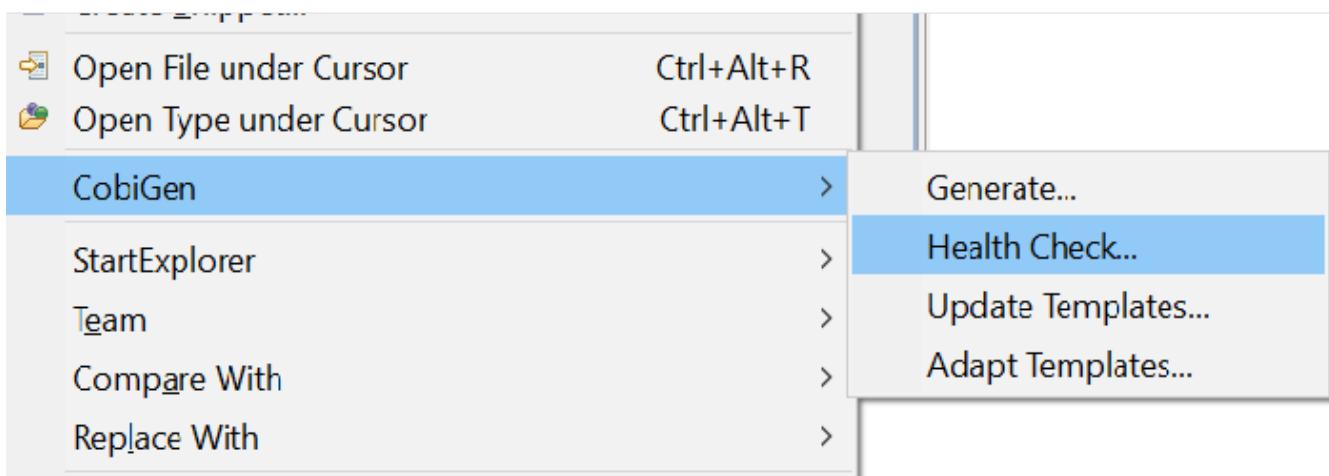
Thus the color encoding differs also a little bit:

- **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.

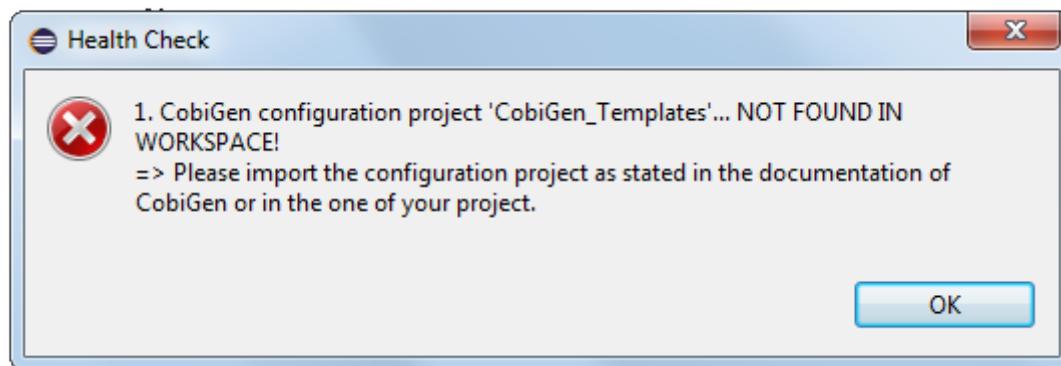
48.2.3. Health Check

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.



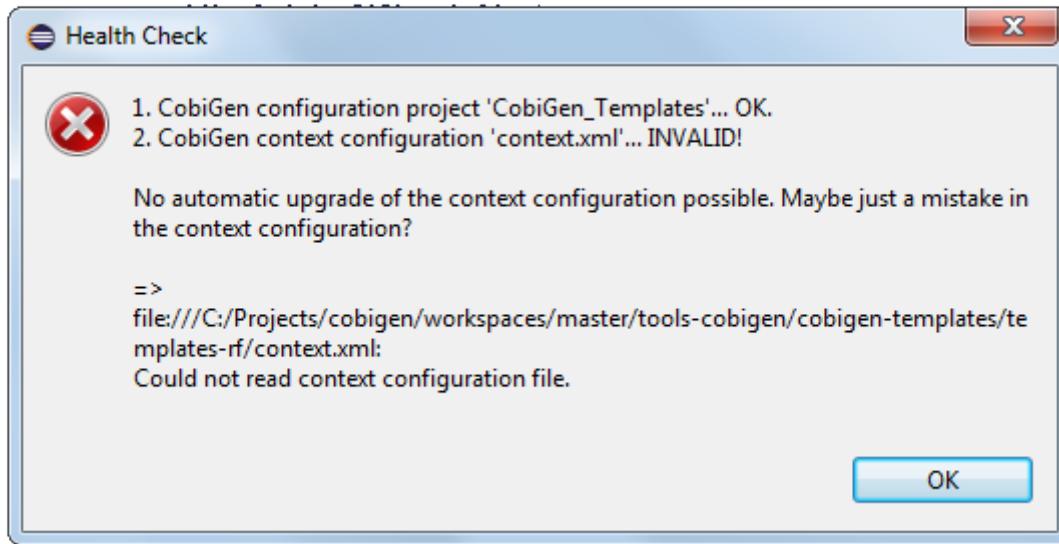
The simple *Health Check* includes 3 checks. As long as any of these steps fails, the *Generate* menu entry is grayed out.

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](#).

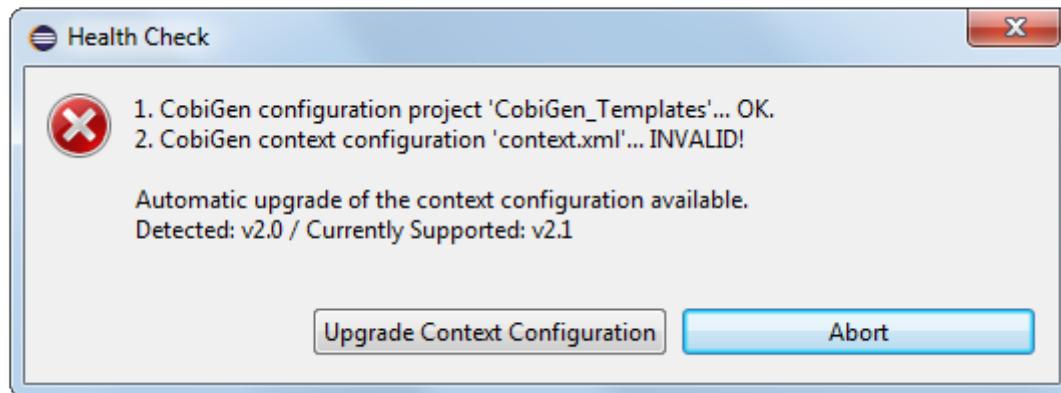
The second step is to check whether the template project includes a valid *context.xml*. If this check fails, you will see the following message:



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.

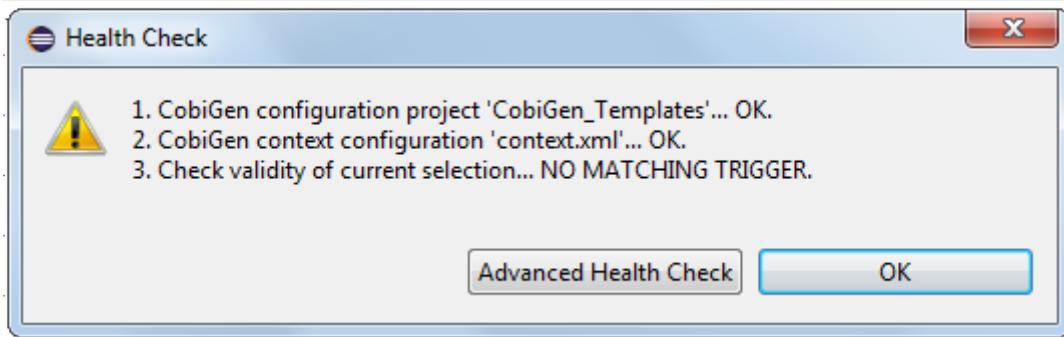
If all this is not the case, such as, there is a *context.xml*, which can be successfully read by CobiGen, you might get the following information:



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.

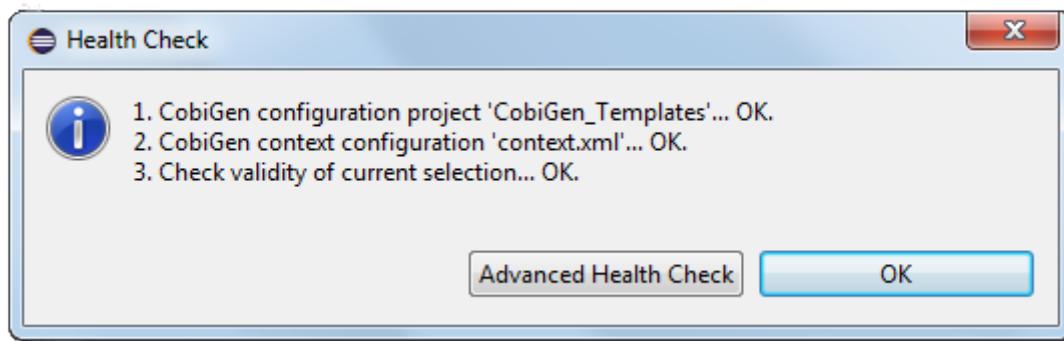
Remark: This will create a backup of your current context configuration and converts your old configuration to the new format. The upgrade will remove all comments from the file, which could be retrieved later on again from the backup. If the creation of the backup fails, you will be asked to continue 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:



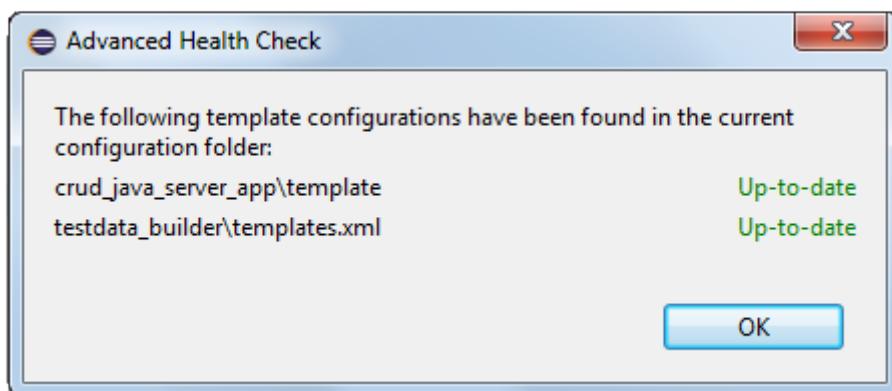
This indicates, that there no trigger has been activated, which matches the current selection. The reason might be that your selection is faulty or that you imported the wrong template project (e.g. you are working on a devon4j project, but imported the Templates for the Register Factory). If you are 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.

48.2.4. 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 .

48.2.5. 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.

48.3. 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>
```

49. Template Development

49.1. Helpful links for template development

- [FreeMarker Root Page](#)
 - [Expressions Cheat Sheet](#)
 - [Complete Language reference](#)
 - [FreeMarker Template Tester](#)
- [Variables to access Java source model](#)

Part IX: devonfw testing

50. Home



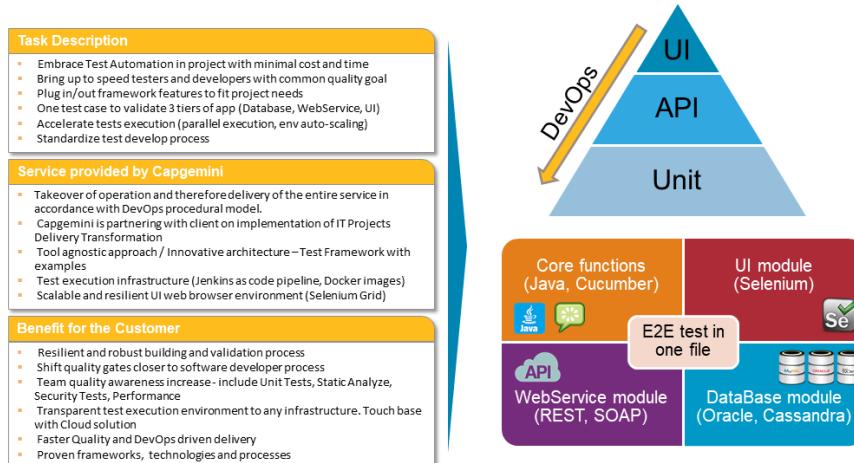
50.1. Contact

In case of any questions, please send mail to: DL PL E2E Test Framework <e2etestframework.pl@capgemini.com>

50.2. What is E2E Mr Checker Test Framework

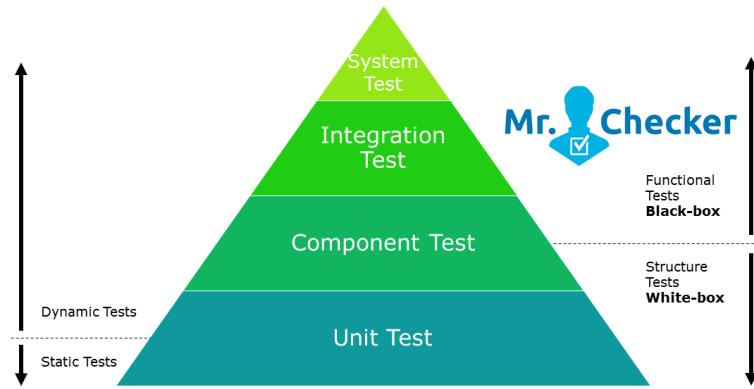
Mr Checker Test Framework is an end-to-end test automation framework written in Java.

E2E Test Framework for DevOps & Smart Automation



50.3. Where Mr Checker applies?

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 he is able to attach his tests to any application interfaces, such as: - IP address - Domain Name - communication protocol - Command Line Interface.

Mr Checker'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

50.3.1. Test stages

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 the enablement of stubs and special test tools.

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 step by step integrated into the bigger components and tested as such.

Integration test

Functions are tested by feeding them input and examining the output, and internal program

structure is rarely considered. The software is step by step completed and tested by the tests covering a collaboration of modules or classes. The integration depends on the kind of system. For example, the steps could be as such: run the operating system first and gradually add one component after another, then check if the black-box tests still are 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.

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 the 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).

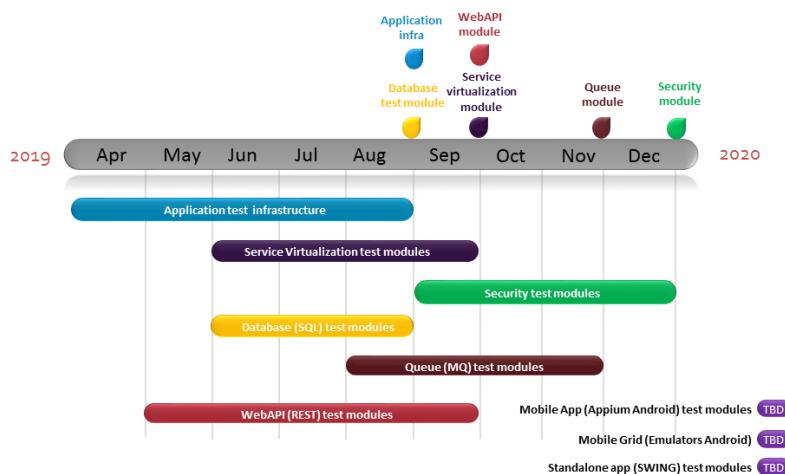
50.4. Benefits for the project

Every customer may benefit from using Mr Checker Test Framework. The main profits for the 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 Analyze, 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.

50.5. Road map plan

Mr Checker E2E Framework road map



All slides: [link](#)

50.6. Wiki Structure

- [How to install Mr Checker Test Framework](#)
- Mr Checker Test Framework modules:
 - Core test module
 - Selenium test module
 - WebAPI test module
 - Security test module
 - DataBase test module
 - Mobile test module
 - Standalone test module
 - DevOps module

51. How to install

There is one important pre-requisite for Mr Checker installation - there has to be Java 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

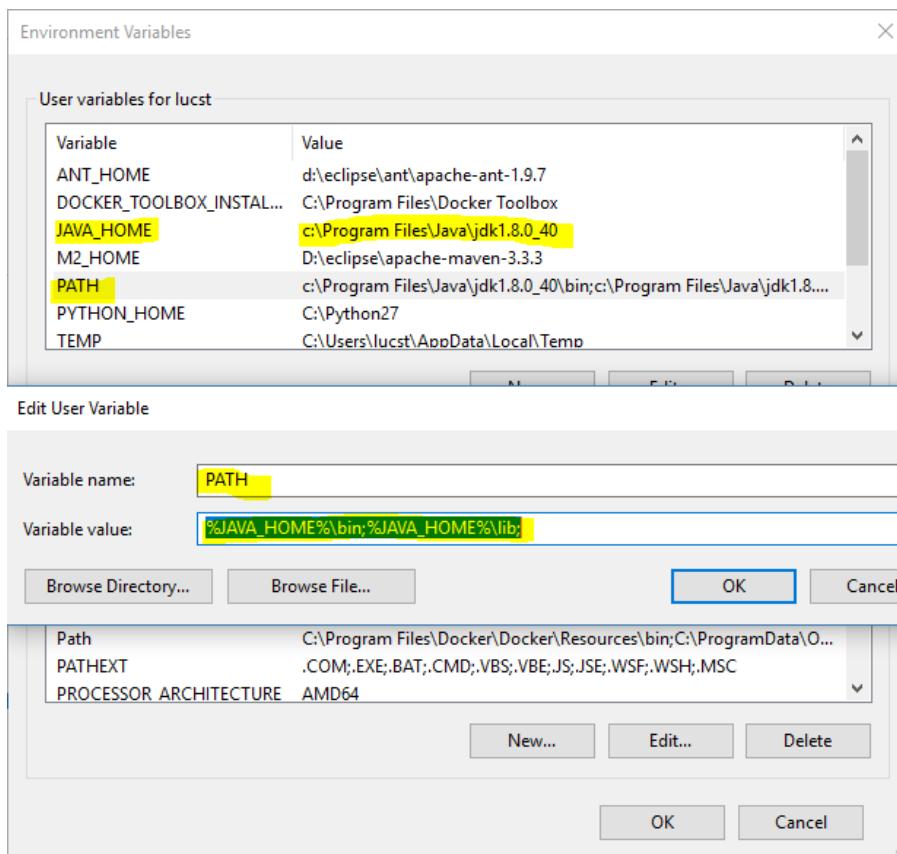
- Download and install [Java download link](#)

Java SE Development Kit 8u131
 You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.87 MB	jdk-8u131-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.81 MB	jdk-8u131-linux-arm64-vfp-hflt.tar.gz
Linux x86	164.66 MB	jdk-8u131-linux-i586.rpm
Linux x86	179.39 MB	jdk-8u131-linux-i586.tar.gz
Linux x64	162.11 MB	jdk-8u131-linux-x64.rpm
Linux x64	176.95 MB	jdk-8u131-linux-x64.tar.gz
Mac OS X	226.57 MB	jdk-8u131-macosx-x64.dmg
Solaris SPARC 64-bit	139.79 MB	jdk-8u131-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.13 MB	jdk-8u131-solaris-sparcv9.tar.gz
Solaris x64	140.51 MB	jdk-8u131-solaris-x64.tar.Z
Solaris x64	96.96 MB	jdk-8u131-solaris-x64.tar.gz
Windows x86	191.22 MB	jdk-8u131-windows-i586.exe
Windows x64	198.03 MB	jdk-8u131-windows-x64.exe

- 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



2. Verify in command line:

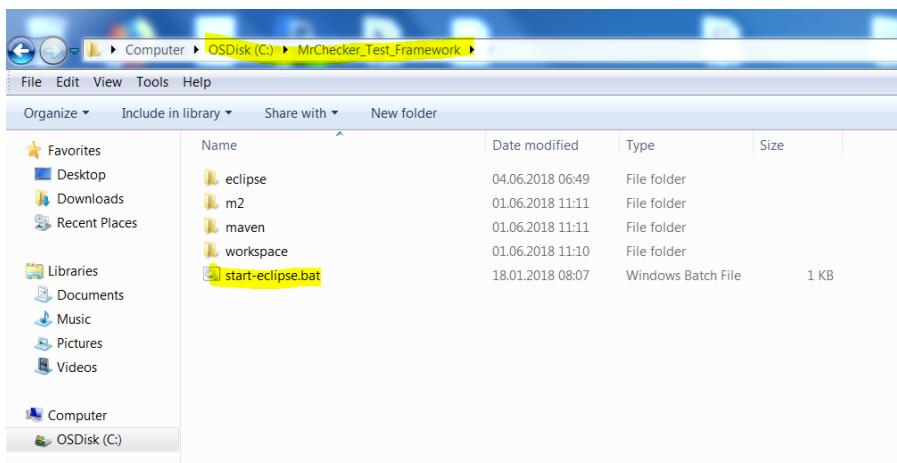
```
> java --version
```

Mr Checker installation can be done in three ways:

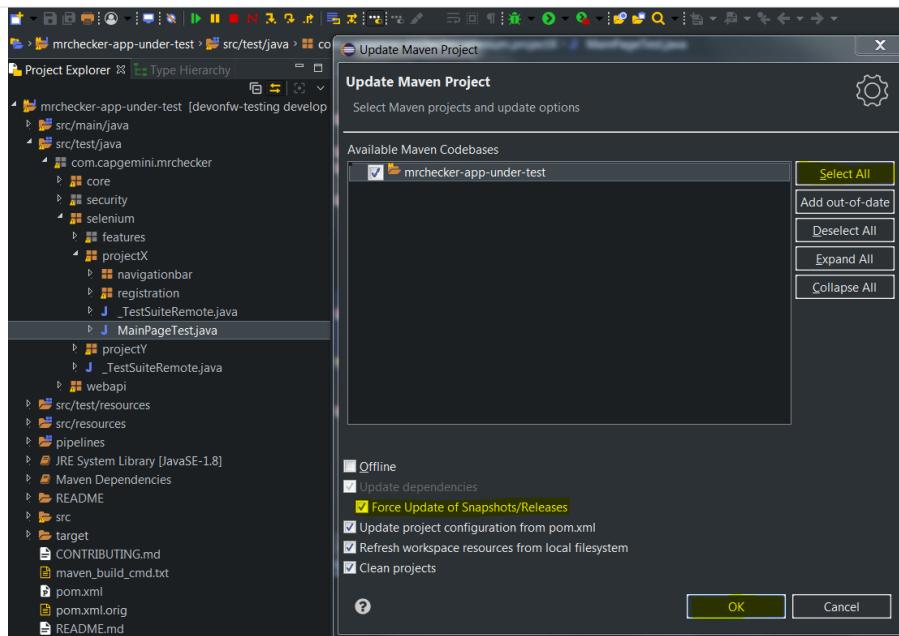
- **Easy out of the box installation** - Fast and easy solution - recommended for all users, who had previously not used any test automation environment. A drawback of these solutions is that it applies not for all Operation Systems
- **Out of the box installation** - with additional steps - when the first way is not working for you
- **Advanced installation** - Manual step by step installation containing all framework ingredients

51.1. Easy out of the box installation

1. Click on the link [Ready to use MrChecker_Test_Environment](#) and download the package
 2. Unzip downloaded MrChecker Test Framework to the folder C:\ on your PC - recommended tool: [7z](#) All necessary components, such as Eclipse, Java and Maven will be pre-installed for you. There is no need for additional installations.
- Note:** Please double check the place in which you have unzipped MrChecker_Test_Framework
3. Go to folder C:\MrChecker_Test_Framework\ , in which Mr.Checker has been unzipped



1. Double click on *start-eclipse.bat*
2. . Update project structure (*ALT + F5*)



If the script is not working for you - try:

51.2. Out of the box installation - with additional steps

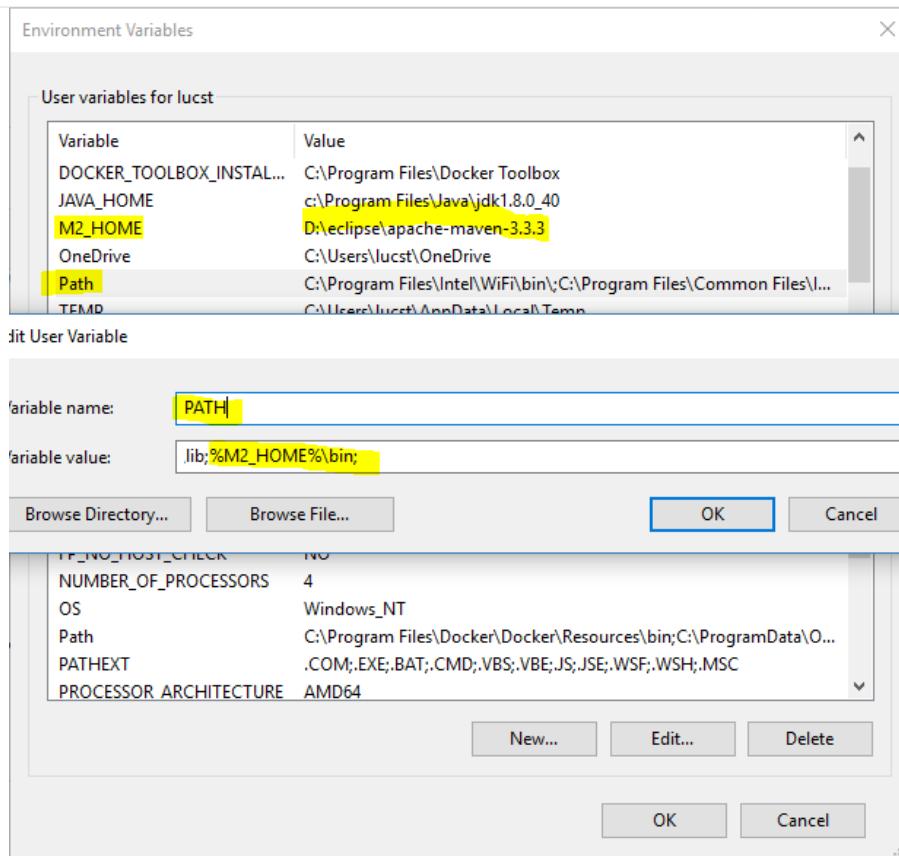
1. Open Eclipse
2. Manually Delete folders that appear in Eclipse
3. Click inside Eclipse with a right mouse click and open Import
4. Select Maven → existing Maven project
5. Select Mr Checker → workspace → devon project and click OK

At this point, all test catalogues should be imported in Eclipse and ready to use.

51.3. Advanced manual step by step installation

Install each component separately, or update the existing ones on your PC.

1. Maven 3.5
 - Download Maven <http://www-eu.apache.org/dist/maven/maven-3/3.5.0/binaries/apache-maven-3.5.0-bin.zip>
 - Unzip Maven in followin location C:\maven
 - Set Windows Local Environment
 - **Variable name:** M2_HOME | **Variable value:** c:\maven
 - **Variable name:** PATH | **Variable value:** %M2_HOME%\bin



- Verify in command line:

```
> mvn --version
```

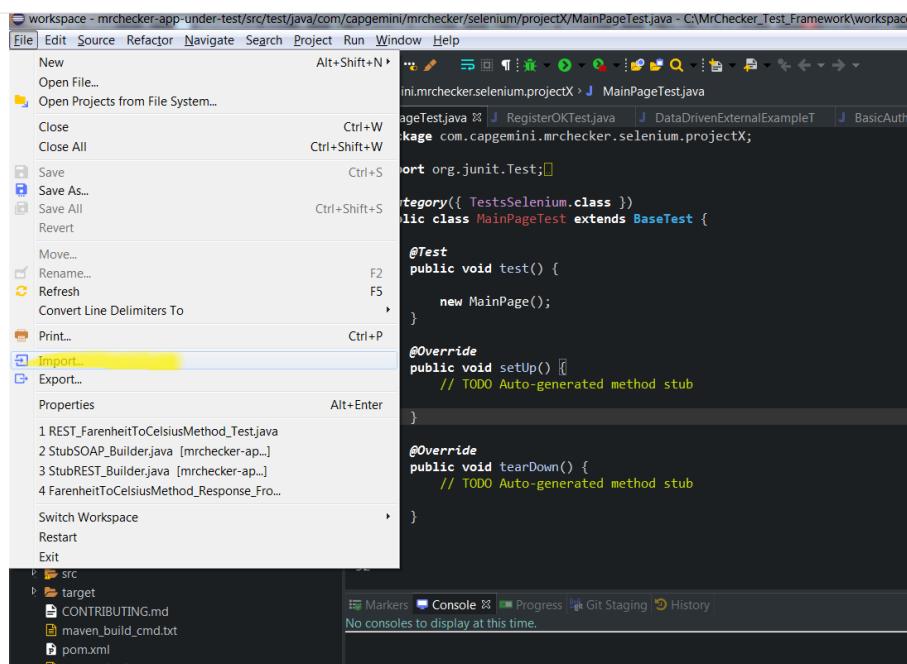
2. Eclipse IDE

- Download and unzip [Eclipse](#)

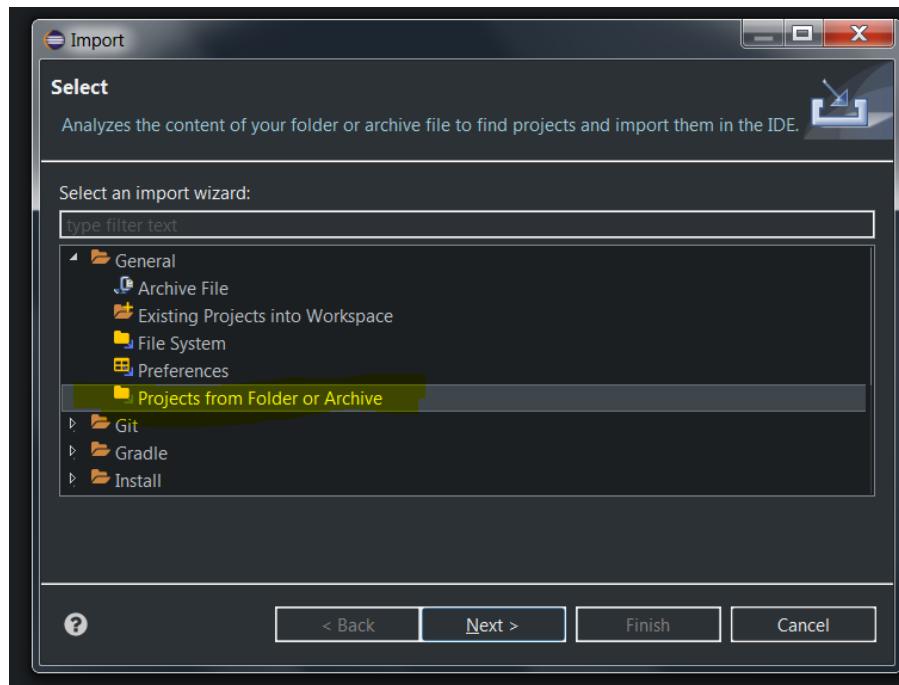
3. Download Mr Checker Test Framework [source code](#)

4. Import projects in Eclipse

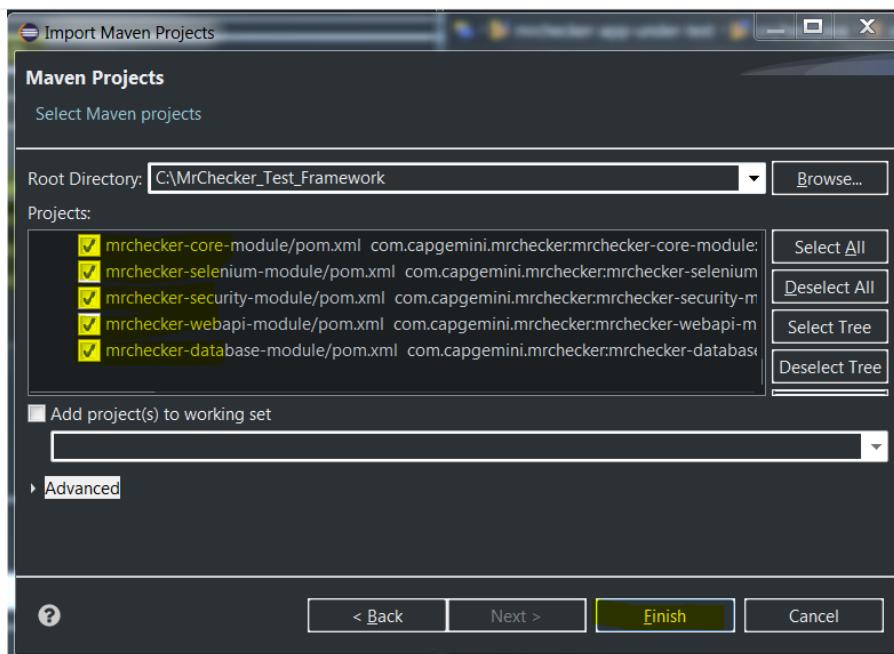
- Import:



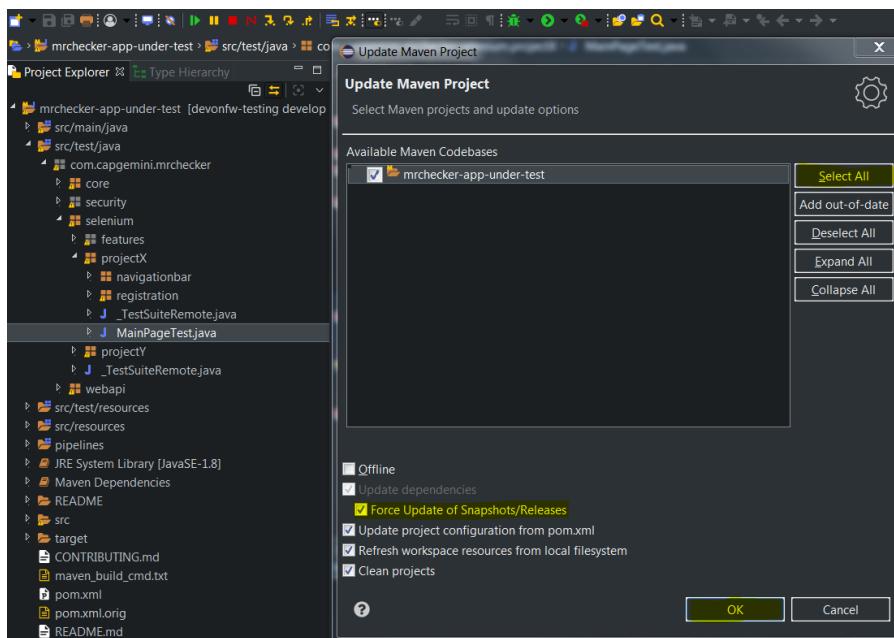
- Projects from folders:



- Open already created projects



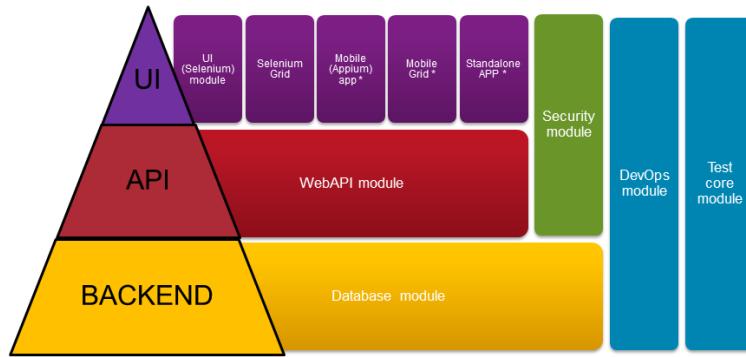
- Update project structure - *ALT + F5*



52. Mr Checker Test Framework modules

This is the structure of Mr Checker Framework Modules:

E2E Test Framework modules



* - task not started, to be defined

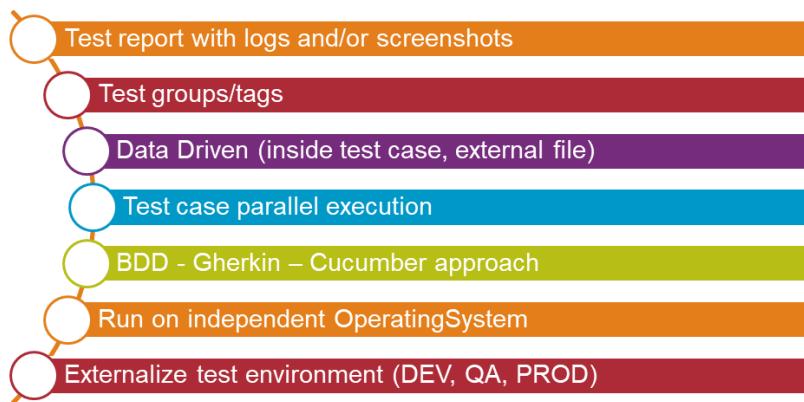
In this section, it is possible to find all information regarding the main modules of Mr Checker:

- [Core Test Module](#)
- [Selenium Test Module](#)
- [WebAPI Test Module](#)
- [Security Test Module](#)
- [Database Test Module](#)
- [Mobile Test Module](#)
- [Standalone Test Module](#)
- [DevOps Module](#)

52.1. Core Test Module

52.1.1. What is Core Test Module

Core functionality ingredients



52.1.2. 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

52.1.3. How to start?

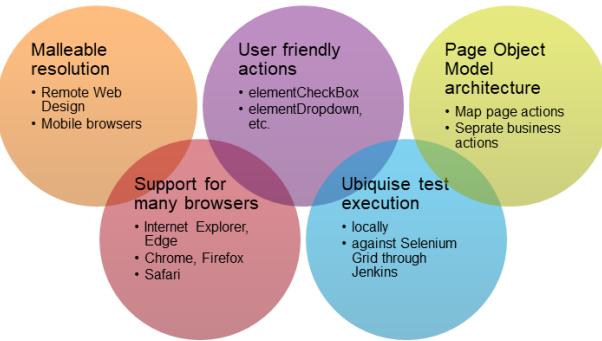
Read: [Framework Test Class](#)

Unresolved directive in devonfw-testing.wiki/master-devonfw-testing.asciidoc
include::framework-test-class.asciidoc[leveloffset=3]

52.2. Selenium Test Module

52.2.1. What is Mr Checker E2E Selenium Test Module

UI Selenium test module ingredients



52.2.2. Selenium Structure

- [What is Selenium](#)
- [What is WebDriver](#)
- [What is Page Object Model/Pattern](#)
- [List of web elements \(Button, Dropdown, Checkbox, Alert Popup, etc.\)](#)

52.2.3. Framework Features

- [Construction of Framework Page Class](#)
 - Every Page class must extend BasePage
 - What is isLoaded(), load() and pageTitle() for
 - How to create selector variable - 'private static final By ButtonOkSelector = By.Css(...)'
 - How to prepare everlasting selector - [documentation](#)
 - Method/action naming convention - [documentation](#)
 - 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 - [documentation](#)
- [Run on different browsers: Chrome, Firefox, IE, Safari, Edge](#)
- [Run with different browser options](#)
- [Run with full range of resolution \(mobile and desktop\): Testing Response Design Webpage](#)

52.2.4. How to start?

Read: [My first Selenium Test](#)

52.2.5. Selenium Best Practices

- [Table of best practices](#)

52.2.6. Selenium UFT Comparison

- [Selenium UFT Comparison](#)

Unresolved directive in devonfw-testing.wiki/master-devonfw-testing.asciidoc - include::Building-basic-Selenium-test.asciidoc[leveloffset=3]

Unresolved directive in devonfw-testing.wiki/master-devonfw-testing.asciidoc - include::WebAPI-test-module.asciidoc[leveloffset=2]

52.3. Security Test Module

52.3.1. 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 due to poor secret management process (which will have nothing to do with code itself), 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 be still needed. Nevertheless, every security requirement which are automatically verified, will prevent code degeneration and misconfiguration in a continuous manner.

52.3.2. How to test Security?

Security tests can be performed in many different ways like:

- **Static Code Analysis** - improves the security by (usually) automated code review. Good way to search after vulnerabilities, which are 'obvious' on the code level (like e.g. SQL injection). The downside is that the 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. Good way to search after 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 maintained tests. They work usually on the HTTP/REST level (as this defines the trust boundary between the client and the server) and run against a working environment. Unit tests are best suited to verify 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 security tests to run on will be **integration** (as 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.

52.3.3. Scope definition

Unresolved directive in devonfw-testing.wiki/master-devonfw-testing.asciidoc - include::DataBase-test-module.asciidoc[leveloffset=2]

52.4. Mobile Test Module

52.5. Standalone Test Module

52.6. DevOps Module

52.6.1. What is DevOps for us?

DevOps consists of a mixture of three key components in a technical project:

- People skills and mindset
- Processes
- Tools

By using **E2E Mr Checker Test Framework** it is possible to cover the majority of these areas.

52.6.2. QA Team Goal

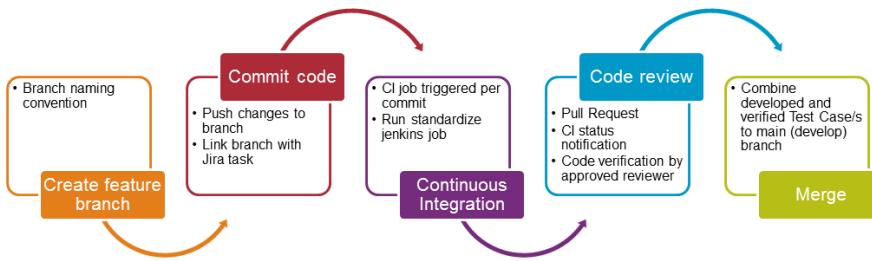
For QA engineers, it is essential to take care of the product code quality.

Therefore, we have to understand, that a **test case is also a code which has to be validated** against quality gates. As a result, we must **test our developed test case** alike it is done during standard Software Delivery Life Cycle.

52.6.3. Well rounded test case production process

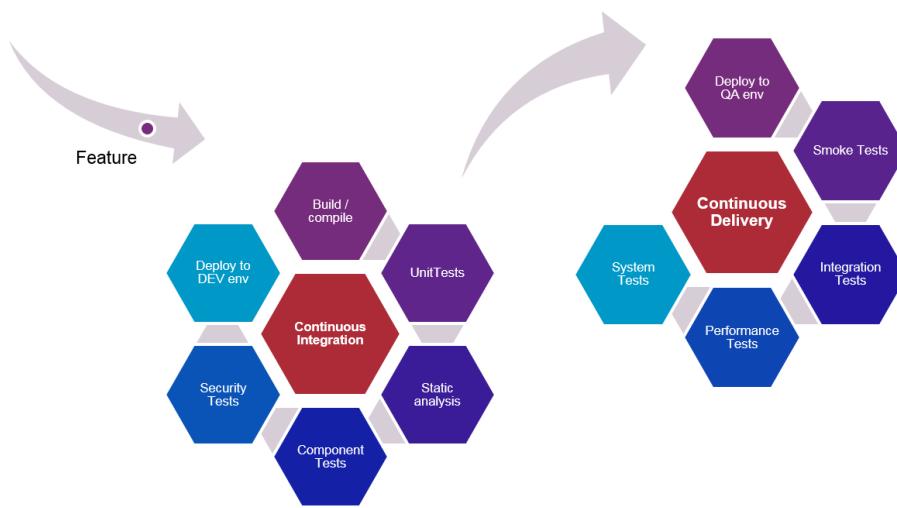
- How do we define top notch test cases development process in **E2E Mr Checker Test Framework**

Well defined Test Case develop process



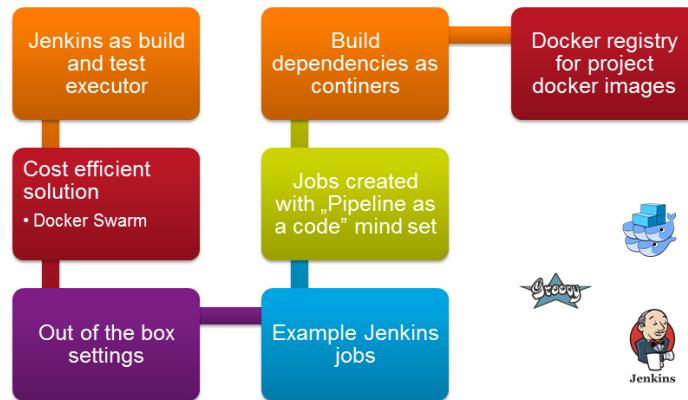
52.6.4. Continuous Integration (CI) and Continuous Delivery (CD)

- **Continuous Integration (CI)** - procedure where quality gates validate test case creation process
- **Continuous Delivery (CD)** - procedure where we include as smoke/regression/security created test cases, validated against CI



52.6.5. What should you receive from this DevOps module

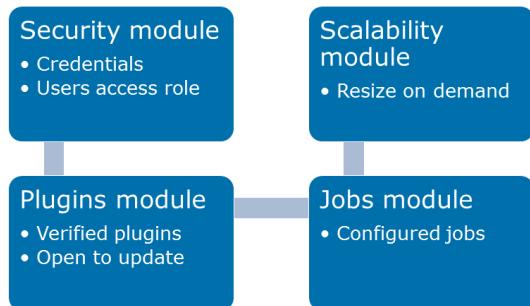
DevOps infrastructure ingredients



52.6.6. What will you gain with our DevOps module

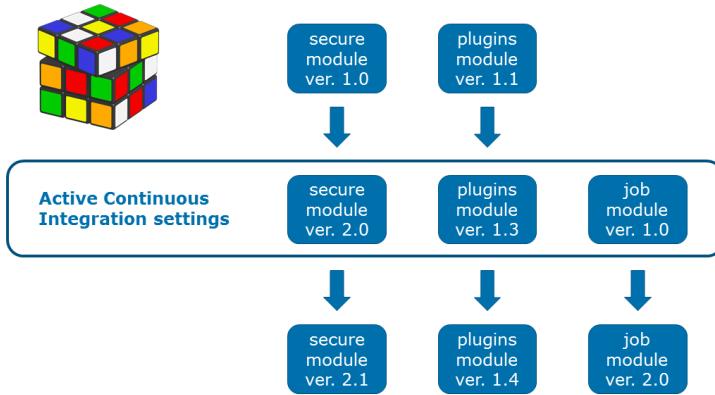
The CI procedure has been divided into transparent modules. This solution makes configuration and maintenance very easy because everyone is able to manage versions and customize the configuration independently for each module. A separate security module ensures the protection of your credentials and assigned access roles regardless of changes in other modules.

Superior Continuous Integration ingredients



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.

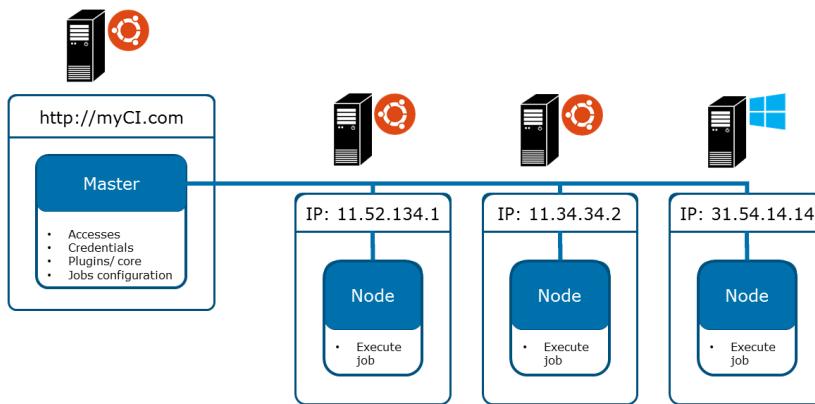
Setup your own proven CI modules



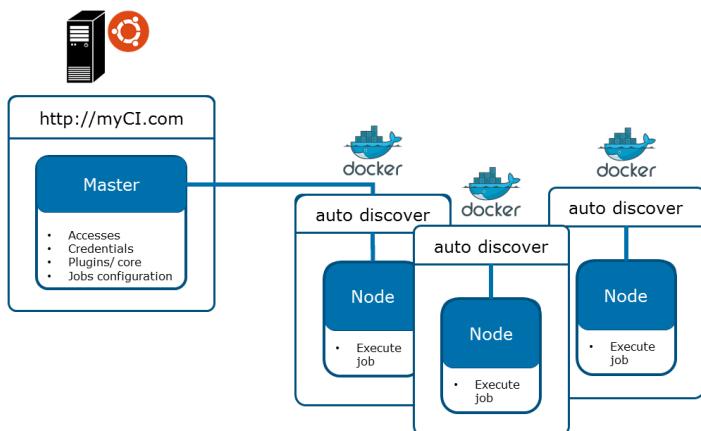
DevOps module supports a delivery model in which executors are made available to the user as needed. It has advantages such as:

- Saving computing resources
- Eliminating guessing on your infrastructure capacity needs
- Not spending time on running and maintaining additional executors

Classic executor architecture for Continuous Integration



Innovative executor architecture for Continuous Integration



Benefits



Modularity

Build your own CI



Secure

Restricted access to credentials



Autoscaling and service discovery

Take as you need

52.6.7. How to build this DevOps module

If you want to install the module, please click the link below. Installation should not take more than a few minutes

- [DevOps module installation](#)

Once you have implemented the module, you can learn more about:

- [Building jobs & Running builds](#)
- [Docker commands](#)

52.6.8. 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 configurations is required (but optional) using this custom Docker image. Additionally, this Jenkins setup allows to be dynamically scaled across multiple machines and even the cloud (AWS, Azure, Google Cloud etc.).

Jenkins Overview

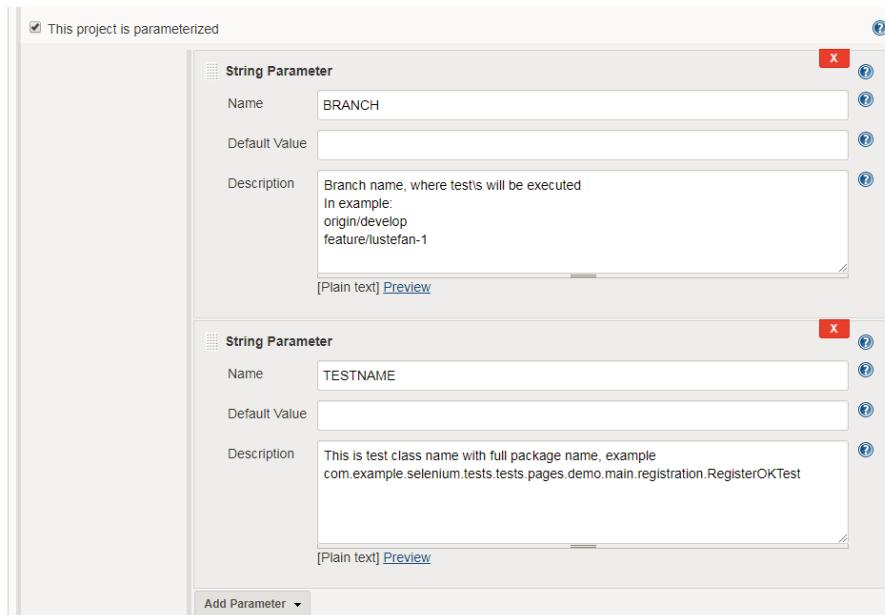
Jenkins is an Open Source Continuous Integration Tool. It allows the user to create automated build jobs which will run remotely on so called *Jenkins Slaves*. A build job can be triggered by several events, for example on new pull request on specified repositories or timed (e.g. at midnight).

Jenkins Configuration

Tests created by using the testing framework can be easily 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 *parametrized* 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.



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.

The screenshot shows the Jenkins 'Execute shell' configuration step. The 'Command' field contains the following script:

```
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
```

Below the command, there is a link to "See the list of available environment variables". A "Advanced..." button is also present.

- Next up, Maven will be used to check if the code can be compiled, without running any tests.

The screenshot shows the Jenkins 'Invoke top-level Maven targets' configuration step. The 'Maven Version' is set to v3.3.9 and the 'Goals' field contains the command: `clean install test-compile -DskipTests=true`. An 'Advanced...' button is at the bottom right.

After making sure that the code can be compiled, the actual tests will be executed.

+ image::images/jenkins-build-3.png["Starting the actual tests", width="450", link="images/jenkins-build-3.png"]

- Finally, reports will be generated.

The screenshot shows the Jenkins 'Invoke top-level Maven targets' configuration step. The 'Maven Version' is set to v3.3.9 and the 'Goals' field contains the command: `site`. An 'Advanced...' button is at the bottom right.

Post Build Configuration

- At first, the results will be imported to the [Allure System](#)

The screenshot shows the Jenkins 'Allure Report' configuration step. The 'Results:' section has a 'Path' field containing `target/allure-results`. Below it, there is a note: "Paths to Allure results directories relative from workspace. E.g. `target/allure-results`". There is also a 'Properties' section with an 'Add' button. An 'Advanced...' button is at the bottom right.

- JUnit test results will be reported as well. Using this step, the test result trend graph will be displayed on the Jenkins job overview.

Publish JUnit test result report

Test report XMLs

Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/*.xml'. Basedir of the fileset is the workspace root.

Retain long standard output/error

Health report amplification factor

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

Allow empty results Do not fail the build on empty test results

- Finally, an E-Mail will be sent to the previously extracted author of the commit.

E-mail Notification

Recipients

Whitespace-separated list of recipient addresses. May reference build parameters like \$PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

Send e-mail for every unstable build

Send separate e-mails to individuals who broke the build

Using the Pre-Configured Custom Docker Image

If you are starting a new Jenkins instance for your tests, we'd suggest to use the pre-configured Docker image. This image already contains all configurations and additional features.

The configurations that are made are e.g. Plugins and Pre-Installed job setup samples. This way, you don't have to set up the entire CI-Environment from ground up.

The additional features from this docker image allow the dynamic creation and deletion of Jenkins slaves, by creating Docker containers. Also, Cloud Solutions can be implemented to allow wide-spread load balancing.

52.6.9. Continuous Delivery

Include quality with Continuous Delivery during product release.

Overview

CD from Jenkins point of view does not change a lot from Continuous Integration one.

Jenkins Overview

For Jenkins CD setup please use the same Jenkins settings as for CI [link](#) The only difference is:

- What type of test you we will execute. Before we have been picking test case(s), however now we will choose test suite(s)
- Who will trigger given Smoke/Integration/Performance job
- What is the name of official branch. This branch ought to be used always in every CD execution. It will be either **master**, or **develop**.

Jenkins for Smoke Tests

In point where we input test name - \$TESTNAME ([link](#)), please input test suite which merge by tags

-([link](#)) how to all test cases need to run only smoke tests.

Jenkins for Performance Tests

Under construction - added when WebAPI module is included.

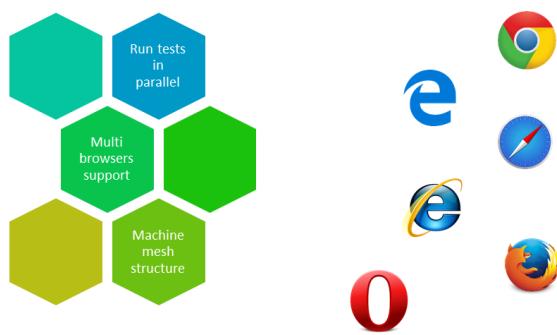
52.6.10. Selenium Grid

What is Selenium Grid

Selenium Grid allows running web/mobile browsers test cases to fulfil bedrock factors, such as:

- Independence 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
- Support multi-type programming languages (java, javascript, python, ...).

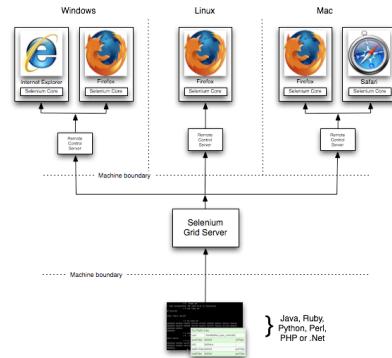
Selenium Grid – where we use it



On a daily basis, a test automation engineer uses his local environments for test case execution/development. However, created browser test case has to be able to run on any other infrastructure. Selenium Grid enables this portability for us.

Selenium Grid Structure

Selenium Grid - structure



Full documentation for Selenium Grid can be found here: [here](#) and [here](#).

'Vanilla flavour' Selenium Grid is based on two, not too 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	• 200 \$ /month (VMs)	• + 20 \$ /month (VM) + 4 hour SeleniumGrid specialist	• Works as centralized solution used across all test departments	• No replication. Manual actions needed to recover	• Manual infrastructure setup, for each browser (Chrome, Firefox, IE, Safari, Edge)
Cloud solution	• 10 \$ /month (VMs)	• + 10 \$ /month (VMs) + 0.25 hour junior team member	• Works as centralized per department or decentralized solution per team	• Auto recovery. Balance number of active instances/browsers through swarm of active VMs	• 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

[[selenium-grid.asciidoc_how-to-use-selenium-grid-with-e2e-mr-checker-test-frameworks]]

==

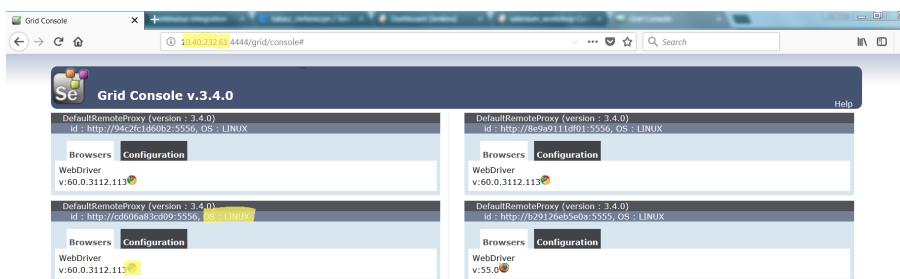
How to use Selenium Grid with E2E Mr Checker Test Frameworks

Run 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 taken during test case execution
- *-Dbrowser=chrome* - what type of browser will be used during test case execution



52.6.11. What is Docker

Docker - open source software platform to create, deploy and manage virtualized application containers on a common operating system (OS), with an ecosystem of allied tools.

52.6.12. 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 the Docker

52.6.13. Exploring basic Docker options

Let's expose 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 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 image by adding container ID to 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"
```

52.6.14. 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)
```

52.6.15. How to build DevOps module

Prerequisites

- 64-bit Linux operating server system (recommended: Ubuntu 16.04 LTS server - [Download](#))
- Non-root user with sudo privileges (the default user created during the operating system installation process)
- Docker - open source software platform to create, deploy and manage virtualized application containers on a common operating system (OS), with an ecosystem of allied tools.
- git - version control system software to clone module code

Docker service installation

```
$ curl -fsSL get.docker.com -o get-docker.sh  
$ sh get-docker.sh
```

Git installation

```
$ sudo apt-get install git
```

Creating special system users

```
$ sudo useradd -M jenkins  
$ sudo usermod -L jenkins  
$ sudo usermod -a -G docker jenkins
```

Cloning the repository

Check Your current directory, create a new for DevOps module in /home/<YourUserName>. Open it and enter git clone command

```
$ pwd  
/home/<YourUserName>/  
$ mkdir dev_ops_module  
$ cd dev_ops_module  
$ pwd  
/home/<YourUserName>/dev_ops_module  
$ git clone https://bitbucket.org/lukasz_stefaniszyn/jenkinsdockercompose.git
```

Enabling Docker remote API

Enable Docker remote API - By default, due to security reasons, Docker runs via a non-networked Unix socket. This solution allows only local communication. The Docker daemon in Ubuntu 16.04 is configured by the system, so You need to modify file `/lib/systemd/system/docker.service`. You'll enable the access to the Docker daemon from specific IP address.

Allow port communication:

```
$ sudo ufw allow 4243/tcp  
$ sudo ufw allow 32000:33000/tcp
```

Open file docker service and replace variable `ExecStart` value:

```
$ sudo nano /lib/systemd/system/docker.service
#
# Find variable ExecStart and change value to: /usr/bin/dockerd -H tcp://0.0.0.0:4243
-H unix:///var/run/docker.sock
#
$ ExecStart=/usr/bin/dockerd -H tcp://0.0.0.0:4243 -H unix:///var/run/docker.sock
```

Finish editing this file: Nano: **Ctrl + X** and **Y** Vi: **Esc + wq**

Restart deamon and docker service:

```
$ sudo systemctl daemon-reload  
$ sudo service docker restart
```

Please, test the configuration:

```
$ curl --noproxy GET http://127.0.0.1:4243/version
```

The answer should be similar to the one shown in the picture:

```
{"Platform": {"Name": ""}, "Components": [ {"Name": "Engine", "Version": "18.01.0-ce", "Details": {"ApiVersion": "1.35", "Arch": "amd64", "BuildTime": "2018-01-10T20:09:37.000000000+00:00", "Experimental": "false", "GitCommit": "03596f5", "GoVersion": "go1.9.2", "KernelVersion": "4.4.0-87-generic", "MinAPIVersion": "1.12", "Os": "linux"}, {"Name": "Version", "Version": "18.01.0-ce", "ApiVersion": "1.35", "MinAPIVersion": "1.12", "GitCommit": "03596f5", "GoVersion": "go1.9.2", "Os": "linux", "Arch": "amd64", "KernelVersion": "4.4.0-87-generic", "BuildTime": "2018-01-10T20:09:37.000000000+00:00"}]}
```

Jenkins with Docker

Execute commands:

```
$ sudo apt-get upgrade -y  
$ sudo apt-get install -y sudo libltdl-dev  
$ GID=$(cut -d: -f3 < <(getent group docker))
```

52.6.16. Running module

Open JenkinsDockerCompose directory and print the path by `pwd` command:

```
$ pwd  
/home/<YourUserName>/dev_ops_module  
$ cd jenkinsdockercompose  
$ pwd  
/home/<YourUserName>/dev_ops_module/jenkinsdockercompose/
```

Let's edit a configuration file `create_and_run.sh`. You can use default system text editors such as "nano" or "vi":

```
$ nano create_and_run.sh
```

Replace variable value in the second line with the path previously displayed:
`/home/<YourUserName>/dev_ops_module/jenkinsdockercompose/`

```
1 echo "Set global variables"  
2 REPO_HOME=/home/<DefaultUserName>/dev_ops_module/jenkinsdockercompose/
```

Finish editing this file: Nano: `Ctrl + X` and `Y` Vi: `Esc + wq` and run the script:

```
$ sudo ./create_and_run.sh
```

Wait until the end of the building process. It can take a few minutes. What happened there?

1. Setting global variables
2. Removing older docker images (if they exist)
3. Building jenkins_home_security image
4. Building jenkins_home_jobs image
5. Building jenkins_home_mng image
6. Start Jenkins

Run web browser with address <http://<server-ip-address>:8080>. If the configuration is correct, the Jenkins main page will be displayed.

The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Manage Jenkins', 'My Views', 'Open Blue Ocean', and 'Credentials'. Below the sidebar are sections for 'Build Queue' (empty) and 'Build Executor Status' (empty). The main area has a table with columns 'S', 'W', 'Name', 'Last Success', and 'Last Failure'. It lists two items: 'Examples' and 'Training', both marked with a yellow sun icon.

Configuring slaves

Login as jenkins admin and run web browser with address <http://<server-ip-address>:8080/configure>

Find API ip address option and change it to <http://<server-ip-address>:4243>

This screenshot shows the 'Docker' configuration page under the 'Cloud' section. It includes fields for 'Name' (set to 'jenkins_slave'), 'Docker URL' (set to 'tcp://192.168.1.220:4243'), 'Docker API Version' (set to 'none'), 'Connection Timeout' (set to '5'), and 'Read Timeout' (set to '15'). A 'Test Connection' button is at the bottom right.

[[How-to-build-jobs-&-run-builds.asciidoc]] = Adding a new item

The easiest way to create a new job is to use the "new item" option. You can create a whole new item or copy the configuration from an already existing job with changing only the parameters we are interested in.

This screenshot shows the Jenkins dashboard after creating a new job named 'Training'. The sidebar on the left is identical to the first screenshot. The main area shows a table with three items under the 'Training' folder: 'selenium_workshop', 'selenium_workshop_cucumber', and 'selenium_workshop_cucumberParallel'. All three items have a grey circle icon next to them and a yellow sun icon.

Click "new item" in the menu and next and on the next page enter an item name and choose the type.

Enter an item name

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

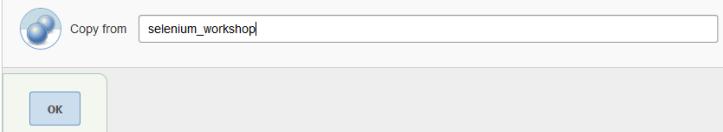
Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

GitHub Organization
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.

If you want, you can copy an existing configuration:

if you want to create a new item from other existing, you can use this option:



[[How-to-build-jobs-&-run-builds.asciidoc]] = Configuring job

Each job can be configured according to your preferences. This is done on the page available in the menu of the job page. Just click on "configure".

Load the main page again and next look at the middle of the page. Two job catalogs are displayed there.

All	+	S	W	Name ↓	Last Success	Last Failure
				Examples	N/A	N/A
				Training	N/A	N/A

Click "Training" directory name. You will be taken to a page with a list of available jobs. In the default configuration there are:

1. selenium_workshop
2. selenium_workshop_cucumber
3. selenium_workshop_cucumberParallel

All	+	S	W	Name ↓	Last Success	Last Failure
				selenium_workshop	N/A	N/A
				selenium_workshop_cucumber	N/A	N/A
				selenium_workshop_cucumberParallel	N/A	N/A

Click on the name - "selenium_workshop". You will be taken to the main page of the selected Job

- [Up](#)
- [Status](#)
- [Changes](#)
- [Workspace](#)
- [Build with Parameters](#)
- [Delete Project](#)
- [Configure](#)
- [Favorite](#)
- [Allure Report](#)
- [Move](#)
- [Open Blue Ocean](#)

Project selenium_workshop

Full project name: Training/selenium_workshop



[Allure Report](#)



[Workspace](#)



[Recent Changes](#)

Permalinks

Build History [trend](#) [RSS for all](#) [RSS for failures](#)

find

↑ ↓

From the menu on the left select "configure".

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Project name:

Description:

[Plain text] [Preview]

Discard old builds [?](#)

Strategy: Log Rotation [?](#)

Days to keep builds:

if not empty, build records are only kept up to this number of days

Max # of builds to keep:

if not empty, only up to this number of build records are kept

[Edit](#) [Advanced...](#)

The configuration categories are displayed there:

1. General
2. Source Code Management
3. Build Triggers
4. Build Environment
5. Build
6. Post-build Actions

[[How-to-build-jobs-&-run-builds.asciidoc]] = Running example build

Let create the first build. Choose selenium workshop main page again.

- Up
- Status
- Changes
- Workspace
- Build with Parameters
- Delete Project
- Configure
- Favorite
- Allure Report
- Move
- Open Blue Ocean

Project selenium_workshop

Full project name: Training/selenium_workshop



[Allure Report](#)



[Workspace](#)



[Recent Changes](#)

Permalinks

[Build History](#) [trend](#) [—](#)

X

[RSS for all](#) [RSS for failures](#)

Click the link "Build with parameters" in the menu on the left. On this page you can choose the configuration of the branch name and the name of the test that will run.

Project selenium_workshop

This build requires parameters:

BRANCH

Branch name, where test's will be executed
In example:
origin/develop
feature/lustefan-1

TESTNAME

This is test class name with full package name, example
com.example.selenium.tests.tests.demo.main.registration.RegisterOKTest

Build

Click the button and see that your build has been targeted.

[Build History](#) [trend](#) [—](#)

X

#1	(pending—Waiting for next available executor)		
----	---	--	--

After assigning to him the executor it will be executed.

Build Queue

No builds in the queue.

Build Executor Status

jenkins_slave-d91354ed9a14

1 [Training » selenium_workshop #1](#)

It is also possible to preview the build page:

Jenkins

Jenkins > Training > selenium_workshop > #1

Back to Project
 Status
 Changes
 Console Output
 Edit Build Information
 Delete Build
 Parameters
 Open Blue Ocean
 Allure Report
 Built on Docker

Build #1 (2018-01-28 18:45:58)

Failed to determine (log)

Build Artifacts
[allure-report.zip](#) 950.96 KB

Started by user Jenkins Admin

This run spent:

- 25 sec waiting in the queue;
- 45 sec building on an executor;
- 1 min 11 sec total from scheduled to completion.

Allure Report

Docker Build Data
Host: tcp://192.168.1.220:4243
Original Container Id: d015ebc54188f3ce93c69db991712b076762da7f91a96d7238a249a39ff006d8

[[How-to-build-jobs-&-run-builds.asciidoc]] = Editing the list of plugins

Please, open `/home/<YourUserName>/dev_ops_module/jenkinsdockercompose/jenkins_home_mng` directory. You should see file `plugins.txt` on the list. Please edit it. It is possible to add or remove plugins that interest you.

```
$ cd /home/<YourUserName>/dev_ops_module/jenkinsdockercompose/jenkins_home_mng
$ ls
$ nano plugins.txt
```

Finish editing this file: Nano: **Ctrl + X** and **Y** Vi: **Esc + wq**. After this operation, the application must be restarted: * [DevOps module installation](#)

The second way is to edit the plugins option in the GUI.

[[How-to-build-jobs-&run-builds.asciidoc]] = User management Please, open /home/<YourUserName>/dev_ops_module/jenkinsdockercompose/jenkins_home_security/jenkins_home/users directory. Print content of it by `ls` command. You should see list of jenkins users.

```
$ cd  
/home/<YourUserName>/dev_ops_module/jenkinsdockercompose/jenkins_home_security/jenkins_home/users  
$ ls
```

Adding new user

The easiest way to create a new user is to copy his configuration file from one of the existing users and substitute the required options.

```
# Please create directory for new user and copy files  
#  
$ mkdir newUserName  
$ cd newUserName  
$ cp ../user1/config.xml .  
$ ls  
# You should see Your config. Please edit chosen option if You want  
$ nano config.xml
```

Finish editing this file: Nano: `Ctrl + X` and `Y`

Removing user

Open default directory of users again: /home/<YourUserName>/dev_ops_module/jenkinsdockercompose/jenkins_home_security/jenkins_home/users. Choose user name to remove and execute it by Unix `rm` command:

```
$ cd  
/home/<YourUserName>/dev_ops_module/jenkinsdockercompose/jenkins_home_security/jenkins_home/users  
$ ls  
# execute command : rm -rf <userName>  
# ! Take special care, executing the following command in the wrong directory can  
cause undesirable effects for other  
# applications or the entire operating system  
$ rm -rf user1
```

Part X: MyThaiStar

53. 1.My Thai Star – Agile Framework

53.1. 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.

53.2. 1.2 Scrum events

53.2.1. 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.

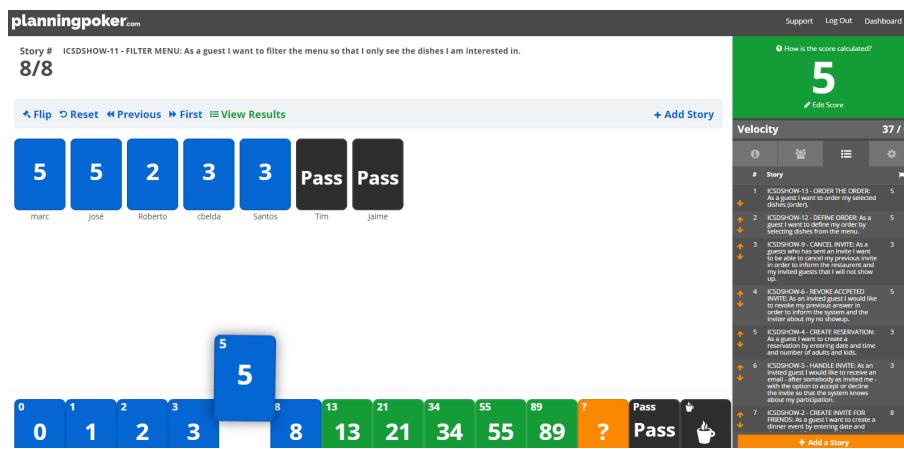


Figure 78. Screenshot of planningpoker.com during Sprint 1 Planning

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.

53.2.2. 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.

53.2.3. 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.

53.2.4. 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.

53.2.5. Backlog refinement

The team decided that the backlog refinement meeting is part of the Sprint Planning meeting.

53.3. 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.



Figure 79. 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 several items, including 'ICSDSHOW-38', 'ICSDSHOW-23', 'ICSDSHOW-24', 'ICSDSHOW-25', 'ICSDSHOW-39', 'ICSDSHOW-73', 'ICSDSHOW-40', 'ICSDSHOW-27', and 'ICSDSHOW-38'. The Done column contains no visible items.

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
		ICSDSHOW-27 Angular view components development
		ICSDSHOW-38 Angular services development

Figure 80. Screenshots of Subtasks during Sprint 2

54. 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.

54.1. 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.

54.2. 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.

54.3. 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.

54.4. 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.

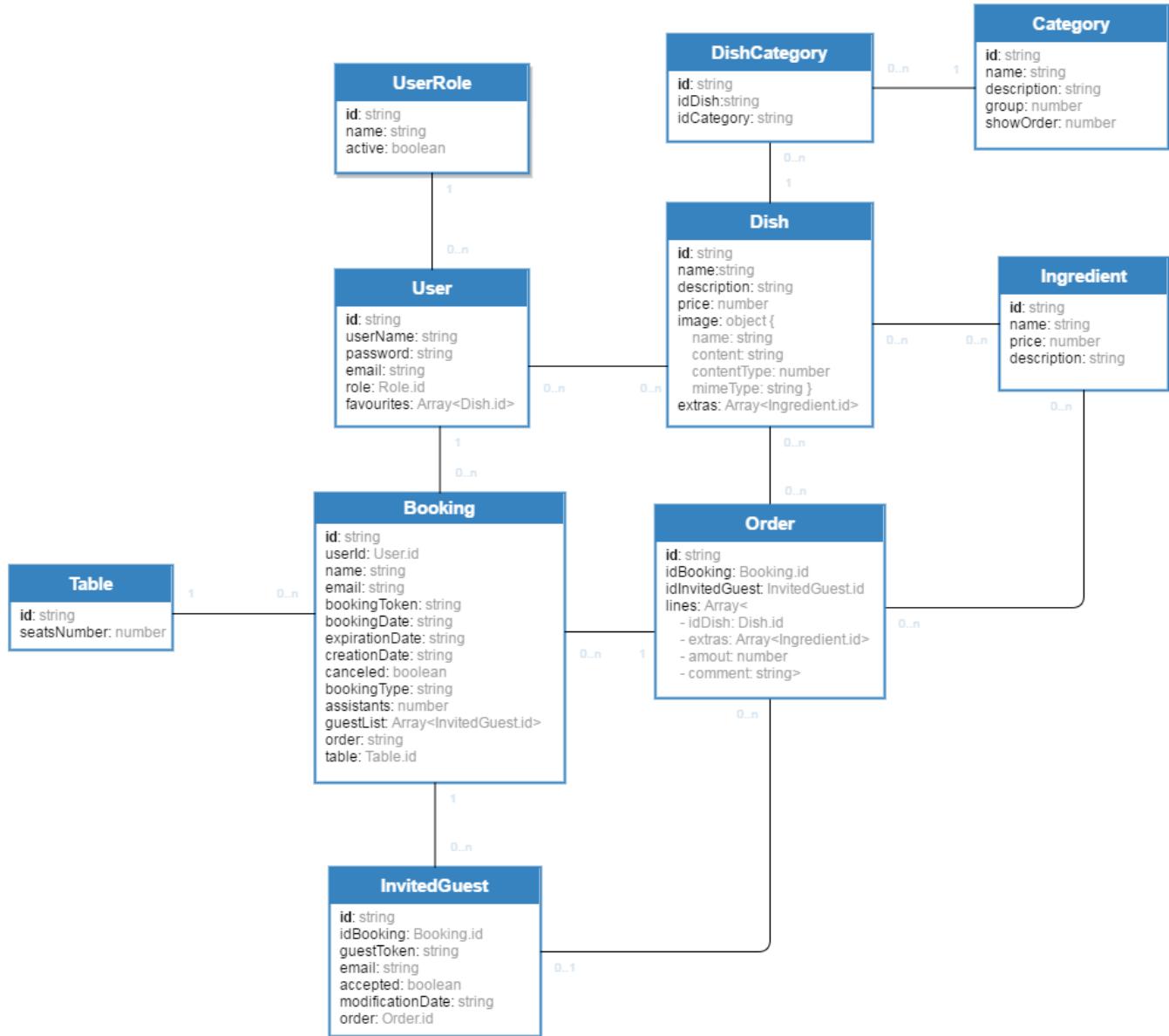
Unresolved directive in my-thai-star.wiki/master-my-thai-star.asciidoc - include::user-stories.asciidoc[leveloffset=1]

55. Technical design

55.1. Data Model

Unresolved directive in my-thai-star.wiki/master-my-thai-star.asciidoc - include::my-thai-star-data-model.asciidoc[leveloffset=3]

55.1.1. NoSQL Data Model



55.2. Server Side

55.2.1. 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:

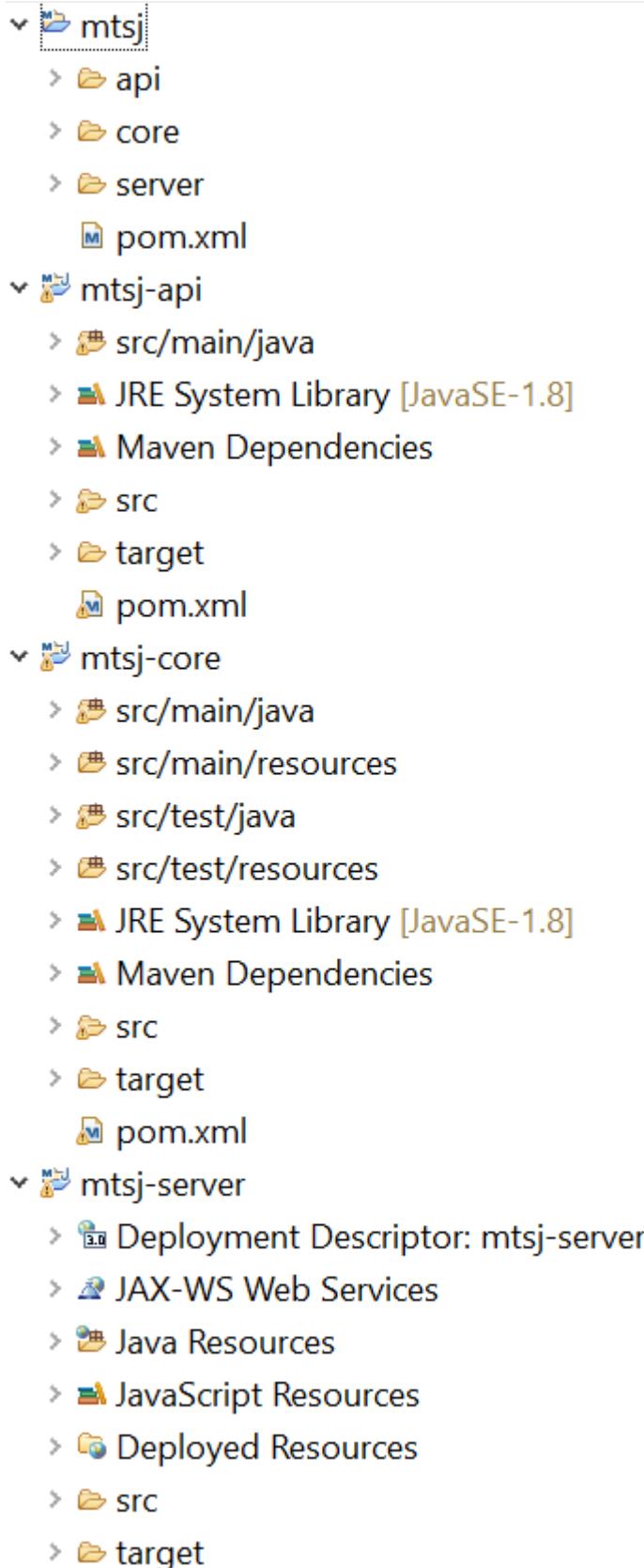
- [DEVON4J](#)
- [Devonfw](#)
- [Cobigen](#)

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

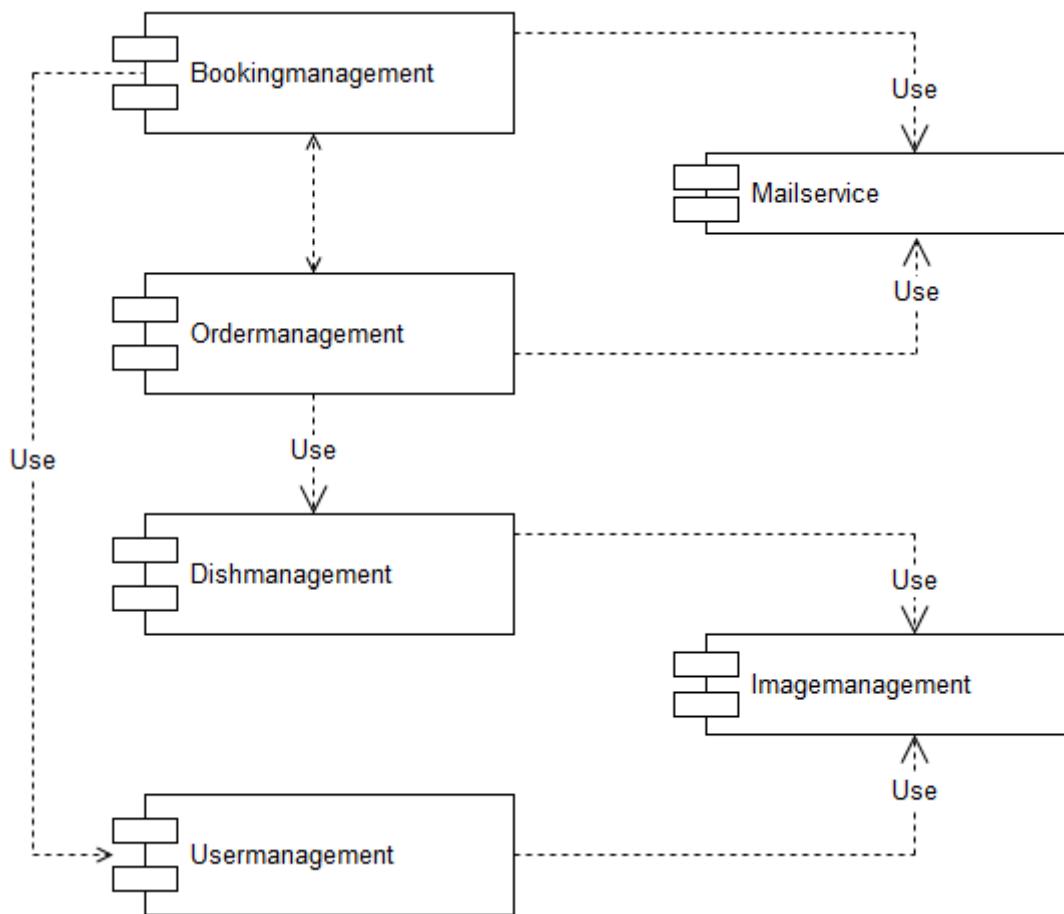


- *api*: Stores all the REST interfaces and corresponding Request/Response objects.
- *core*: Stores all the logic and functionality of the application.
- *server*: Configures the packaging of the application.

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*):

- *Imagemanagement*: 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 *Imagemanagement* 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.

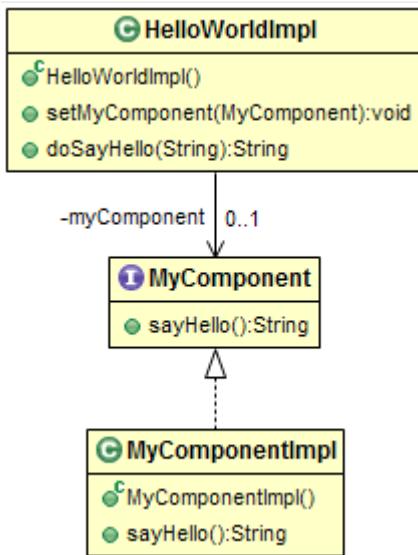
Component structure

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.

```

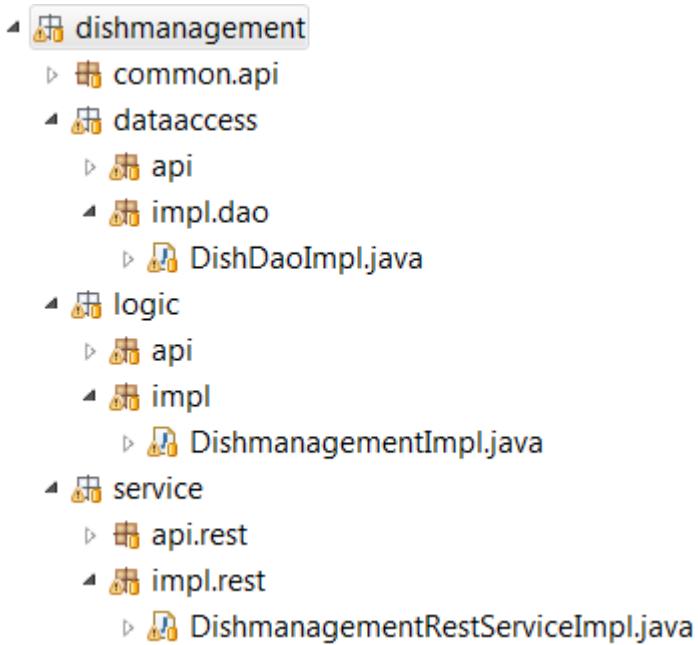
@Named
public class MyBeanImpl implements MyBean {
    @Inject
    private MyOtherBean myOtherBean;

    @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);
    }

    ...
}
```



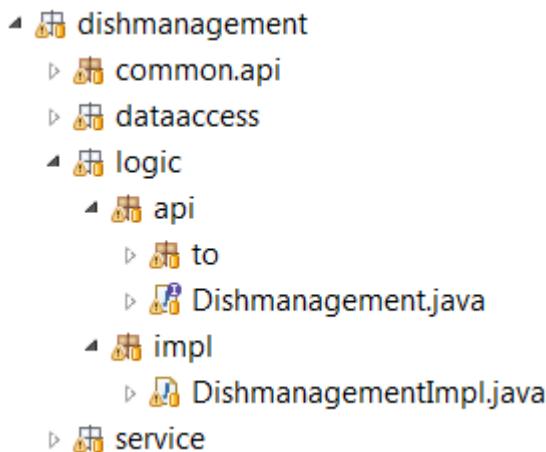
You can see the Devon4j conventions for REST services [here](#). And the My Thai Star services definition [here](#) as part of the [My Thai Star project](#).

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](#).

SAP Hana

Download/Install Vmware/SAP hana

- Download VMware Workstation Player to run SAP Hana database <https://www.vmware.com/in/products/workstation-player/workstation-player-evaluation.html>
- Install VMware and open. Using VMware browse check if you can find hxe.ova file. This file is not visible through windows explorer. This is the SAP Hana Express Edition image to be run inside VMware. If available open it in VMware. Otherwise, download Download Manager for SAP Hana from the link below. It may ask to register which is a simple process. <https://www.sap.com/cmp/ft/crm-xu16-dat-hddedft/typ.html>
- Run the download manager and on the screen use the defaults and click download.

Run SAP Hana Database Server

- Once the .ova file has been opened inside VMware workstation. Click on the image and go to Edit Virtual Machine Settings. Set the memory allocation to 5GB. And Network Connection to NAT . NAT shows the IP for the virtual machine which will be used to establish JDBC connection
- Click Play Virtual Machine. When first time the virtual machine runs it display following. Copy the IP address which will be used for JDBC connection
- Type hxeadm, which is the username and hit Enter. Next it will ask for password which is HXEHana1. Once successfully logged in it will ask to set a new password. Choose a new password and remember.

- You need to set Master password for SAP Hana database. Set it as you like and remember.
- For “proceed with configuration” type y and hit Enter. Hana database has started in the background.
- Try connecting with following command, replace the password with the master password

```
hxehost:hxeadm>hdbsql
\c -d SYSTEMDB -n localhost:39013 -u SYSTEM -p <>
```

Setting up Database for MTSJ

Once you have install SAP hana with VMware , you need to setup the DB.

Connect to DB

- After you start Vmware, login with hxeadm as login and the password. At the prompt - hxehost:hxeadm>hdbsql Please note the IP address, that need to be put in mtsj java backend
- On prompt hdbsql> type below to connect to the DB

```
\c -d SYSTEMDB -n localhost:39013 -u SYSTEM -p <password>
```

- Type below query to see, if you have access to tenant database i.e. HXE

```
SELECT DATABASE_NAME, ACTIVE_STATUS FROM SYS.M_DATABASES ORDER BY 1;
```

Enabling the script server

Run the below for enabling the script server

```
ALTER DATABASE HXE ADD 'scriptserver'
```

To check if the script server is enable, execute below statement

```
SELECT SERVICE_NAME, PORT, ACTIVE_STATUS FROM SYS.M_SERVICES ORDER BY 1;
```

It should see the scriptserver in it.

Creating a User on HXE

- Connect using the below

```
\c -d hxe -n localhost:39013 -u system -p <password>
```

- To create a user

```
Create user hanauser1 password <password> no force_first_password_change
```

- Grant below permission to the user

```
GRANT AFLPM_CREATOR_ERASER_EXECUTE TO hanauser1
GRANT AFL__SYS_AFL_AFLPAL_EXECUTE TO hanauser1 – here we have 2 underscore
grant AFL__SYS_AFL_AFLPAL_EXECUTE_WITH_GRANT_OPTION to hanauser1
grant AFLPM_CREATOR_ERASER_EXECUTE to hanauser
GRANT DATA ADMIN TO hanauser1
GRANT IMPORT TO hanauser1

GRANT EXECUTE on _SYS_REPO.GRANT_ACTIVATED_ROLE TO hanauser1
GRANT EXECUTE ON system.afl_wrapper_generator to hanauser1

GRANT EXECUTE ON system.afl_wrapper_eraser to hanauser1
GRANT MODELING TO hanauser1
```

- Now connect to HXE tenant using below

```
\c -d hxe -n localhost:39013 -u hanauser1 -p <password>
```

Setting up MTSJ Java backend

- Update application.properties file

```
# update the below
spring.flyway.locations=classpath:db/migration,classpath:db/specific/hana
# Add the below
spring.jpa.database=default
spring.jpa.database-platform=org.hibernate.dialect.HANAColumnStoreDialect
spring.datasource.driver-class-name=com.sap.db.jdbc.Driver
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true

#Comment the below
#spring.profiles.active=h2mem

spring.profiles.active=hana
```

- Update config/application.properties file

```
# update the below
spring.flyway.locations=classpath:db/migration,classpath:db/specific/hana
spring.datasource.url=jdbc:sap://ip:port/?databaseName=hxe
spring.datasource.username=username
spring.datasource.password=password
```

Enabling prediction usecase in MTSJ

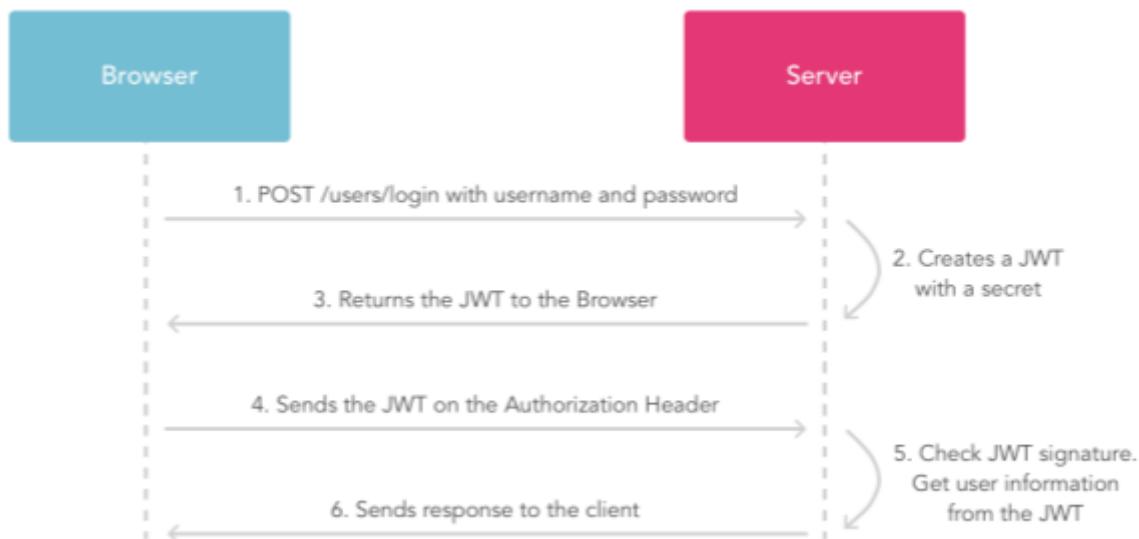
- Please refer link <https://github.com/devonfw/my-thai-star/wiki/angular-design> to enable prediction in gui
- Setting up data for Predictive use case, please refer to <https://github.com/SAP/hana-my-thai-star-data-generator>

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);
}
```

55.2.2. .NET design

TODO

55.2.3. Node.js design (deprecated)

Introduction

The Node.js backend for My Thai Star application is going to be based on:

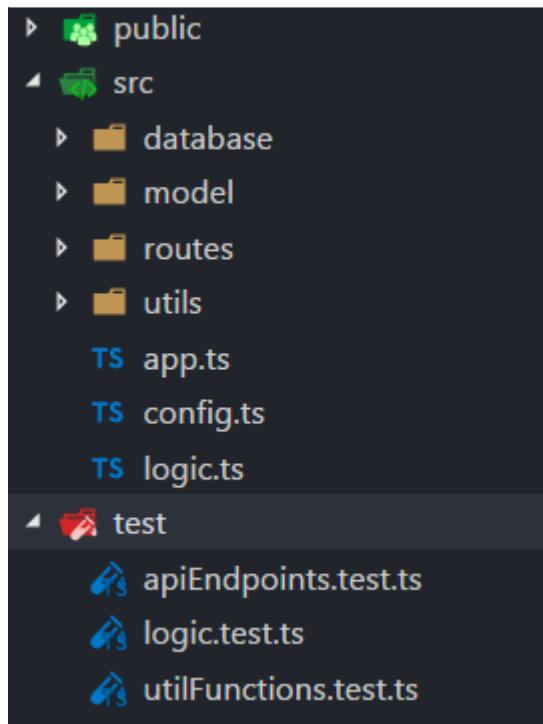
- **Express.js** as the web application framework
- **OASP4Fn** as data access layer framework
- **DynamoDB** as NoSQL Database

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/bookingmanagement/v1/booking/search`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order/search`.
- (POST) `/mythaistar/services/rest/ordermanagement/v1/order/filter`.

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'));
```

55.2.4. Serverless design (deprecated)

Introduction

The Node.js backend for My Thai Star application is going to be based on:

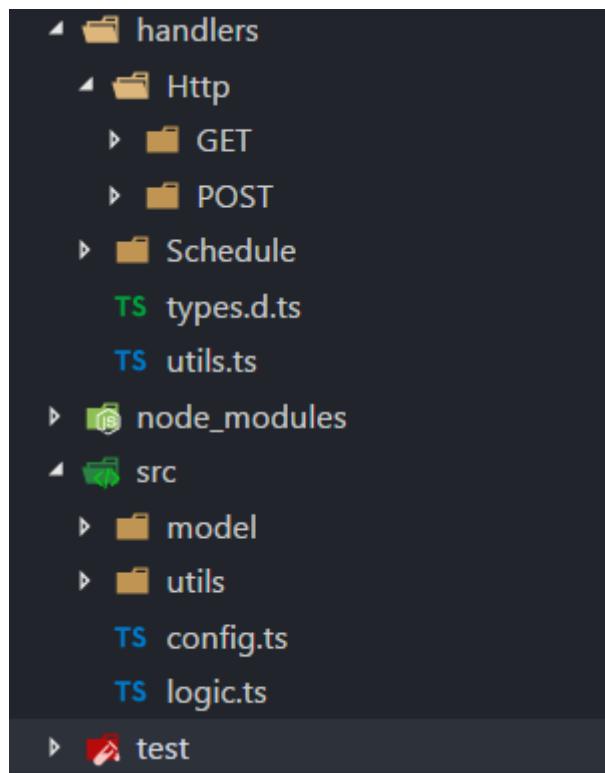
- **Serverless** as serverless framework
- **OASP4Fn** as data access layer framework
- **DynamoDB** as NoSQL Database

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* for 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}`));
    }
});
}

```

55.2.5. GraphQL design

TODO

55.3. Client Side

55.3.1. 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.

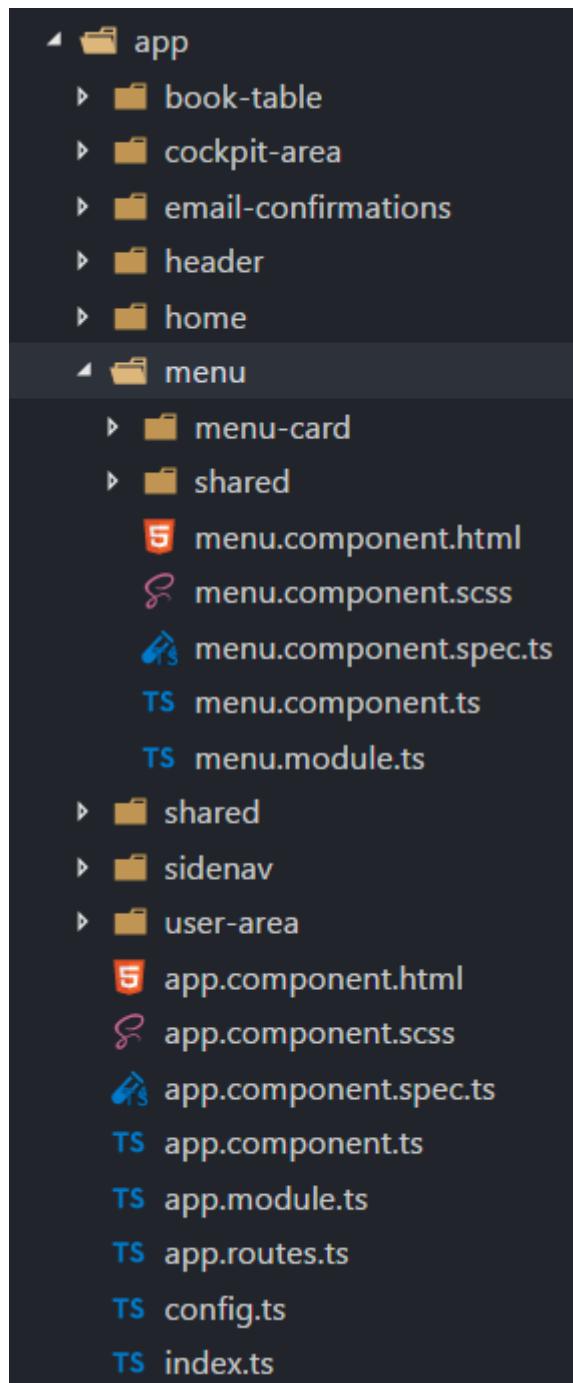
Basic project structure

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

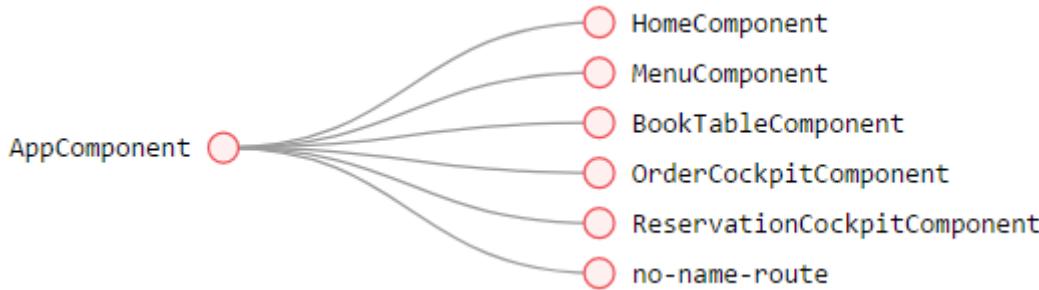
- components
 - sub-components
 - shared
 - component files
- main app component
- assets folder
- environments folder
- rest of angular files

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 cancelation 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.

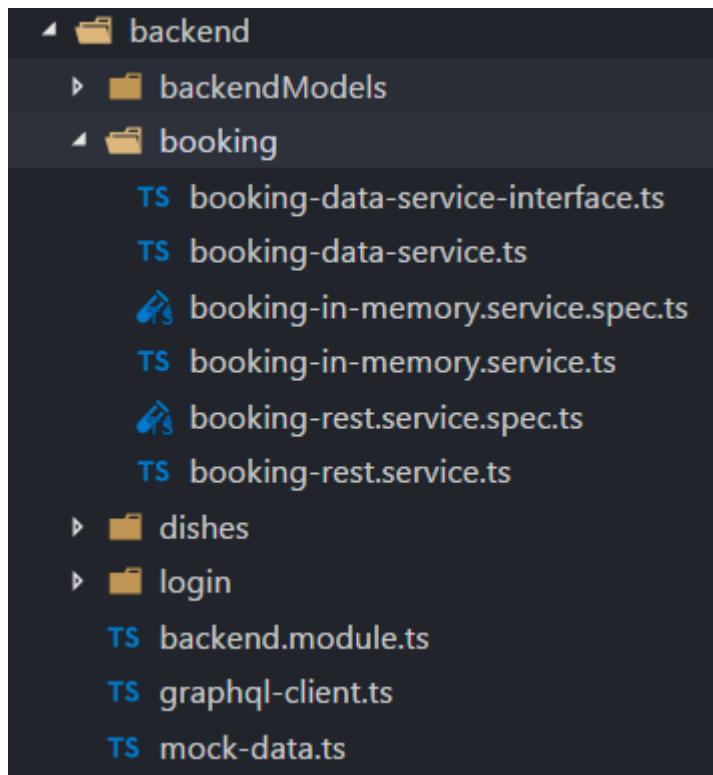
Equal-validator-directive

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.

SAP Hana

Setting up MTSJ angular

update the following property in config file in my-thai-star\angular\src\app\core\config

```
enablePrediction: true,
```

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

Jason Web Token consist in 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 it 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 services 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.

55.3.2. Xamarin design

TODO

56. Security

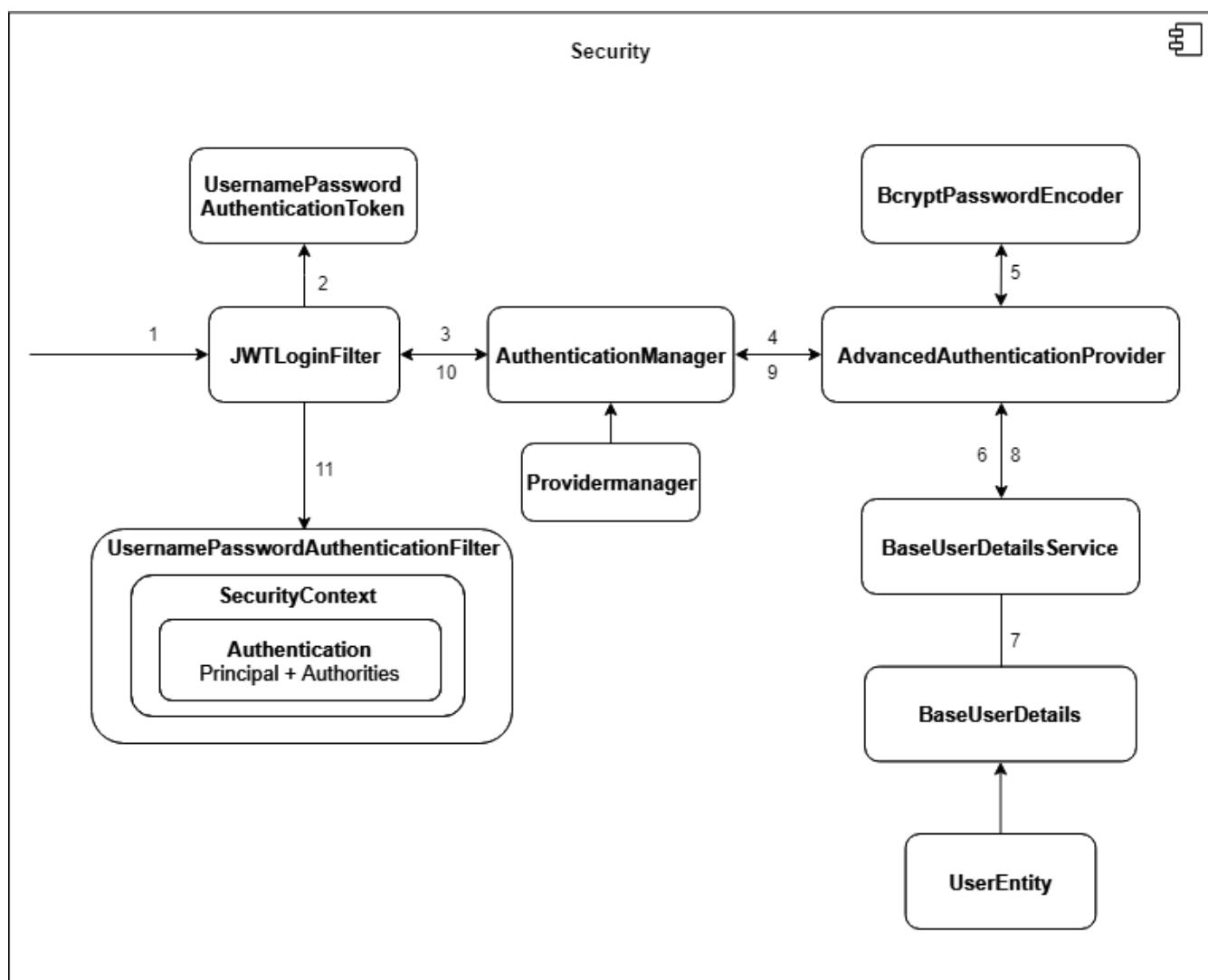
56.1. Two-Factor Authentication

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.

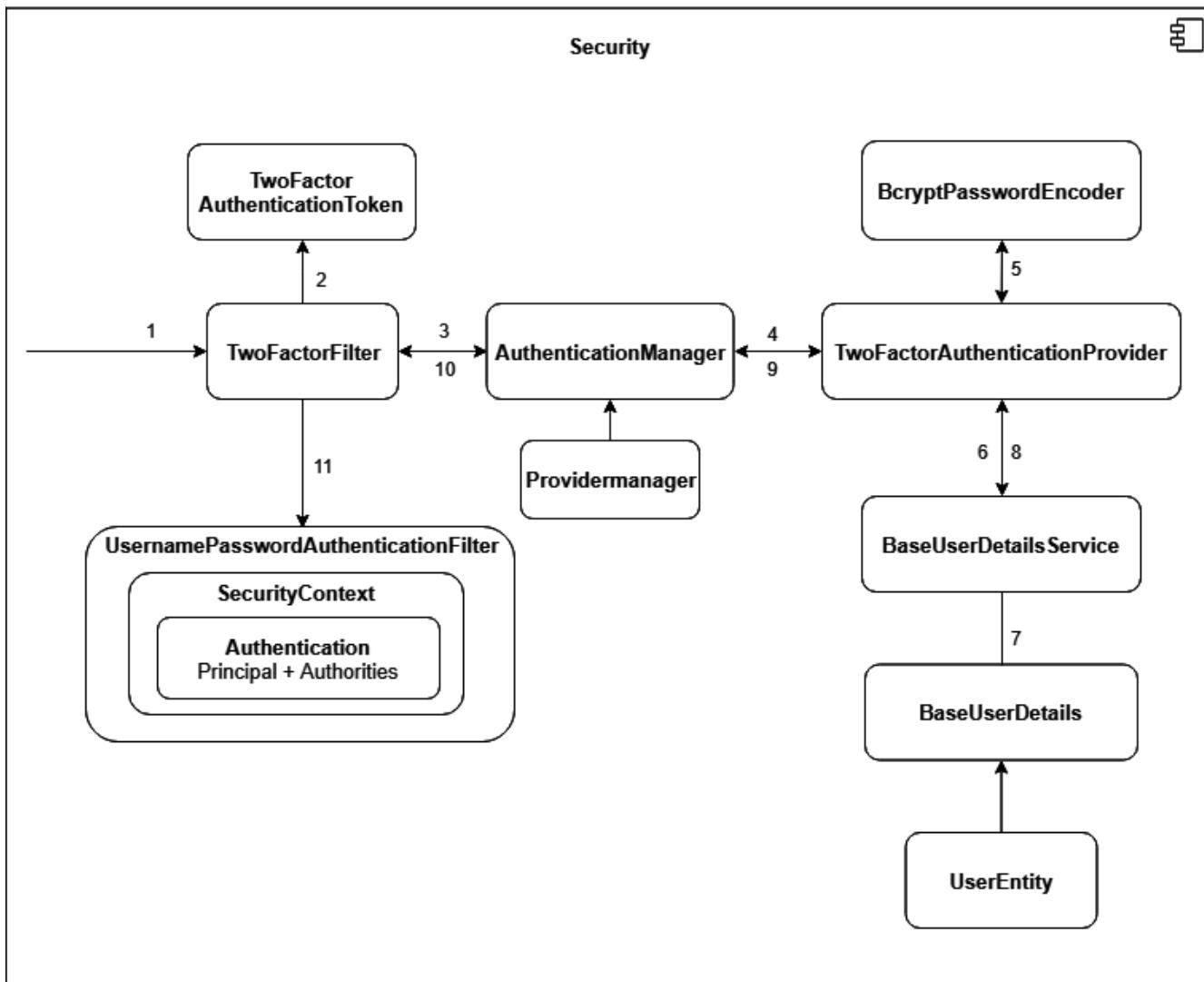
By enabling 2FA, to log into your account an additional one-time password is required what requires access to your paired device. This massively increases the barrier for an attacker to break into your account.

Backend mechanism

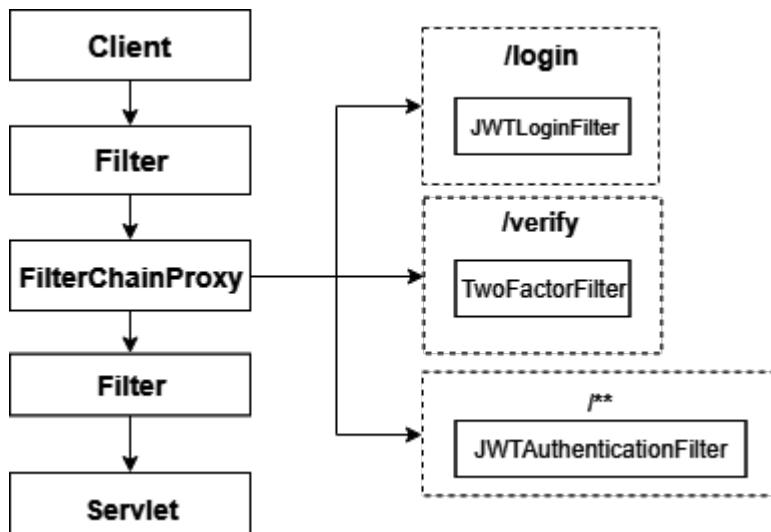
In the backend, we utilize Spring Security for any authentication. Following the arrows, one can see all processes regarding authentication. The main idea is to check all credentials depending on their 2FA status and then either grant access to the specific user or deny access. This picture illustrates 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 by 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 switch. Below the navigation, a table lists several 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 instructions: 'Please scan the QR code with the Google Authenticator or use the secret key'. There are two toggle switches: one for 'Two Factor Authentication Status' (which is turned on) and another for 'QR Code'. A 'Close' button is at the bottom right of the modal. The footer of the page includes the text 'MY THAI STAR 2019 - docker-version' and 'Devonfw'.

★ My Thai Star

ORDERS RESERVATIONS waiter

Filter

ORDERS

Reservation Date	Email	Reference Number
Jun 30, 2019 3:12 PM	user0@mail.com	CB_20170509_123502555Z
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 logo with a stylized crown or star shape containing the text "MY THAI STAR" in large letters, with the tagline "More than just delicious food" below it. At the top left, there is a star icon and the text "My Thai Star". At the top right, there are links for "HOME", "MENU", "BOOK TABLE", and user/account icons. A white modal box is overlaid on the page, containing the text "One Time Password *", a text input field with the value "123456", and two buttons labeled "CANCEL" and "APPLY". Below the modal, there are two smaller images: one showing the restaurant's interior with hanging lamps and shelves, and another showing a plate of food.

57. Testing

57.1. Server Side

57.1.1. 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() {

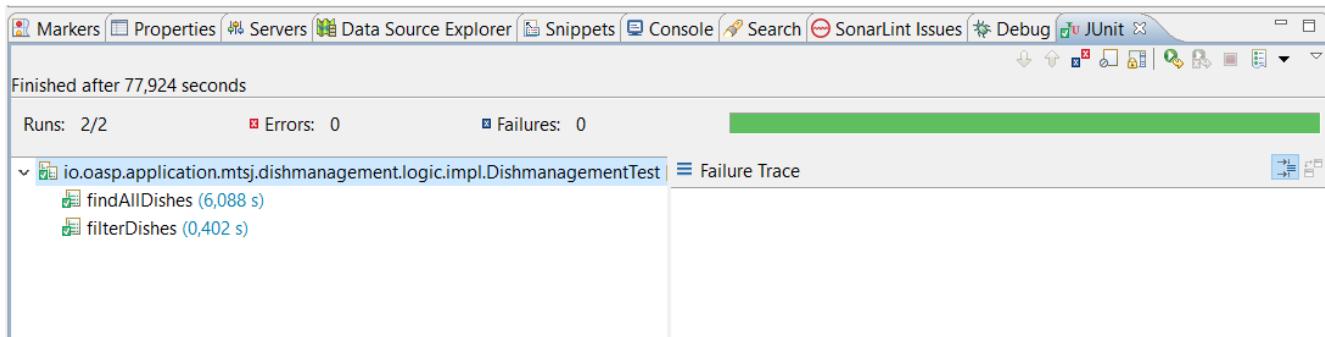
        PaginatedListTo<DishCto> result = this.dishmanagement.findDishes();
        assertThat(result).isNotNull();
    }

    ...
}
```

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] -----
```

57.1.2. .NET testing

TODO

57.1.3. Node.js testing

TODO

57.1.4. GraphQL testing

TODO

57.2. Client Side

57.2.1. Angular testing

The screenshot shows a green header bar with the text "Karma v1.4.1 - connected" and a "DEBUG" button. Below it, a white status bar says "Chrome 58.0.3029 (Windows 7 0.0.0) is idle". The main area has a light gray background with a green footer bar at the bottom. The footer bar contains the text "Jasmine 2.5.2", "finished in 20.533s", "45 specs, 0 failures", and a "raise exceptions" button.

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

57.2.2. Xamarin testing

TODO

57.3. End to end

57.3.1. 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](#).

58. UI design

58.1. Style guide

MY THAI STAR STYLE GUIDE

Colors

Main color	Secondary colors
BR: #46362C	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

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

Links

Normal	M1	This text is a link
Hover	S2	<u>This text is a link</u>
Active / Pressed	S1	<u>This text is a link</u>

Buttons

Main buttons

Normal
NORMAL

Hover
HOVER

Active / Pressed
ACTIVE / PRESSED

Normal
Background color: GR1
Text color: H4 Regular
Text size: WH

Hover
Background color: GR2

Active/Pressed
Background color: GR1

Buttons without background color

NORMAL
NORMAL

HOVER
HOVER

ACTIVE / PRESSED
ACTIVE / PRESSED

Normal
Background color: Transparent
Text size: H4 Regular
Text color: GR1

Hover
Background color: GR3

Active/Pressed
Background color: Transparent

Secondary buttons

NORMAL
NORMAL

HOVER
HOVER

ACTIVE / PRESSED
ACTIVE / PRESSED

Normal
Background color: G3
Text size: H4 Regular
Text color: WH

Hover
Background color: G2

Active/Pressed
Background color: G3

Secondary buttons without background color

NORMAL
NORMAL

HOVER
HOVER

ACTIVE / PRESSED
ACTIVE / PRESSED

Normal
Background color: Transparent
Text size: H4 Regular
Text color: G2

Hover
Background color: G5

Active/Pressed
Background color: Transparent

Form components

Field empty

Label

Field content

Label	Content	Field border
Field	Text color: G3 Text size: H5 Regular	Border width: 2px Border color: G4

Field focused

Label	Content	Field border
Field	Text color: GR1 Text size: H5 Regular	Border width: 2px Border color: G4

Checkbox unselected

Label	Box	
Unselected	Text color: G1 Text size: H4 Regular	Border width: 1px Border color: G4 Background color: WH

Checkbox selected

Label	Box	Icon
Selected	Text color: G1 Text size: H4 Regular	Border width: 1px Border color: GR1 Background color: GR1

Slider

Slider	Label	Border	Circle
Slider	Text color: G3 Text size: H5 Regular	Border color: G4 Border color active: GR1	Background color: WH Border color: GR1

Google Material Design Icons

Icon
Font size icons: 24px Font size big icons: 36px

58.2. 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](#)

59. CI/CD

59.1. 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

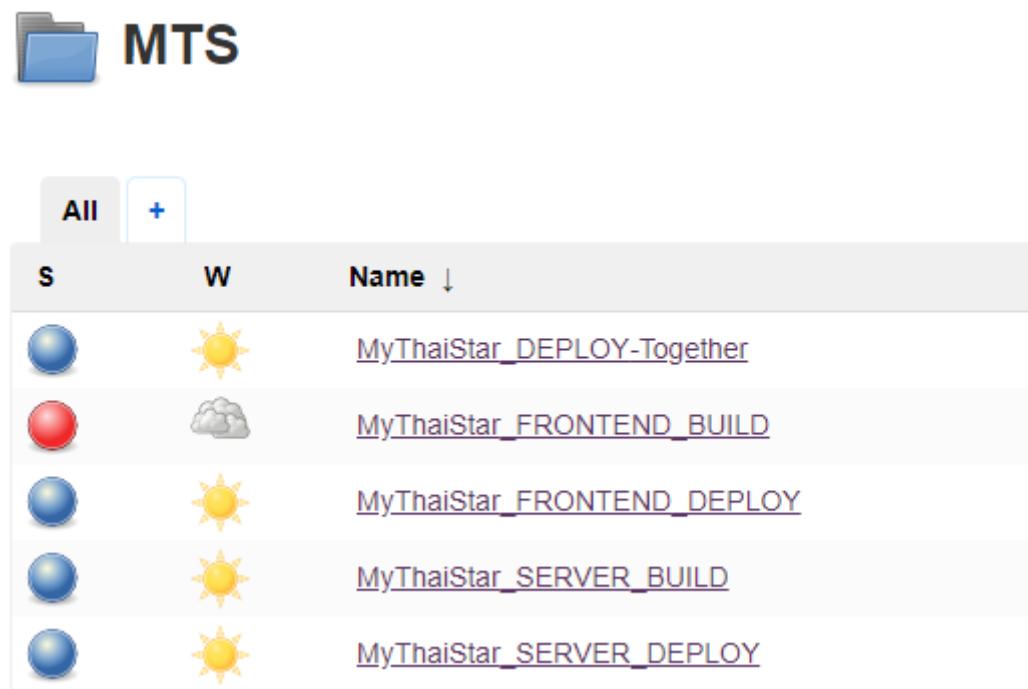
Nexus manages software "artifacts" required for development. It is possible to both download dependencies from Nexus and publish artifacts as well. It allows to share resources within an organization.

3. SonarQube

It is a platform for continuous inspection of the code. It is going to be used for the Java back-end.

59.1.3. Where can I find all My Thai Star Pipelines?

They are located under the **MTS** folder of the PL instance:



The screenshot shows a Jenkins pipeline list titled "MTS". There are two tabs at the top: "All" (selected) and "+". The table has three columns: "S" (Status), "W" (Workflow), and "Name". The "Name" column is sorted by name. The pipelines listed are:

S	W	Name
		MyThaiStar_DEPLOY-Together
		MyThaiStar_FRONTEND_BUILD
		MyThaiStar_FRONTEND_DEPLOY
		MyThaiStar_SERVER_BUILD
		MyThaiStar_SERVER_DEPLOY

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](#).

59.1.4. CI in My Thai Star stack

- [Angular CI](#)
- [Java CI](#)

59.1.5. How to configure everything out of the box

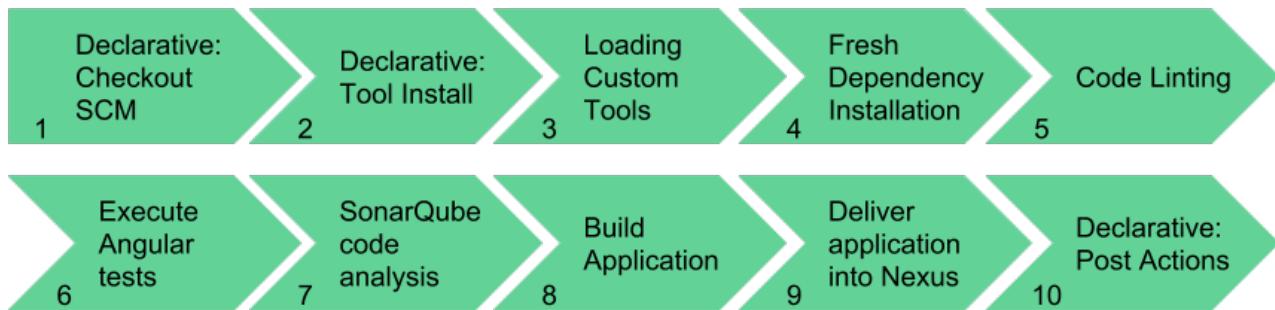
Production Line currently has a template to integrate My Thai Star. All information can be found at [devon production line repository](#)

59.1.6. 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

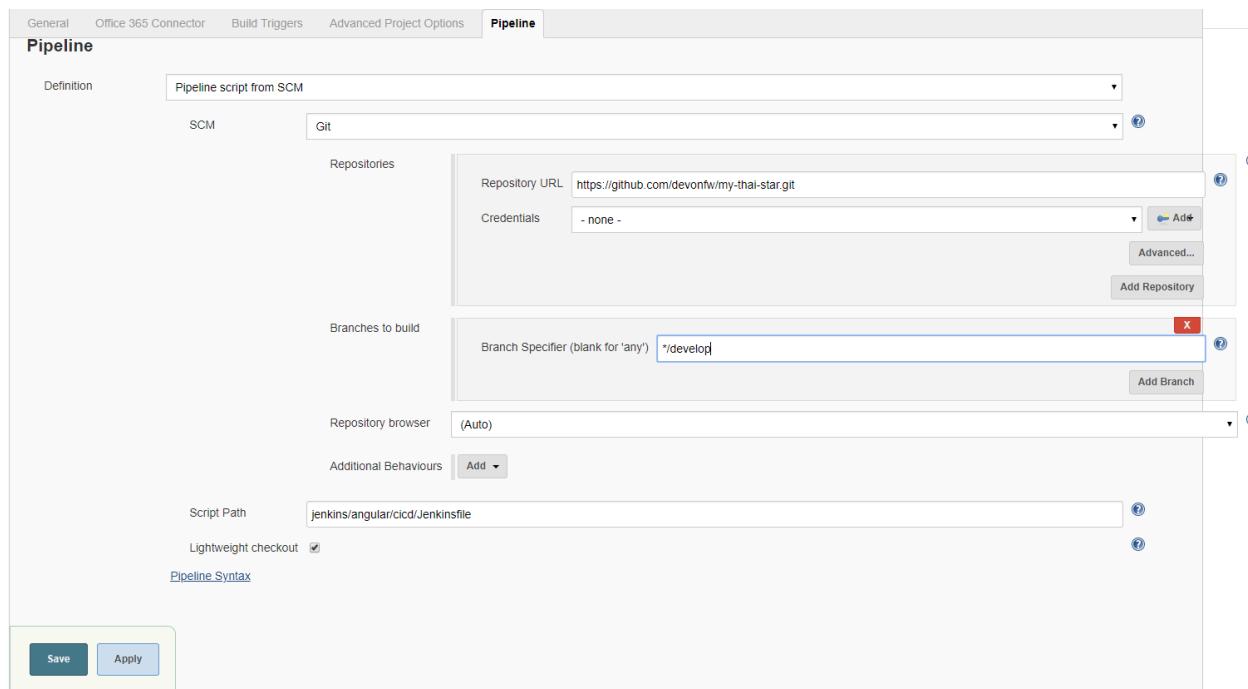
The Pipeline for the Angular client-side is going to be called **MyThaiStar_FRONTEND_BUILD**. It is located in the PL instance, under the [MTS folder](#) (as previously explained). It is going to follow a process flow like this one:



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** (10.14.0 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 **Addaptation** section of document). This script just executes the following commands:

```
ng test --browsers ChromeHeadless --watch=false
```

7. 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.

8. 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
```

9. Deliver application into Nexus

Once the scripts produce the Angular artifact (`/dist` folder), it's time to package it and store into nexus.

10. 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

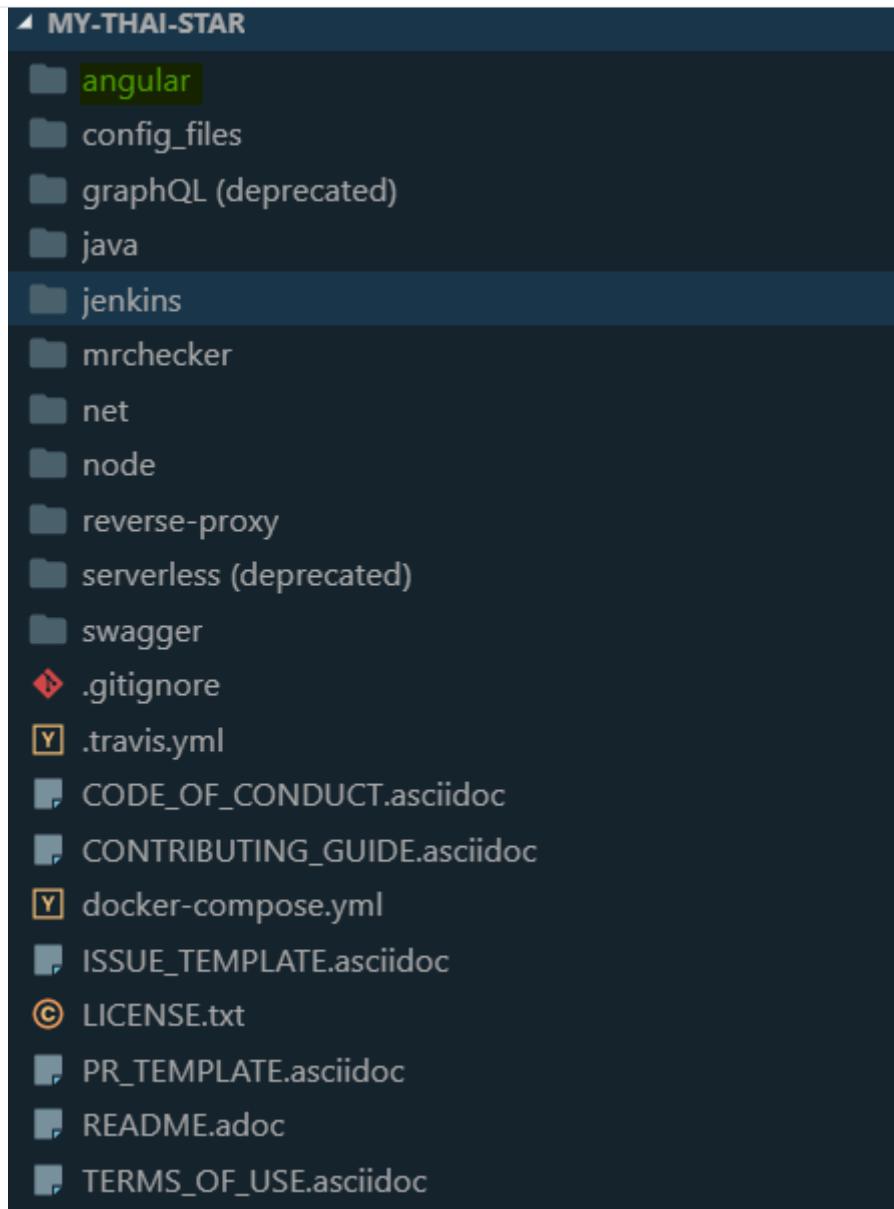
Example (package.json):

```
{  
  "name": "mythaistar-restaurant",  
  ...  
  "scripts": {  
    ...  
    "test:ci": "npm run postinstall:web && ng test --browsers ChromeHeadless  
--watch=false",  
    ...  
  }  
  ...  
}
```

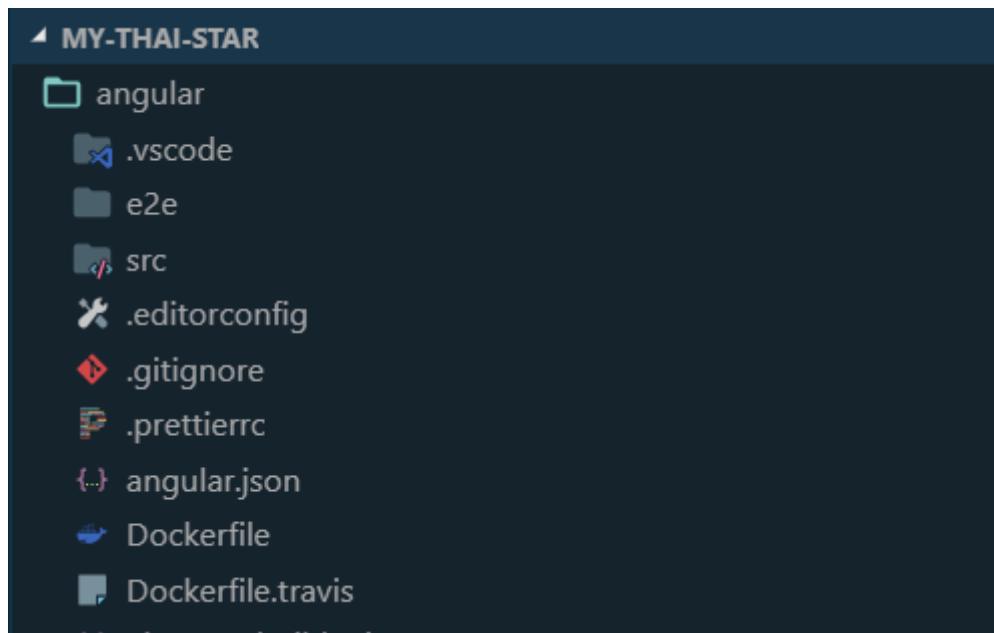
This will be used as follows:

```
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.

Custom tool

Name: Chrome-stable

Install automatically

Run Shell Command

Label:

Command:

```
if I which google-chrome > /dev/null; then
    sudo su << EOF
    wget -q -O - https://dl-ssl.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
```

Tool Home: /opt/chrome

Delete Installer

- **sonarTool:** Name of the sonarQube scanner installation.

SonarQube Scanner

SonarQube Scanner installations

Name: SonarQube

Install automatically

Install from Maven Central

Version: SonarQube Scanner 3.2.0.1227 ▾

Delete Installer

Add Installer ▾

Delete SonarQube Scanner

- **sonarEnv:** Name of the sonarQube environment. SonarQube is the default value for PL.

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

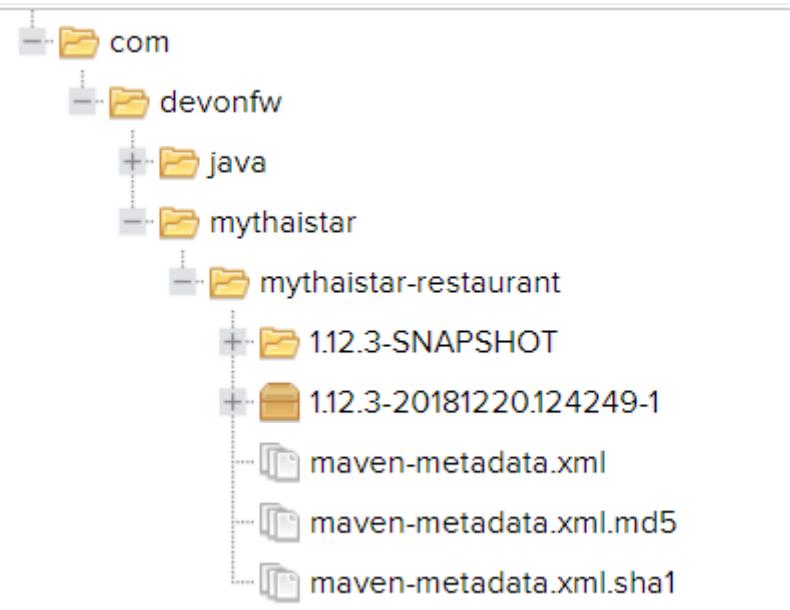
Server URL: http://sonarqube-core:9000/sonarqube
Default is http://localhost:9000

Server authentication token: [REDACTED]
SonarQube authentication token. Mandatory when anonymous access is disabled.

Add SonarQube

List of SonarQube installations

- **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.

This screenshot shows the 'Server Credentials' section of the Maven Global Settings configuration. It includes fields for 'ServerId' (set to 'pl-nexus'), 'Credentials' (set to 'admin/******** (Admin credentials to access Nexus)'), and buttons for 'Add', 'Delete', and help.

- **repositoryUrl**: The url of the repository.
- **globalSettingsId**: The id of the global settings file.

This screenshot shows the 'The configuration' section of the Maven Global Settings configuration. It includes fields for 'ID' (set to 'MavenSettings'), 'Name' (set to 'MyGlobalSettings'), 'Comment' (set to 'global settings'), and a 'Replace All' checkbox.

- **mavenInstallation**: The name of the maven tool.

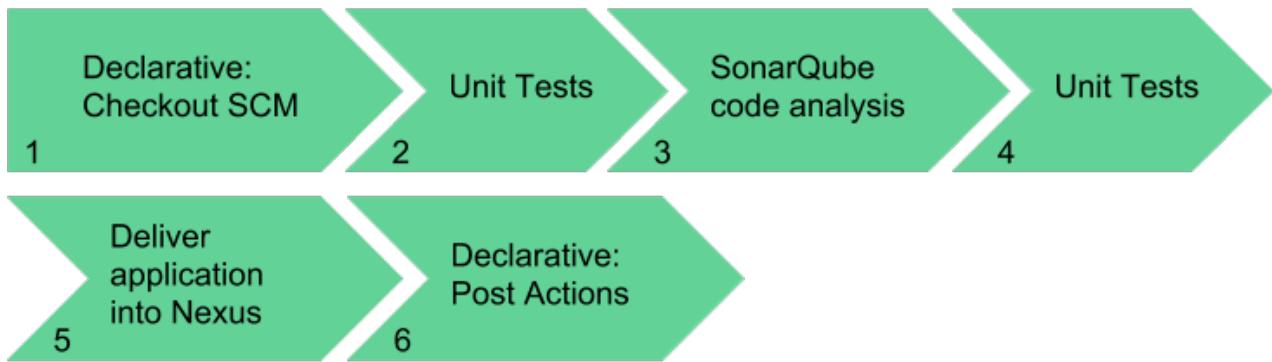
This screenshot shows the 'Maven' section of the Maven Global Settings configuration. It includes a 'Maven installations' table with one entry for 'Maven3' (Name: 'Maven3', Install automatically checked, Version: '3.6.0'). Buttons for 'Delete Installer' and 'Delete Maven' are also present.

59.1.7. 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. Unit Tests

This step will execute the project unit test with maven.

```
mvn clean test
```

3. 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 qq = waitForQualityGate()
if (qq.status != 'OK') {
    error "Pipeline aborted due to quality gate failure: ${qq.status}"
}

```

4. Deliver application into Nexus

Store all artifacts into nexus.

```
mvn deploy -Dmaven.test.skip=true
```

Adjustments

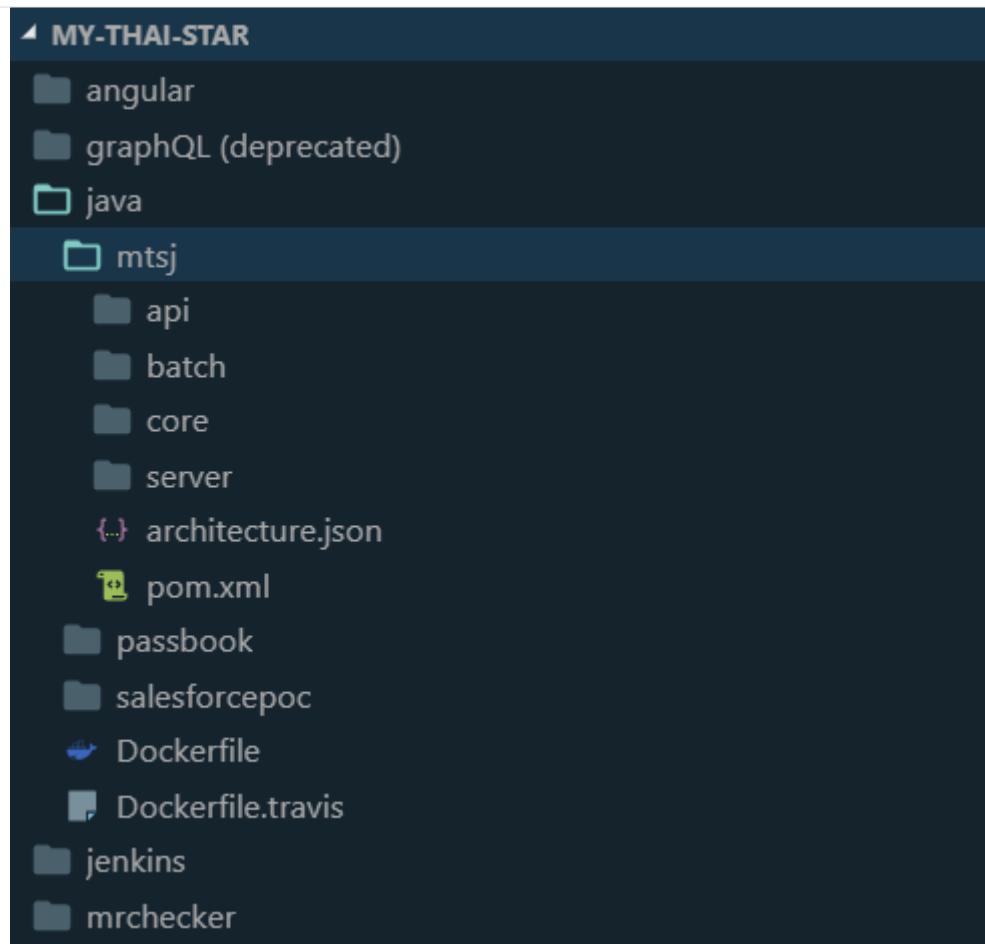
Pipeline Environment

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'  
}
```

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	<ul style="list-style-type: none"> <input checked="" type="checkbox"/> Maven
Name	Maven3
<input checked="" type="checkbox"/> Install automatically	
<ul style="list-style-type: none"> <input checked="" type="checkbox"/> Install from Apache 	
Version 3.6.0 ▾	
Delete Installer	
Add Installer ▾	
Delete Maven	

Distribution management

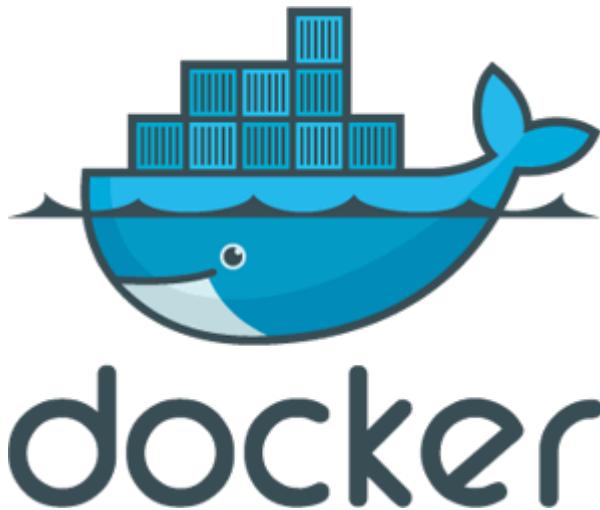
The only *extra* thing that needs to be added to the Java server-side is some information that determines where the artifact of the project is going to be stored in **Nexus**. This is going to be a

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>
```

59.2. 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.

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.

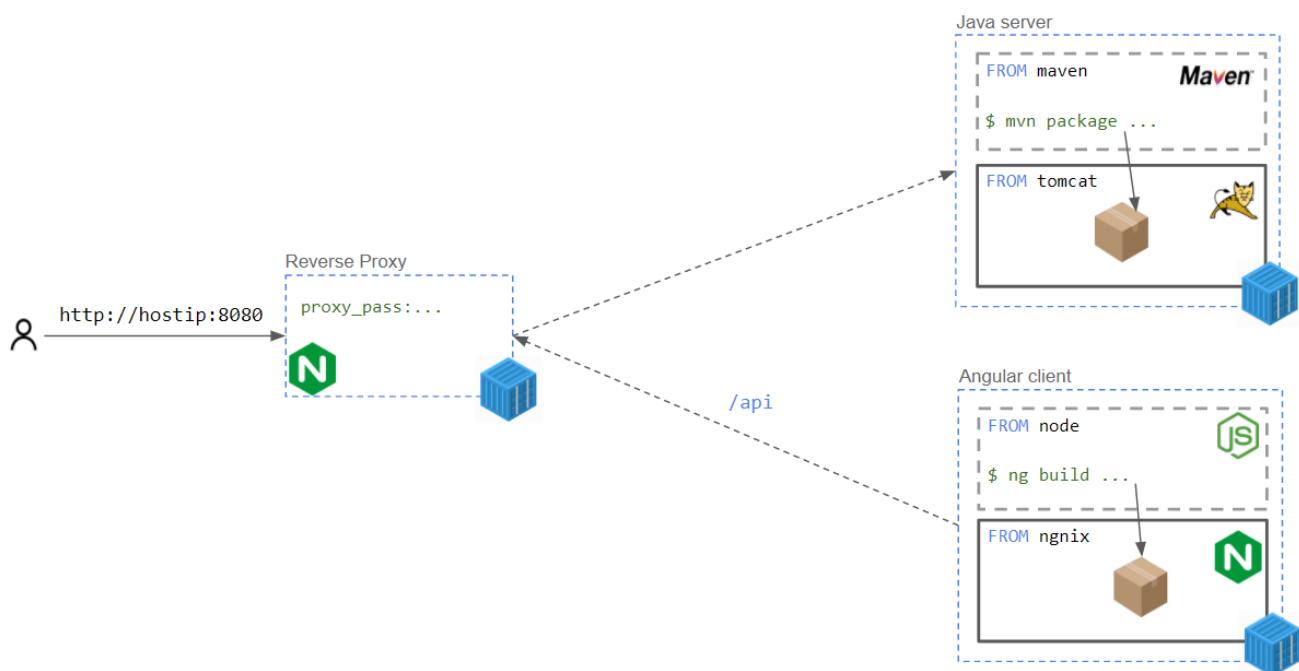
This external server will be located in <http://de-mucdevondepl01> .

59.3. 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. rever-proxy to angular client or angular client to java server) will be done by using, as hostnames, the service name `docker-compose` defines for each of them, followed by the internal 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.

59.4. Run My Thai Star

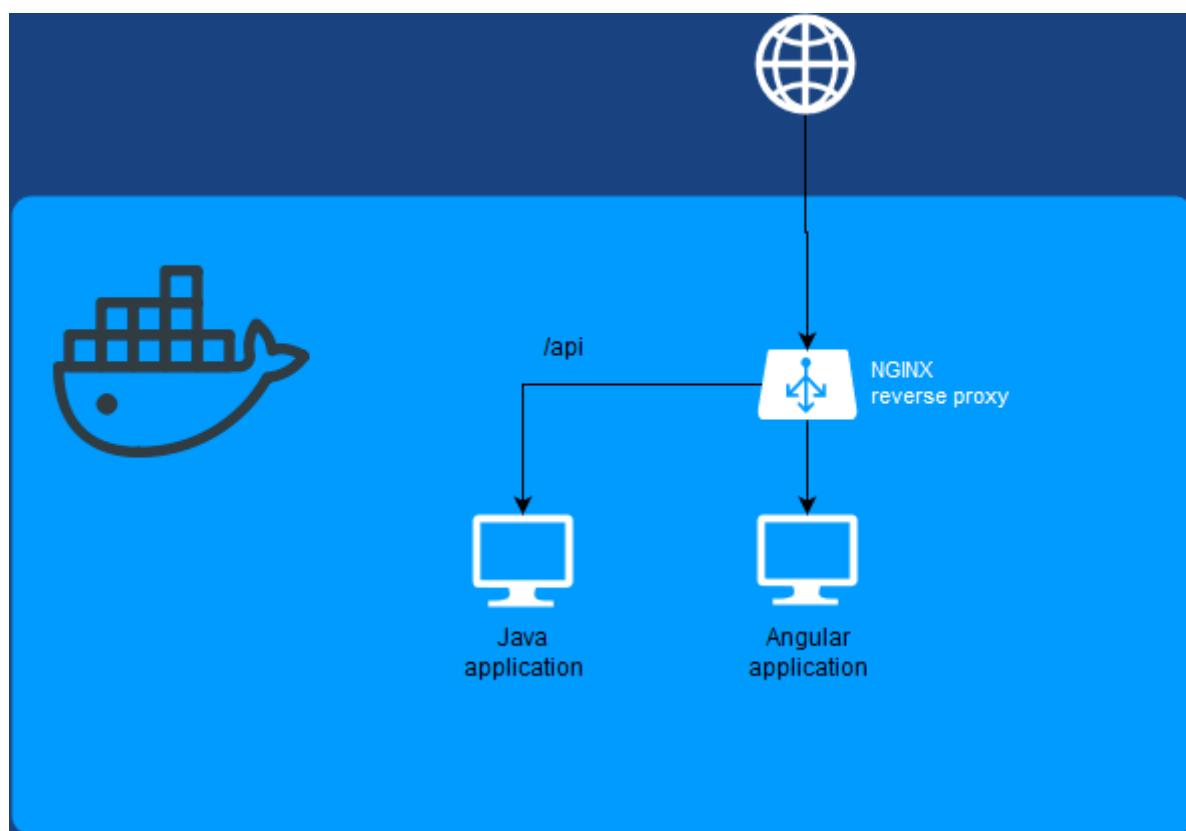
The steps to run **My Thai Star** are:

1. Clone the repository `$ git clone https://github.com/devonfw/my-thai-star.git`
2. Run the docker compose command: `$ docker-compose up`

59.4.1. Deployment Pipelines

As PL does not support deployments, we have created separate pipelines for this purpose. Those pipelines are: `MyThaiStar_DEPLOY-Together`, `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_DEPLOY-Together Pipeline

The MyThaiStar_DEPLOY-Together pipeline will deploy all parts of My Thai Star application into a remote server via ssh.

Parameters

- **sshAgentCredentials**: The SSH private key to connecto to remote server. The public key must be included in the remote server as a authorized_keys.
- **nexusApiUrl**: The url to the nexus api. <http://nexus3-core:8081/nexus3> is the default url for PL.
- **nexusCredentialsId**: The nexus credentials.
- **repository**: Name of the repository where the artifacts are stored. maven-snapshots is the default value for PL.

- **JAVA_VERSION:** The version of the java project that you want to deploy.
- **ANGULAR_VERSION:** The version of the java project that you want to deploy.
- **EXTERNAL_SERVER_IP:** The IP of the remote server where you will deploy My Thai Star.
- **APPLICATION_DIR:** The folder of the application.

Pipeline steps

- **Copy files to remote server:** Copy all files required for the deployment to the remote server. Those files are all files inside the reverse_proxy folder in the My Thai Star repository.
- **Deploy java application:** Call to **MyThaiStar_SERVER_DEPLOY** pipeline.
- **Deploy angular application:** Call to **MyThaiStar_FRONTEND_DEPLOY** pipeline.

MyThaiStar_SERVER_DEPLOY Pipeline

Deploys on the server the Java part of My Thai Star.



You need to run the MyThaiStar_DEPLOY-Together pipeline at least once before you run this one.

Parameters

- **sshAgentCredentials:** The SSH private key to connecto to remote server. The public key must be included in the remote server as a authorized_keys.
- **nexusApiUrl:** The url to the nexus api. <http://nexus3-core:8081/nexus3> is the default url for PL.
- **nexusCredentialsId:** The nexus credentials.
- **repository:** Name of the repository where the artifacts are stored. maven-snapshots is the default value for PL.
- **JAVA_VERSION:** The version of the java project that you want to deploy.
- **EXTERNAL_SERVER_IP:** The IP of the remote server where you will deploy My Thai Star.
- **APPLICATION_DIR:** The folder of the application.

Pipeline steps

- **Download artifact from Nexus:** Download the artifact from nexus using the nexus3 API. The downloaded artifact will be the last snapshot of the version specified in the parameters.
- **Deployment:** Deploy the download artifact in the remote server. It will create a new docker image and redeploy the java docker container.

MyThaiStar_FRONTEND_DEPLOY

Deploys on the server the Angular part of My Thai Star



You need to run the MyThaiStar_DEPLOY-Together pipeline at least once before you run this one.

Parameters

- **sshAgentCredentials:** The SSH private key to connecto to remote server. The public key must be included in the remote server as a authorized_keys.
- **nexusApiUrl:** The url to the nexus api. <http://nexus3-core:8081/nexus3> is the default url for PL.
- **nexusCredentialsId:** The nexus credentials.
- **repository:** Name of the repository where the artifacts are stored. maven-snapshots is the default value for PL.
- **ANGULAR_VERSION:** The version of the java project that you want to deploy.
- **EXTERNAL_SERVER_IP:** The IP of the remote server where you will deploy My Thai Star.
- **APPLICATION_DIR:** The folder of the application.

Pipeline steps

- **Download artifact from Nexus:** Download the artifact from nexus using the nexus3 API. The downloaded artifact will be the last snapshot of the version specified in the parameters.
- **Deployment:** Deploy the download artifact in the remote server. It will create a new docker image and redeploy the angular docker container.

59.4.2. Deployment Strategies

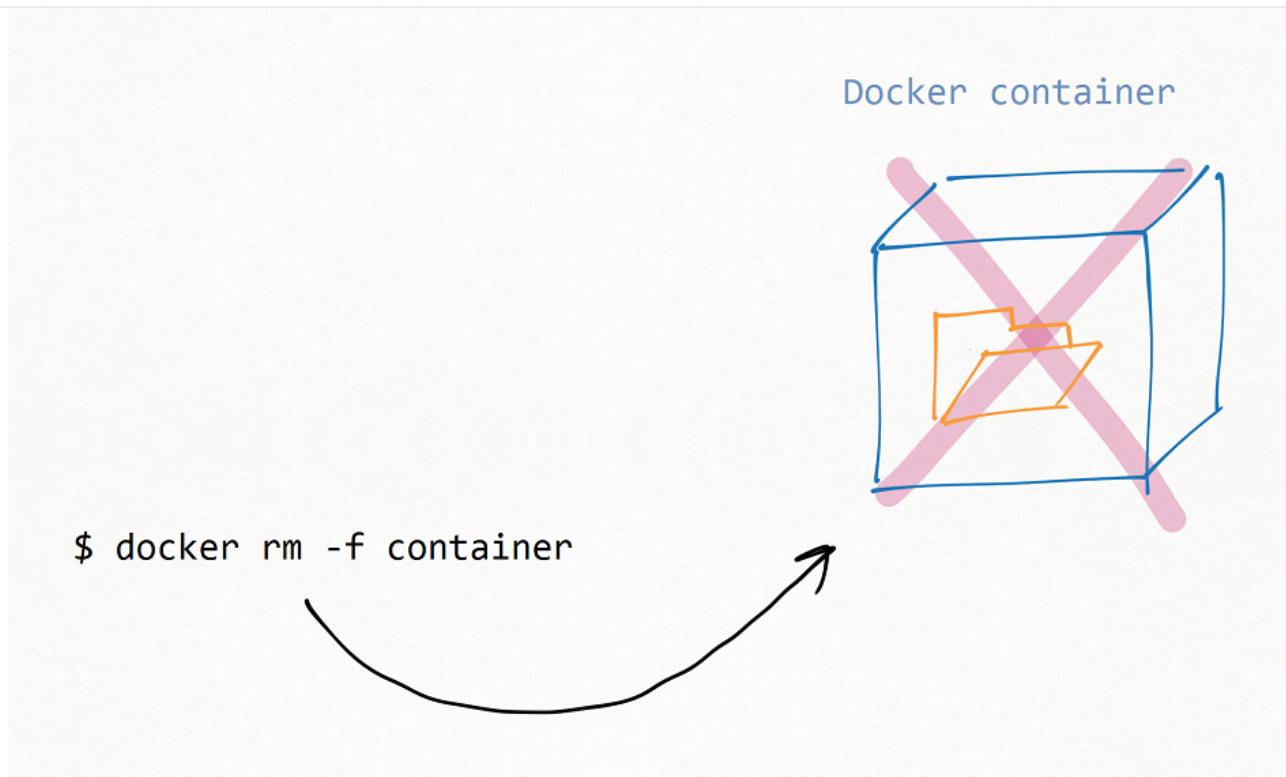
In this chapter different way of deploying My Thai Star are explained. Everything will be based in Docker.

Independent Docker containers

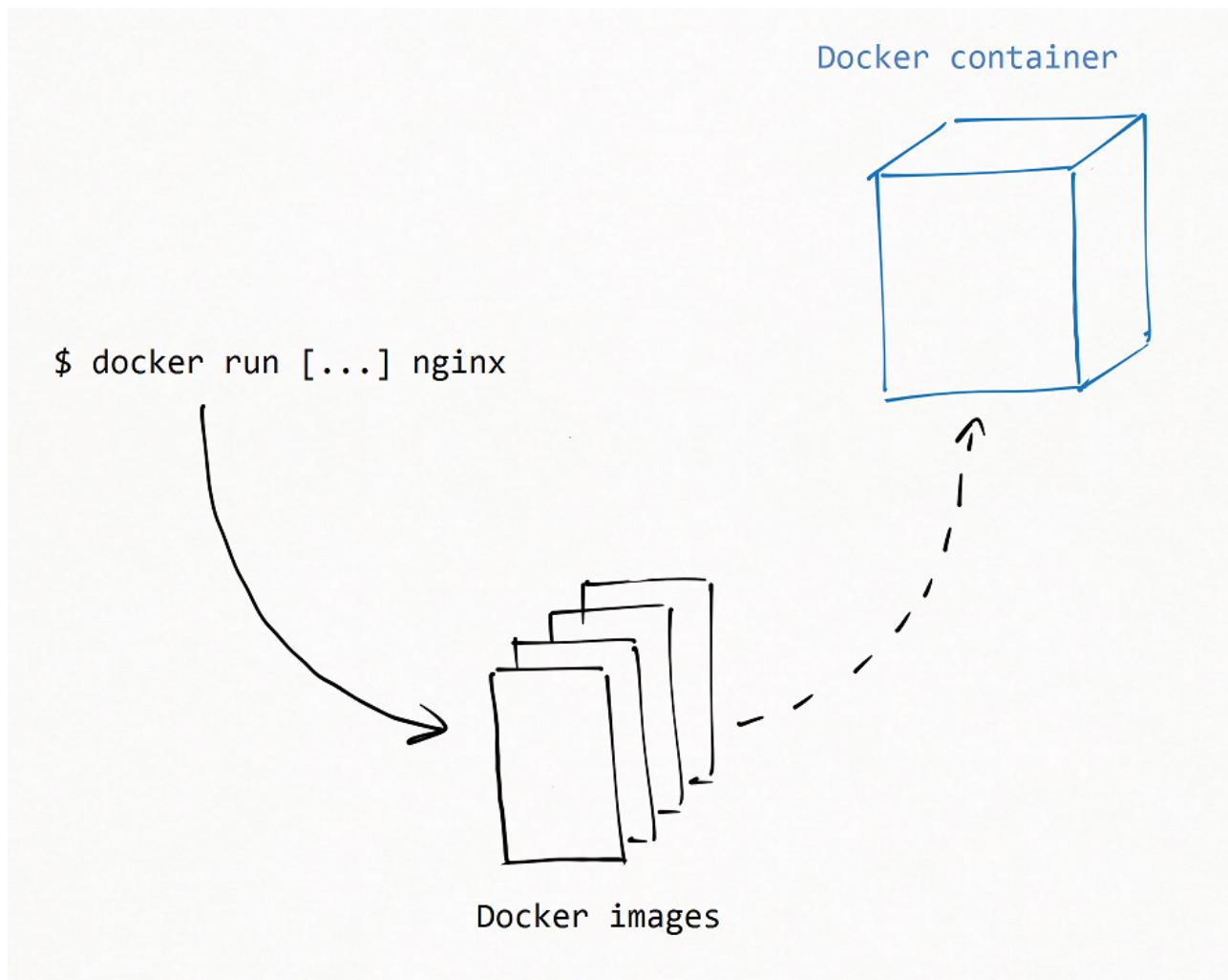
The first way of deployment will use isolated Docker containers. That means that if the client-side container is deployed, it does not affect the server-side container's lifecycle and viceversa.

Let's show how the containers will behave during their life cycle.

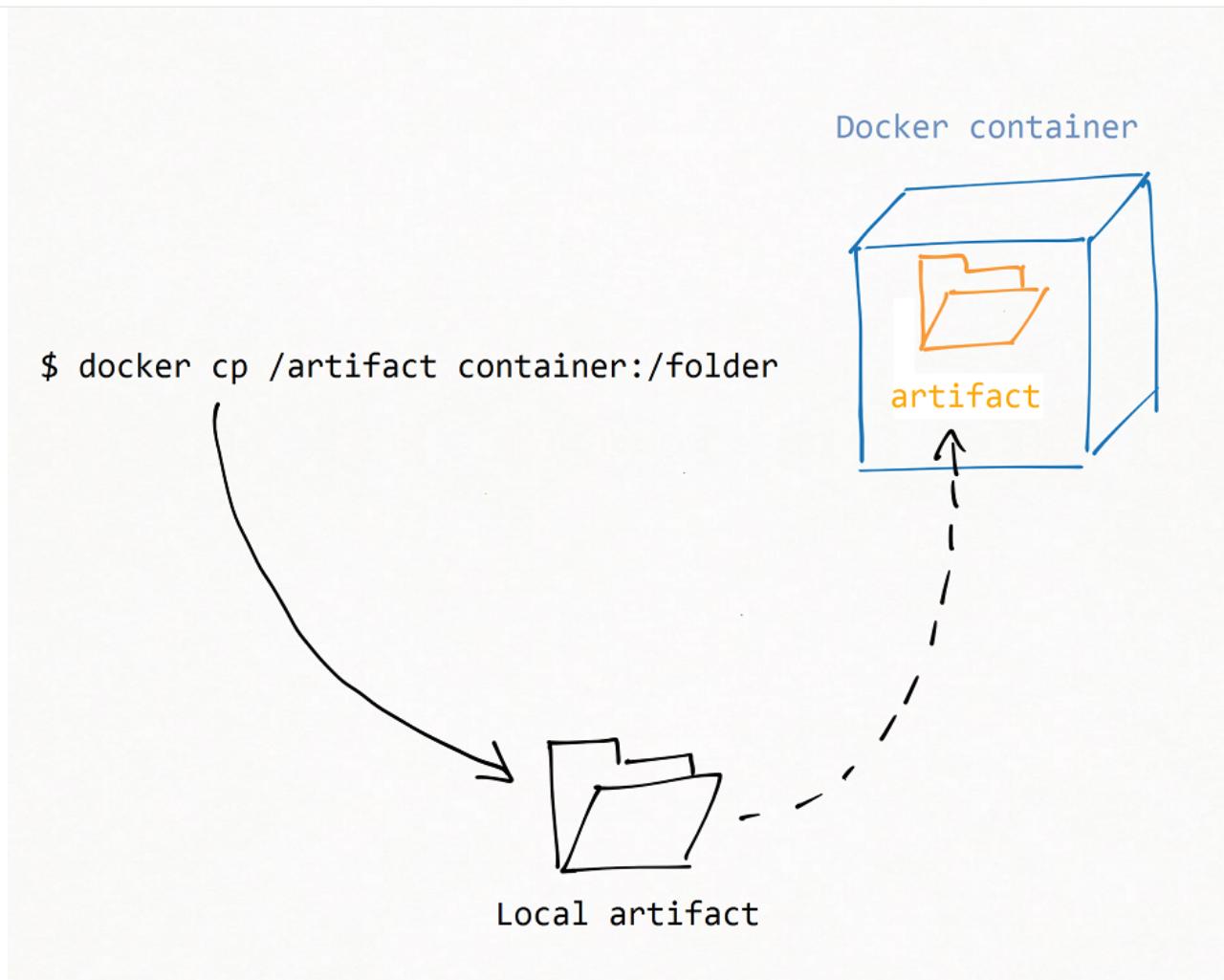
- 0) Copy everything you need into the Deployment Server directory
- 1) Remove existing container (Nginx or Tomcat)



- 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:

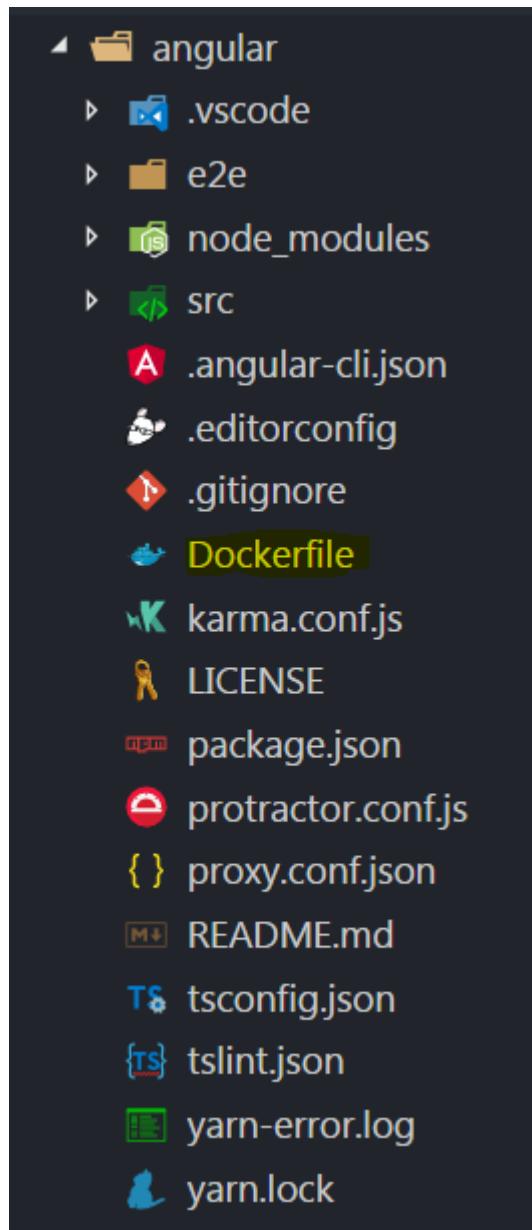
```
ssh -o StrictHostKeyChecking=no root@10.40.235.244
```

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.

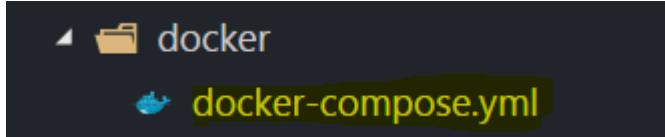


The second file is the **Dockerfile** for the Java server-side. Its function will be quite similar to the Angular one. It will run a **tomcat** Docker image and copy the Java artifact (`mythaistar.war` file) in its deployment folder.



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 Thai 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](#).

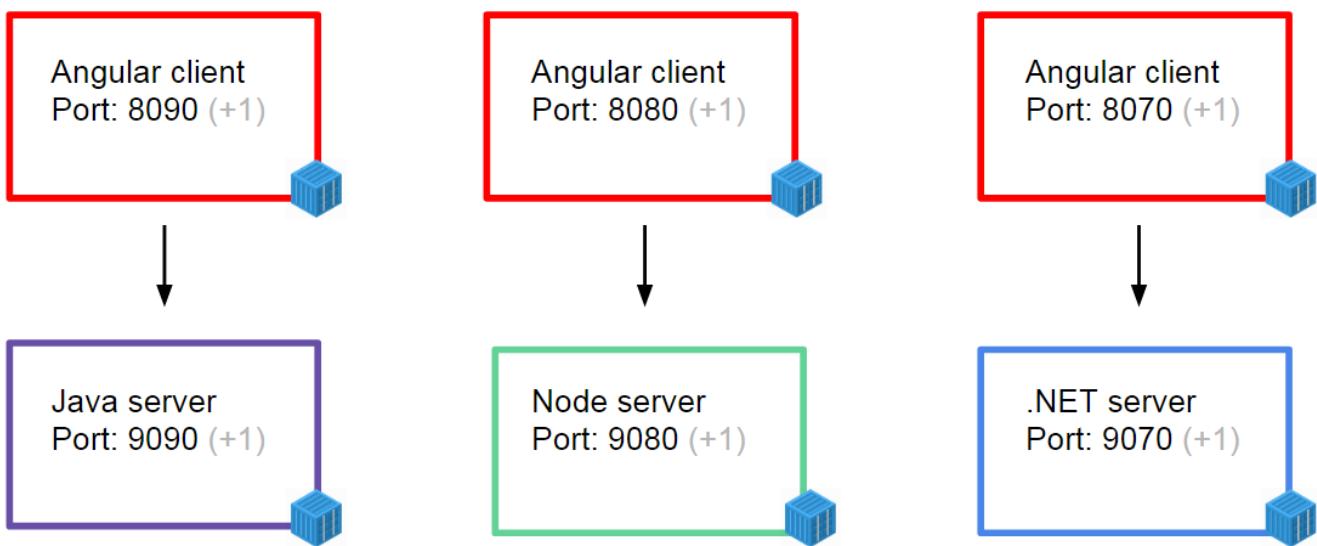
59.4.3. Future Deployment

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 backends: **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.frontend.rule=PathPrefixStrip:/api/;AddPrefix:/mythaistar"
  -
  "traefik.backend.healthcheck.path=/mythaistar/services/rest/dishmanagement/v1/category
  /0/"
  - "traefik.backend.healthcheck.interval=10s"
  - "traefik.backend.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.

Part XI: Contributing Guide

60. Code Contributions

60.1. Notes on Code Contributions

Both projects, devon and OASP, are intended to be easy to contribute to. One service allowing such simplicity is GitHub was therefore selected as preferred collaboration platform.

In order to contribute code, git and GitHub specific pull-requests are being used.

It is mandatory to follow the [code of conduct](#) that must be present in the root of every OSS or private project as [CODE_OF_CONDUCT.asciidoc](#) or [CODE_OF_CONDUCT.md](#).

60.2. Introduction to Git and GitHub

Git is a version control system used for a coordinated and versioned collaboration of computer files. It enables a project to be easily worked on by multiple developers and contributors.

GitHub is an online repository used by deonvfw and OASP in order to host the corresponding files. Using the command line tool or the GUI Tool "GitHub Desktop" a user can easily manage project files. There are private and public repositories. Public ones (like OASP) can be accessed by everyone, private repositories (like devon) require access permissions.

60.2.1. Creating a new user account

The devon and OASP projects use GitHub as hosting service. Therefore you'll need an account to allow collaboration. Visit [this page](#) to create a new account. If available, use **your CORP username** as GitHub username and **your CORP email address**.

A GitHub account is essential for contributing code and gaining permissions to access private repositories.

60.2.2. Git Basics

An in-depth documentation on basic Git syntax and usage can be found on the [official Git homepage](#). Another helpful and easy to follow instruction can be found [here](#).

60.3. Structure of our projects

In total, there are three GitHub projects regarding OASP and devon:

- [oasp-forge](#)

Repository used for work on the guide - Similar to the according devon repository [devon-guide](#)

- [oasp](#)

The official *Open Application Standard Platform* project repository. Usually, two main branches exist:

- **develop**

This branch contains software in the state of being in development.

- **master**

This branch contains software in release state.

- [devonfw](#)

This is a private repository. You have to be logged in and have permissions to access the project and its repositories. Similar to OASP, there are usually two branches:

- **develop**

- **master**

60.4. Contributing to our projects

In order to contribute to our projects, developers must follow the following [development guidelines](#). Other sources about contributing to devon/OASP:

- [devonfw code contributions](#)
- [devonfw documentation](#)
- [Devon collaboration](#)

Every project must include the following files in order to establish the contributing rules and facilitate the process:

- **CONTRIBUTING.asciidoc** that establishes the specific guidelines of contributing in a project repository.
- **CODE_OF_CONDUCT.asciidoc** mandatory to contribute.
- **ISSUE_TEMPLATE.asciidoc** that defines the appropriated way to submit an issue in a project repository.
- **PULL_REQUEST_TEMPLATE.asciidoc** that specifies the rules in order to submit a pull request in a project repository.

This files should be included at the root folder or in a [docs](#) folder. [This repository](#) is a good resource to find the perfect templates for issues and pull requests that fit in your repository.

60.4.1. Process of contributing code to the devon/OASP projects

- Use the issue tracker to check whether the issue you would like to be working on exists. Otherwise create a new issue.

The screenshot shows the GitHub interface for the oasp/oasp4j repository. The top navigation bar includes links for Code, Issues (97), Pull requests (10), Projects (5), Wiki, and Insights. The main content area displays a list of 97 open issues. A search bar at the top allows filtering by 'is:issue is:open'. The issues are listed with their titles, descriptions, labels (e.g., SCM, task, enhancement, bug, persistence), and a link to view the issue details.

Figure 81. Using GitHub's issue tracker

- Before making more complex changes you should probably notify the community. The worst case would be you investing time and effort into something that'll be later rejected. Oftentimes the [Devon Community](#) on Yammer will have the right answer.
- Assign yourself to the issue you would like to work on. If a member was already assigned to your preferred issue, get in contact to contribute to the same issue.
- Fork the desired repository to your corporate GitHub account. Afterwards you'll have your own copy of the repository you'd like to work on.
- Create a new branch for your feature/bugfix. Check out the develop branch for the upcoming release. The following changes will afterwards be merged when the new version is released.
- Please read the [Working with forked repositories](#) document to learn all about this topic.
 - Check out the develop branch

```
git checkout develop-x.y.z
```

- Create a new branch

```
git checkout -b myBranchName
```

- Apply your modifications according to the [coding conventions](#) to the newly created branch
- Verify your changes to only include relevant and required changes.
- Commit your changes locally
 - When committing changes please follow this pattern for your commit message:

```
#<issueId>: <change description>
```

- When working on multiple different repositories, the actual repository name of the change should also be declared in the commit message:

```
<project>/<repository>#<issueId>: <change description>
```

For example:

```
devonfw/devon4j#1: added REST service for tablemanagement
```

Note: Starting directly with a # symbol will comment out the line when using the editor to insert a commit message. Instead, you should use a prefix like a space or simply typing "Issue". E.g.:

```
Issue #4: Added some new feature, fixed some bug
```

The language to be used for commit messages is English.

- Push the changes to your Fork of the repository
- After completing the issue/bugfix/feature, use the *pull request* function in GitHub. This feature allows other members to look over your branch, automated CI systems may test your changes and finally apply the changes to the corresponding branch (if no conflicts occur).

Use the tab "Pull requests" and the button labeled "New pull request". Afterwards you can *Choose different branches or forks above to discuss and review changes*.

60.5. Reviewing Pull Requests

Detailed information about revieweing can be found on the [official topic on GitHub Pull Requests](#).

There are two different methods to review Pull Requests:

- **Human based reviews**

Other project members are able to discuss the changes made in the pull request by having insight into changed files and file differences by commenting.

The screenshot shows a pull request interface with three comments:

- hohwille commented on 22 Sep 2016**: Could you please give some rationale for introducing `SpringBootTestApp`? This means that tests stop testing the actual `SpringBootApp` hence bugs might not be discovered and extra maintenance overhead may happen. In our project we are using only `application.properties` to tweak our test world.
- jomora commented on 22 Sep 2016**: Good point. I'll have a look at it...
- hohwille requested changes on 22 Sep 2016**: A diff view showing a change in `pom.xml`:


```

samples/core/pom.xml
...
... @@ -213,6 +213,7 @@
213 213 <dependency>
214 214   <groupId>org.springframework.boot</groupId>
215 215   <artifactId>spring-boot-starter-web</artifactId>
216 +   <scope>provided</scope>
      
```

hohwille on 22 Sep 2016 Owner
I am fine with this but I would have expected to remove this here and also move the app to the server module. But this will have other implications. So just some thoughts for discussion

Figure 82. People can add comments to pull requests and suggest further changes

• CI based reviews

CI Systems like [Jenkins](#) or [Travis.ci](#) are able to listen for new pull requests on specified projects. As soon as the request was made, Travis for example checks out the to-be-merged branch and builds it. This enables an automated build which could even include testcases. Finally, the CI approves the pull requests if the build was built and tested successfully, otherwise it'll let the project members know that something went wrong.

The screenshot shows a CI status summary with the following items:

- All checks have failed**: 1 errored check
- continuous-integration/travis-ci/pr**: The Travis CI build could not complete due to...
- This branch has no conflicts with the base branch**: Only those with write access to this repository can merge pull requests.

Figure 83. If Travis fails to build a project, it'll post the results directly to the pull request

Combining these two possibilities should accelerate the reviewing process of pull requests.



This document is the **Official Covenant Code of Conduct** that must be present in every OASP or devonfw project at the root folder as `CODE_OF_CONDUCT.asciidoc` or `CODE_OF_CONDUCT.md`. Please, include this contents in your repository and the **Product Owner email address** in the right place below.

61. Contributor Covenant Code of Conduct

61.1. Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

61.2. Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

61.3. Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

61.4. Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is

representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

61.5. Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at **[INSERT PRODUCT OWNER EMAIL ADDRESS]**. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

61.6. Attribution

This Code of Conduct is adapted from the [Contributor Covenant](#), version 1.4, available at <https://www.contributor-covenant.org/version/1/4/code-of-conduct.html>

62. Development Guidelines

- **Always ask before creating a pull request.** To avoid duplication efforts, its better to discuss it with us first or create an issue.
- **All code must be reviewed via a pull request.** Before anything can be merged, it must be reviewed by other developer and ideally at least 2 others.
- **Use git flow processes.** Start a feature, release, or hotfix branch, and you should never commit and push directly to master.
- **Code should adhere to lint and codestyle tests.** While you can commit code that doesn't validate but still works, it is encouraged to validate your code. It saves other's headaches down the road.
- **Code must pass existing tests when submitting a pull request.** If your code breaks a test, it needs to be updated to pass the tests before merging.
- **New code should come with proper tests.** Your code should come with proper test coverage, ideally 95%, minimum 80%, before it can be merged.
- **Bug fixes must come with a test.** Any bug fixes should come with an appropriate test to verify the bug is fixed, and does not return.
- **Code structure should be maintained.** The structure of the repo and files has been carefully crafted, and any deviations from that should be only done when agreed upon by the entire community.

63. Working with forked repositories

63.1. Fork a repository

A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

Most commonly, forks are used to either propose changes to someone else's project or to use someone else's project as a starting point for your own idea.

63.1.1. Propose changes to someone else's project

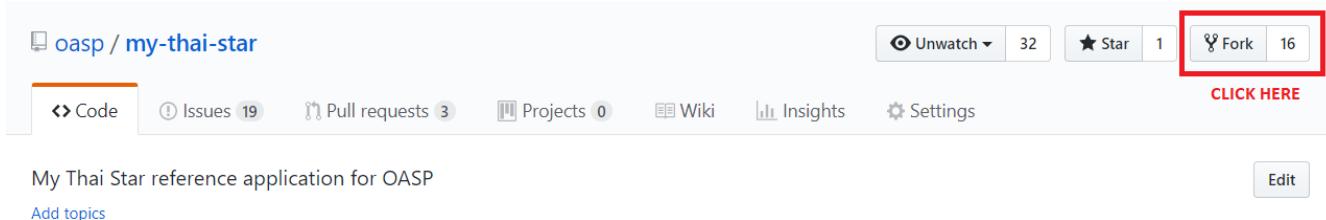
A great example of using forks to propose changes is for bug fixes. Rather than logging an issue for a bug you've found, you can:

1. Fork the repository.
2. Make the fix.
3. Submit a pull request to the project owner.

If the project owner likes your work, they might pull your fix into the original repository!

63.1.2. How to fork a repository

GitHub, GitLab and Bitbucket have a very accessible option to fork any repository you can access to. For example, at GitHub you will only need to do the following:



The screenshot shows a GitHub repository page for 'oasp / my-thai-star'. At the top right, there is a 'Fork' button with a count of '16' next to it. This button is highlighted with a red box. Below the header, there is a navigation bar with links for 'Code', 'Issues 19', 'Pull requests 3', 'Projects 0', 'Wiki', 'Insights', and 'Settings'. To the right of the navigation bar, there is a 'CLICK HERE' button. The main content area displays the repository's description: 'My Thai Star reference application for OASP' and a 'Edit' button. There is also a 'Add topics' link.

In order to work locally you will need to pull your forked repository. Open the Terminal or Git Bash and run the following command:

```
$ git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY
```

63.2. Configuring a remote for a fork

You must configure a remote that points to the upstream repository in Git to sync changes you make in a fork with the original repository. This also allows you to sync changes made in the original repository with the fork.

1. Open Terminal or Git Bash.
2. List the current configured remote repository for your fork.

```
$ git remote -v  
origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)  
origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
```

3. Specify a new remote upstream repository that will be synced with the fork.

```
$ git remote add upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git
```

4. Verify the new upstream repository you've specified for your fork.

```
$ git remote -v  
origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)  
origin https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)  
upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (fetch)  
upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git (push)
```

63.3. Syncing a fork

Sync a fork of a repository to keep it up-to-date with the upstream repository.

Before you can sync your fork with an upstream repository, you must configure a remote that points to the upstream repository in Git.

1. Open Terminal or Git Bash.
2. Change the current working directory to your local project.
3. Fetch the branches and their respective commits from the upstream repository. Commits to **master** will be stored in a local branch, **upstream/master**.

```
$ git fetch upstream  
remote: Counting objects: 75, done.  
remote: Compressing objects: 100% (53/53), done.  
remote: Total 62 (delta 27), reused 44 (delta 9)  
Unpacking objects: 100% (62/62), done.  
From https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY  
* [new branch]      master    -> upstream/master
```

4. Check out your fork's local **master** branch.

```
$ git checkout master  
Switched to branch 'master'
```

5. Merge the changes from **upstream/master** into your local master branch. This brings your fork's **master** branch into sync with the upstream repository, without losing your local changes.

```
$ git merge upstream/master
Updating a422352..5fdff0f
Fast-forward
 README           |    9 -----
 README.md        |    7 ++++++
 2 files changed, 7 insertions(+), 9 deletions(-)
 delete mode 100644 README
 create mode 100644 README.md
```

If your local branch didn't have any unique commits, Git will instead perform a "fast-forward":

```
git merge upstream/master
Updating 34e91da..16c56ad
Fast-forward
 README.md          |    5 +---
 1 file changed, 3 insertions(+), 2 deletions(-)
```

6. Push the changes to update your fork on GitHub, GitLab, Bitbucket, etc.

64. Wiki Contributions

Our wikis are written in the so called *AsciiDoc* format. Check the [AsciiDoc cheatsheet](#) and the [AsciiDoc quick reference](#) for more information. Knowing the following basic features should allow you to convert your Word documents into the Wiki friendly AsciiDoc format.

It is mandatory to follow the [code of conduct](#) that must be present in the root of every OSS or private project as `CODE_OF_CONDUCT.asciidoc` or `CODE_OF_CONDUCT.md`.

64.1. Text styles

Italic Text

Italic Text

Bold Text

Bold Text

Mono Spaced Text

+Mono Spaced Text+

Text in ^{Superscript}

Text in ^{^Superscript^}

Text in _{Subscript}

Text in _{~Subscript~}

64.2. Titles

A title can be initiated like this:

```
[[Contributing-Wiki.asciidoc]]
= Level 1 header
[[contributing-wiki.asciidoc_level-2-header]]
== Level 2 header
[[contributing-wiki.asciidoc_level-3-header]]
== Level 3 header
...
...
```

64.3. Lists

Ordered and unordered lists can be created like this:

```
Ordered list:
```

- . Item 1
- . Item 2
- . Item 3
-

```
Unordered list:
```

- * Item 1
- * Item 2
- * Item 3
- * ...

64.4. Tables

The following example shows how a table can be created. Note that the *header* flag is optional.

```
[options="header"]
|===
|Header 1|Header 2| Header 3
| Item 1| Item 2| Item 3
| ... | ... | ...
|===
|...
```

64.5. Source Code

If you want to show off some code examples, you can use the *code block*:

```
[source]
-----
Some source code
-----
```

You can also specify which script language is used. This will allow GitHub to use a matching color scheme. Therefore, just type in the type of code used:

```
[source, bash]
```

or

```
[source, java]
```

Part XII: Release Notes

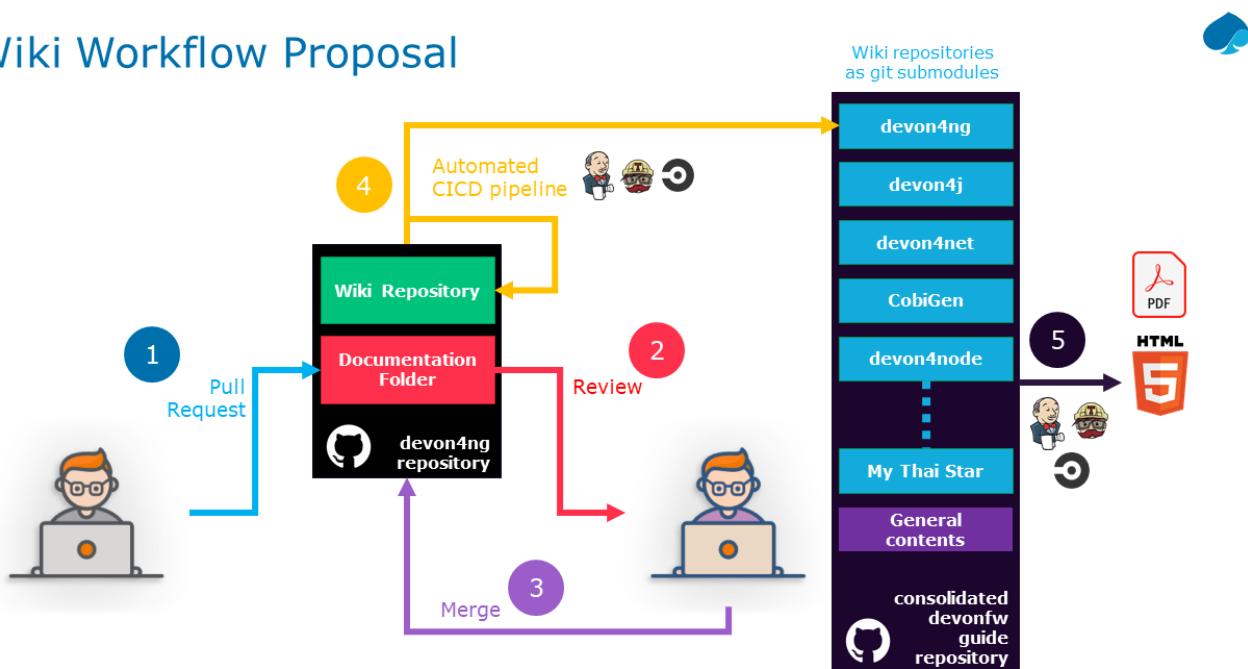
65. devonfw Release notes 3.1 “Goku”

65.1. 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



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>.

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>.

65.2. Changes and new features

65.2.1. 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 preconfigured 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

65.2.2. My Thay 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 toggable (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

65.2.3. 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

65.2.4. 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](#).

65.2.5. 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

65.2.6. devon4net

The following changes have been incorporated in devon4net:

- New circuit breaker component to communicate microservices via HTTP
- Resolved the update packages issue

65.2.7. 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>

65.2.8. 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.

65.2.9. 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.

65.2.10. 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>

65.2.11. 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

65.2.12. 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

66. devonfw Release notes 3.0 “Fry”

66.1. 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.

66.2. Changes and new features

66.2.1. 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 preconfigured 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

66.2.2. My Thay 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

66.2.3. 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.

66.2.4. 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.

66.2.5. 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.

66.2.6. 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.

66.2.7. 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.

66.2.8. 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>

66.2.9. 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.

66.2.10. 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.

66.2.11. 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>

66.2.12. 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>.

67. devonfw Release notes 2.4 “EVE”

67.1. 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.

67.2. Changes and new features

67.2.1. 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.

67.2.2. 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 preconfigured 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

67.2.3. My Thay 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.

67.2.4. 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.

67.2.5. 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.

67.2.6. 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.

67.2.7. 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>

67.2.8. 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.

67.2.9. 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.

67.2.10. 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>

67.2.11. 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>

67.2.12. 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

67.2.13. 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

68. devonfw Release notes 2.3 "Dash"

68.1. 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.

68.2. 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.

68.3. Changes and new features

68.3.1. 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/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 included.
 - Python3 integrated
 - Spyder3 IDE integrated in python3 installation
 - OASP4JS-application-template for Angular5 at workspaces/examples
 - Devon4sencha starter templates updated

68.3.2. 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>

68.3.3. 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/tools-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/tools-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

68.3.4. 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

68.3.5. 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. Asciidoctor is used for devonfw guide PDF generation. See: <https://github.com/devonfw/devon-guide/wiki>

68.3.6. 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.

68.3.7. 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>

68.3.8. 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

68.3.9. 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>

68.3.10. 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>

68.3.11. 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.)
 - Ubíquese 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>

68.3.12. 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)

68.3.13. 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>

68.3.14. 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 Holwiller. 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 Cobige 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

68.3.15. 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>

69. devonfw Release notes 2.2 "Courage"

69.1. 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.

69.2. 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>

69.3. 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.

69.4. 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/tools-cobigen/releases>

69.5. 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>

69.6. 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>

69.7. 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>

69.8. Microservices Netflix

devonfw now includes a microservices implementation based on Spring Cloud Netflix. It provides a Netflix OSS integrations for Spring Boot apps through autoconfiguration 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/devon/wiki/devon-microservices>

69.9. devonfw distribution based on Eclipse OOMPH

The new Eclipse devonfw distribution is now based on Eclipse OOMPH, which allows us, at 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)

69.10. 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>

69.11. 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>

69.12. 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>

69.13. 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>

69.14. 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 Cloudfoundtry

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>

69.15. 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>

69.16. 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*.

70. Release notes devonfw 2.1.1 "Balu"

70.1. 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:

70.1.1. 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.

70.1.2. 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>

70.1.3. 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](#)

70.1.4. 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](#).

70.1.5. WebSphere & Wildfly deployment documentation

The new version of devonfw contains more elaborate and updated documentation about deployment on [WebSphere](#) and [Wildfly](#).

70.2. Version 2.1.1 Updates, fixes & some new features

70.2.1. 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](#)

70.2.2. 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.

70.2.3. 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.)

70.2.4. 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](#)

70.2.5. 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.

70.3. Version 2.1 New features, improvements and updates

70.3.1. 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.

70.3.2. 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

70.3.3. 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](#)

70.3.4. 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.

70.3.5. 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](#)

70.3.6. 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](#)

70.3.7. 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](#)

70.3.8. 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](#)

70.3.9. 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](#)

70.3.10. 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](#)

70.3.11. 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](#)

70.3.12. 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](#)

70.3.13. 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.

70.3.14. 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 preconfigured set of essential plugins. See: [OASP/devonfw Atom editor \("IDE"\) settings & packages](#)

70.3.15. 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](#)

70.3.16. 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](#)

70.3.17. 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](#)

70.3.18. 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](#)

70.3.19. 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.

70.3.20. 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.

70.3.21. 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!