

Capgemini

[] | /home/travis/build/devonfw/devonfw-guide/theme/images/devonfw.jpg

devonfw guide

Table of Contents

Getting Started	1
Devcon	21
devon4j documentation	64
devon4ng documentation	65
Arquitecture	65
Architecture	66
Overview	67
Layers	67
Modules	68
Meta arquitecture	68
Meta Architecture	69
Introduction	70
Purpose of this document	70
Goal of the Client Architecture	70
Architecture Views	70
devonfw Reference Client Architecture	71
Client Architecture	71
Dialog Container Architecture	72
Dialog Component Architecture	74
Appendix	77
Notes about Quasar Client	77
References	78
Layers	78
Components Layer	79
Smart and Dumb Components	80
Interaction of Smart and Dumb Components	82
Services Layer	84
Boundaries	85
Store	86
Use Case Service	87
Adapter	88
Interaction of Smart Components through the services layer	89
Pattern	89
Example	90
Guides	91
Package Managers	92
Semantic Versioning	93
Do not modify package.json and lock files by hand	94

What does the lock file do	95
devon4ng NPM-Yarn Workflow	97
Introduction	97
.1. Angular	106
Accessibility	107
1. Key Concerns of Accessible Web Applications	108
1.1. Semantic Markup	108
1.2. Keyboard Accessibility	109
2. Visual Assistance	111
2.1. Color Contrast	111
2.2. Visual Information	111
2.3. Audiovisual Media	111
3. Accessibility with Angular Material	112
3.1. ListKeyManager	112
3.2. FocusKeyManager	112
3.3. ActiveDescendantKeyManager	112
3.4. FocusTrap	113
3.5. Regions	113
3.6. InteractivityChecker	113
3.7. LiveAnnouncer	113
3.8. API reference for Angular CDK a11y	114
4. What are Angular Elements?	115
5. Why use Angular Elements?	116
6. Negative points about Elements	117
7. How to use Angular Elements?	118
7.1. Installing Angular Elements	118
7.2. Preparing the components in the modules	118
7.3. A component example	119
7.4. Solving the error	121
7.5. Building the Angular Element	122
7.6. Using the Angular Element	124
8. Angular Element within another Angular project	128
8.1. Copy bundled script and css to resources	128
8.2. Add bundled script to angular.json	128
8.3. Using Angular Element	129
Lazy loading	131
An example with Angular	132
9. Implementation	134
Conclusion	144
10. Angular Library	145
11. Whats a library?	146

12. How to build a library	147
12.1. Creating an empty application	147
12.2. Generating a library	147
12.3. Generating/Modifying in our library	147
12.4. Exporting the generated things	147
12.5. Building our library	148
12.6. Packing the library	148
12.7. Publishing to npm repository (optional)	148
12.8. Installing our library in other projects	148
12.9. Using the library	149
12.10. devon4ng libraries	151
Angular Material Theming	152
Theming basics	153
Prebuilt themes	155
Custom themes	157
13. Basics	158
14. Basic custom theme	159
15. Full custom theme	163
16. Multiple themes and overlay-based components	166
Useful resources	167
Angular Progressive Web App	168
Assumptions	169
Sample Application	170
17. Step 1: Create a new project	171
18. Step 2: Create a service	172
19. Step 3: Use the service	173
20. Step 4: Structures, styles and updates	174
21. Step 5: Make it Progressive.	175
22. Step 6: Configure the app	177
23. Step 7: Check that your app is a PWA	178
APP_INITIALIZER	181
24. What is the APP_INITIALIZER pattern	182
25. What is APP_INITIALIZER	183
26. Creating a APP_INITIALIZER configuration	187
27. Setting up the config files	188
27.1. Docker external configuration (Optional)	188
27.2. External json configuration	188
28. Setting up the proxies	190
28.1. Docker (Optional)	190
28.2. External Configuration	190
29. Adding the loadExternalConfig boolean to the environments	191

30. Creating core configuration service	192
30.1. Using the Config Service.....	194
30.2. Final steps	195
Component Decomposition	196
31. Defining Components.....	197
32. Defining state	200
33. When are Dumb Components needed	203
Consuming REST services	204
34. Defining Adapters	205
35. Token management	208
36. Global Error Handler in angular	209
37. What is ErrorHandler	210
38. Creating your custom ErrorHandler step by step	211
38.1. Creating the custom ErrorHandler class	211
38.2. Creating a ErrorInterceptor	212
38.3. Creating a Error Module	212
39. Handling Errors	214
39.1. Using MatSnackBarService and NgZone	214
File Structure	215
40. Toplevel	216
41. Feature Modules	217
42. Components Layer	218
Internationalization	220
43. devon4ng i18n approach	221
43.1. Install NGX Translate	221
43.2. Configure NGX Translate	221
43.3. Usage	222
43.4. Using the service, pipe or directive	224
Routing	226
44. Defining Routes	227
44.1. Example 1 - No Lazy Loading	227
44.2. Example 2 - Lazy Loading	228
45. Triggering Route Changes	230
46. Guards	231
46.1. Example 1 - CanActivate and CanActivateChild guards	231
46.2. Example 2 - CanLoad guard	233
Testing	235
47. Testing Concept	236
48. Testing Smart Components	237
49. Testing state transitions performed by stores	240
50. Testing services	241

51. Angular CLI common issues	244
52. Angular CLI update guide	245
Working with Angular CLI	246
53. Installing Angular CLI	247
54. Running a live development server	248
55. Running Unit Tests	249
56. Linting the code quality	250
57. Generating Code	251
57.1. Creating a new Angular CLI project	251
57.2. Creating a new feature module	251
57.3. Creating a new component	251
58. Configuring an Angular CLI project	253
58.1. Ionic	254
Ionic: Getting started	255
Why Ionic?	256
Basic environment set up	257
59. Install Ionic CLI	258
60. Update Ionic CLI	259
Basic project set up	260
Ionic: From code to android	262
Assumptions	263
From ionic 4 to Android project	264
61. Modifications	265
62. Build	266
From Android project to emulated device	267
From Android project to real device	270
63. Send APK to Android through USB	271
64. Send APK to Android through email	273
Result	274
Ionic Progressive Web App	275
Assumptions	276
Sample Application	277
65. Step 1: Create a new project	278
66. Step 2: Structures and styles	279
67. Step 3: Add functionality	280
68. Step 4: PWA Elements	281
69. Step 5: Make it Progressive	282
70. Step 6: Configure the app	283
71. Step 7: Check that your app is a PWA	284
71.1. Layouts	286
Angular Material Layout	287

Let's begin	288
Adding Angular Material library to the project	289
Development	292
Conclusion	302
.1. NgRx.....	302
NgRx	303
72. The need for client side state management	304
73. Why NgRx?	305
74. Setup.....	306
75. Concept	307
Creating a Simple Store	308
76. Initializing NgRx	309
77. Create an entity model and initial state.....	311
78. Select the current watchlist	313
79. Dispatching an action to update watched minutes.....	316
79.1. Creating the action	316
79.2. Dispatch	316
79.3. State reduction.....	317
79.4. Alternative state mapping with immer	318
79.5. Redux devtools	319
80. NgRx Effects	320
80.1. Obtaining the recommendation list from the server	320
81. NgRx Entity	324
81.1. Cookbook	326
Abstract Class Store	327
82. Add Electron to an Angular application	330
82.1. Add Electron and other relevant dependencies.....	330
82.2. Add Electron build configuration	331
82.3. Create the necessary typescript configurations	332
82.4. Modify angular.json	333
82.5. Add Angular Electron directives	333
82.6. Add access Electron APIs	334
82.7. Create the electron window in main.ts	335
82.8. Add the electron window and improve the package.json scripts	337
83. devon4net documentation	340
83.1. Arquitecture basics	340
83.2. Coding conventions	348
83.3. Environment	352
83.4. User guide	353
OASP4NET Guide	354
.1. Cheat Sheet	363

.2. Console runner return codes	364
.3. Packages	374
Packages overview	375
The packages	376
Required software	394
.1. Templates	394
Templates	395
.1. Samples	395
Samples	396
84. devonfw shop floor documentation	406
85. cicdgen documentation	407
86. devonfw testing with MrChecker documentation	408
87. What is E2E Mr Checker Test Framework?	409
87.1. Home	409
88. How to install Mr Checker?	413
88.1. How to install	413
89. Mr Checker Test Framework modules	419
89.1. Core Test Module	419
89.2. Selenium Test Module	420
89.3. Security Test Module	422
89.4. Mobile Test Module	423
89.5. Standalone Test Module	423
89.6. DevOps Module	423
90. My Thai star documentation	438
90.1. Contributing Guide	438
90.2. Release Notes	449

Getting Started

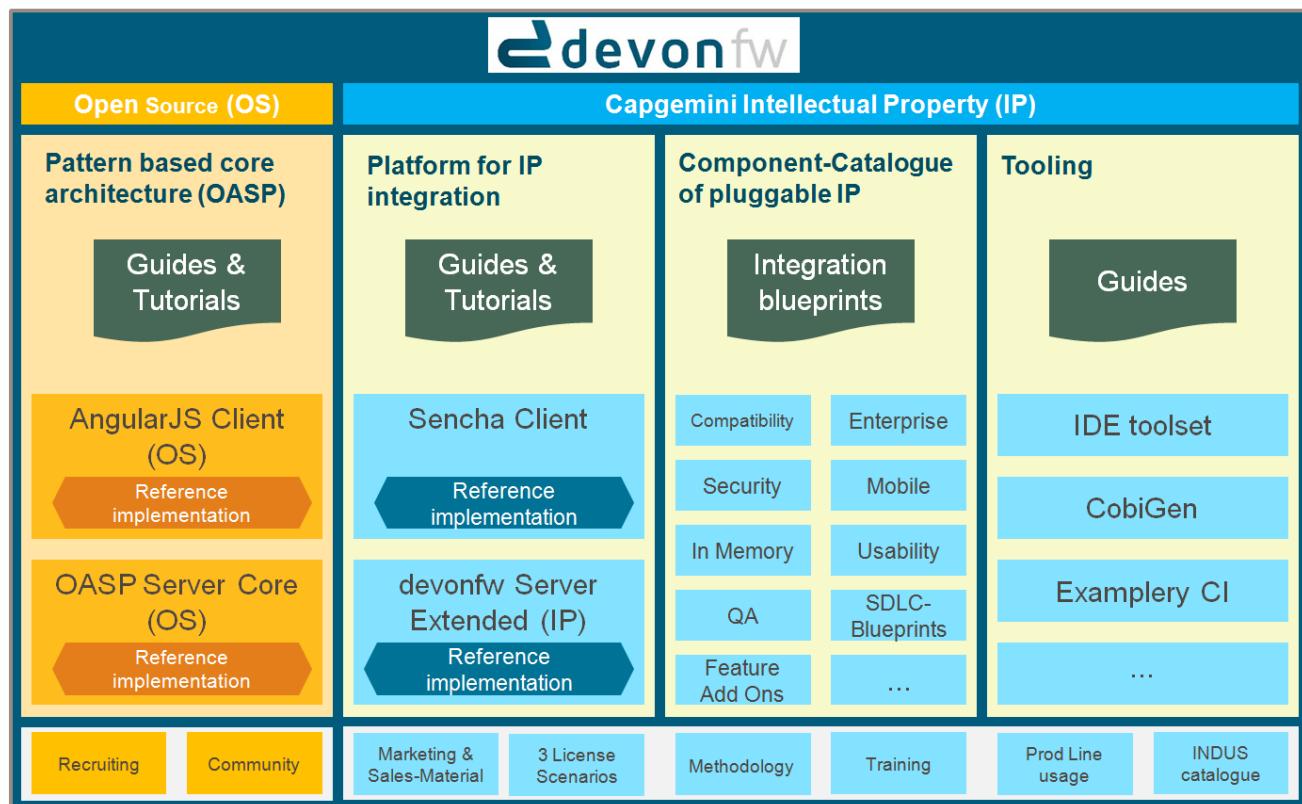
devonfw Introduction



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.

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.

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

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)

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.

Custom Tools

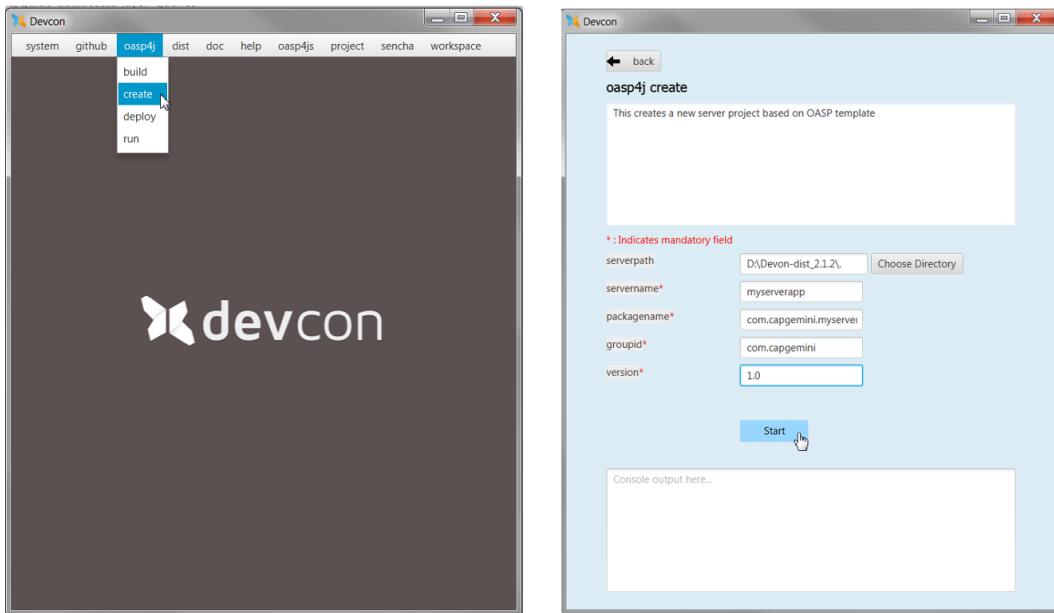
Pre-installed Software

- *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.

Devcon

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

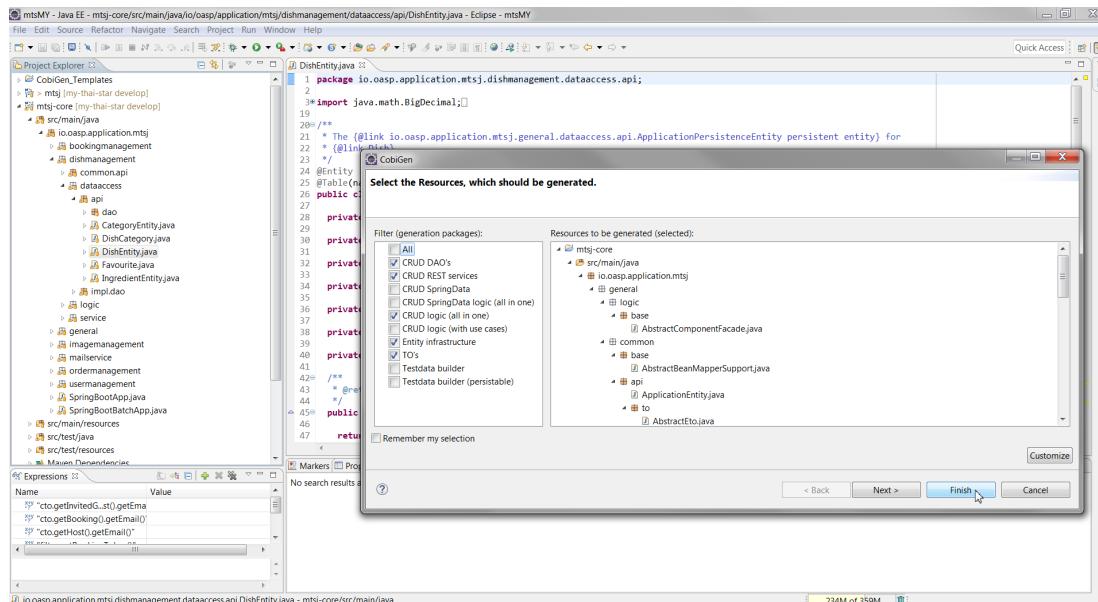
With *Devcon*, users can automate the creation of new projects (both server and client), build and run those and even, for server projects, deploy locally on Tomcat.



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.

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.



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](#).

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.

Objectives

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

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

Deliver & Improve Business Value



Efficiency

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

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.

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.

Features

Everything in a single zip

The devonfw distributions is packaged in a *zip* file that includes all the [Custom Tools](#), [Software](#) and configurations.

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

devonfw, the package

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](#)

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.

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

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



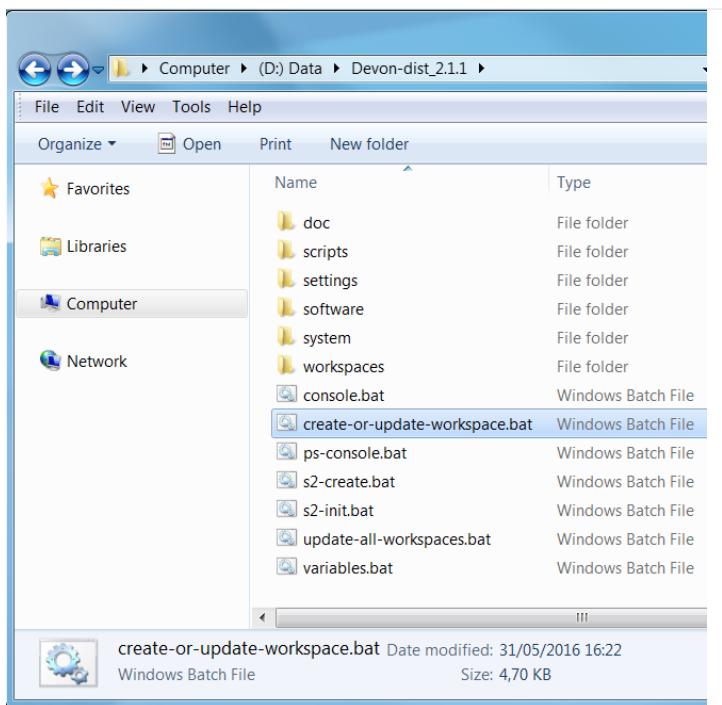
Using devcon



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](#).

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.

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>
</-->

```

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

Then update the settings to match your proxy configuration.

```

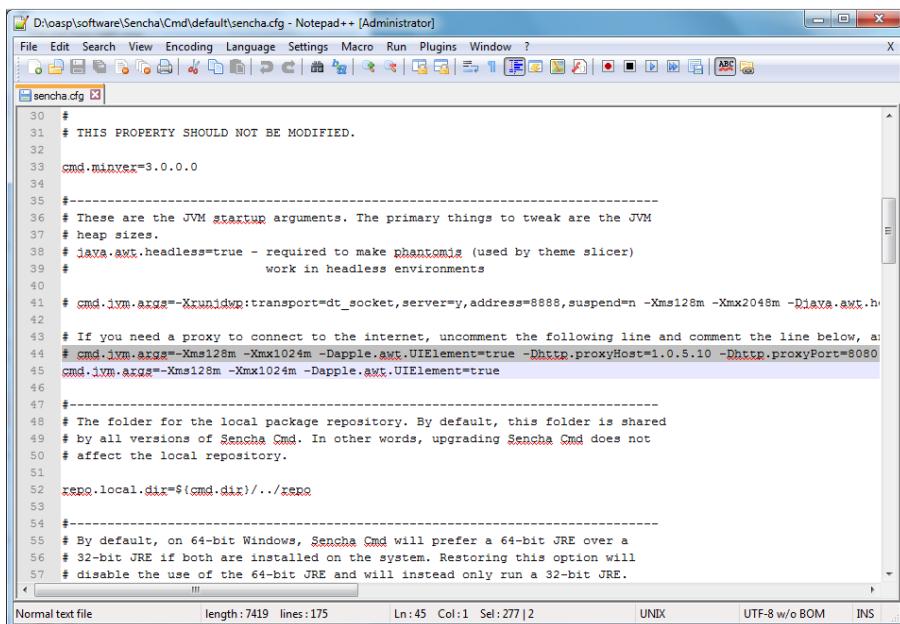
<?xml version="1.0" encoding="UTF-8"?>
<!-- $!ds -->
<settings>
  <!-- If you connect to the internet via a proxy, uncomment the following section and fill out
       host and port values. Delete username and password entries, if your proxy does not require
       authentication. -->
  <!-->
  <proxies>
    <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. -->

```

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, ai
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.

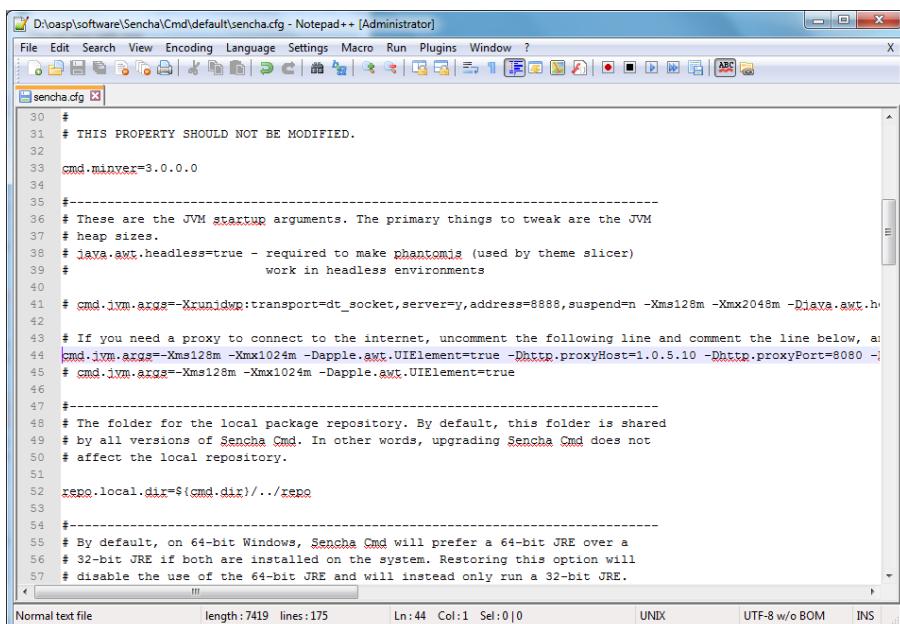
```

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, ai
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.

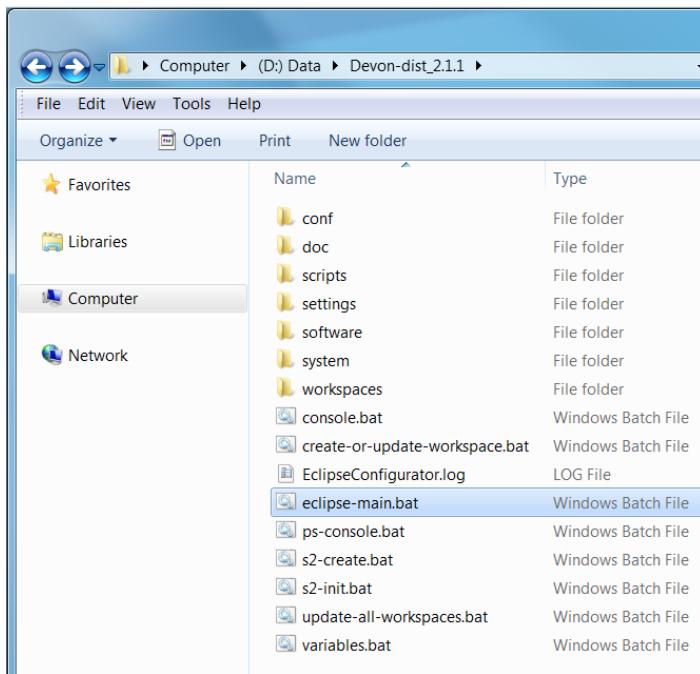
```

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

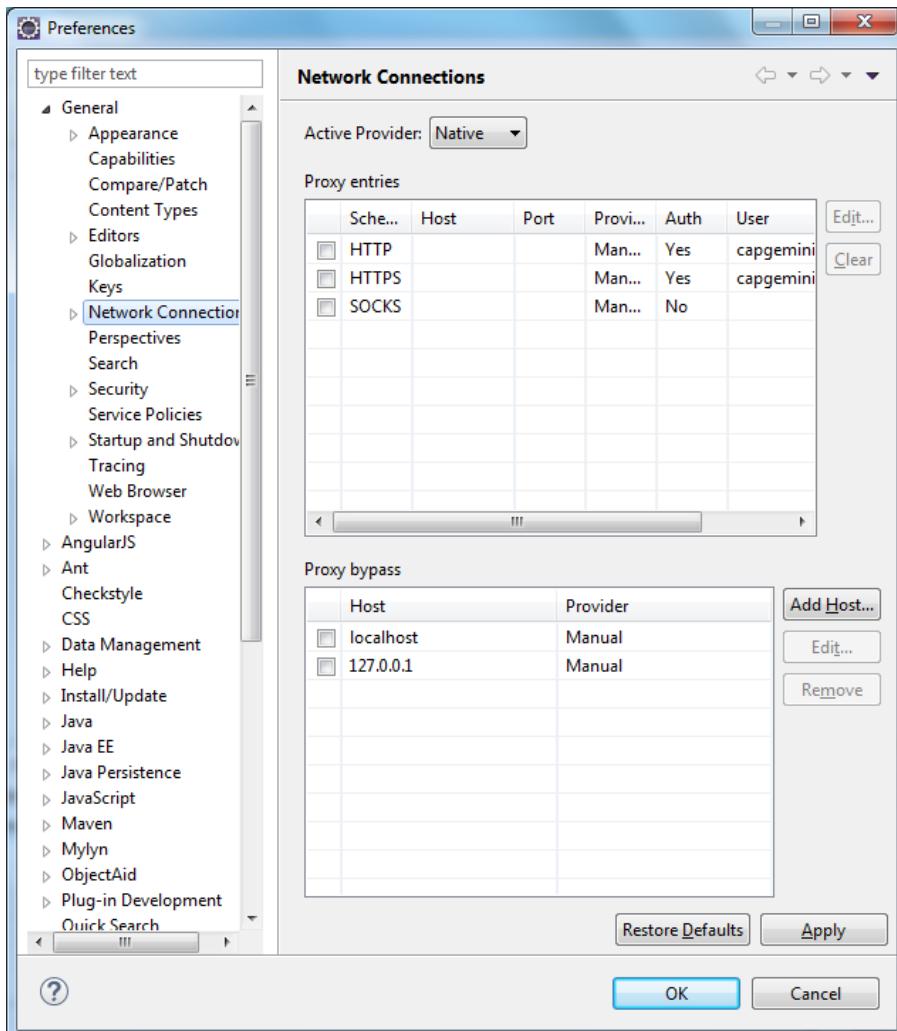
If your proxy does not require authentication, simply remove the "-Dhttp.proxyUser", "-DhttpProxyPassword", "-Dhttps.proxyUser" and "-Dhttps.proxyPassword" parameters.

Eclipse

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

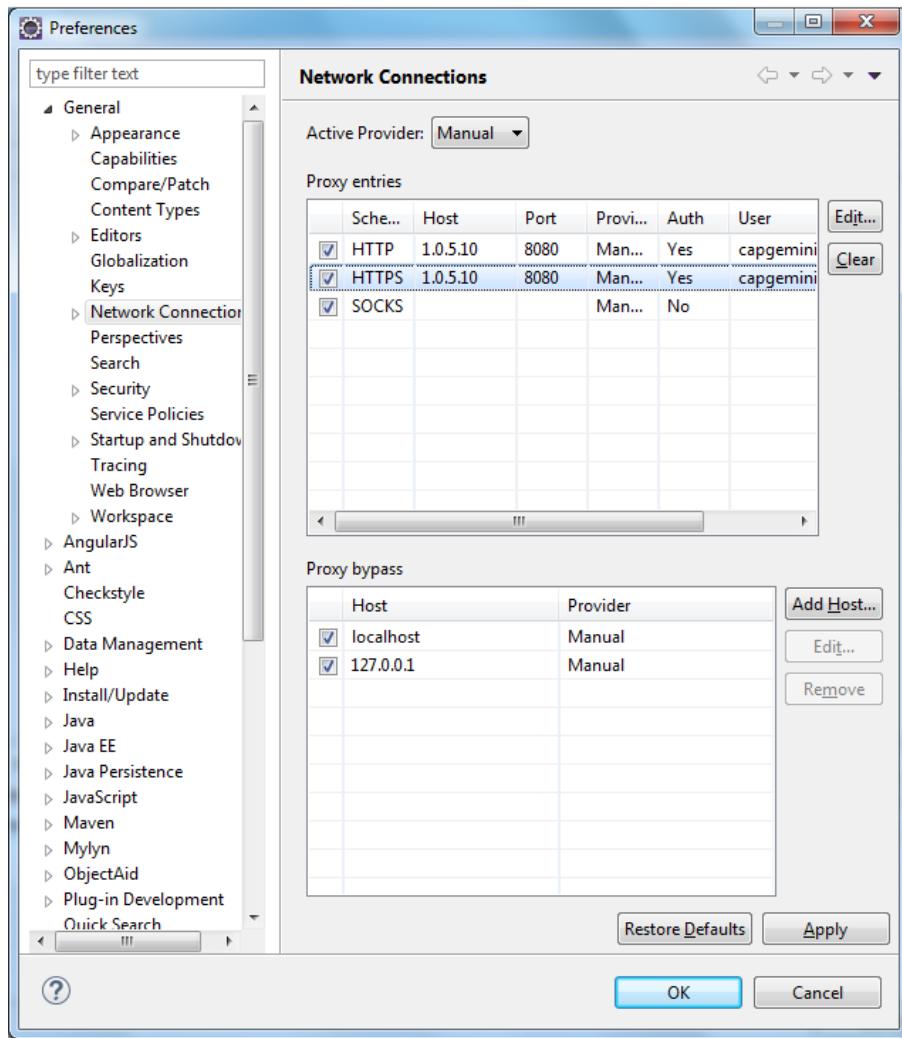


In the Eclipse preferences dialog, go to "General - Network Connection".



Switch from "Native" to "Manual"

Enter your proxy configuration



Thats All!!!

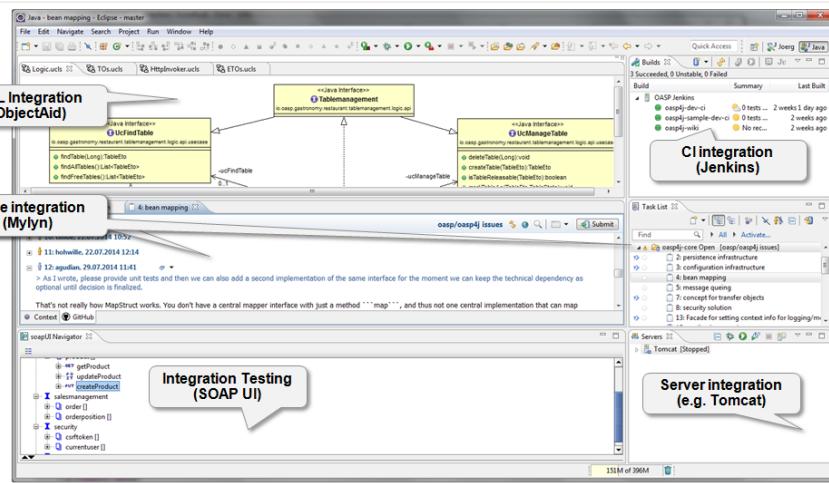
The Devon IDE

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.

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.



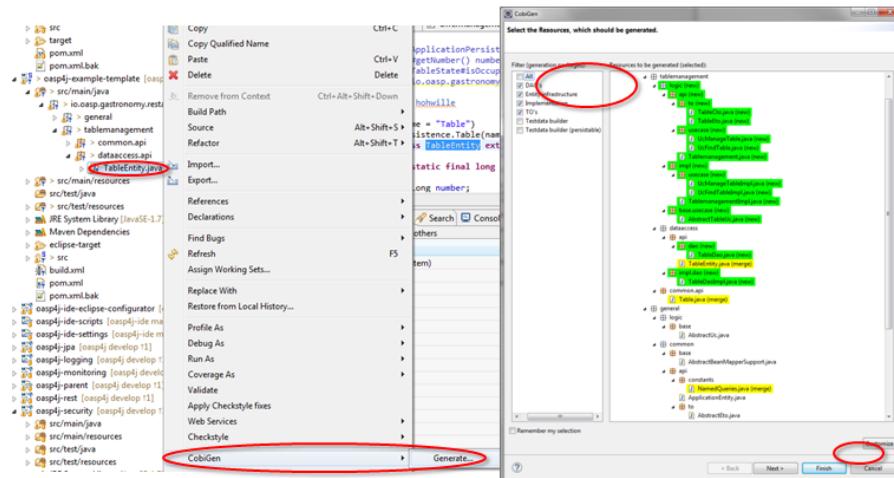
As with the remainder of the distribution, the advantage of this approach is that you can have as many instances of the -ide "installed" on your machine for different projects with different tools, tool versions and configurations. No physical installation and no tweaking of your operating system required.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

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.

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](#).

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.

CheckStyle

What is CheckStyle

[CheckStyle](#) is an 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

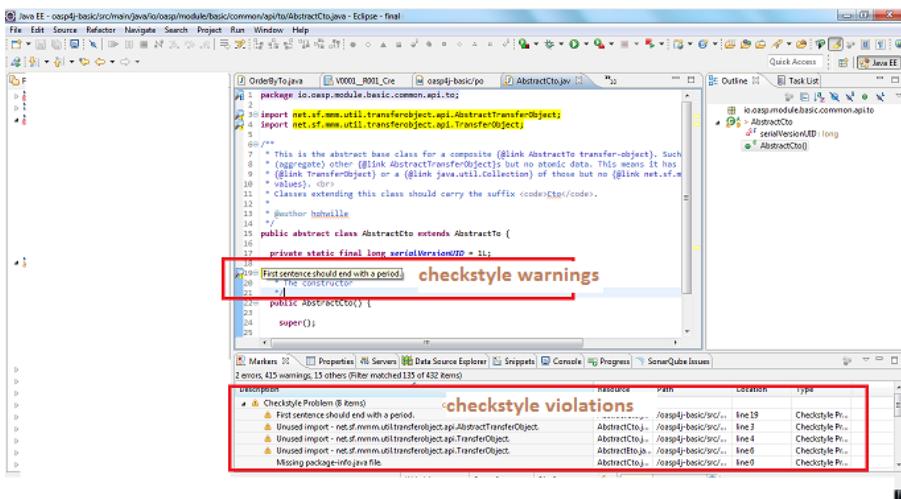
If your development team consists of more than one person, then obviously a common ground for coding standards (formatting rules, line lengths etc.) must be agreed upon - even if it is just for practical reasons to avoid superficial, format related merge conflicts. Checkstyle Plugin helps you define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

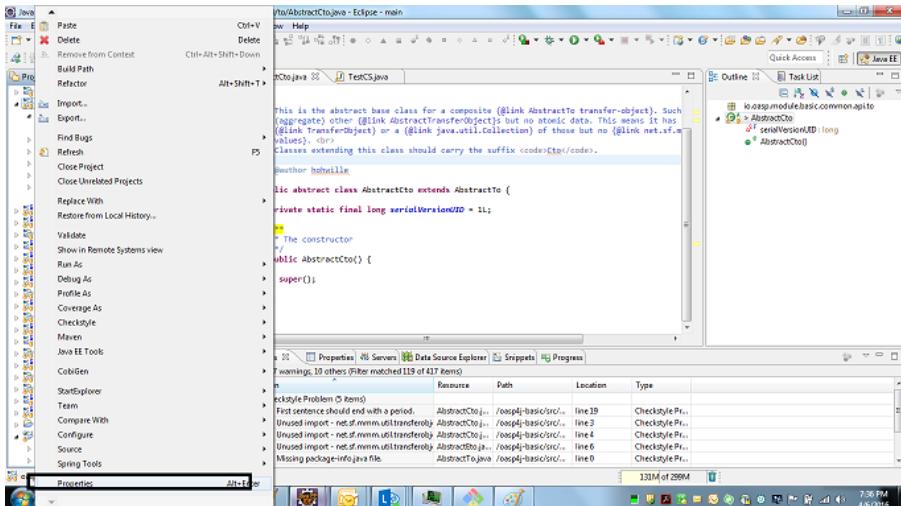
You can see violations in your workspace as shown in below figure.



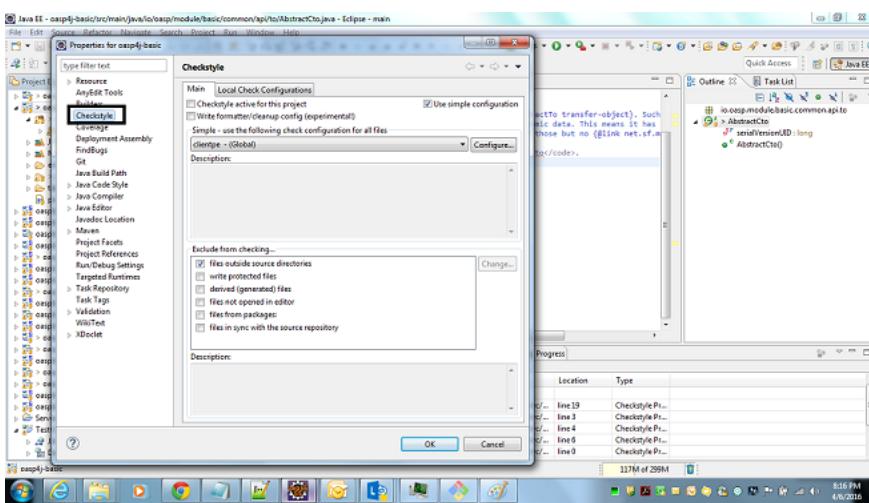
Usage

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

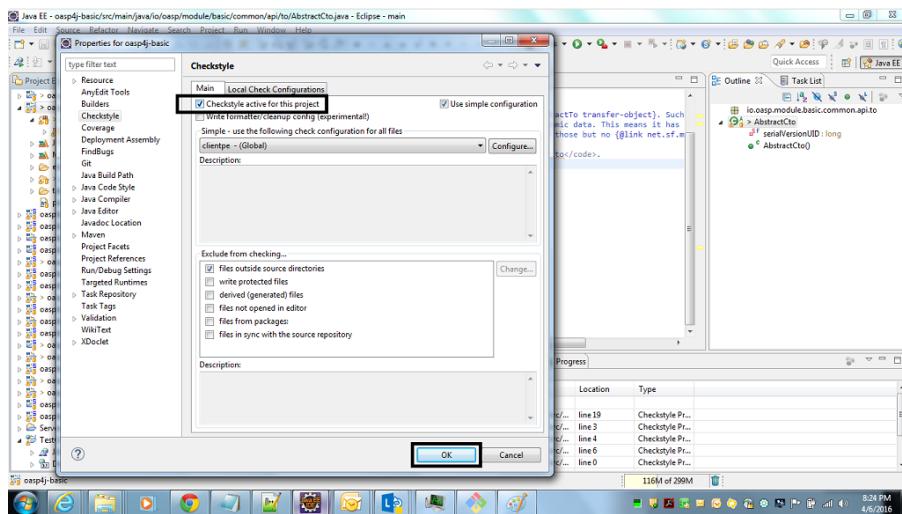
1. Open the properties of the project you want to get checked.



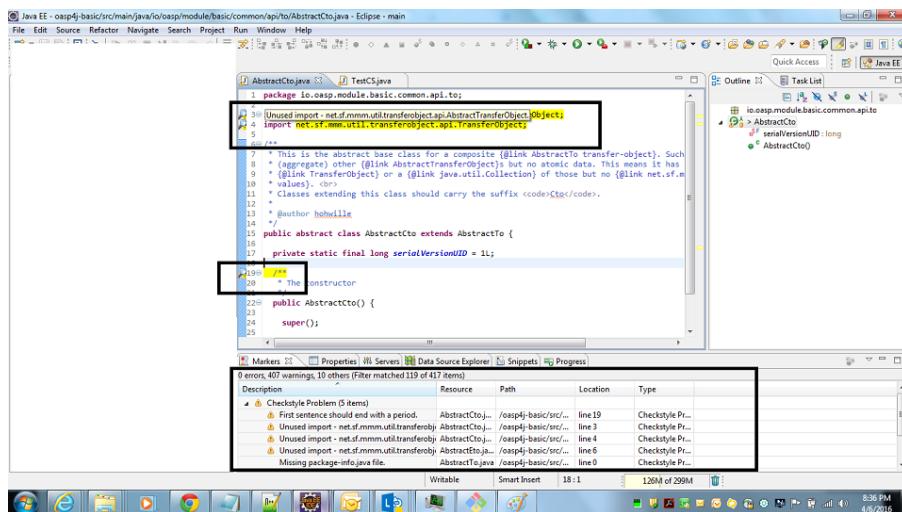
2. Select the Checkstyle section within the properties dialog .



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 your 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.

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

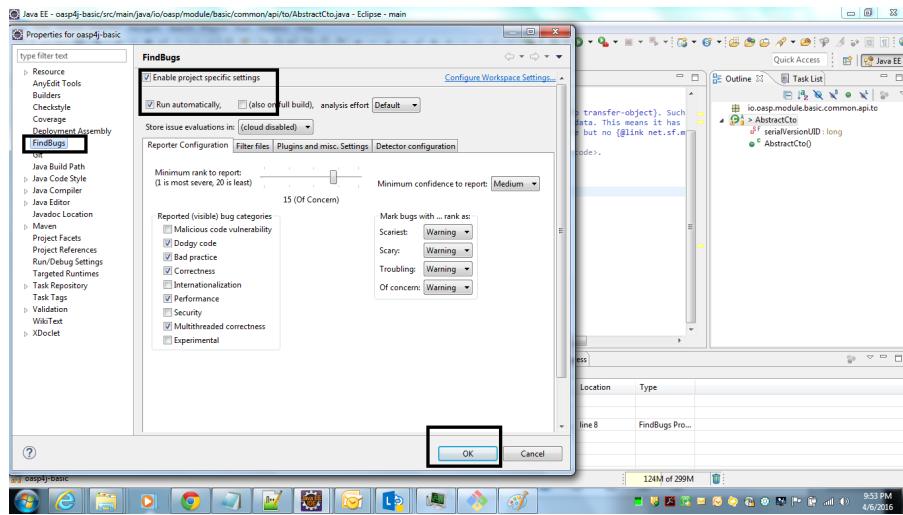
It scans your code for bugs, breaking down the list of bugs in your code into a ranked list on a 20-point scale. The lower the number, the more hardcore the bug. This helps the developer to access

these problems early in the development phase.

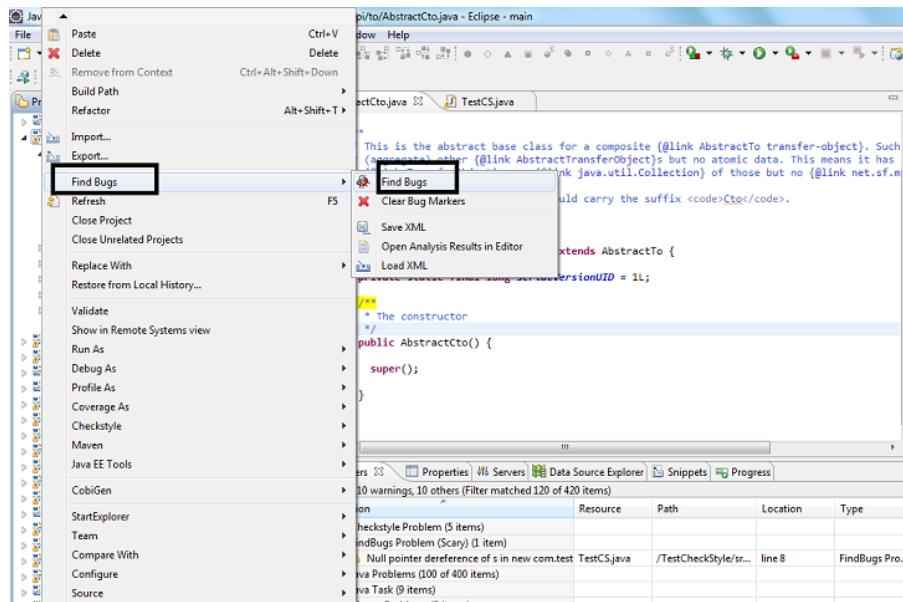
Installation and Usage of FindBugs

IDE comes preinstalled with FindBugs plugin.

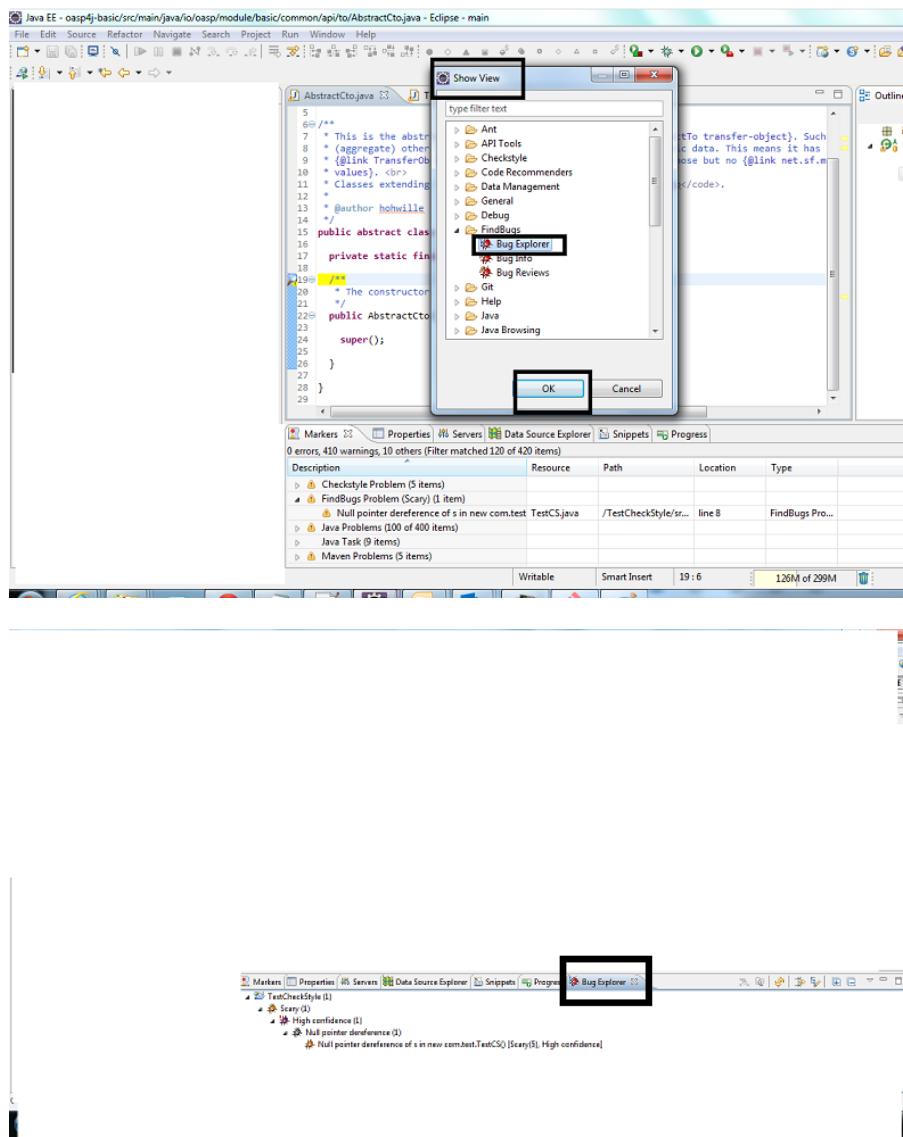
You can configure that FindBugs should run automatically for a selected project. For this right-click on a project and select Properties from the popup menu. via the project properties. Select FindBugs → Run automatically as shown below.



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.



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.

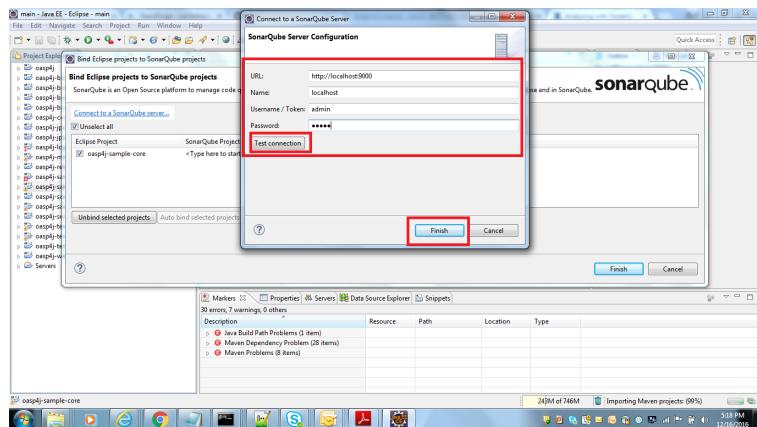
Installation and usage of SonarLint

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.

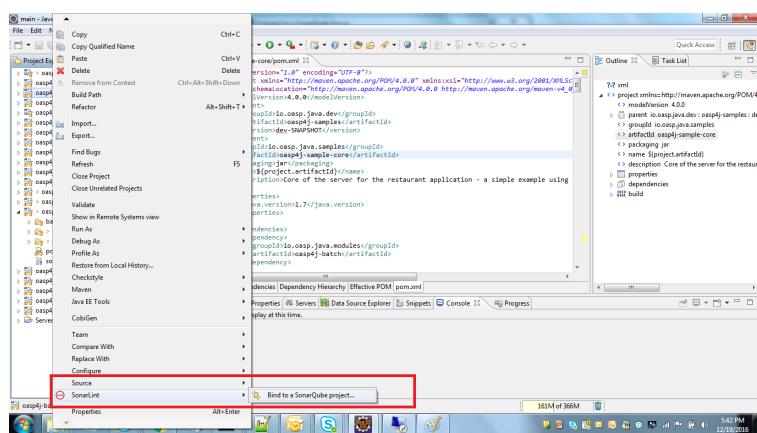
If your project is not already under analysis, you'll need to declare it through the SonarQube web interface as described [here](#). Once your project exists in SonarQube, you're ready to get started with SonarQube in Eclipse.

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.

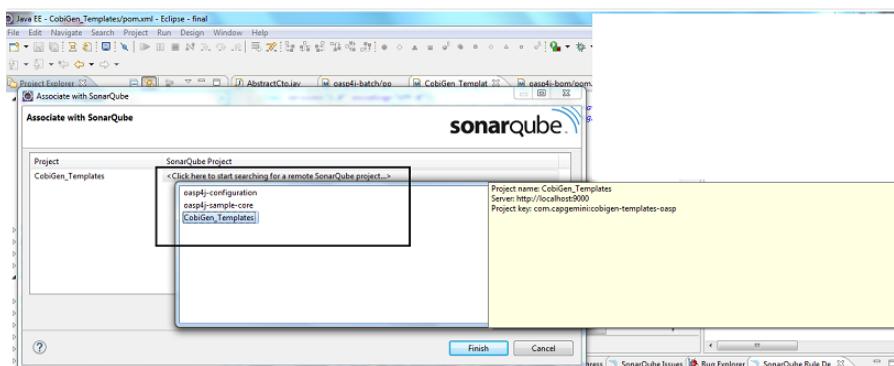


For getting a project analysed on sonar, refer this <http://docs.sonarqube.org/display/SONAR/Analyzing+Source+Code> [link].

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:

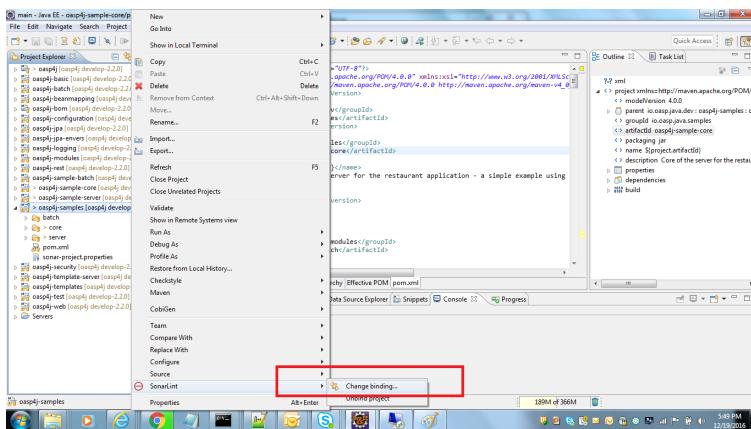


Click on Finish. Your project is now associated to one analyzed on your SonarQube server.

Changing Binding

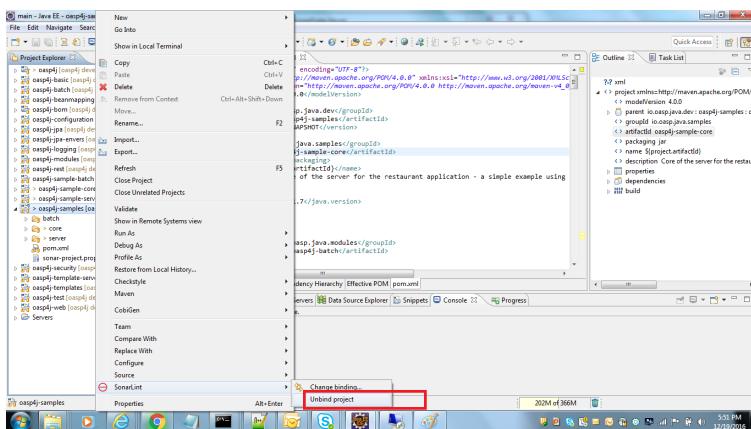
At any time, it is possible to change the project association.

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



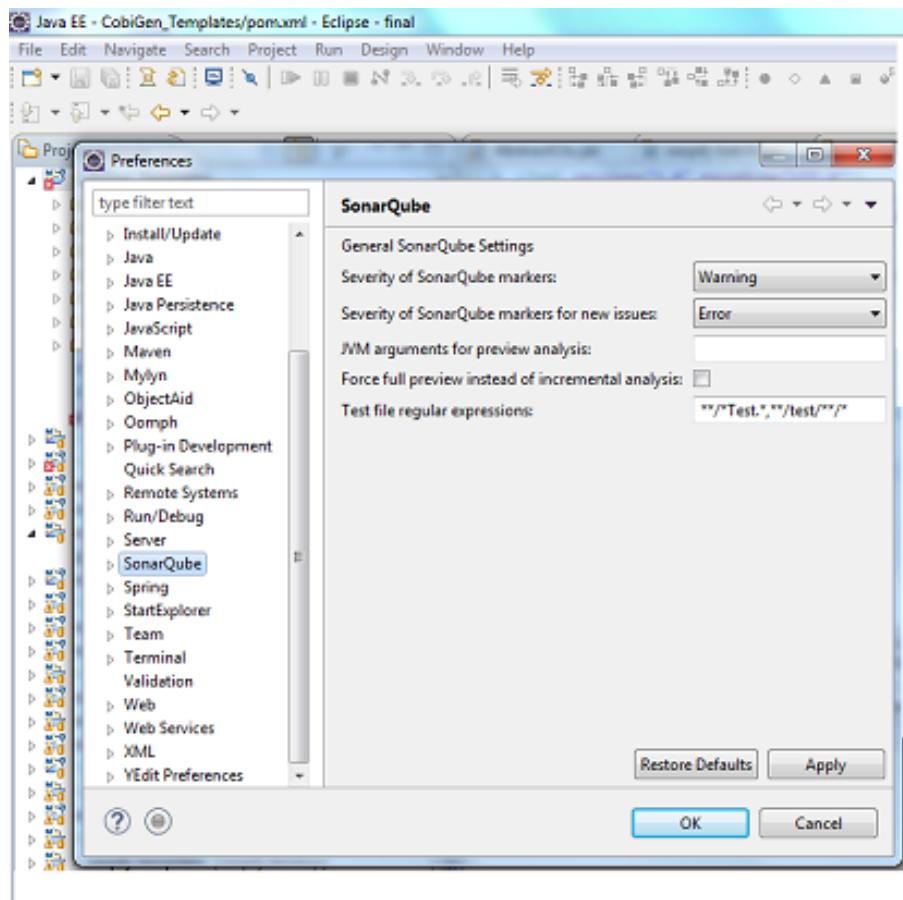
Unbinding a Project

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

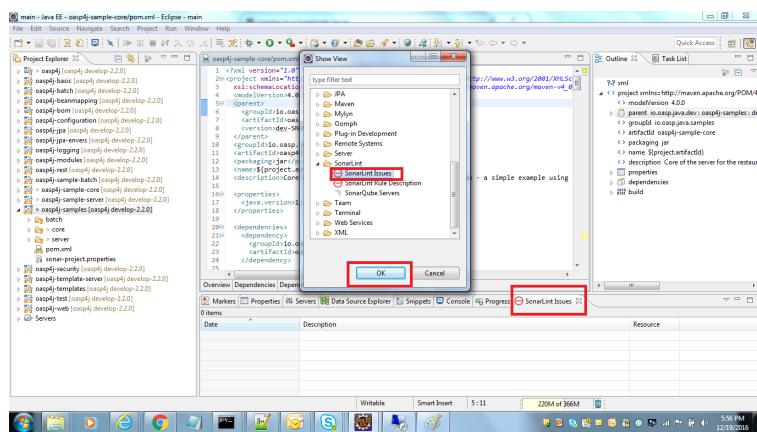


Advanced Configuration

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.

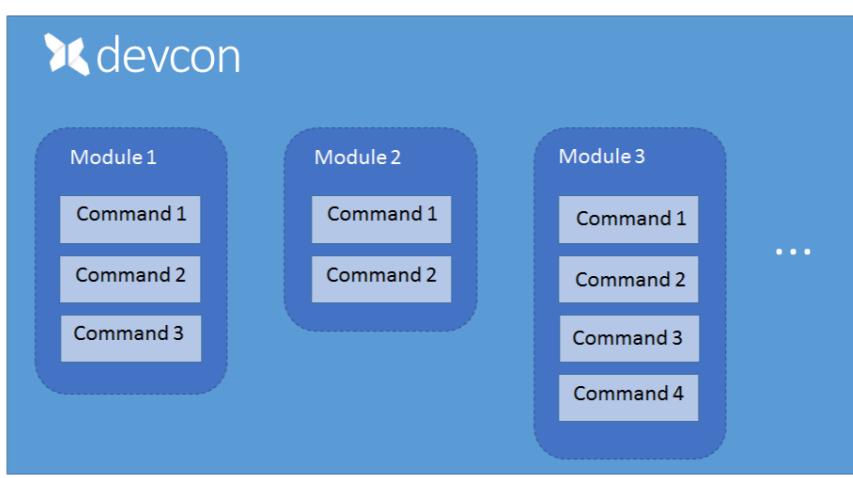
Devcon

Devcon Command Developer's guide

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*.



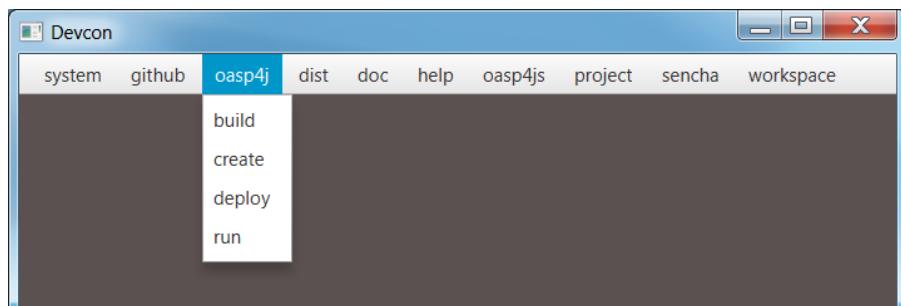
where each module represents an area of Devon and groups commands that are related to some specific task of that area.

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.

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.

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 de 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.).

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 direcly 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*).



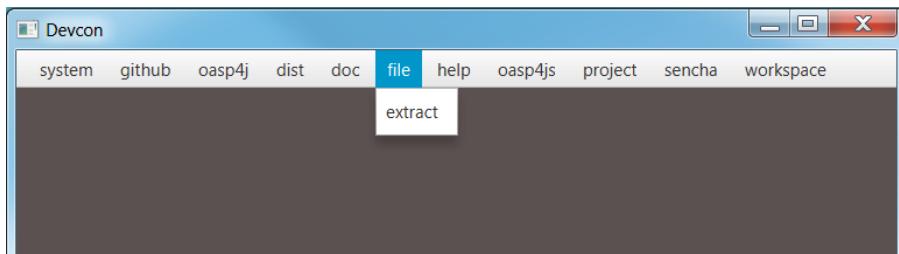
If you don't have Devcon installed you can see how to install it [here](#)

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

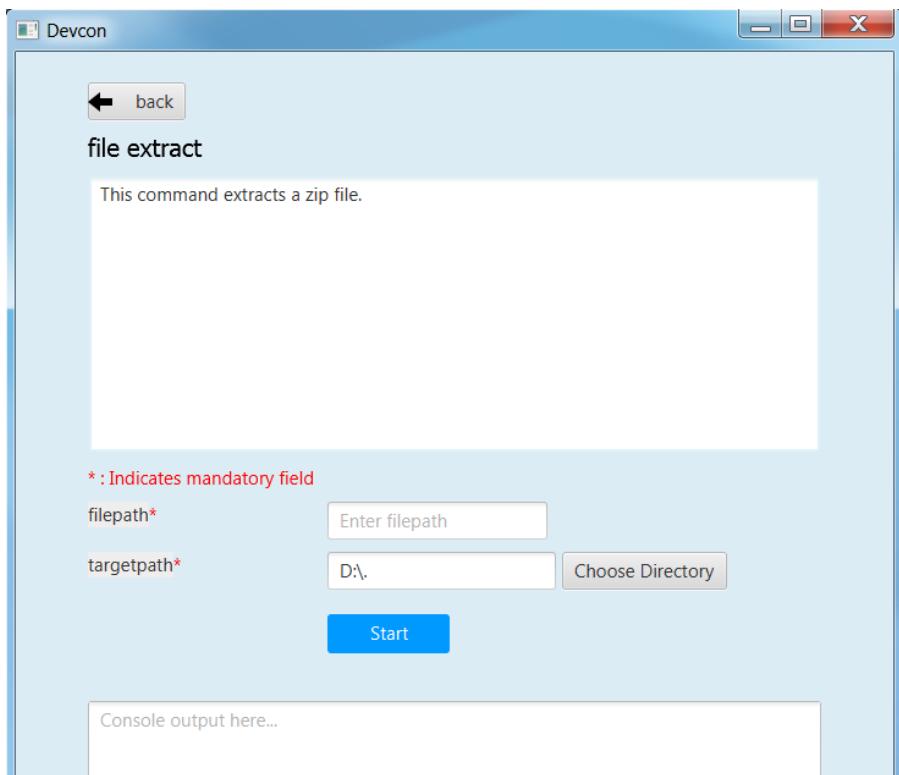
distributions you only need to copy the `devcon.jar` you just created, to `Devon-distribution\software\devcon` replacing the `devcon.jar` that is inside of that directory with your new `devcon.jar`

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`

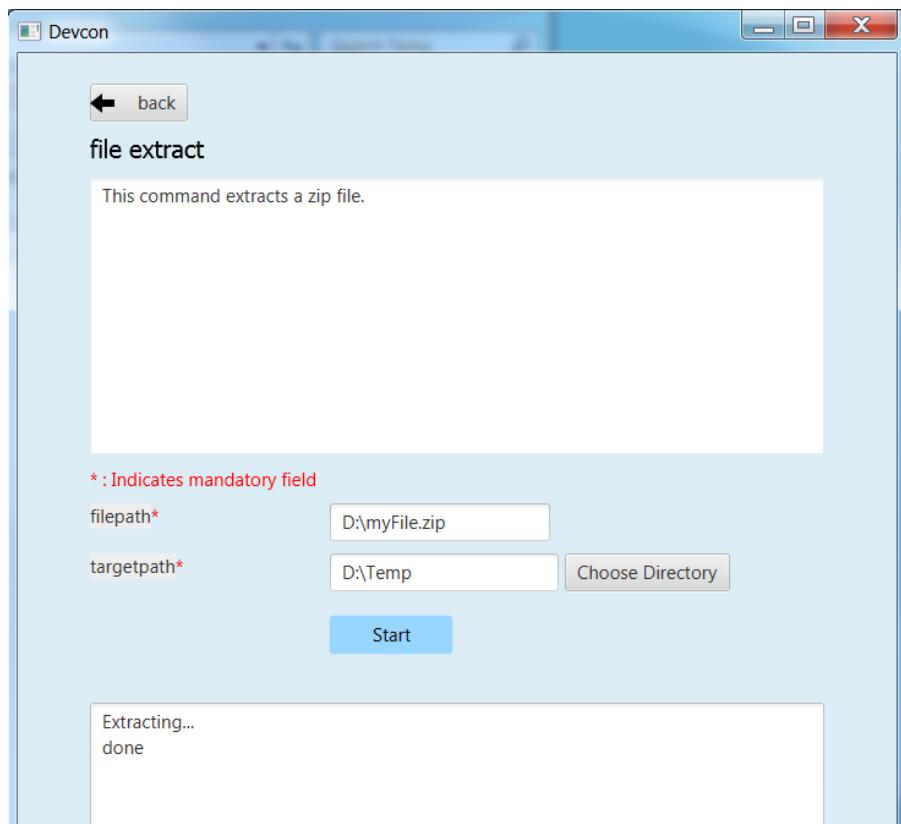
Using our module and command

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.

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

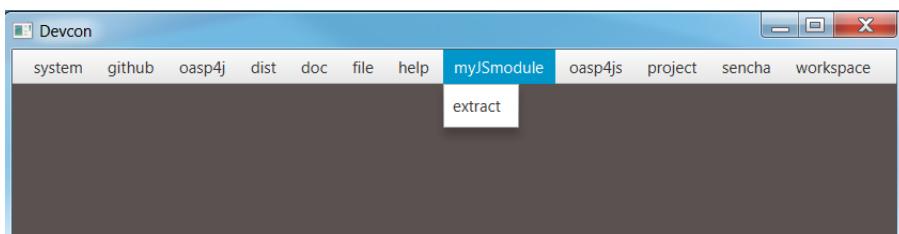
We have finished the implementation of the new Javascript module so now we can start using it.

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).

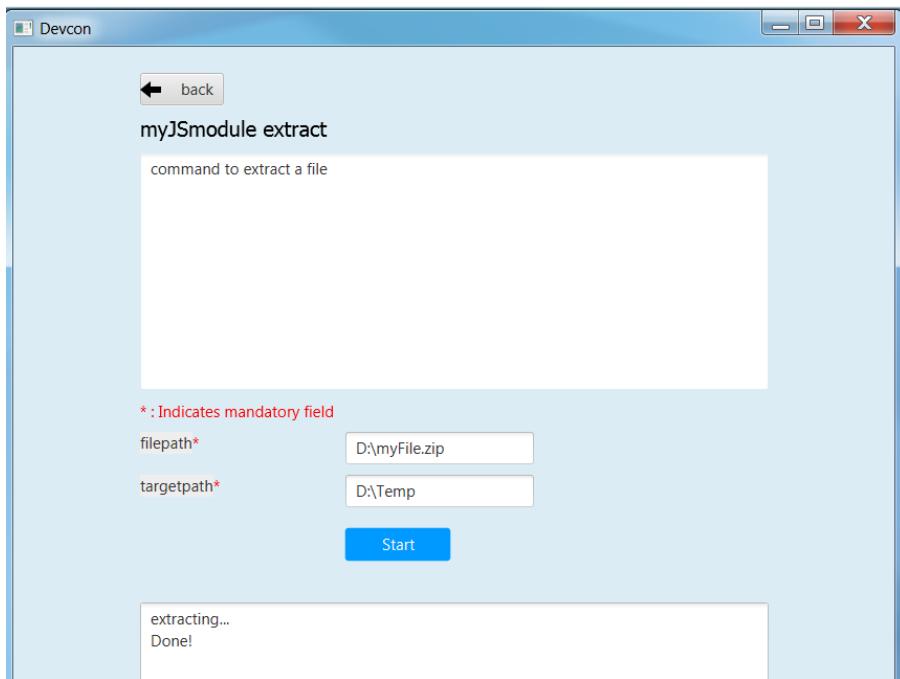
using the gui

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 myJModule 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!
```

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.

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`

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.

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.

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).

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.
- **ciaas:** 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 *ciaas* parameter

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

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.

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.

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.

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
```

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 devon

```
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
```

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

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**.

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.

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.

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
```

[...]

The Spring Boot logo is a complex, abstract graphic composed of various symbols like slashes, parentheses, and dots, arranged in a grid-like pattern. It features a central vertical axis with horizontal bars extending from it. The overall shape is roughly rectangular and has a digital, futuristic feel.

```
2016-07-01 11:13:59.006 INFO 6116 --- [           main] i.d.application.MyApp
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.MyApp
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
```

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
```

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");
```

devon4ng

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

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.

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
```

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>.

project

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

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 paremeters 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.
- **createsencha:** 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] -----  
[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.

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'.

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
```

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 accessible.



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
```

project run

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

project run requirements

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

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).

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.

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.

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
```

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
```

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)

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:\>
```

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.

workspace

This module handles all tasks related to distribution workspaces.

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.

system

This module contains system wide commands related to devcon.

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
```

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
```

devon4j documentation

Unresolved directive in master.asciidoc - include::devon4j.wiki/master-devon4j.asciidoc[]

devon4ng documentation

Arquitecture

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

Overview

When using Angular the web client architecture is driven by the framework in a certain way. Google and the Angular community think about web client architecture. Angular gives an opinion on how to look at architecture. It is component based like devon4j but uses different terms which are common language in web application development. The important term is *module* which is used instead of *component*. The primary reason is the naming collision with the *Web Components* standard (see [Web Components](#)).

To clarify this:

- A *component* describes an UI element containing HTML, CSS and JavaScript - structure, design and logic encapsulated inside a reusable container called component.
- A *module* describes an applications feature area. The application flight-app may have a module called booking.

An application developed using Angular consists of multiple modules. There are feature modules and special modules described by the Angular Styleguide - [core](#) and [shared](#). Angular or Angular Styleguide give no guidance on how to structure a module internally. This is where this architecture comes in.

Layers

The architecture describes two layers. The terminology is based on common language in web development.

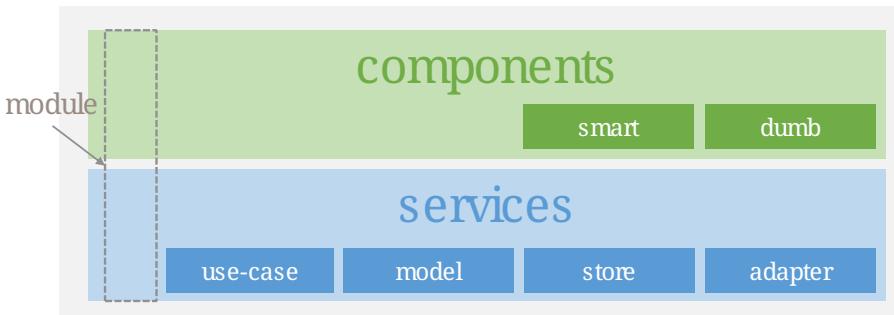


Figure 1. Layers

- **Components Layer** encapsulates components which present the current application state. Components are separated into *Smart* and *Dumb Components*. The only logic present is view logic inside *Smart Components*.
- **Services Layer** is more or less what we call 'business logic layer' on the server side. The layer defines the applications state, the transitions between state and classic business logic. Stores contain application state over time to which *Smart Components* subscribe to. Adapters are used to perform XHRs, WebSocket connections, etc. The business model is described inside the module. Use case services perform business logic needed for use cases. A use case services interacts with the store and adapters. Methods of use case services are the API for *Smart Components*. Those methods are *Actions* in reactive terminology.

Modules

Angular requires a module called *app* which is the main entrance to an application at runtime - this module gets bootstrapped. Angular Styleguide defines feature modules and two special modules - *core* and *shared*.

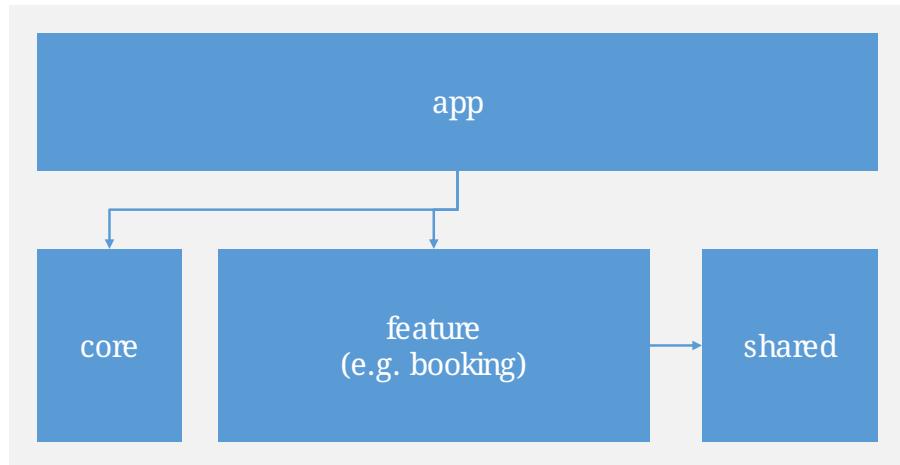


Figure 2. Modules

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.

Meta arquitecture

Meta Architecture

Introduction

Purpose of this document

In our business applications, the client easily gets underestimated. Sometimes the client is more complex to develop and design than the server. While the server architecture is nowadays easily to agree as common sense, for clients this is not as obvious and stable especially as it typically depends on the client framework used. Finding a concrete architecture applicable for all clients may therefore be difficult to accomplish.

This document tries to define on a high abstract level, a reference architecture which is supposed to be a mental image and frame for orientation regarding the evaluation and appliance of different client frameworks. As such it defines terms and concepts required to be provided for in any framework and thus gives a common ground of understanding for those acquainted with the reference architecture. This allows better comparison between the various frameworks out there, each having their own terms for essentially the same concepts. It also means that for each framework we need to explicitly map how it implements the concepts defined in this document.

The architecture proposed herein is neither new nor was it developed from scratch. Instead it is the gathered and consolidated knowledge and best practices of various projects (s. References).

Goal of the Client Architecture

The goal of the client architecture is to support the non-functional requirements for the client, i.e. mostly maintainability, scalability, efficiency and portability. As such it provides a component-oriented architecture following the same principles listed already in the devonfw architecture overview. Furthermore it ensures a homogeneity regarding how different concrete UI technologies are being applied in the projects, solving the common requirements in the same way.

Architecture Views

As for the server we distinguish between the business and the technical architecture. Where the business architecture is different from project to project and relates to the concrete design of dialog components given concrete requirements, the technical architecture can be applied to multiple projects.

The focus of this document is to provide a technical reference architecture on the client on a very abstract level defining required layers and components. How the architecture is implemented has to be defined for each UI technology.

The technical infrastructure architecture is out of scope for this document and although it needs to be considered, the concepts of the reference architecture should work across multiple TI architecture, i.e. native or web clients.

devonfw Reference Client Architecture

The following gives a complete overview of the proposed reference architecture. It will be built up incrementally in the following sections.

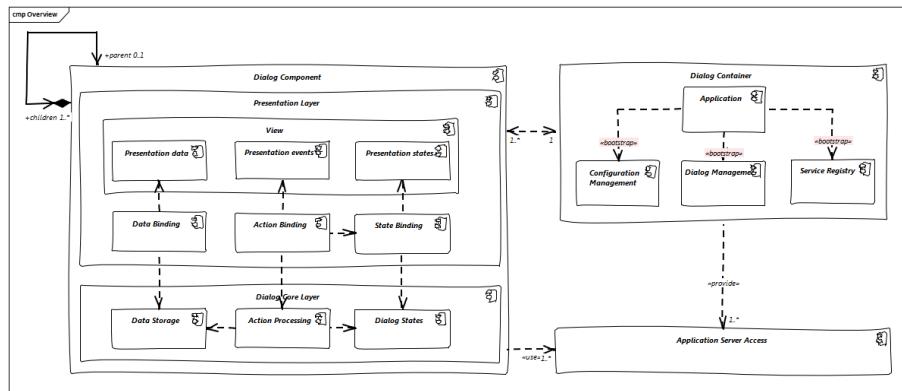


Figure 1 Overview

Client Architecture

On the highest level of abstraction we see the need to differentiate between dialog components and their container they are managed in, as well as the access to the application server being the backend for the client (e.g. an devon4j instance). This section gives a summary of these components and how they relate to each other. Detailed architectures for each component will be supplied in subsequent sections

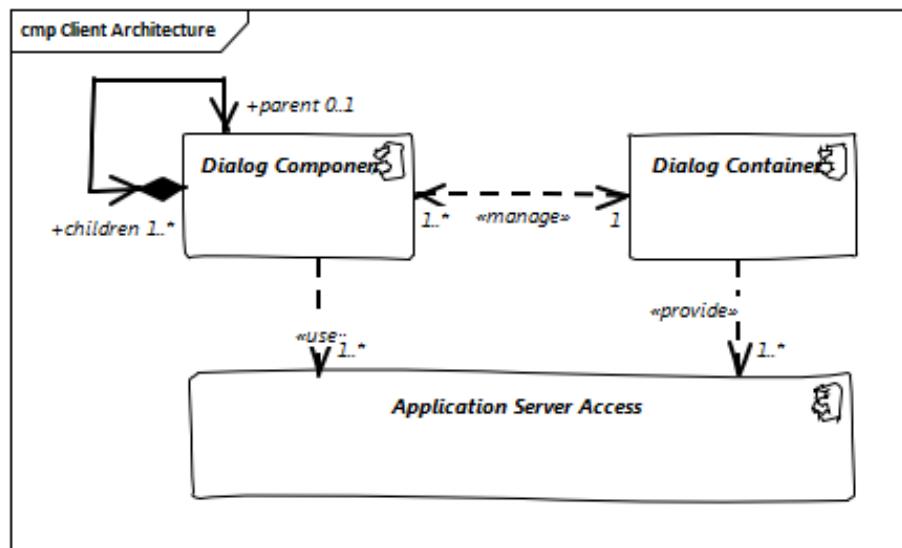


Figure 2 Overview of Client Architecture

Dialog Component

A dialog component is a logical, self-contained part of the user interface. It accepts user input and actions and controls communication with the user. Dialog components use the services provided by the dialog container in order to execute the business logic. They are self-contained, i.e. they possess their own user interface together with the associated logic, data and states.

- Dialog components can be composed of other dialog components forming a hierarchy
- Dialog components can interact with each other. This includes communication of a parent to its children, but also between components independent of each other regarding the hierarchy.

Dialog Container

Dialog components need to be managed in their lifecycle and how they can be coupled to each other. The dialog container is responsible for this along with the following:

- Bootstrapping the client application and environment
 - Configuration of the client
 - Initialization of the application server access component
- Dialog Component Management
 - Controlling the lifecycle
 - Controlling the dialog flow
 - Providing means of interaction between the dialogs
 - Providing application server access
 - Providing services to the dialog components
(e.g. printing, caching, data storage)
- Shutdown of the application

Application Server Access

Dialogs will require a backend application server in order to execute their business logic. Typically in an devonfw application the service layer will provide interfaces for the functionality exposed to the client. These business oriented interfaces should also be present on the client backed by a proxy handling the concrete call of the server over the network. This component provides the set of interfaces as well as the proxy.

Dialog Container Architecture

The dialog container can be further structured into the following components with their respective tasks described in own sections:

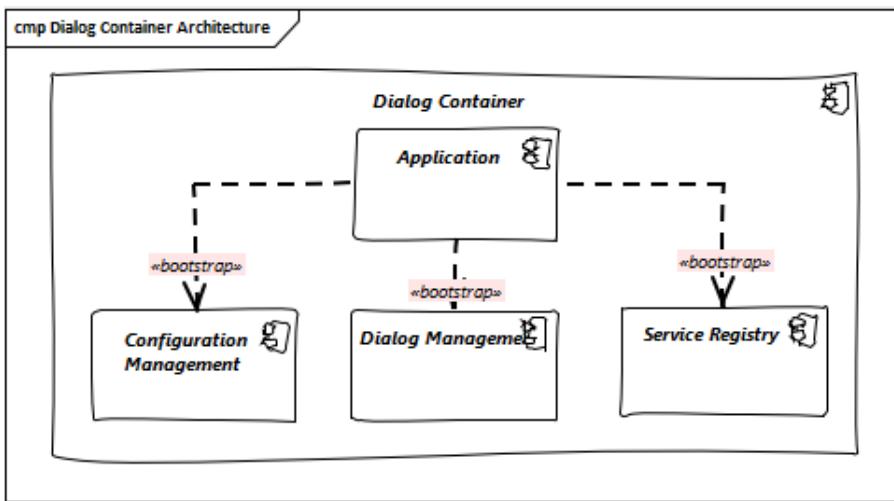


Figure 3 Dialog Container Architecture

Application

The application component represents the overall client in our architecture. It is responsible for bootstrapping all other components and connecting them with each other. As such it initializes the components below and provides an environment for them to work in.

Configuration Management

The configuration management manages the configuration of the client, so the client can be deployed in different environments. This includes configuration of the concrete application server to be called or any other environment-specific property.

Dialog Management

The Dialog Management component provides the means to define, create and destroy dialog components. It therefore offers basic lifecycle capabilities for a component. In addition it also allows composition of dialog components in a hierarchy. The lifecycle is then managed along the hierarchy, meaning when creating/destroying a parent dialog, this affects all child components, which are created/destroyed as well.

Service Registry

Apart from dialog components, a client application also consists of services offered to these. A service can thereby encompass among others:

- Access to the application server
- Access to the dialog container functions for managing dialogs or accessing the configuration
- Dialog independent client functionality such as Printing, Caching, Logging, Encapsulated business logic such as tax calculation
- Dialog component interaction

The service registry offers the possibility to define, register and lookup these services. Note that these services could be dependent on the dialog hierarchy, meaning different child instances could

obtain different instances / implementations of a service via the service registry, depending on which service implementations are registered by the parents.

Services should be defined as interfaces allowing for different implementations and thus loose coupling.

Dialog Component Architecture

A dialog component has to support all or a subset of the following tasks:

- (T1) Displaying the user interface incl. internationalization
- (T2) Displaying business data incl. changes made to the data due to user interactions and localization of the data
- (T3) Accepting user input including possible conversion from e.g. entered Text to an Integer
- (T4) Displaying the dialog state
- (T5) Validation of user input
- (T6) Managing the business data incl. business logic altering it due to user interactions
- (T7) Execution of user interactions
- (T8) Managing the state of the dialog (e.g. Edit vs. View)
- (T9) Calling the application server in the course of user interactions

Following the principle of separation of concerns, we further structure a dialog component in an own architecture allowing us the distribute responsibility for these tasks along the defined components:

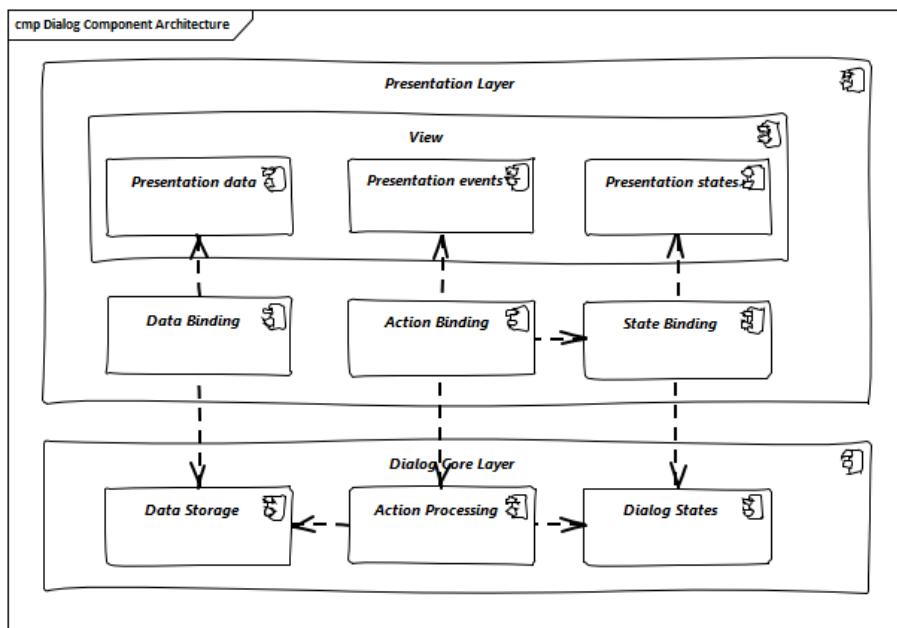


Figure 4 Overview of dialog component architecture

Presentation Layer

The presentation layer generates and displays the user interface, accepts user input and user actions and binds these to the dialog core layer (T1-5). The tasks of the presentation layer fall into two categories:

- Provision of the visual representation (View component)

The presentation layer generates and displays the user interface and accepts user input and user actions. The logical processing of the data, actions and states is performed in the dialog core layer. The data and user interface are displayed in localized and internationalized form.

- **Binding of the visual representation to the dialog core layer**

The presentation layer itself does not contain any dialog logic. The data or actions entered by the user are then processed in the dialog core layer. There are three aspects to the binding to the dialog core layer. We refer to “data binding”, “state binding” and “action binding”. Syntactical and (to a certain extent) semantic validations are performed during data binding (e.g. cross-field plausibility checks). Furthermore, the formatted, localized data in the presentation layer is converted into the presentation-independent, neutral data in the dialog core layer (parsing) and vice versa (formatting).

Dialog Core Layer

The dialog core layer contains the business logic, the control logic, and the logical state of the dialog. It therefore covers tasks T5-9:

- **Maintenance of the logical dialog state and the logical data**

The dialog core layer maintains the logical dialog state and the logical data in a form which is independent of the presentation. The states of the presentation (e.g. individual widgets) must not be maintained in the dialog core layer, e.g. the view state could lead to multiple presentation states disabling all editable widgets on the view.

- **Implementation of the dialog and dialog control logic**

The component parts in the dialog core layer implement the client specific business logic and the dialog control logic. This includes, for example, the manipulation of dialog data and dialog states as well as the opening and closing of dialogs.

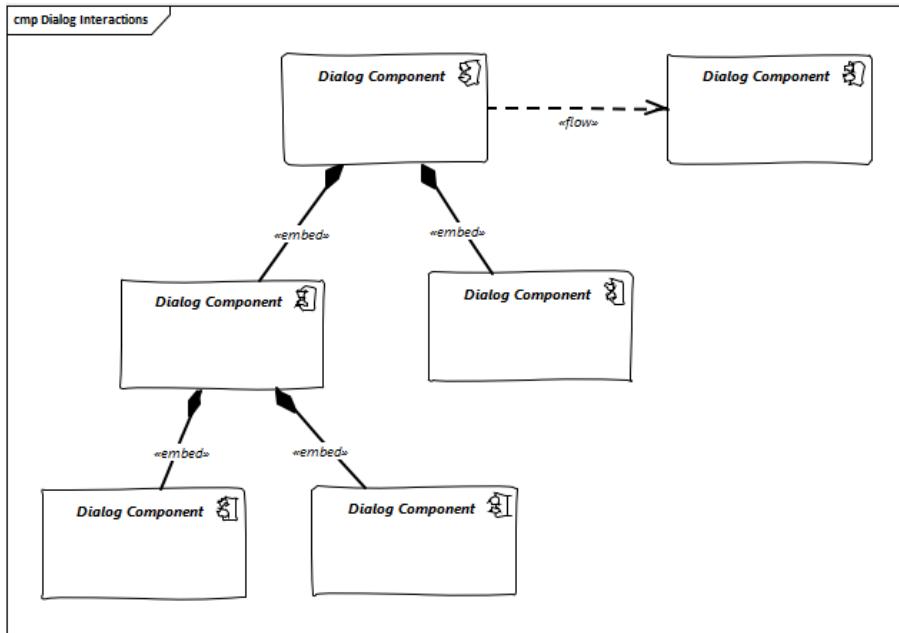
- **Communication with the application server**

The dialog core layer calls the interfaces of the application server via the application server access component services.

The dialog core layer should not depend on the presentation layer enforcing a strict layering and thus minimizing dependencies.

Interactions between dialog components

Dialog components can interact in the following ways:



- **Embedding of dialog components**

As already said dialog components can be hierarchically composed. This composition works by embedding one dialog component within the other. Apart from the lifecycle managed by the dialog container, the embedding needs to cope for the visual embedding of the presentation and core layer.

- **Embedding dialog presentation**

The parent dialog needs to either integrate the embedded dialog in its layout or open it in an own model window.

- **Embedding dialog core**

The parent dialog needs to be able to access the embedded instance of its children. This allows initializing and changing their data and states. On the other hand the children might require context information offered by the parent dialog by registering services in the hierarchical service registry.

- **Dialog flow**

Apart from the embedding of dialog components representing a tight coupling, dialogs can interact with each other by passing the control of the UI, i.e. switching from one dialog to another.

When interacting, dialog components should interact only between the same or lower layers, i.e. the dialog core should not access the presentation layer of another dialog component.

Appendix

Notes about Quasar Client

The Quasar client architecture as the consolidated knowledge of our CSD projects is the major source for the above drafted architecture. However, the above is a much simplified and more agile version thereof:

- Quasar Client tried to abstract from the concrete UI library being used, so it could decouple the business from the technical logic of a dialog. The presentation layer should be the only one knowing the concrete UI framework used. This level of abstraction was dropped in this reference architecture, although it might of course still make sense in some projects. For fast-moving agile projects in the web however introducing such a level of abstraction takes effort with little gained benefits. With frameworks like Angular 2 we would even introduce one additional seemingly artificial and redundant layer, since it already separates the dialog core from its presentation.
- In the past and in the days of Struts, JSF, etc. the concept of session handling was important for the client since part of the client was sitting on a server with a session relating it to its remote counterpart on the users PC. Quasar Client catered for this need, by very prominently differentiating between session and application in the root of the dialog component hierarchy. However, in the current days of SPA applications and the lowered importance of servers-side web clients, this prominent differentiation was dropped. When still needed the referenced documents will provide in more detail how to tailor the respective architecture to this end.

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>

Layers

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

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*.

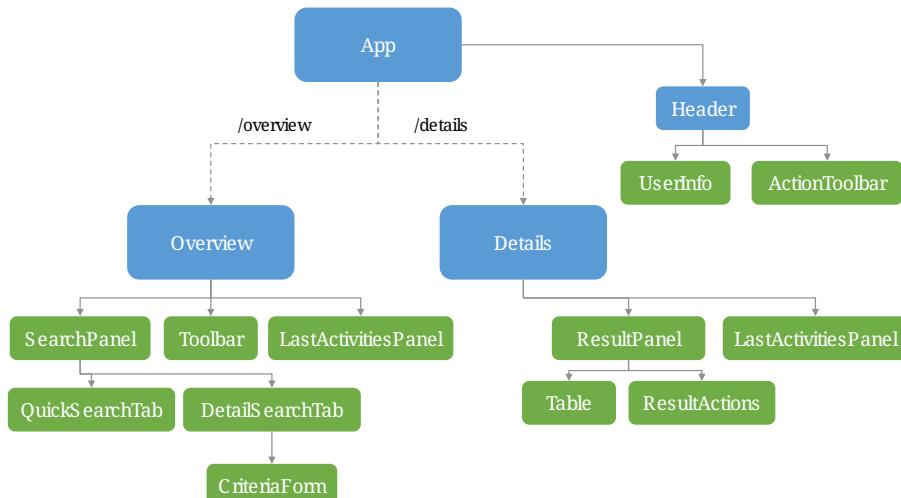


Figure 3. Component tree example

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 1. 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

<i>Smart Components</i>	<i>Dumb Components</i>
use services	do not use services
consists of n <i>Dumb Components</i>	is independent of <i>Smart Components</i>

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*.

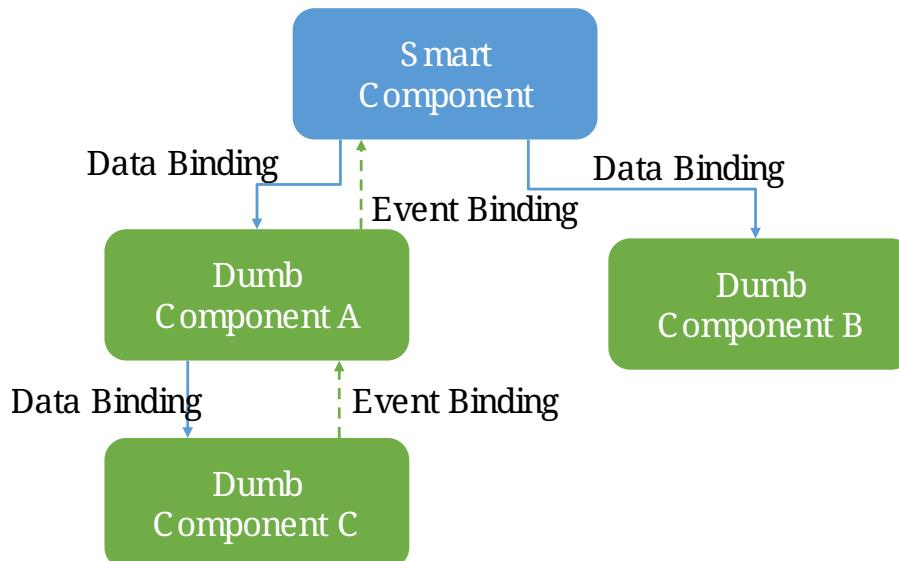


Figure 4. Smart and Dumb Component Interaction

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 1. 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 2. 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()`).

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](#).

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.

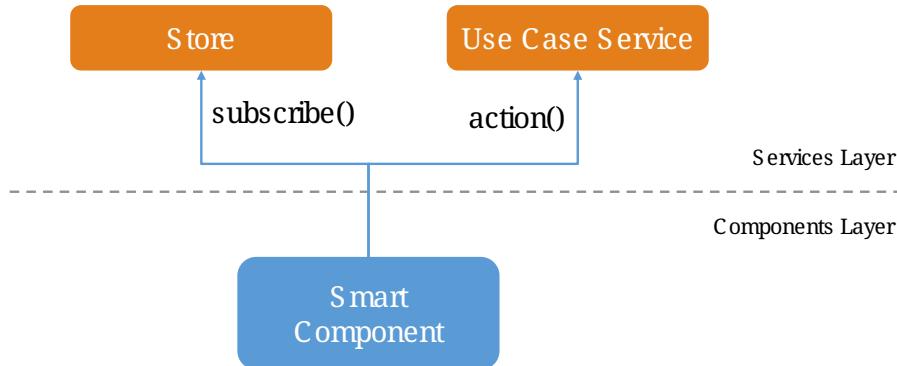


Figure 5. Boundaries to components layer

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 3. 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.

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**).

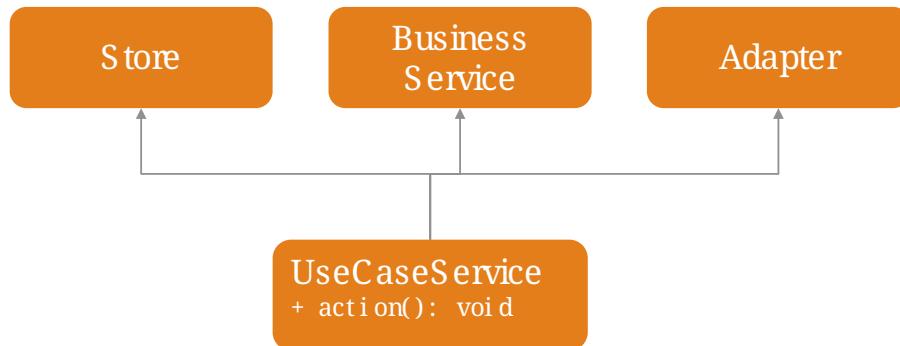


Figure 6. Use case services are the main API to trigger state transitions

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](#).

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 4. 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');
    }

}
```

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.

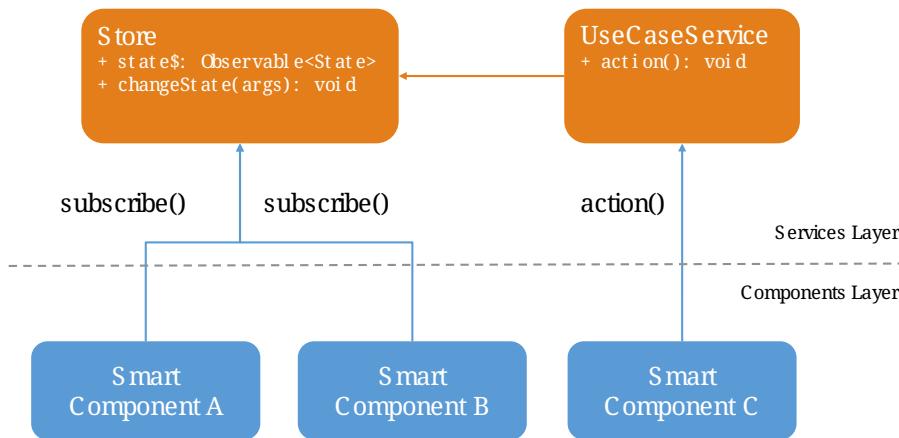


Figure 7. Smart Component interaction

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

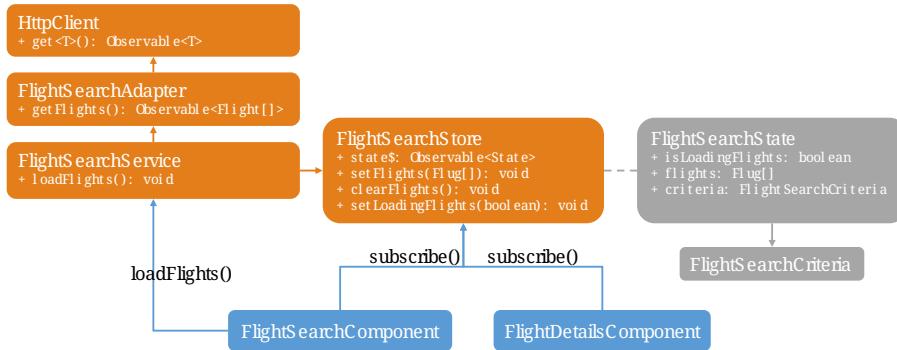


Figure 8. Smart Components interaction example

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.

Guides

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.

Semantic Versioning

When working with package managers it is very important to understand the concept of [semantic versioning](#).

Table 2. 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.

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
```

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 5. `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.

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 ^[1]. 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](#)

..1. 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](#).

..2. 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 command.

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 command **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 6. 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<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<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](#)

..3. 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.

.1. Angular

[1] A package is a file or directory that is described by a package.json. .

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.

Chapter 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

1.1. 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-described 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 {
  ...
}
```

1.2. 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

1.2.1. 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.

1.2.2. 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.

1.2.3. 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);
      }
    })
  }
}
```

Chapter 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.

2.1. 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.

2.2. 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.

2.3. 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.

Chapter 3. Accessibility with Angular Material

The `a11y` package provides a number of tools to improve accessibility. Import

```
import { A11yModule } from '@angular/cdk/a11y';
```

3.1. 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;  
}
```

3.1.1. Types of ListKeyManager

There are two varieties of `ListKeyManager`, `FocusKeyManager` and `ActiveDescendantKeyManager`.

3.2. FocusKeyManager

Used when options will directly receive browser focus. Each item managed must implement the `FocusableOption` interface:

```
interface FocusableOption extends ListKeyManagerOption {  
  focus(): void;  
}
```

3.3. 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.

3.4. 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>
```

3.5. 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>
```

3.6. InteractivityChecker

`InteractivityChecker` is used to check the interactivity of an element, capturing disabled, visible, tabbable, and focusable states for accessibility purposes.

3.7. 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");  
  }  
}
```

3.8. API reference for Angular CDK a11y

[API reference for Angular CDK a11y](#)

Chapter 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.

Chapter 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)

Chapter 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.

Chapter 7. How to use Angular Elements?

In a generalized way, a simple [Angular component](#) could be transformed to an [Angular Element](#) with this steps:

7.1. 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
```

7.2. 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 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 dont 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);  
    }  
}  
....
```

7.3. 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.

7.4. 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](#)

7.5. Building the Angular Element

First, before building the **Angular Element**, every element inside that app component except the module need to be removed. After that, a bash script is created in the root folder,. This script will allow to put every necessary file into a js.

```
ng build " projectName " --prod --output-hashing=none && cat  
dist/" projectName "/runtime.js dist/" projectName "/polyfills.js  
dist/" projectName "/scripts.js dist/" projectName "/main.js >  
. /dist/" projectName "/" "nameWantedAngularElement".js
```

After executing the bash script, it will generate inside the path **dist/" projectName "** a js file named **"nameWantedAngularElement".js** and a css file.

Building with **ngx-build-plus** (Recommended)

The library **ngx-build-plus** allows to add different options when building. In addition, it solves some errors that will occur when trying to use multiple angular elements in an application. In order to use it, yarn or npm can be used:

Yarn

```
yarn add ngx-build-plus
```

Npm

```
npm install ngx-build-plus
```

If you want to add it to a specific sub project in your projects folder, use the --project:

```
.... ngx-build-plus --project "project-name"
```

Using this library and the following command, an isolated **Angular Element** which won't have conflict with others can be generated. This **Angular Element** will not have a polyfill so, the project where we use them will need to include a **polyfill** with the **Angular Element** requirements.

```
ng build " projectName " --output-hashing none --single-bundle true --prod --bundle  
-styles false
```

This command will generate three things:

1. The main js bundle
2. The script js
3. The css

These files will be used later instead of the single js generated in the last step.

Extra parameters

Here are some extra useful parameters that **ngx-build-plus** provides:

- **--keep-polyfills**: This parameter is going to allow us to keep the polyfills. This needs to be used with caution, avoiding using multiple different polyfills that could cause an error if necessary.
- **--extraWebpackConfig webpack.extra.js**: This parameter allows us to create a javascript file inside our **Angular Elements** project with the name of different libraries. Using **webpack** these libraries will not be included in the **Angular Element**. This is useful to lower the size of our **Angular Element** by removing libraries shared. Example:

```
const webpack = require('webpack');

module.exports = {
  "externals": {
    "rxjs": "rxjs",
    "@angular/core": "ng.core",
    "@angular/common": "ng.common",
    "@angular/common/http": "ng.common.http",
    "@angular/platform-browser": "ng.platformBrowser",
    "@angular/platform-browser-dynamic": "ng.platformBrowserDynamic",
    "@angular/compiler": "ng.compiler",
    "@angular/elements": "ng.elements",
    "@angular/router": "ng.router",
    "@angular/forms": "ng.forms"
  }
}
```



If some libraries are excluded from the 'Angular Element' you will need to add the bundled umd files of those libraries manually.

7.6. 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 7. 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 8. 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.

Chapter 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:

8.1. Copy bundled script and css to resources

First copy the generated `.js` and `.css` inside assets in the corresponding folder.

8.2. 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.

8.3. 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>
```

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.

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.

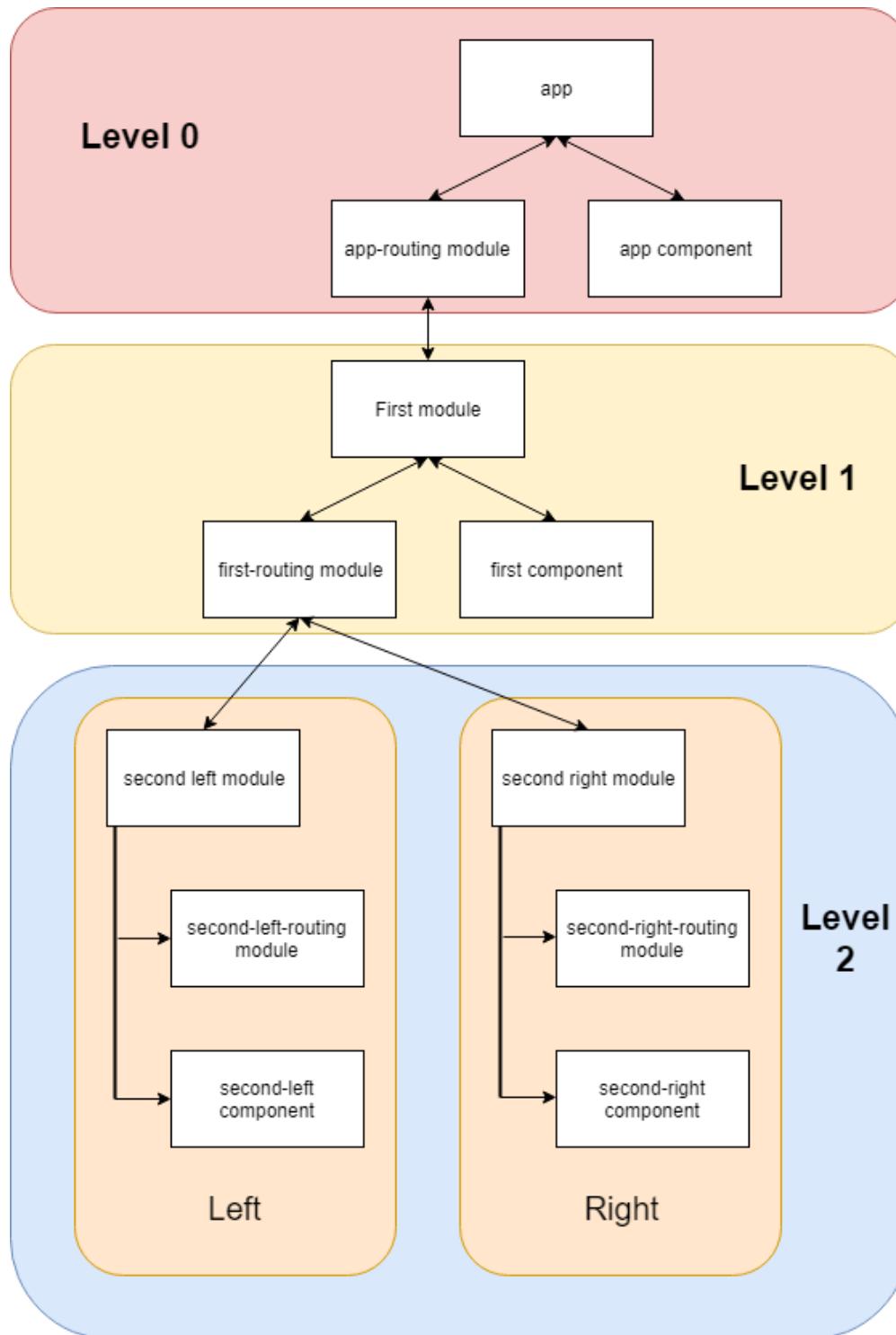


Figure 9. Levels app structure.

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](#).

Chapter 9. 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 9. 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 10. 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 11. 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 12. 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 13. 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 listening 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 10. 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.

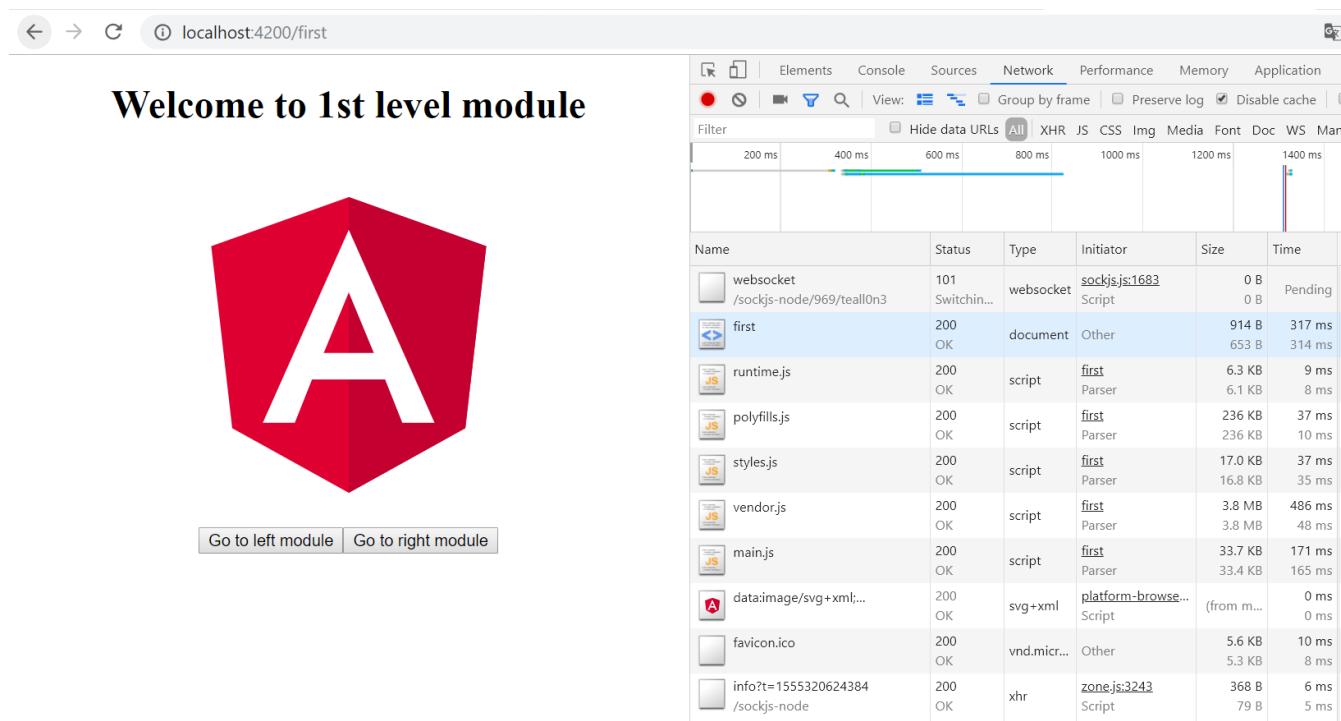


Figure 11. First level eager.

If we click on **[Go to right module]** a second level module opens, but there is no 'second-right' document.

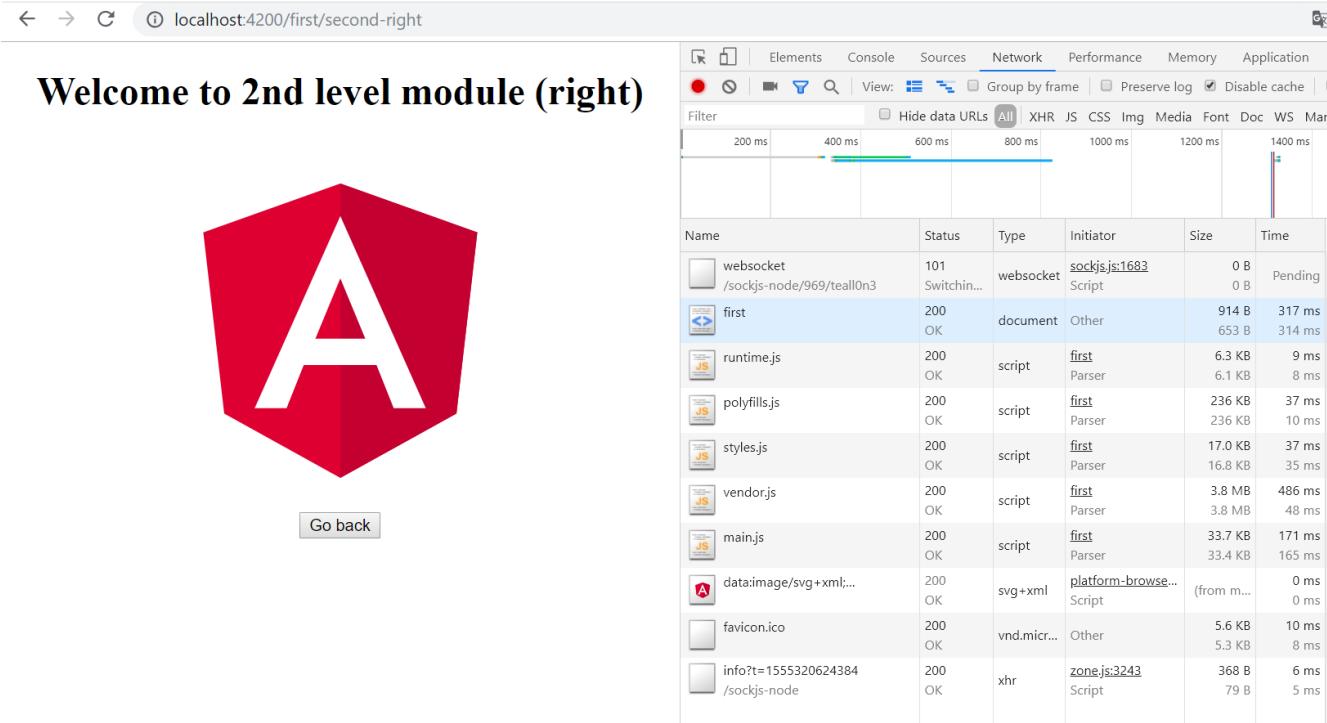


Figure 12. Second level right eager.

But, typing the url directly will load 'second-right' but no 'first', even if we click on [Go back]

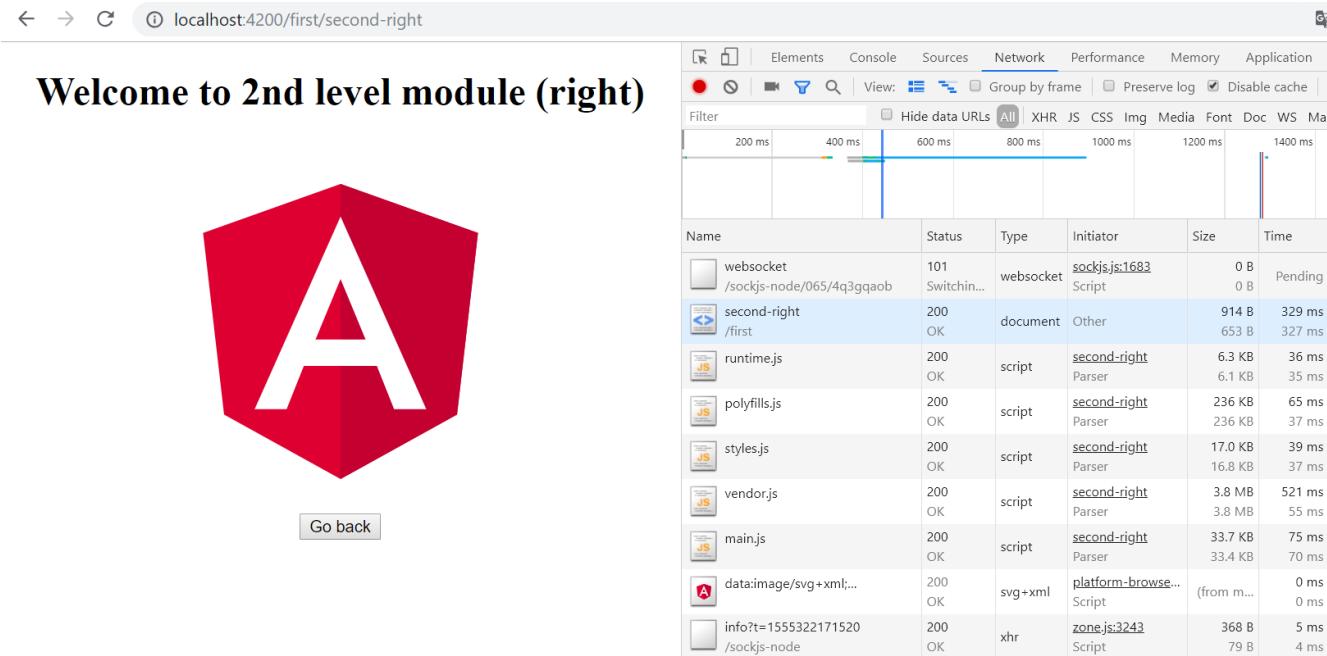


Figure 13. Second level right eager direct url.

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 14. 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 15. 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'.

Welcome to 1st level module



[Go to left module](#) [Go to right module](#)

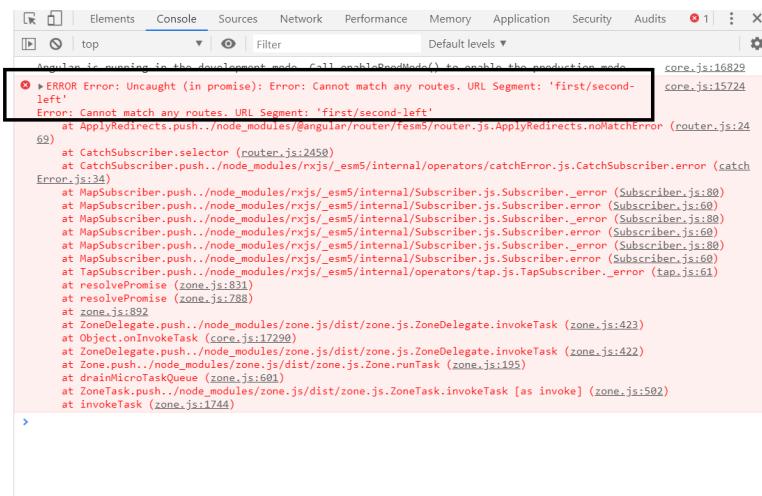


Figure 14. First level lazy wrong path.

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 15. 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 timeline at the top with markers for 200 ms, 400 ms, 600 ms, 800 ms, and 1000 ms. Below the timeline is a list of network requests. The requests are as follows:

- websocket /sockjs-node/846/iehjyl5q
- first (highlighted in blue)
- runtime.js
- polyfills.js
- styles.js
- vendor.js
- main.js
- first-first-module.js (highlighted in blue)

On the right side of the Network tab, there are sections for "General" and "Response Headers". The "General" section shows the Request URL: `http://localhost:4200/first`, Request Method: GET, Status Code: 200 OK, Remote Address: 127.0.0.1, and Referrer Policy: no-referrer. The "Response Headers" section lists several headers including Accept, Accept-Encoding, Accept-Language, Cache-Control, Connection, and Cookie.

Figure 16. 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 16. 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 17. 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 17. Second level lazy loading compilation.

Clicking on **[Go to left module]** triggers the load of SecondLeftModule.

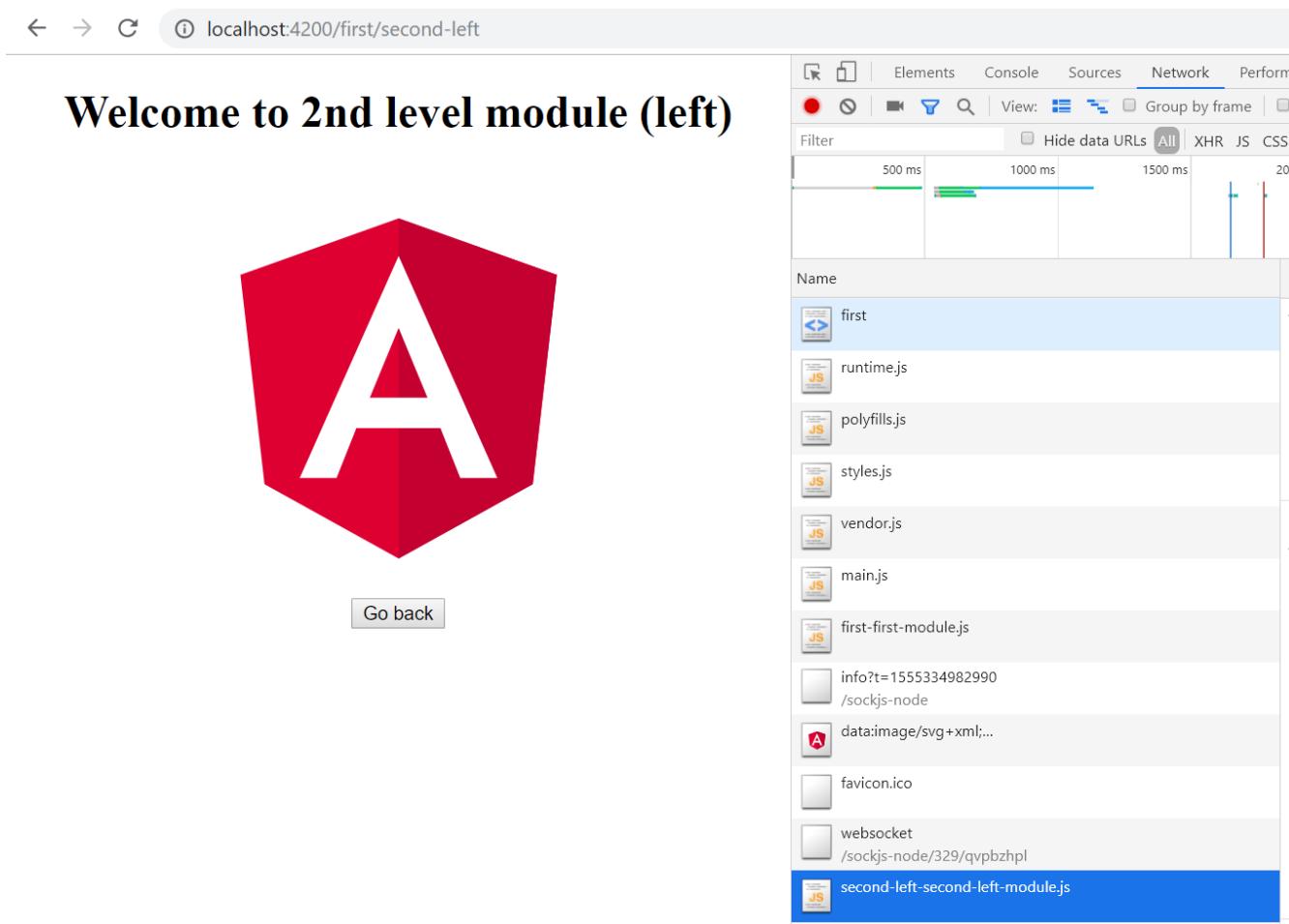


Figure 18. Second level lazy loading network.

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.

Chapter 10. 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.

Chapter 11. 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.

Chapter 12. How to build a library

In this section, a library is going to be build step by step.

12.1. 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
```

12.2. 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"
```

12.3. Generating/Modifying in our library

In the last step we generated a library. This generates automaticly 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"
```

12.4. 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`.

12.5. 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"
```

12.6. 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 18. Packing using npm

```
npm pack
```

Listing 19. Packing using yarn

```
yarn pack
```

12.7. Publishing to npm repository (optional)

- Add a `README.md` and `LICENSE` file. The text inside `README.md` will be used in your 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: <a href="https://npmjs.com/package/<library-name>" class="bare">https://npmjs.com/package/<library-name>;

12.8. Installing our library in other projects

In this step we are going to install/add the library on other projects.

12.8.1. 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 [id-packing-library].
- **Option 2:** run `npm install "library-name"` to install it from npm repository.

12.8.2. 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 [id-packing-library].
- **Option 2:** run `yarn add "library-name"` to install it from npm repository.

12.9. 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!

12.10. 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.

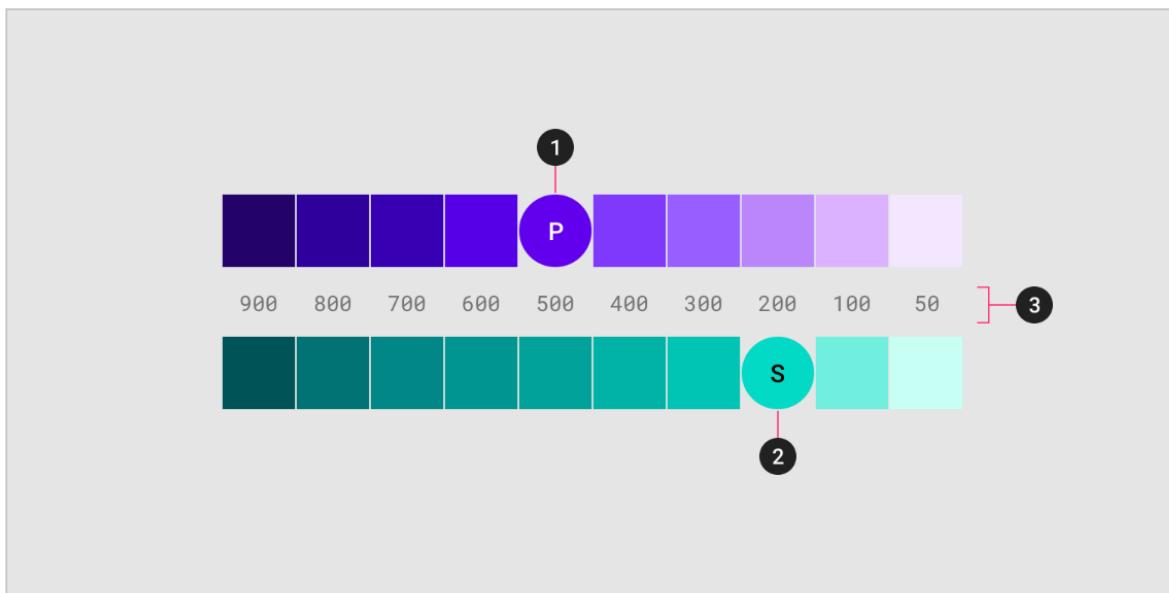
Angular Material Theming

Angular Material library offers UI components for developers, those components follows Google Material desing baselines but characteristics like colors can be modified in order to adapt them to the needs of the client: corporative colors, corporative identity, dark themes, ...

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 19. Palettes and variants.

In angular material, a palette is represented as a scss map.

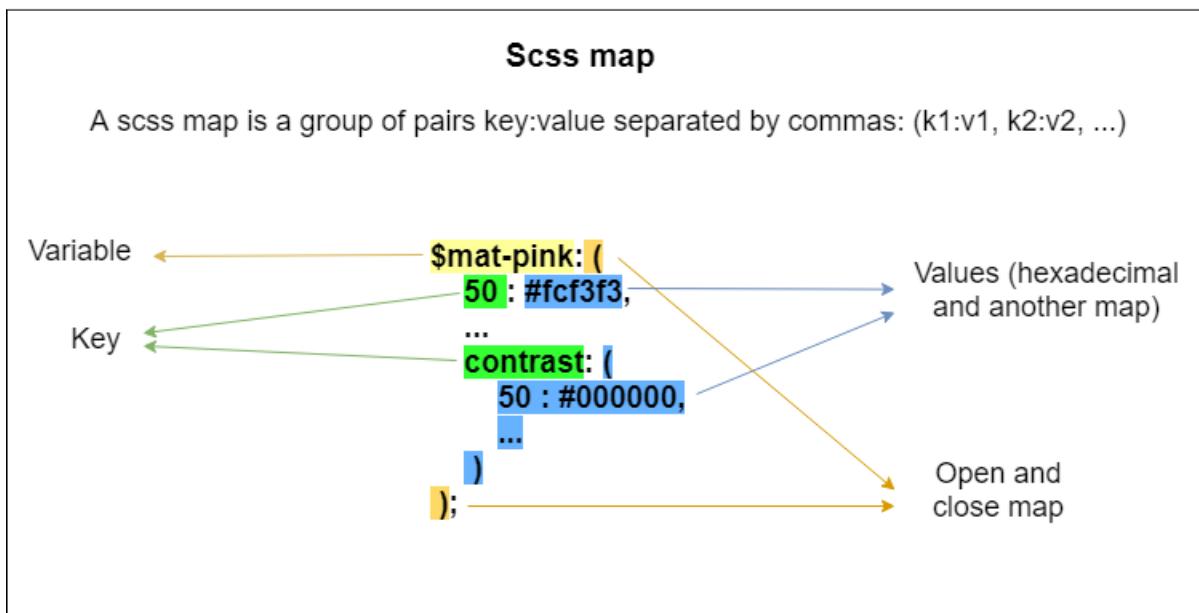


Figure 20. Scss map and palettes.



Some components can be forced to use primary, accent or warn palettes using the attribute **color**, for example: <mat-toolbar color="primary">.

Prebuilt themes

Available prebuilt themes:

- deeppurple-amber.css

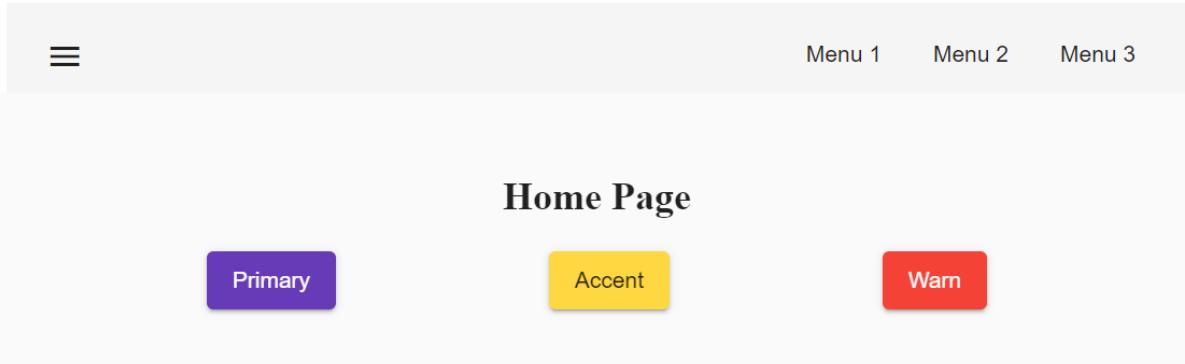


Figure 21. deeppurple-amber theme.

- indigo-pink.css

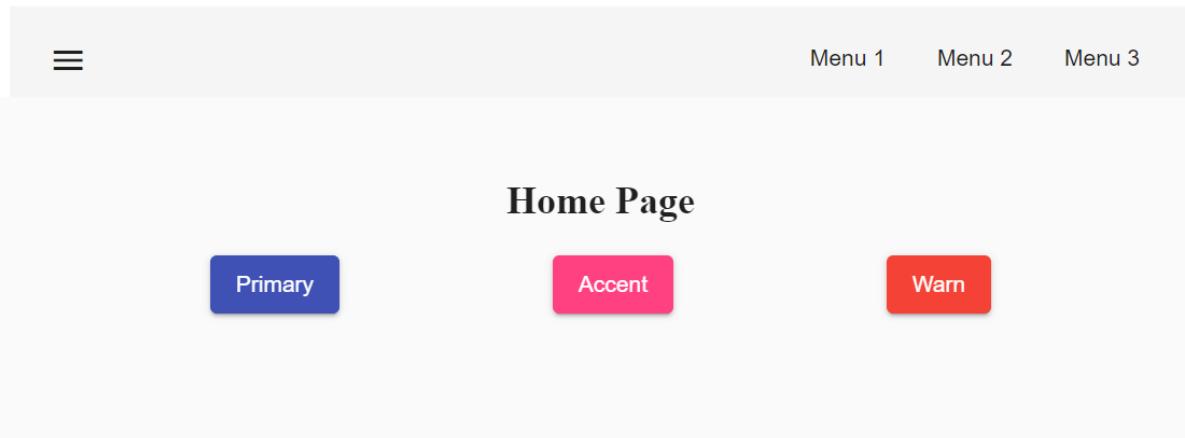


Figure 22. indigo-pink theme.

- pink-bluegrey.css

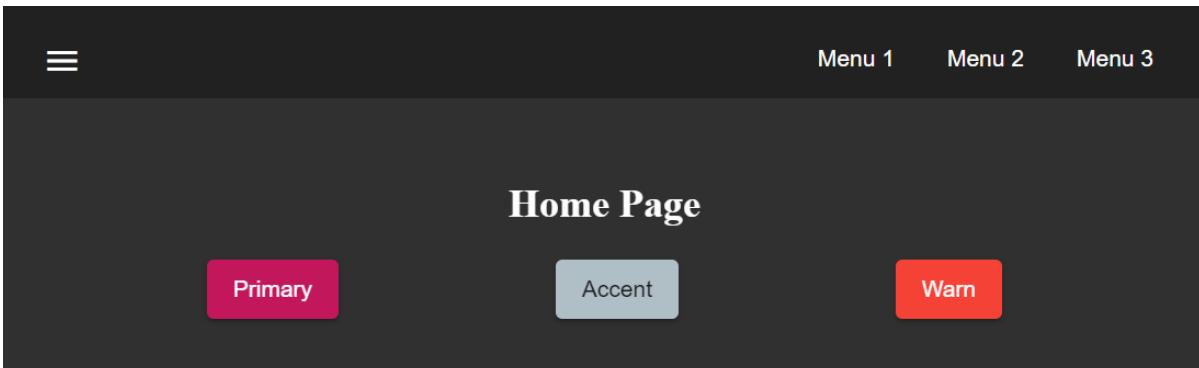


Figure 23. ink-bluegrey theme.

- purple-green.css

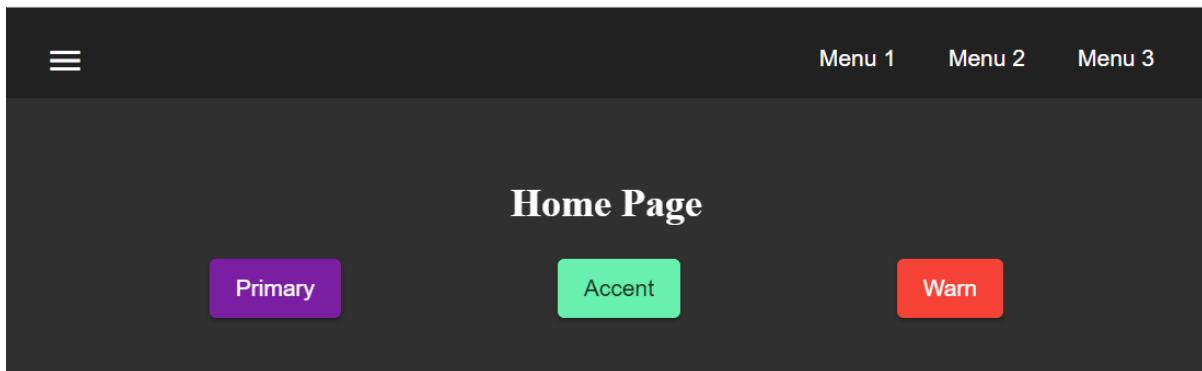


Figure 24. purple-green theme.

The prebuilt themes can be added using `@import`.

```
@import '@angular/material/prebuilt-themes/deeppurple-amber.css';
```

Custom themes

Sometimes prebuild themes do not meet the needs of a project, because color schemas are too specific or do not incorporate branding colors, in those situations custom themes can be built to offer a better solution to the client.

For this topic, we are going to use a basic layout project that can be found in [devon4ng repository](#).

Chapter 13. Basics

Before starting writing custom themes, there are some necessary things that have to be mentioned:

- Add a default theme: The project mentioned before has just one global scss stylesheet **styles.scss** that includes `indigo-pink.scss` which will be the default theme.
- Add `@import '~@angular/material/theming'`; at the begining of the every stylesheet to be able to use angular material prebuilt color palettes and functions.
- Add `@include mat-core(); once` per project, so if you are writing multiple themes in multiple files you could import those files from a 'central' one (for example `styles.scss`). This includes all common styles that are used by multiple components.

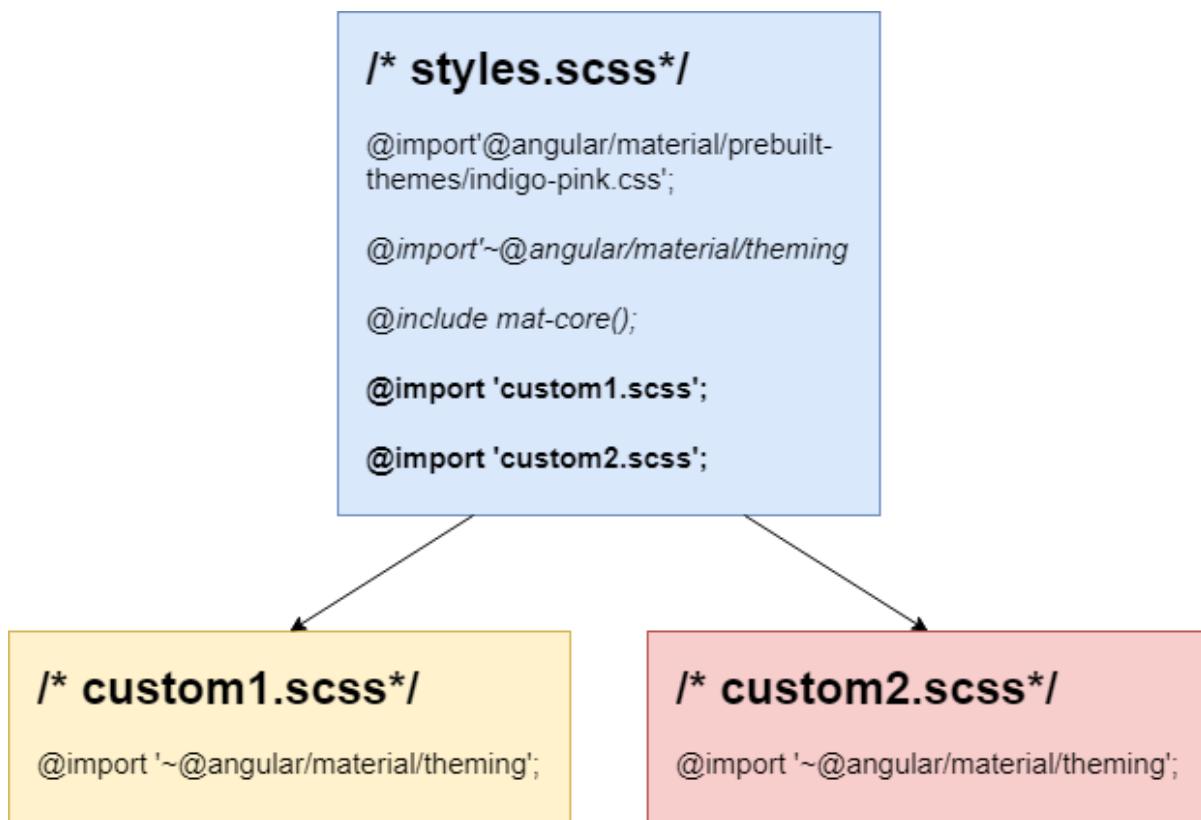


Figure 25. Theme files structure.

Chapter 14. 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 20. 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 21. 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.

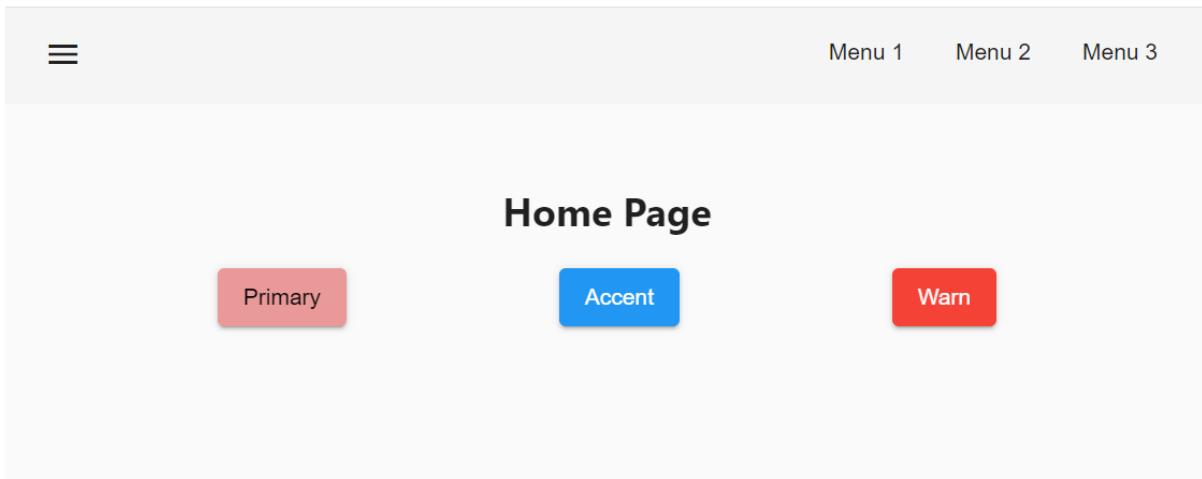


Figure 26. Custom light theme.

- **mat-dark-theme:** Similar to mat-light-theme but returns a theme whose foreground is basically white and background black.

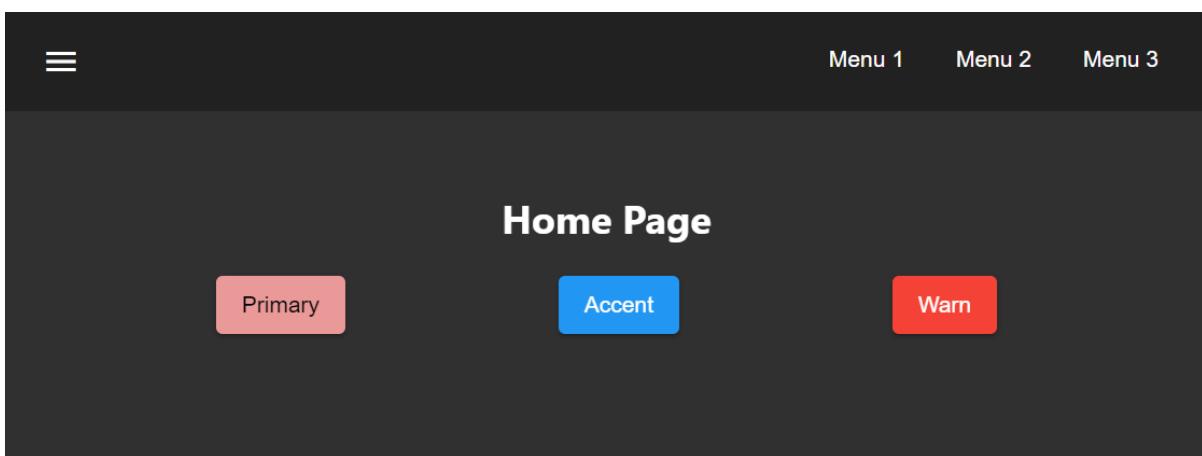


Figure 27. Custom dark theme.

For this example we are going to use mat-dark-theme and save its result in \$custom-dark-theme.

Listing 22. 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 23. 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 24. 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 25. File src/index.html.

```
...
<app-root class="custom-dark-theme"></app-root>
...
```

This will apply **\$custom-dark-theme** theme for the entire application.

Chapter 15. 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 26. 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 see, *mat-light-theme* takes three arguments and returns 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 27. 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 28. 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 29. 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 30. File styles.scss updated with custom-cap-theme class.

```
...  
@import 'styles-custom-cap.scss';  
.custom-cap-theme {  
  @include angular-material-theme($custom-cap-theme);  
}
```

Chapter 16. 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 31. File app.module.ts

```
import {OverlayContainer} from '@angular/cdk/overlay';

@NgModule({
  // ...
})
export class AppModule {
  constructor(overlayContainer: OverlayContainer) {
    overlayContainer.getContainerElement().classList.add('custom-cap-theme');
  }
}
```

Useful resources

- [Angular Material's official theming guide](#)
- [Material Design: Color theme creation](#)
- [Palette generator](#)
- [SCSS tutorial](#)

Angular Progressive Web App

Progresive web applications (PWAs) are web application that offer better user experience than the traditional ones. In general, they solve problems related with reliability and speed:

- *Reliability*: PWAs are stable. In this context stability means than even with slow connections or even with no network at all, the application still works. To achieve this, some basic resources like styles, fonts, requests, ... are stored; due to this caching, it is not possible to assure that the content is always up-to-date.
- *Speed*: When an users opens an application, he or she will expect it to load almost 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.

Assumptions

This guide assumes that you already have installed:

- Node.js
- npm package manager
- Angular CLI

Sample Application

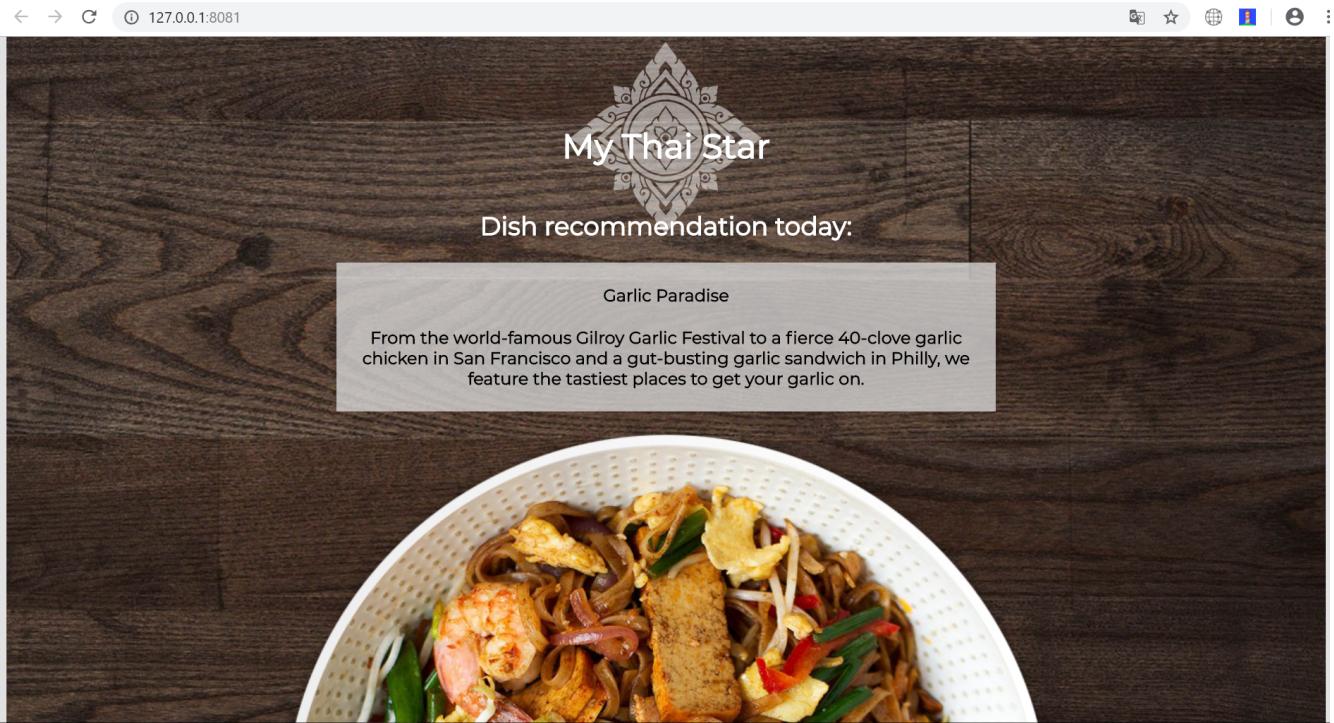


Figure 28. Basic angular PWA.

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.

Chapter 17. 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**.

Chapter 18. 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';
...
```

Chapter 19. 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,
          };
        },
      );
  }
}
```

Chapter 20. 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());  
    });  
  }  
  ...  
}
```

Chapter 21. 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](#).

- ngsyw-config.json

ngsw-config.json specifies which files and data URLs have to be cached and updated by the Angular service worker.

Fields

- *index*: File that serves as index page to satisfy navigation requests.
- *assetGroups*: Resources that are part of the app version that update along with the app.
 - *name*: Identifies the group.
 - *installMode*: How the resources are cached (prefetch or lazy).
 - *updateMode*: Caching behaviour when a new version of the app is found (prefetch or lazy).
 - *resources*: Resources to cache. There are three groups.

- *files*: Lists patterns that match files in the distribution directory.
- *urls*: URL patterns matched at runtime.
- *dataGroups*: UsefulIdentifies the group. for API requests.
 - *name*: Identifies the group.
 - *urls*: URL patterns matched at runtime.
 - *version*: Indicates that the resources being cached have been updated in a backwards-incompatible way.
 - *cacheConfig*: Policy by which matching requests will be cached
 - *maxSize*: The maximum number of entries, or responses, in the cache.
 - *maxAge*: How long responses are allowed to remain in the cache.
 - d: days. (5d = 5 days).
 - h: hours
 - m: minutes
 - s: seconds. (5m20s = 5 minutes and 20 seconds).
 - u: milliseconds
 - *timeout*: How long the Angular service worker will wait for the network to respond before using a cached response. Same dataformat as maxAge.
 - *strategy*: Caching strategies (performance or freshness).
- *navigationUrls*: List of URLs that will be redirected to the index file.

Additional information can be found [here](#).

Chapter 22. 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"  
    }  
  }]  
}
```

Chapter 23. Step 7: Check that your app is a PWA

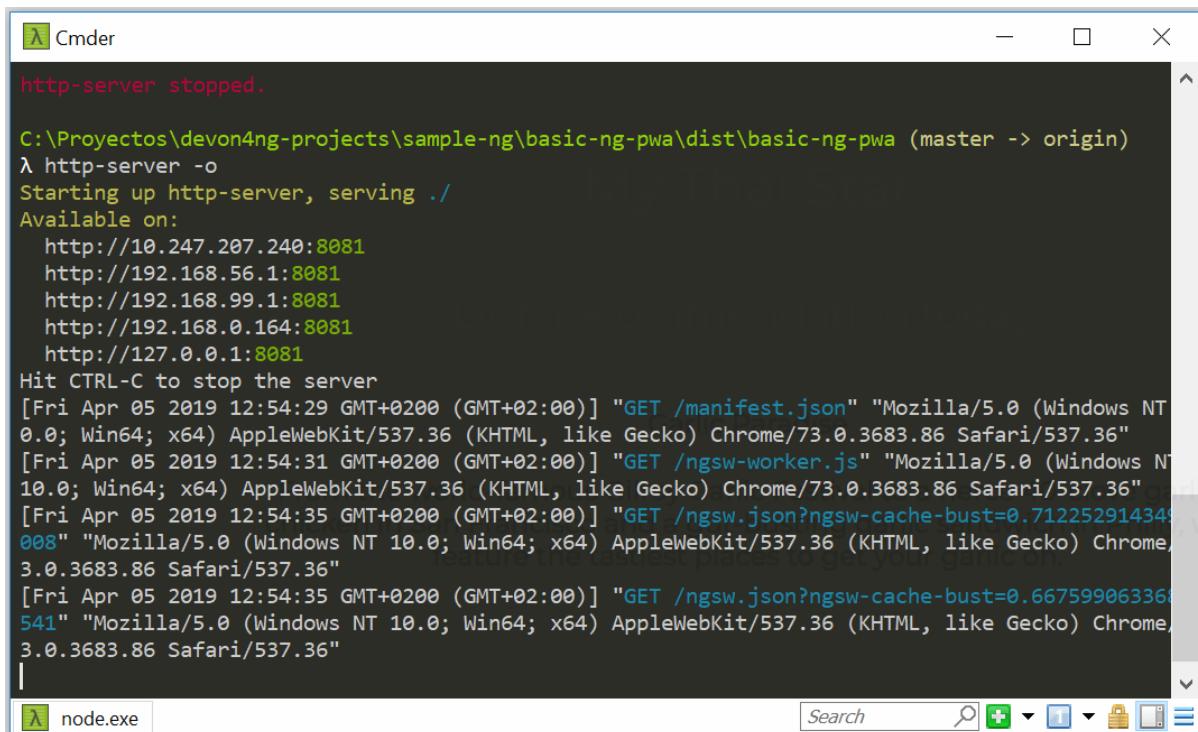
To check if an app is a PWA lets compare its normal behaviour against itself but built for production. Run in the project's root folder the commands below:

`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.



```
λ Cmder
http-server stopped.

C:\Proyectos\devon4ng-projects\sample-ng\basic-ng-pwa\dist\basic-ng-pwa (master -> origin)
λ http-server -o
Starting up http-server, serving .
Available on:
  http://10.247.207.240:8081
  http://192.168.56.1:8081
  http://192.168.99.1:8081
  http://192.168.0.164:8081
  http://127.0.0.1:8081
Hit CTRL-C to stop the server
[Fri Apr 05 2019 12:54:29 GMT+0200 (GMT+02:00)] "GET /manifest.json" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36"
[Fri Apr 05 2019 12:54:31 GMT+0200 (GMT+02:00)] "GET /ngsw-worker.js" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36"
[Fri Apr 05 2019 12:54:35 GMT+0200 (GMT+02:00)] "GET /ngsw.json?ngsw-cache-bust=0.712252914349008" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36"
[Fri Apr 05 2019 12:54:35 GMT+0200 (GMT+02:00)] "GET /ngsw.json?ngsw-cache-bust=0.667599063364541" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36"
```

Figure 29. 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 SourceM
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 30. 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.

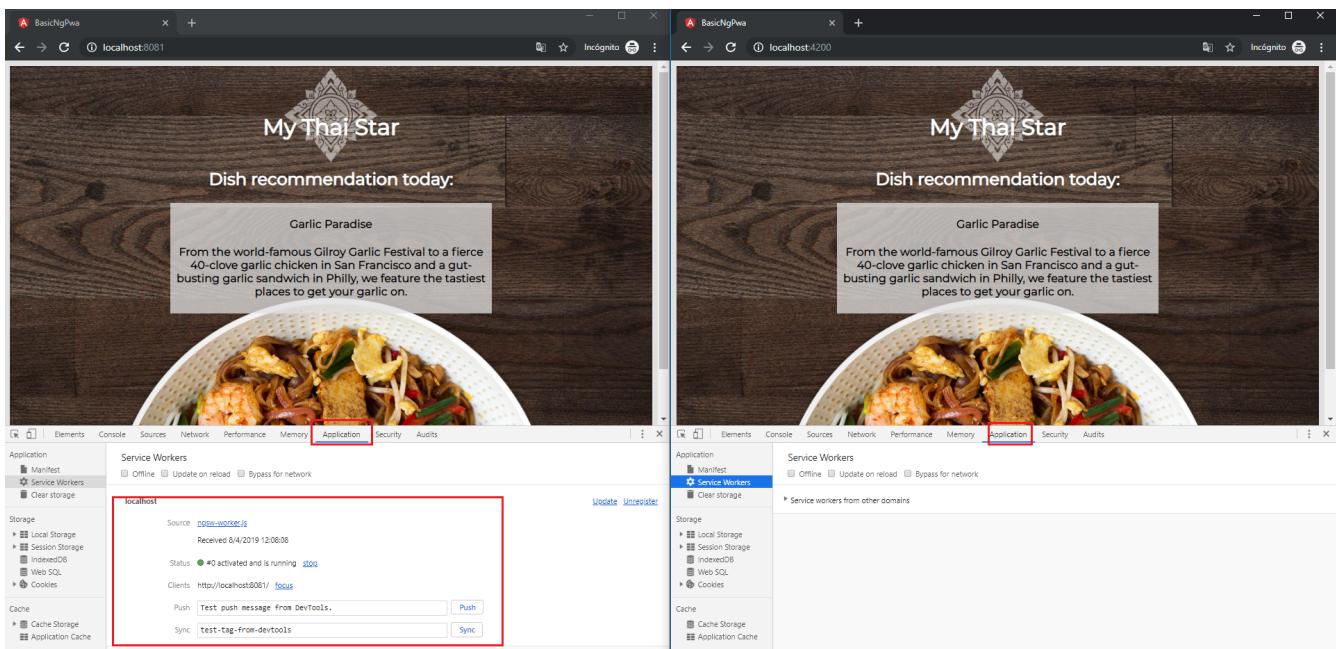


Figure 31. Application service worker comparison.

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.

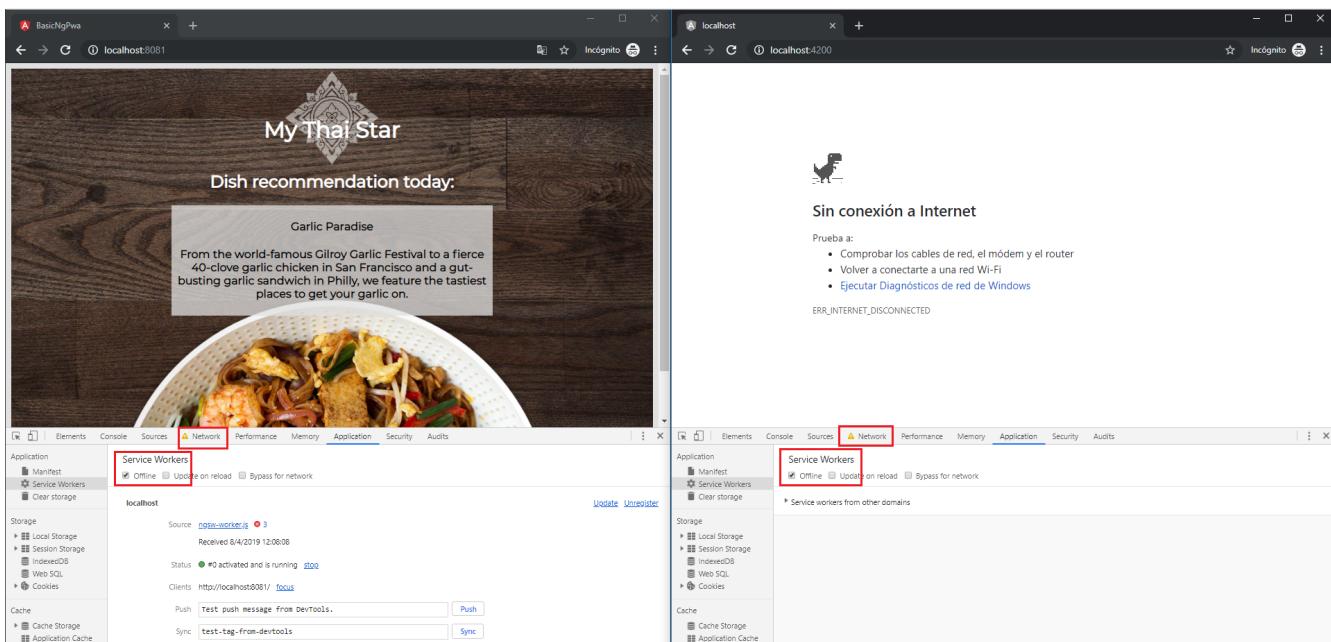


Figure 32. Offline application.

Finally, browser extensions like [Lighthouse](#) can be used to test whether an application is progressive or not.

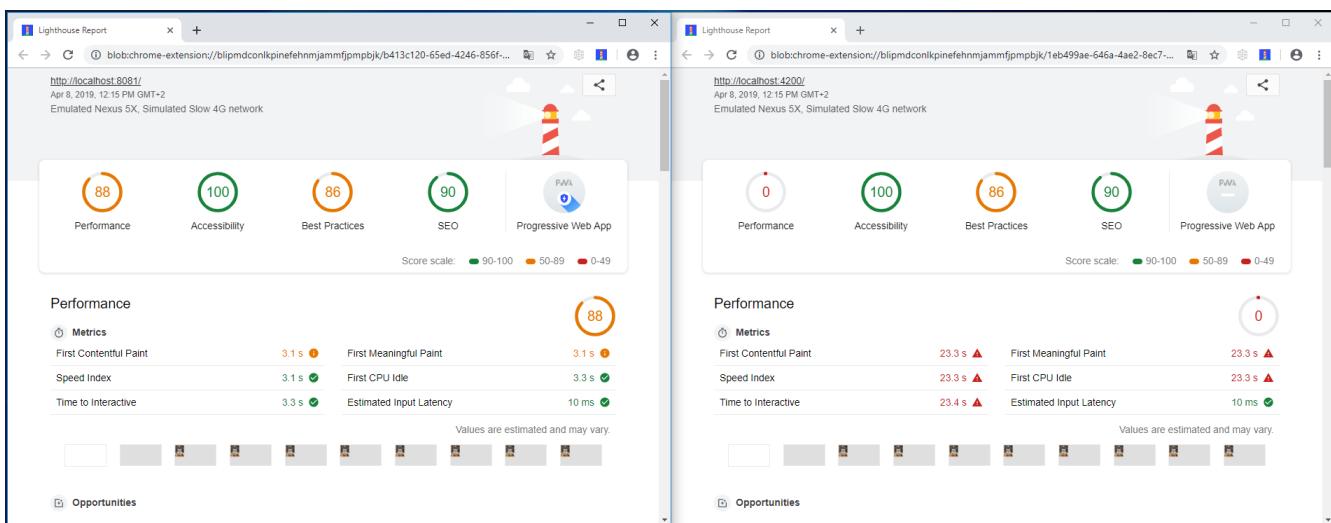


Figure 33. Lighthouse report.

APP_INITIALIZER

Chapter 24. 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 theres no need to recompile the whole application to switch from configuration.

Chapter 25. 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 };
```

Chapter 26. 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.

Chapter 27. Setting up the config files

27.1. Docker external configuration (Optional)

This section is only done if there's 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  
....
```

27.2. 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"  
        ]  
        ....  
    }  
}
```

Chapter 28. 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`.

28.1. 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;  
}  
....
```

28.2. 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",  
    ....
```

Chapter 29. 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.

Chapter 30. 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`.

30.1. 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>
```

30.2. 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,
};
```

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.

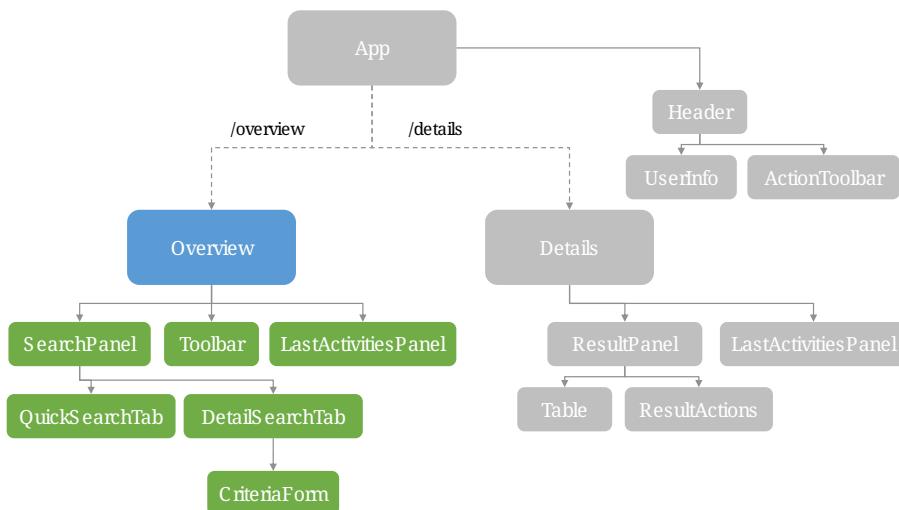


Figure 34. Component Tree with highlighted subtree

Chapter 31. 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.

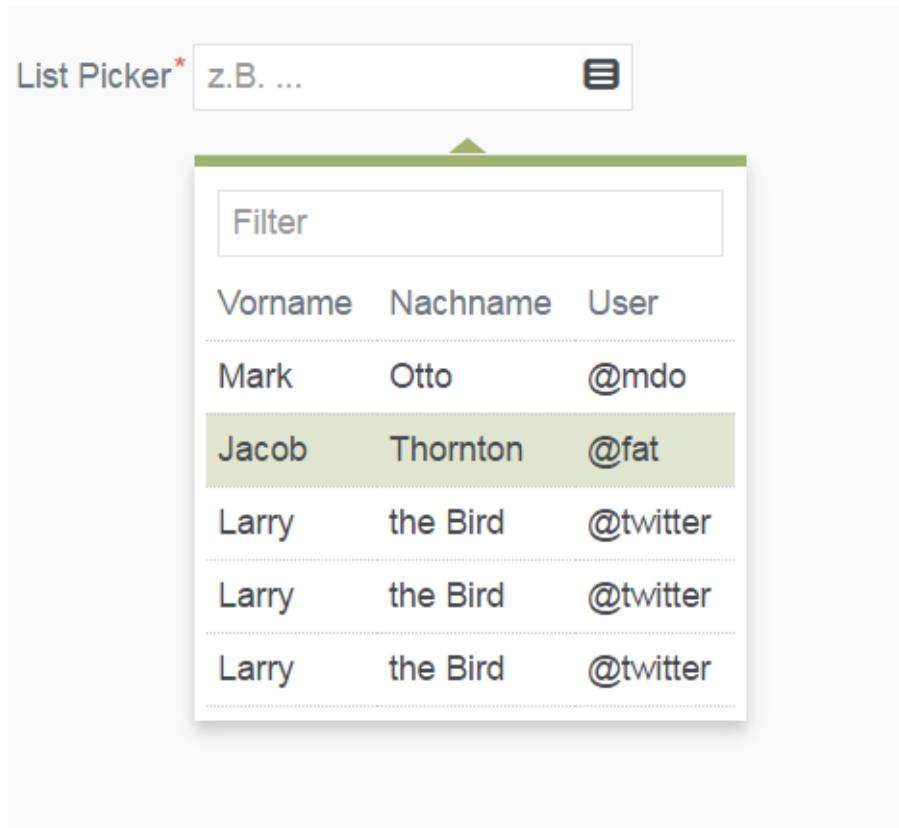


Figure 35. Component decomposition example before

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*.

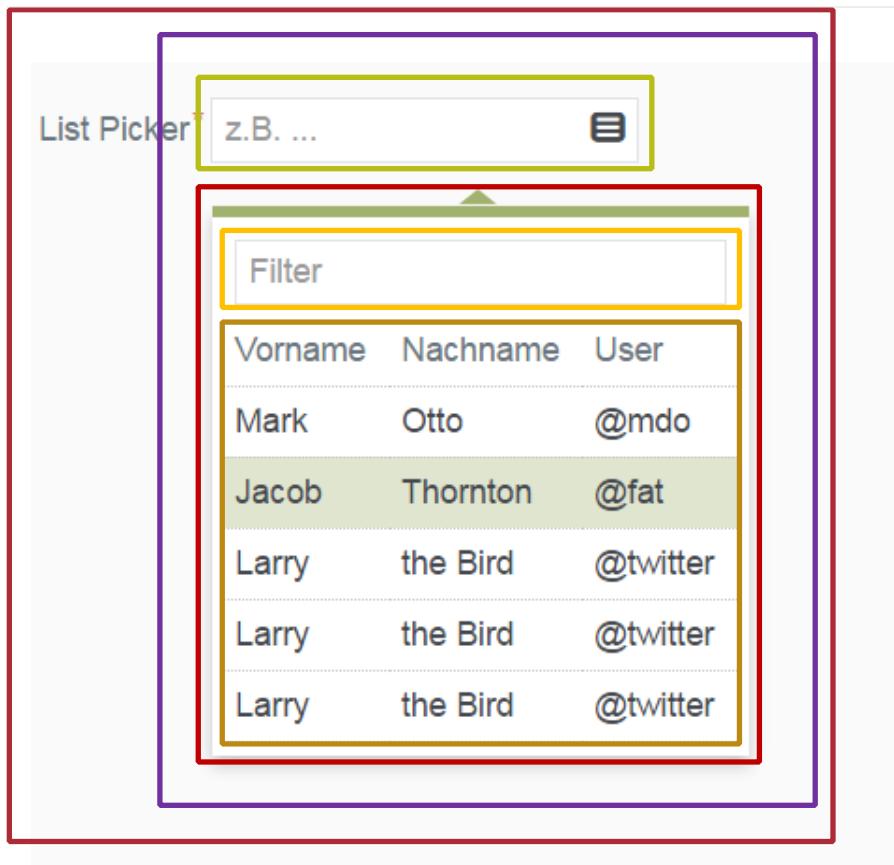


Figure 36. Component decomposition example after

This leads to the following component tree.

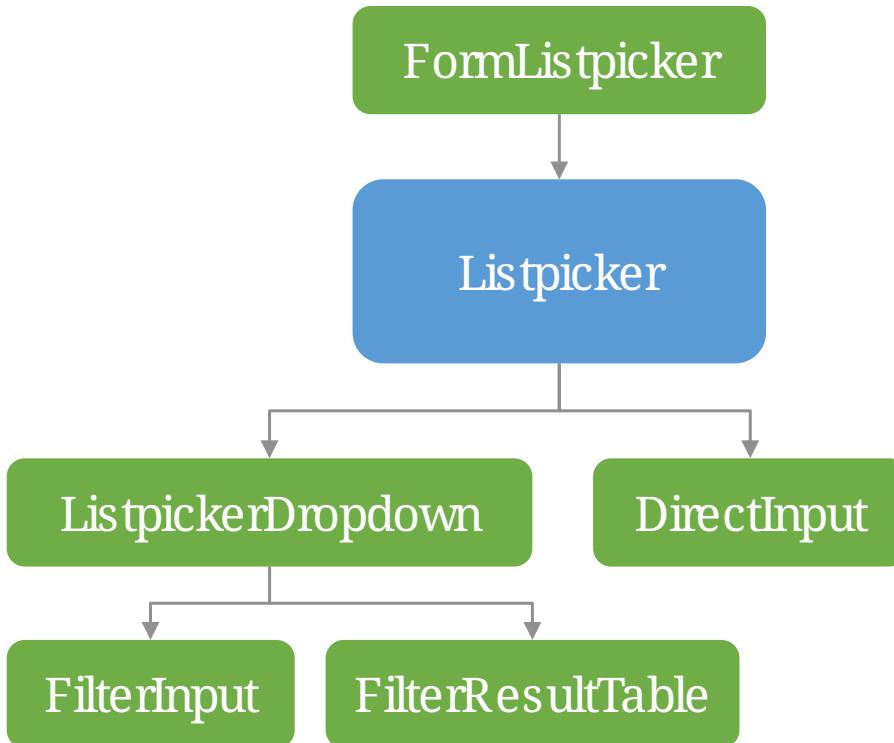


Figure 37. Component decomposition example component tree

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.

Chapter 32. 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 32. 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 33. 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.

Chapter 33. 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.

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.

Chapter 34. 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.

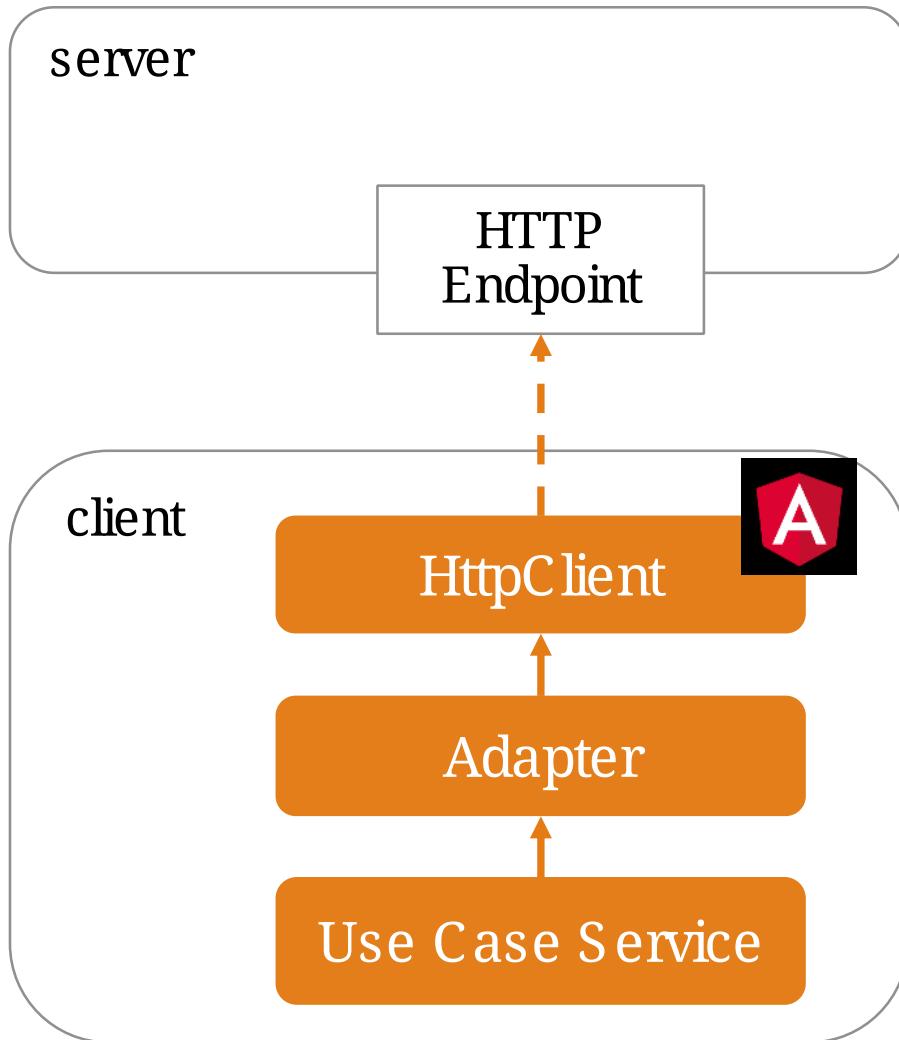


Figure 38. Adapters handle backend communication

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 34. 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 35. 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
  }
}
```

Chapter 35. Token management

Chapter 36. 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.

Chapter 37. 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.

Chapter 38. Creating your custom ErrorHandler step by step

In order to create a custom `ErrorHandler` three steps are going to be needed:

38.1. 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.

38.2. 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.

38.3. 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,  
  ....
```

Chapter 39. Handling Errors

As a final step, handling these errors is necessary. There's 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.

39.1. 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.

File Structure

Chapter 40. 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 36. 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.

Chapter 41. Feature Modules

A feature module contains the modules definition and two folders representing both layers.

Listing 37. 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](#)).

Chapter 42. Components Layer

The component layer reflects the distinction between *Smart Components* and *Dumb Components*.

Listing 38. 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 39. 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 40. 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!

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](#)

Chapter 43. 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/>.

43.1. 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.

43.2. 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`.

43.3. 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**.

43.4. Using the service, pipe or directive

43.4.1. 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'  
});
```

43.4.2. Pipe

The use of pipes can be possible too:

template:

```
<div>{{ 'HELLO' | translate:param }}</div>
```

component:

```
param = {value: 'world'};
```

43.4.3. 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.

Routing

A basic introduction to the Angular Router can be found in [Angular Docs](#).

This guide will show common tasks and best practices.

Chapter 44. 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.

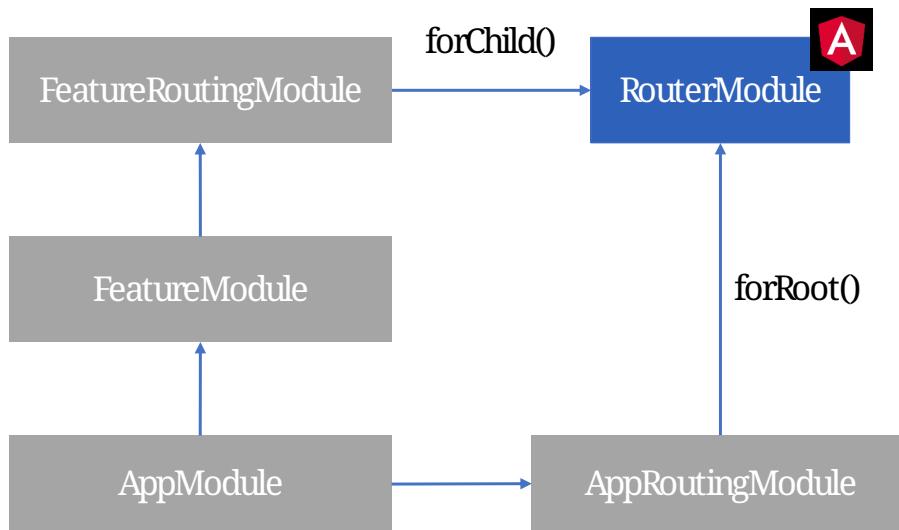


Figure 39. Routing module declaration

It is important to define routes inside app routing module with `.forRoot()` and in feature routing modules with `.forChild()`.

44.1. 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 41. 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 42. 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.

44.2. 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 43. 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 44. 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 { }
```

Chapter 45. 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*.

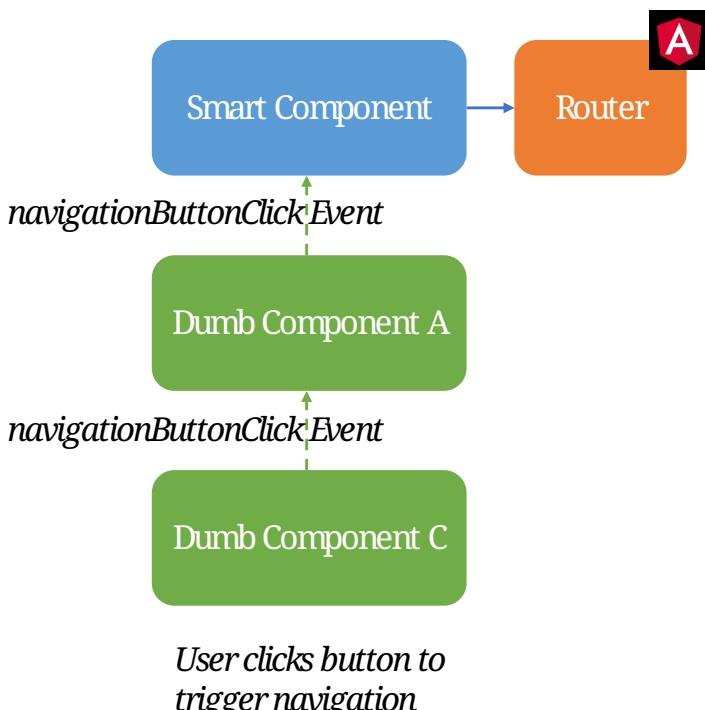


Figure 40. Triggering navigation

Chapter 46. 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.

46.1. Example 1 - CanActivate and CanActivateChild guards

46.1.1. 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 45. 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 46. 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 47. 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.

46.1.2. CanActivateChild guard

To use the guard on nested (children) routes, we add it to the `canActivateChild` property like so:

Listing 48. 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}
    ]
};
```

46.2. 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 49. 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 50. Implementing the guard

```
...
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'admin', loadChildren: 'app/admin/admin.module#AdminModule', canLoad:
  [ExampleCanLoadGuard] }
];
```

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.

Chapter 47. Testing Concept

The following figure shows you an overview of the application architecture devided in testing areas.

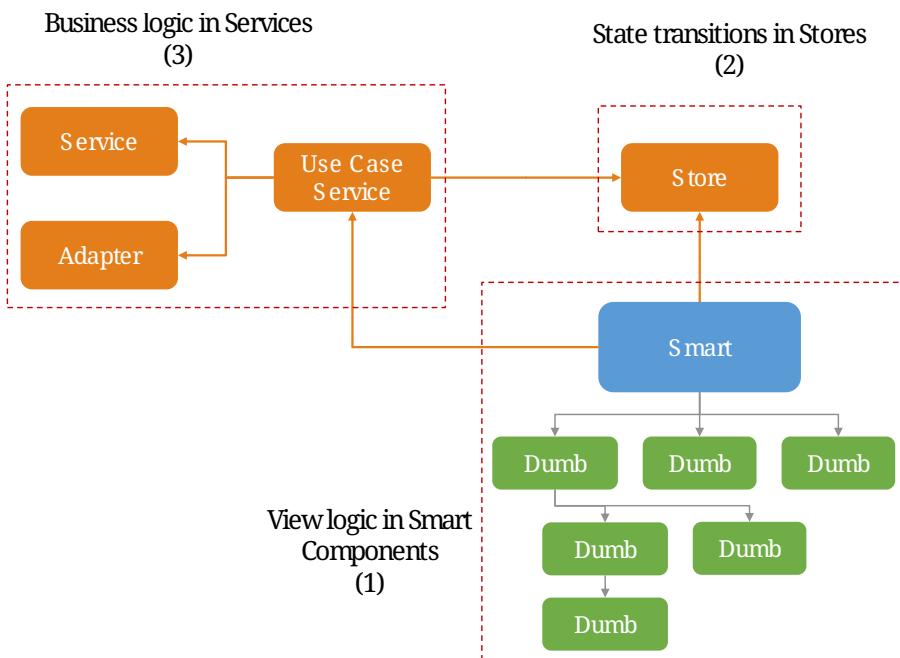


Figure 41. Testing Areas

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 fail and warn you, that there is something wrong with the new version. Most of the time Dumb Components do not need to be tested because they mainly display data and do not contain any logic. Smart Components are always 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 tested 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.

Chapter 48. 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 51. 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 52. 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.

Chapter 49. 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 53. 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();
  });
});
```

Chapter 50. 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 54. 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 55. 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 56. 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 57. 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)');
}));
```

Chapter 51. 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.

Chapter 52. 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

A complex musical score for two voices, featuring multiple staves with various note heads and rests.

Angular CLI: 7.2.3

Node: 10.13.0

OS: win32 x64

Angular:

■ ■ ■

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.

Chapter 53. 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.

A screenshot of a terminal window on a Mac OS X system. The title bar says "Terminal". The command "C:\>ng --version" is entered at the prompt. The output shows the Angular CLI version as 1.7.3, Node.js version as 8.9.4, and the operating system as win32 x64. There is also some decorative ASCII art at the top of the terminal window.

Figure 42. Printing Angular CLI Version

Chapter 54. 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
```

Chapter 55. 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.

Chapter 56. 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.

Chapter 57. Generating Code

57.1. 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
```

57.2. 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 43. 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.

57.3. 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 44. 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.

Chapter 58. 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
```

58.1. Ionic

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.

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.

Basic environment set up

Chapter 59. Install Ionic CLI

Although the devonfw distribution comes with and already installed Ionic CLI, here are the steps to install it. The instalation of Ionic is easy, just one command has to be written:

```
npm install -g ionic
```

Chapter 60. 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
```

Basic project set up

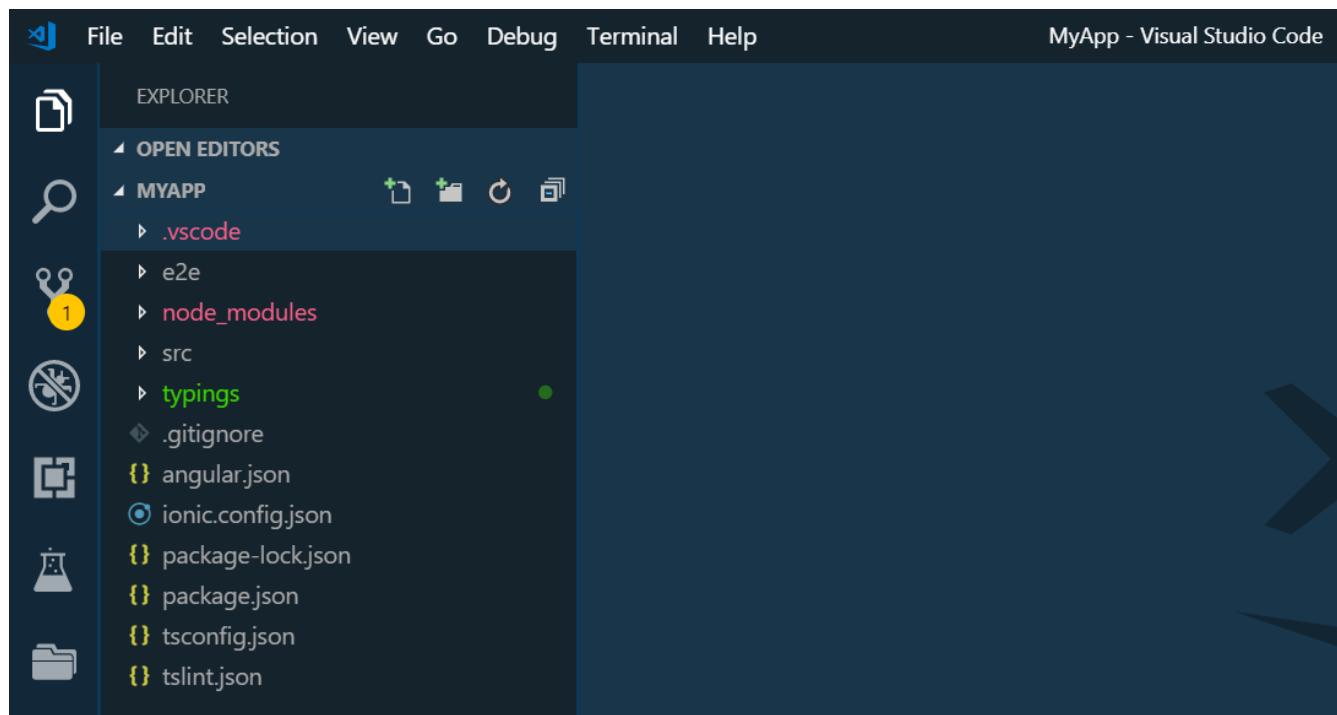
The set up of an ionic application is pretty immediate and can be done in one line:

```
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	michecker-template	update-ionic-clip	ionic-blank-proje
name	project type	ct.PNG	
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).

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.

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.

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`

Chapter 61. 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.

```

ts environments.ts ...
environment.android.ts ...
angular.json ...

```

```

1 // This file can be replaced during build by using the 'fileReplacements' ...
2 // in '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
13 * to ignore zone related error stack frames such as 'zone.run'.
14 * This import should be commented out in production mode because
15 * on performance if an error is thrown.
16 */
17
18 // Import 'zone.js/dist/zone-error'; // Included with Angular
19

```

```

1 export const environment = {
2   production: false,
3 };
4
5 export const SERVER_URL = 'http://10.0.2.2:8081';
6

```

```

5 "newProjectRoot": "projects",
6 "projects": {
7   "app": {
8     "root": "",
9     "sourceRoot": "src",
10    "projectType": "application",
11    "prefix": "app",
12    "schematics": {},
13    "architect": {
14      "build": {
15        "builder": "@angular-devkit/build-angular:browser",
16        "options": {
17          "outputPath": "dist/app",
18          "index": "src/index.html",
19          "main": "src/main.ts",
20          "polyfills": "src/polyfills.ts",
21          "tsConfig": "tsconfig.app.json",
22          "assets": [
23            "src/favicon.ico",
24            "src/assets"
25          ],
26          "styles": [
27            "src/styles.css"
28          ],
29          "scripts": []
30        },
31        "configurations": {
32          "production": {
33            "outputPath": "dist/app",
34            "index": "src/index.html",
35            "main": "src/main.ts",
36            "polyfills": "src/polyfills.ts",
37            "tsConfig": "tsconfig.app.json",
38            "assets": [
39              "src/favicon.ico",
40              "src/assets"
41            ],
42            "styles": [
43              "src/styles.css"
44            ],
45            "scripts": []
46          }
47        }
48      },
49      "serve": {
50        "builder": "@angular-devkit/build-angular:dev-server",
51        "options": {
52          "browserTarget": "app:build"
53        },
54        "configurations": {
55          "production": {
56            "browserTarget": "app:build:production"
57          },
58          "ci": {
59            "progress": false
60          }
61        }
62      }
63    }
64  }
65}

```

```

"build": {
...
  "configurations": {
    ...
      "android": {
        "fileReplacements": [
          {
            "replace": "src/environments/environment.ts",
            "with": "src/environments/environment.android.ts"
          }
        ]
      },
    }
  }
}

```

Chapter 62. Build

Once configured, it is necessary to build the Ionic 4 app using this new configuration:

- `ionic build --configuration=android`

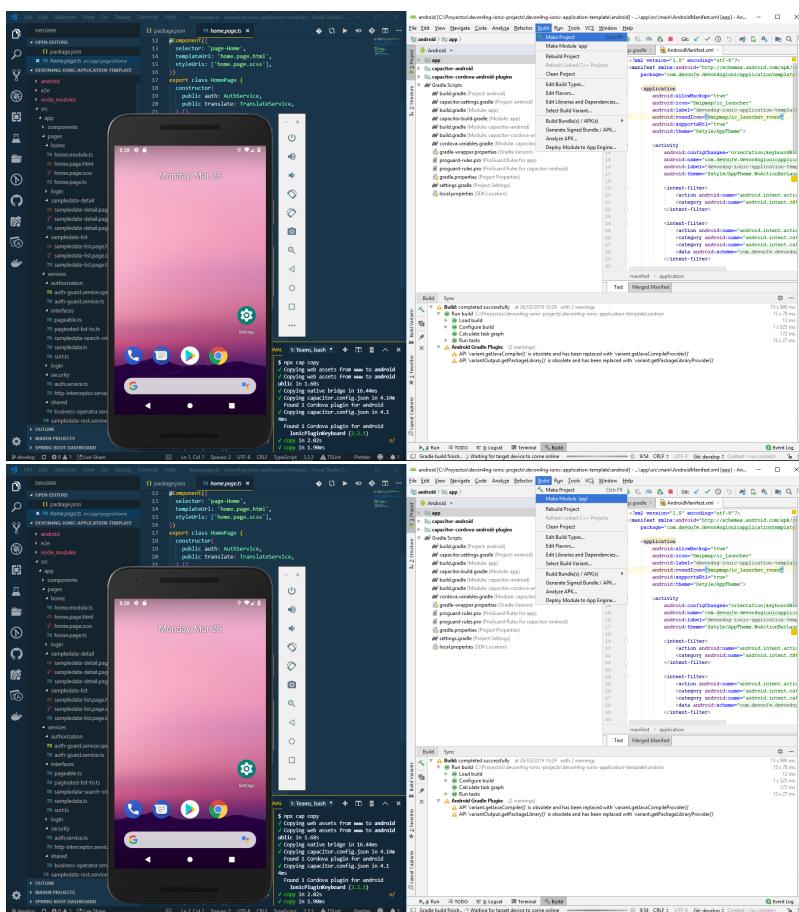
The next commands copy the build application on a folder named android and open android studio.

- `npx cap add android`
- `npx cap copy`
- `npx cap open android`

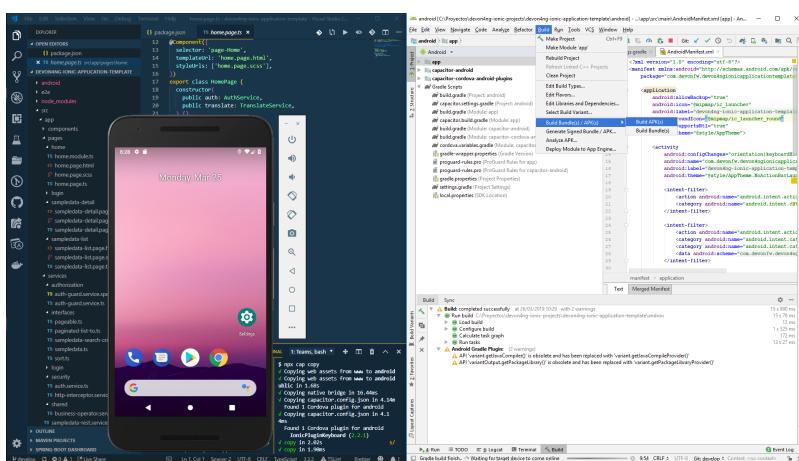
From Android project to emulated device

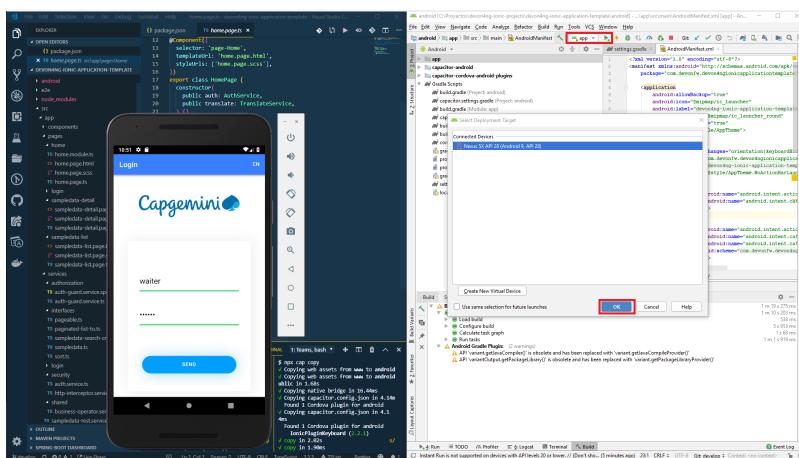
Once Android Studio is opened, follow these steps:

1. Click on "Build" → Make project.
2. Click on "Build" → Make Module 'app' (default name).



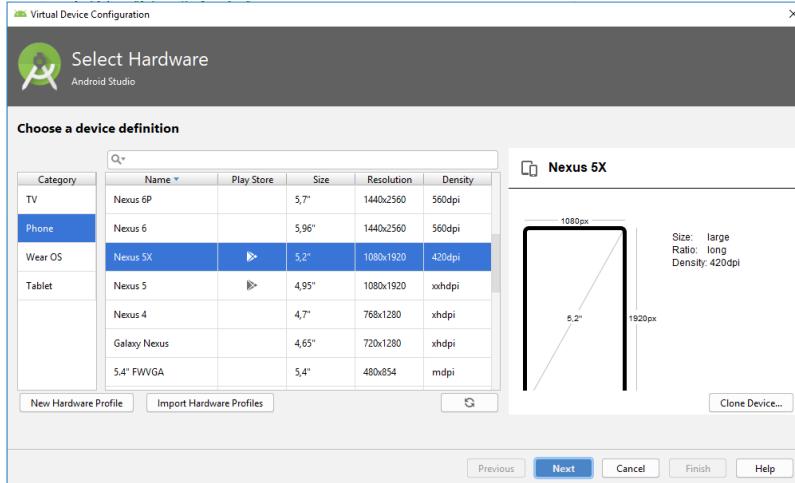
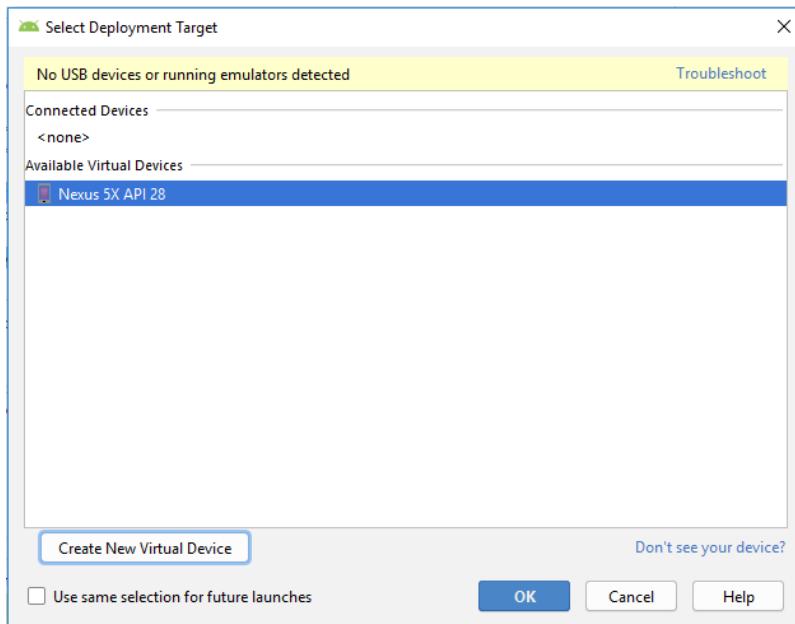
3. Click on "Build" → Build Bundle(s) / APK(s) → Build APK(s).
4. Click on run and choose a device.





If there are no devices available, a new one can be created:

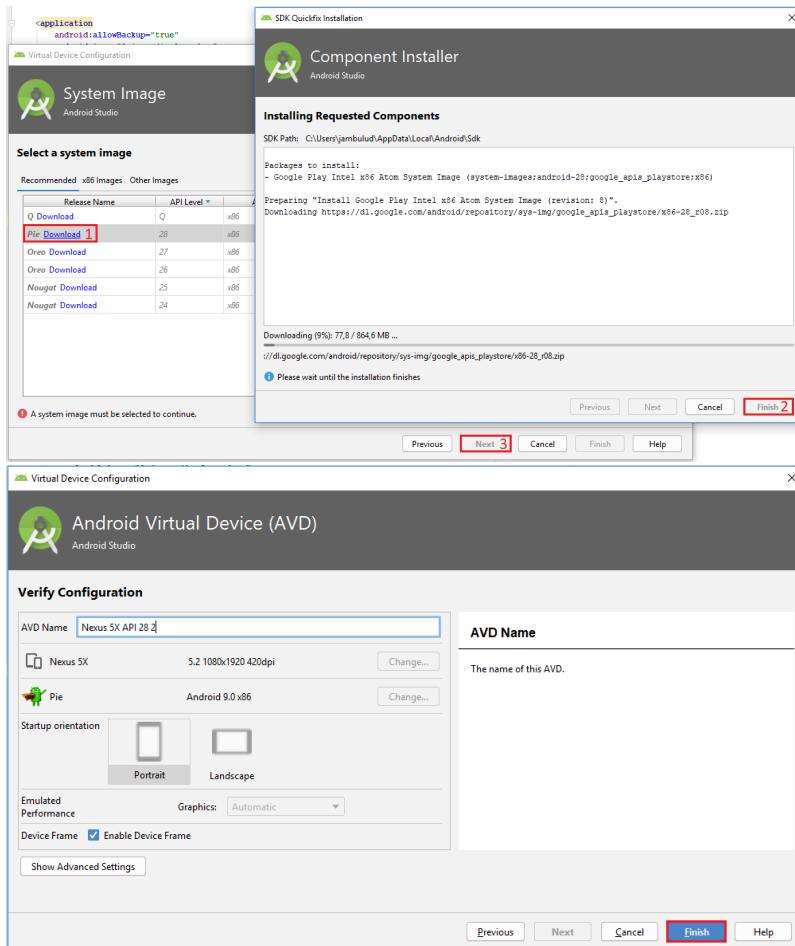
1. Click on "Create new device"
2. Select hardware and click "Next". For example: Phone → Nexus 5X.



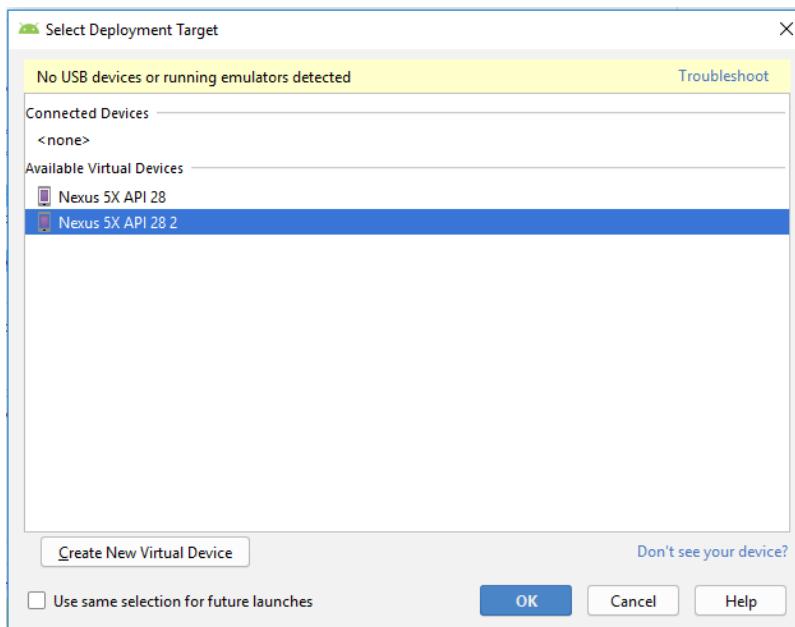
3. Download a system image.
 1. Click on download.
 2. Wait until the installation finished and then click "Finish".

3. Click "Next".

4. Verify configuration (default configuration should be enough) and click "Next".



5. Check that the new device is created correctly.



From Android project to real device

To test on a real android device, an easy approach to communicate a smartphone (front-end) and computer (back-end) is to configure a Wi-fi hotspot and connect the computer to it. A guide about this process can be found at <https://support.google.com/nexus/answer/9059108?hl=en>

Once connected, run `ipconfig` on a console if you are using windows or `ifconfig` on a linux machine to get the IP address of your machine's Wireless LAN adapter Wi-fi.

```
Connection-specific DNS Suffix . . . . .  
Wireless LAN adapter Wi-Fi:  
  Connection-specific DNS Suffix . . . :  
  Link-local IPv6 Address . . . . . : fe80::90a:3d66:7659:1963%17  
  IPv4 Address . . . . . : 192.168.43.17  
    Subnet Mask . . . . . : 255.255.255.0  
    Default Gateway . . . . . : 192.168.43.1  
  
Ethernet adapter Bluetooth Network Connection:  
  Media State . . . . . : Media disconnected  
  Connection-specific DNS Suffix . . :  
  
Tunnel adapter Teredo Tunneling Pseudo-Interface:  
  Media State . . . . . : Media disconnected  
  Connection-specific DNS Suffix . . :
```

This obtained IP must be used instead of "localhost" or "10.0.2.2" at environment.android.ts.



The screenshot shows a code editor with two tabs: `environment.android.ts` and `capacitor.config.json`. The `environment.android.ts` file contains the following code:

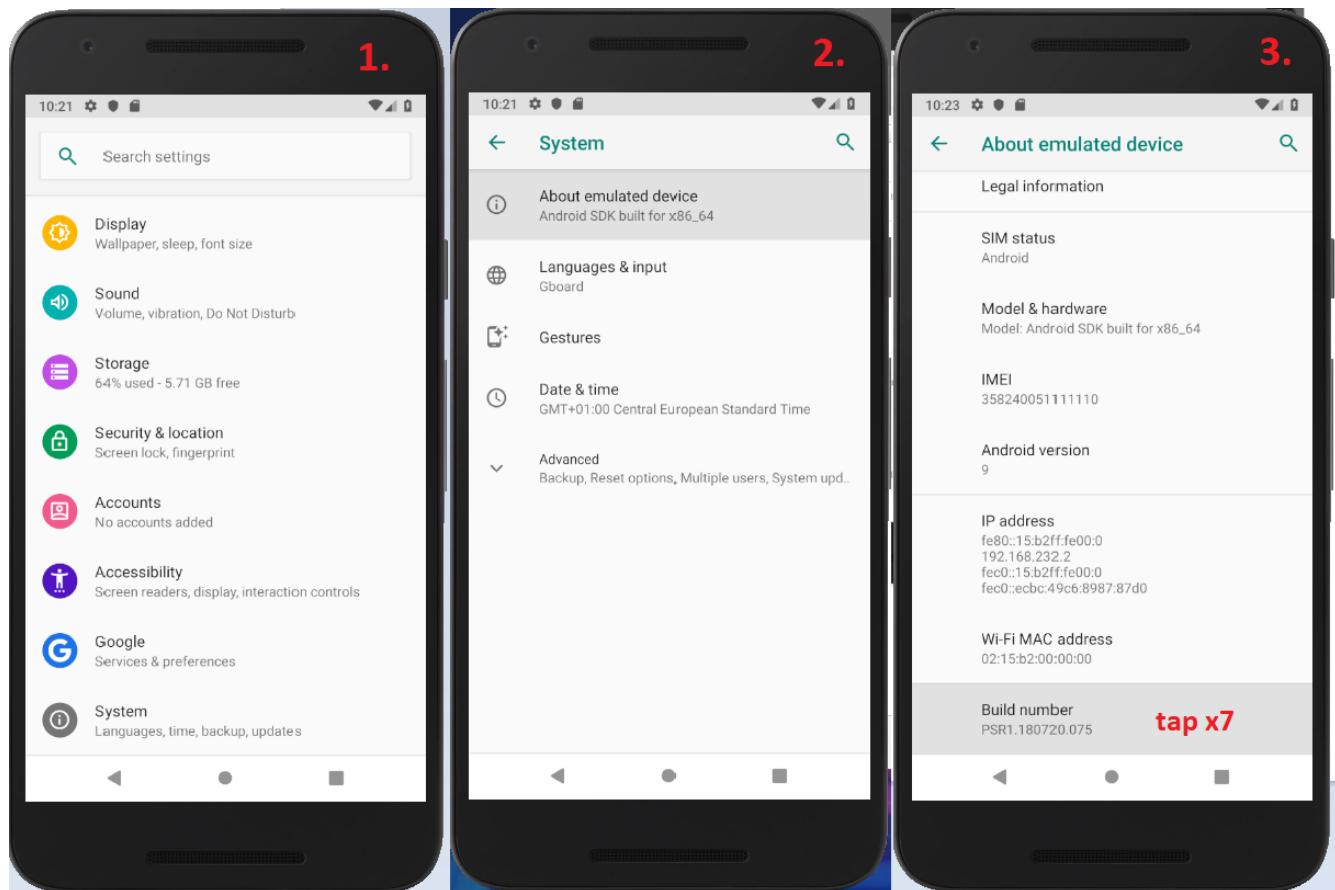
```
1 export const environment = {  
2   production: false,  
3 };  
4  
5 export const SERVER_URL = 'http://192.168.43.17:8081/';  
6
```

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".

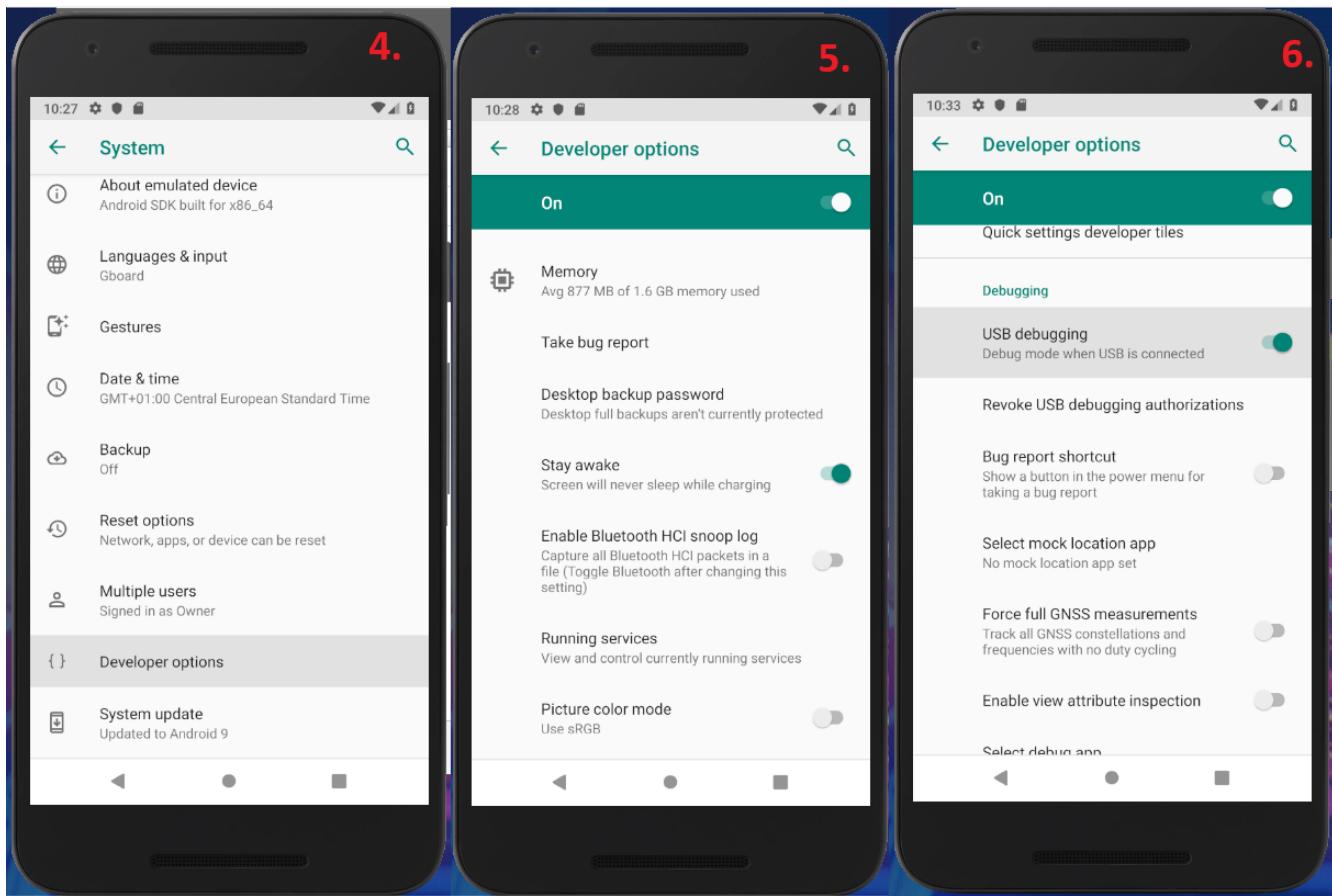
Chapter 63. Send APK to Android through USB

To send the built application to a device, you can connect computer and mobile through USB, but first, it is necessary to unlock developer options.

1. Open "Settings" and go to "System".
2. Click on "About".
3. Click "Build number" seven times to unlock developer options.



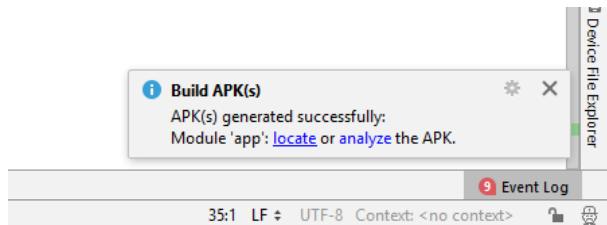
4. Go to "System" again and then to "Developer options"
5. Check that the options are "On".
6. Check that "USB debugging" is activated.



After this, do the step four in "From Android project to emulated device" and choose the connected smartphone.

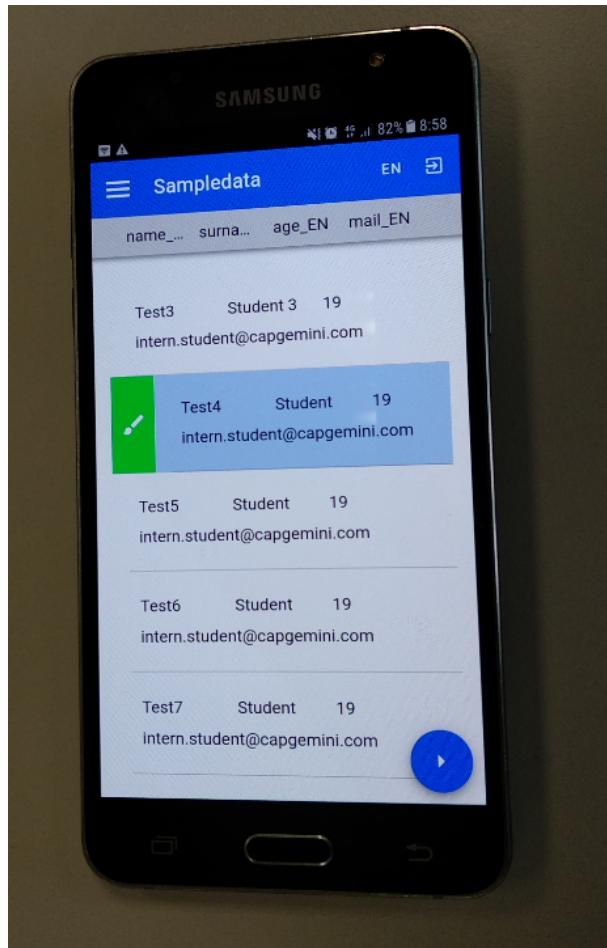
Chapter 64. Send APK to Android through email

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.



Result

If everything goes correctly, the Ionic 4 application will be ready to be tested.



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.

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.

Sample Application

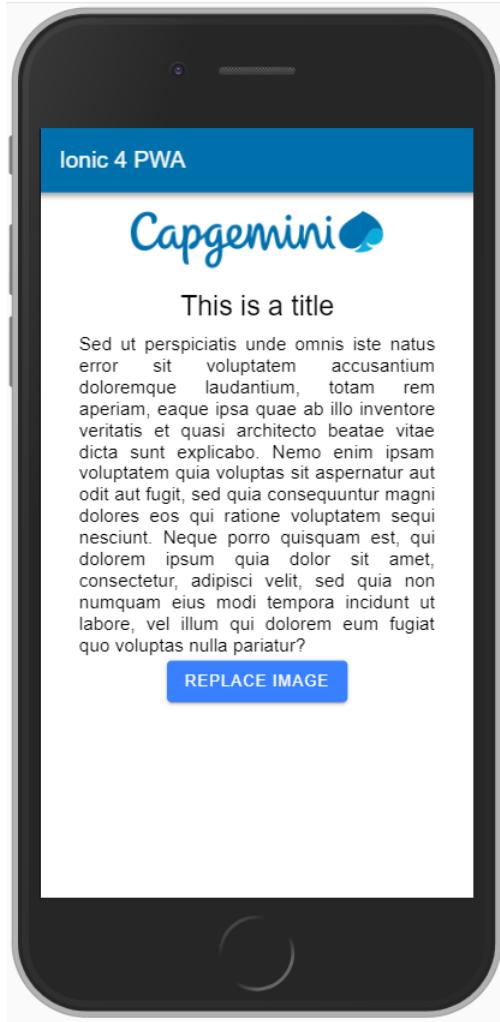


Figure 45. Basic ionic PWA.

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.

Chapter 65. 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**.

Chapter 66. 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](#).

Chapter 67. 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,
    });
  }
}
```

Chapter 68. 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);
```

Chapter 69. 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.

Chapter 70. 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/**"]
```

Chapter 71. 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.

```
C:\Proyectos\devon4ng-projects\sample-ion\basic-ion-pwa\www (master -> origin)
λ http-server -o
Starting up http-server, serving .
Available on:
  http://10.80.132.29:8081
  http://192.168.56.1:8081
  http://192.168.99.1:8081
  http://127.0.0.1:8081
Hit CTRL-C to stop the server
[Mon Apr 08 2019 08:35:37 GMT+0200 (GMT+02:00)] "GET /" "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/73.0.3683.86 Safari/537.36"
[Mon Apr 08 2019 08:35:42 GMT+0200 (GMT+02:00)] "GET /ngsw.json?ngsw-cache-bust=0366" "Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X) AppleWebKit/604.1 (KHTML, like Gecko) Version/11.0 Mobile/15A372 Safari/604.1"
```

Figure 46. 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 locally. It hasn't been reviewed for security issues.
[ng] Binding this server to an open connection can result in compromising your application or computer. Using a different host than the one passed to the "--host" flag might result in websocket connection issues. You might need to use "--disableHostCheck" if that's the 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 47. 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.

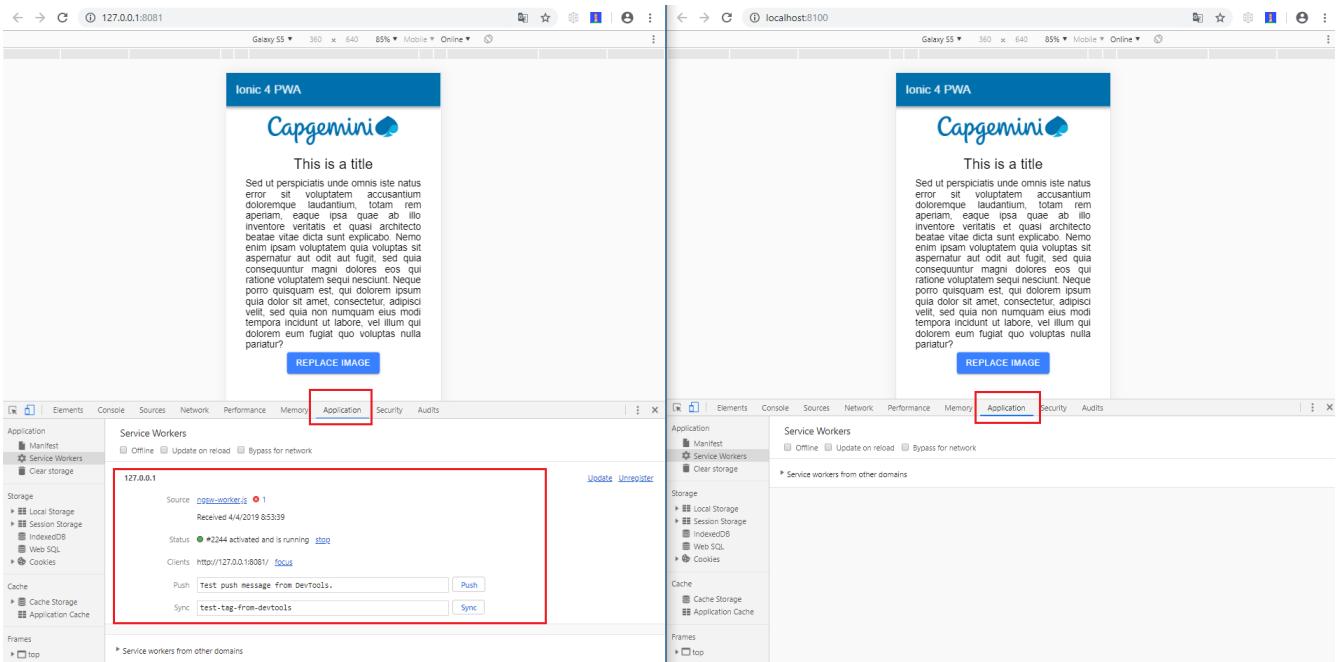


Figure 48. Application service worker comparison.

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.

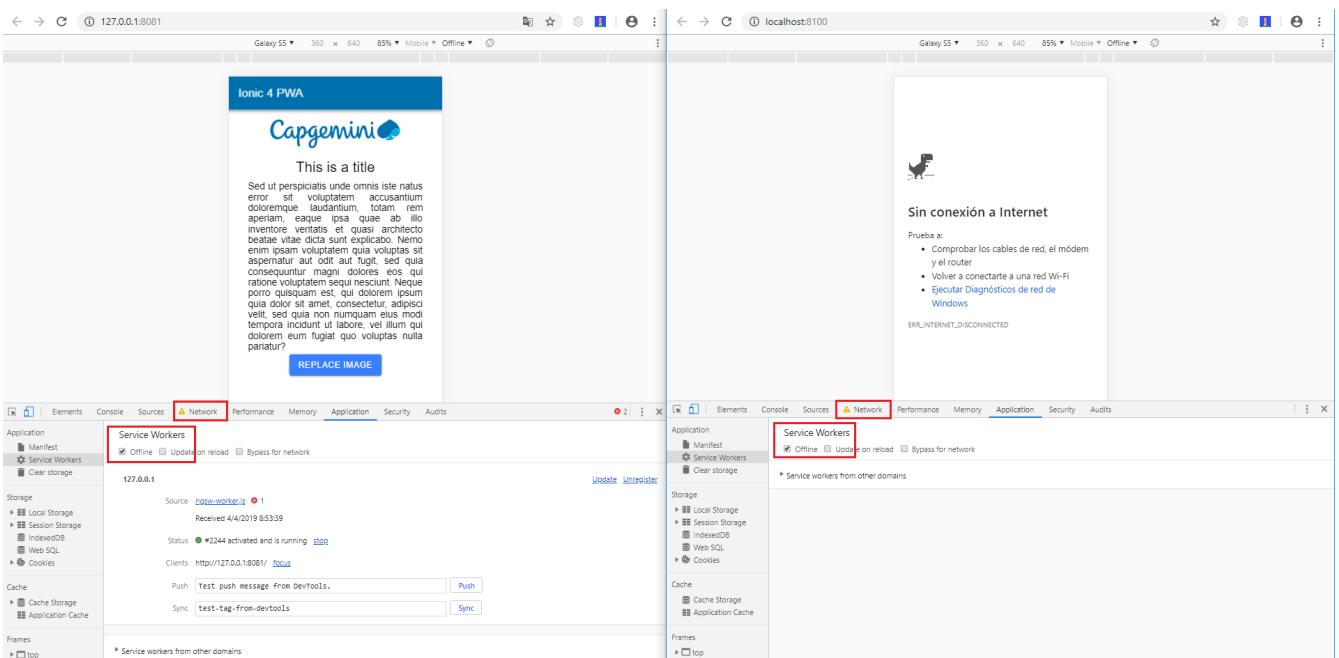


Figure 49. Offline application.

Finally, plugins like [Lighthouse](#) can be used to test whether an application is progressive or not.

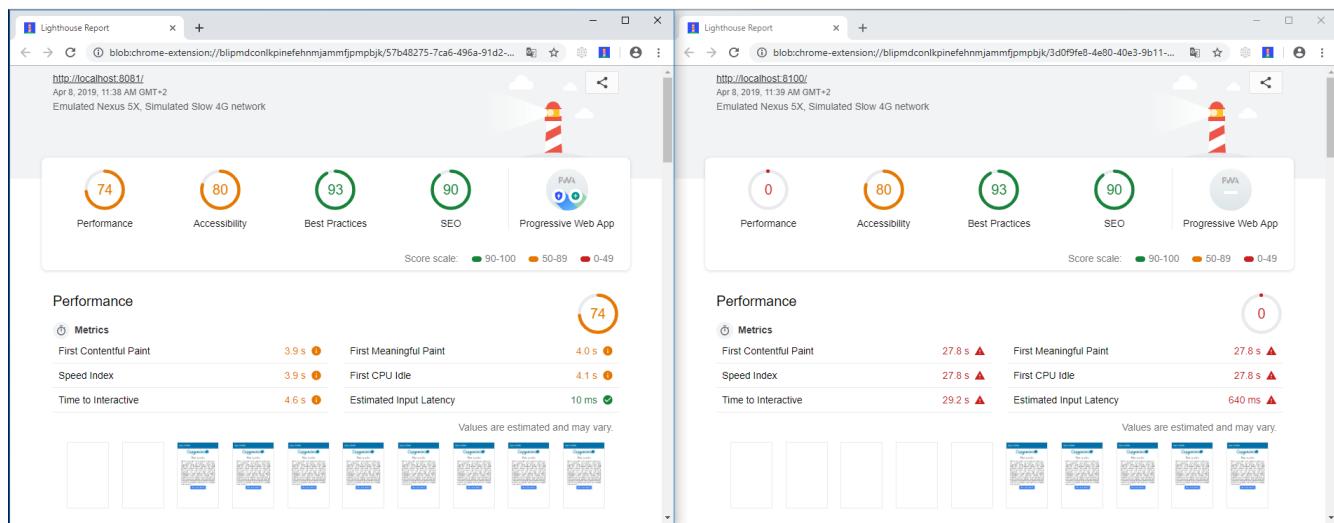


Figure 50. Lighthouse report.

71.1. Layouts

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.

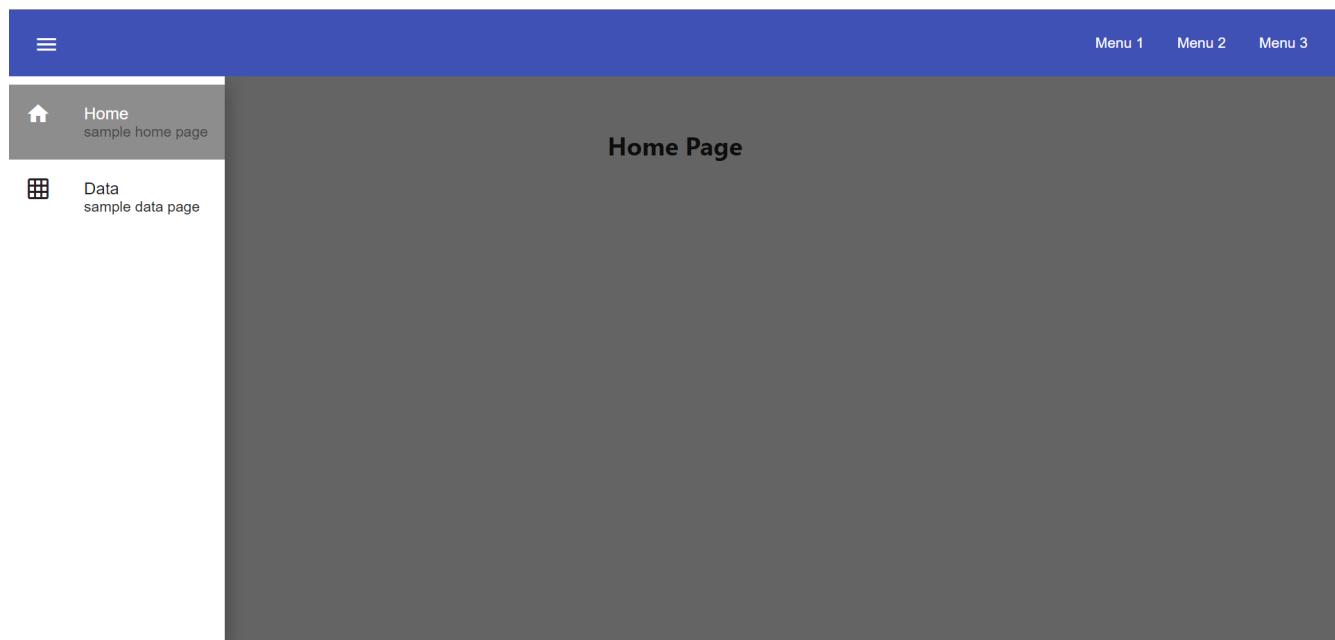


Figure 51. This is what the finished application will look like

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 52. Blank application

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 58. 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 59. 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 60. 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 61. In src/index.html:

```
<link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
```

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 62. 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 53. 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 63. 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 54. 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 64. `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 65. `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 55. 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 66. `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 67. 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 68. home.component.html:

```
<h2>Home Page</h2>
```

Listing 69. home.component.scss:

```
h2 {
  text-align: center;
  margin-top: 50px;
}
```

Listing 70. data.component.html:

```
<h2>Data Page</h2>
```

Listing 71. data.component.scss:

```
h2 {
  text-align: center;
  margin-top: 50px;
}
```

The pages look somewhat better now:



Figure 56. Home page



Figure 57. 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 72. 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 73. app.component.html:

```
...
<button mat-icon-button aria-label="menu" class="menu" (click)="sidenav.toggle()">
  <mat-icon>menu</mat-icon>
</button>
...
```

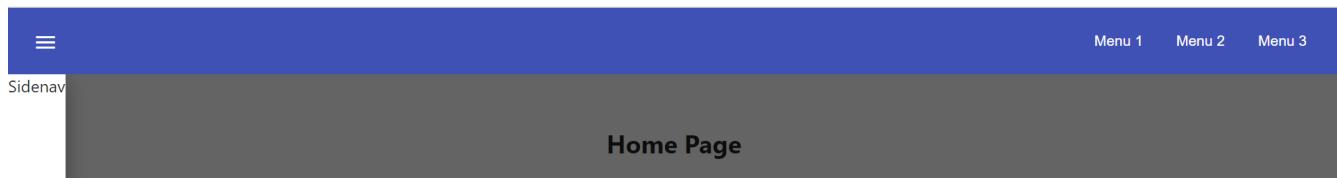


Figure 58. Sidenav is implemented

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 74. 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 75. 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 76. 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.

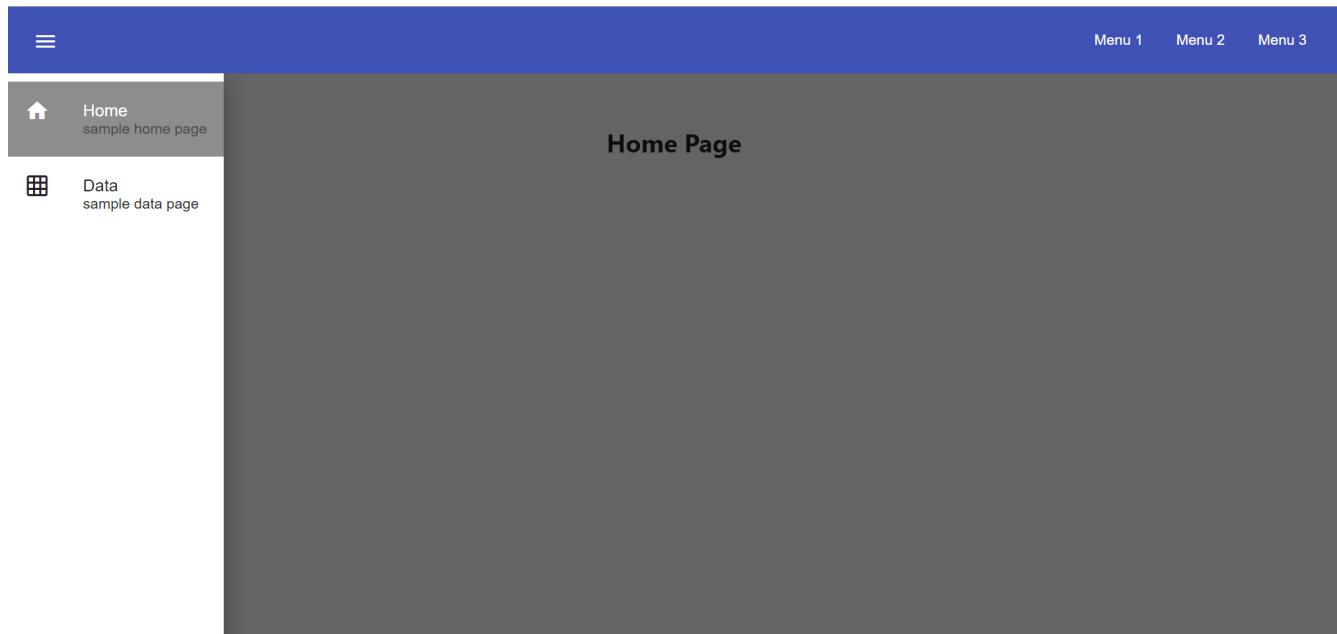


Figure 59. Finished application

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.

.1. NgRx

NgRx

NgRx is a state management framework for Angular based on the [Redux](#) pattern.

Chapter 72. 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 managment libraries for Angular provide tools for storing, retrieving, and updating client-side state.

Chapter 73. 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

Chapter 74. 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](#).

Chapter 75. Concept

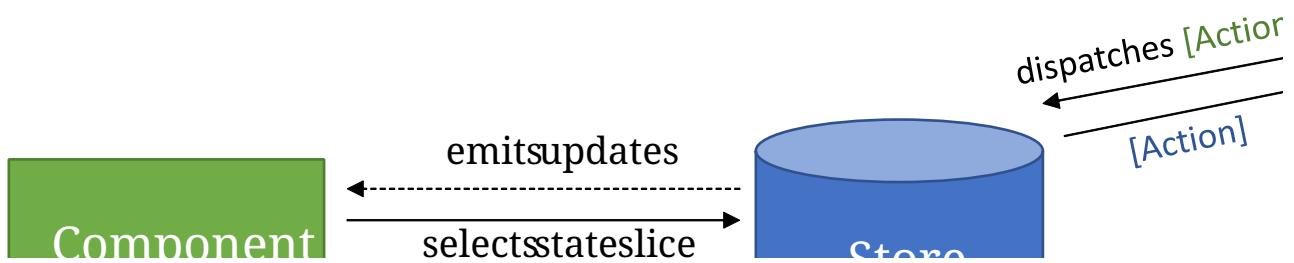


Figure 60. NgRx architecture overview

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](#).

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.

Chapter 76. 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 { }
```

Chapter 77. 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(),
    }
  ]
};
```

Chapter 78. 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>
```

Chapter 79. 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.

79.1. 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).

79.2. 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  
}));  
}
```

79.3. 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.

79.4. 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.

79.5. 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.

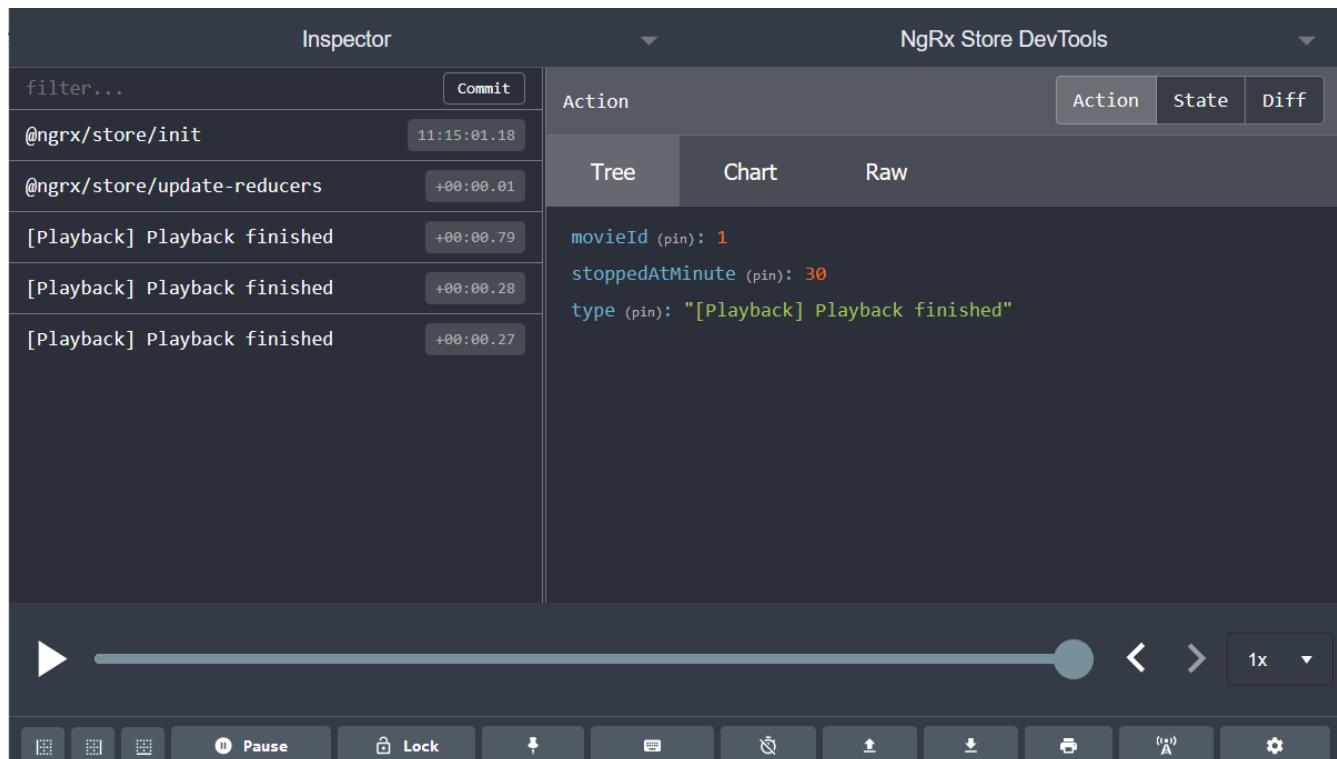


Figure 61. Redux devtools

Continue with [learning about effects](#)

Chapter 80. 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.

80.1. Obtaining the recommendation list from the server

Create a module for recommendations and add stores and states as in the previous chapter. Add `EffectsModule.forRoot([])` to the imports in `AppModule` below `StoreModule.forRoot()`. Add effects to the feature module:

```
ng generate effect recommendation/Recommendation -m  
recommendation/recommendation.module.ts
```

We need actions for loading the movie list, success and failure cases:

recommendation/actions/index.ts

```
import { createAction, props, union } from '@ngrx/store';
import { Movie } from 'src/app/watchlist/models/movies';

export const loadRecommendedMovies = createAction('[Recommendation List] Load
movies');
export const loadRecommendedMoviesSuccess = createAction('[Recommendation API] Load
movies success', props<{movies: Movie[]}>());
export const loadRecommendedMoviesFailure = createAction('[Recommendation API] Load
movies failure', props<{error: any}>());

const actions = union({
  loadRecommendedMovies,
  loadRecommendedMoviesSuccess,
  loadRecommendedMoviesFailure
});

export type ActionsUnion = typeof actions;
```

In the reducer, we use a loading flag so the UI can show a loading spinner. The store is updated with arriving data.

recommendation/actions/index.ts

```
export interface State {
  items: Movie[];
  loading: boolean;
}

export const initialState: State = {
  items: [],
  loading: false
};

export function reducer(state = initialState, action:
recommendationActions.ActionsUnion): State {
  switch (action.type) {
    case '[Recommendation List] Load movies':
      return {
        ...state,
        items: [],
        loading: true
      };

    case '[Recommendation API] Load movies failure':
      return {
        ...state,
        loading: false
      };

    case '[Recommendation API] Load movies success':
      return {
        ...state,
        items: action.movies,
        loading: false
      };

    default:
      return state;
  }
}

export const getAll = (state: State) => state.items;
export const isLoading = (state: State) => state.loading;
```

We need an API service to talk to the server. For demonstration purposes, we simulate an answer delayed by one second:

recommendation/services/recommendation-api.service.ts

```
@Injectable({
  providedIn: 'root'
})
export class Recommendation ApiService {

  private readonly recommendedMovies: Movie[] = [
    {
      id: 2,
      title: 'The Hunger Games',
      genre: 'sci-fi',
      releaseYear: 2012,
      runtimeMinutes: 144
    },
    {
      id: 4,
      title: 'Avengers: Endgame',
      genre: 'fantasy',
      releaseYear: 2019,
      runtimeMinutes: 181
    }
  ];

  loadRecommendedMovies(): Observable<Movie[]> {
    return of(this.recommendedMovies).pipe(delay(1000));
  }
}
```

Here are the effects:

recommendation/services/recommendation-api.service.ts

```
@Injectable()
export class RecommendationEffects {

  constructor(
    private actions$: Actions,
    private recommendationApi: Recommendation ApiService,
  ) { }

  @Effect()
  loadBooks$ = this.actions$.pipe(
    ofType(recommendationActions.loadRecommendedMovies.type),
    switchMap(() => this.recommendationApi.loadRecommendedMovies().pipe(
      map(movies => recommendationActions.loadRecommendedMoviesSuccess({ movies })),
      catchError(error => of(recommendationActions.loadRecommendedMoviesFailure({
        error })))
    ))
  );
}
```

Effects are always observables and return actions. In this example, we consume the actions observable provided by NgRx and listen only for the `loadRecommendedMovies` actions by using the `ofType` operator. Using `switchMap`, we map to a new observable, one that loads movies and maps the successful result to a new `loadRecommendedMoviesSuccess` action or a failure to `loadRecommendedMoviesFailure`. In a real application we would show a notification in the error case.



If an effect should not dispatch another action, return an empty observable.

Continue reading how to simplify CRUD (Create Read Update Delete) operations using [@ngrx/entity](#).

Chapter 81. 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](#).

81.1. Cookbook

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 77. 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 78. 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 newProbs) {  
    result[prop] = { from: undefined, to: (<any>after)[prop] };  
}  
  
for (const prop of sameProbs) {  
    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();  
}  
}
```

Chapter 82. 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.

82.1. 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
```

82.2. 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.

82.3. 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.

82.3.1. 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"  
  ]  
  ....  
}
```

82.3.2. 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"  
  ]  
}
```

82.4. Modify angular.json

`angular.json` has to be modified so the project is build inside `/dist` without an intermediate folder.

```
{  
  ....  
  "architect": {  
    ....  
    "build": {  
      outputPath": "dist",  
      ....  
    }  
  }  
}
```

82.5. 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 79. File webview.directive.ts

```
import { Directive } from '@angular/core';

@Directive({
  selector: '[webview]',
})
export class WebviewDirective {}
```

82.6. 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**

82.6.1. Update `app.module.ts` and `app-routing.module.ts`

As an example, the `webview.directive.ts` file is located inside a `shared` module:

Listing 80. 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,  
},  
)],
```

82.6.2. 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](#).

82.7. 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;
}
```

82.8. 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
```

Chapter 83. devon4net documentation

83.1. 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:

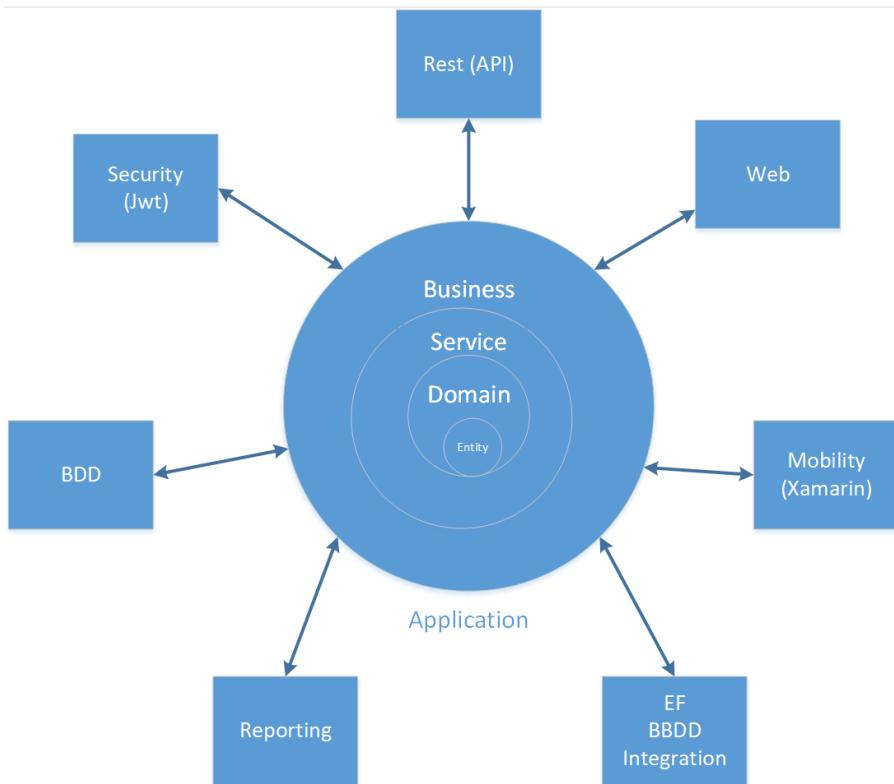


Figure 62. High level architecture representation

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.

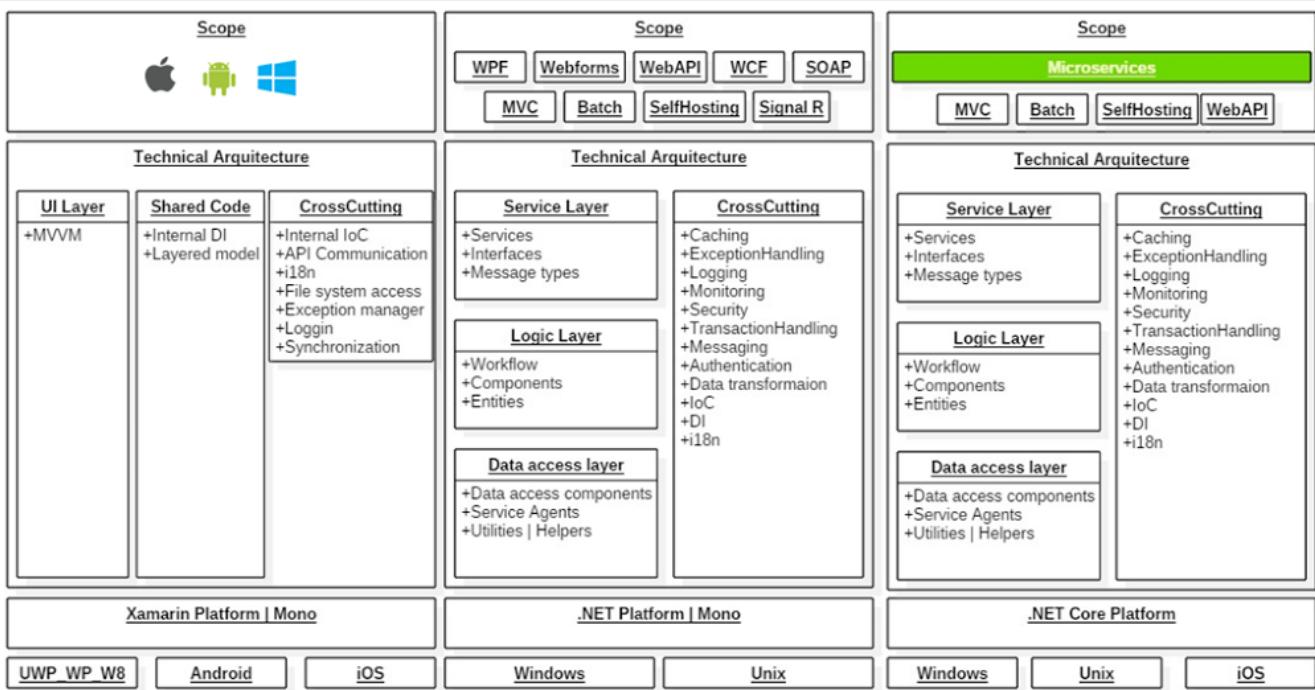


Figure 63. Net application stack

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.

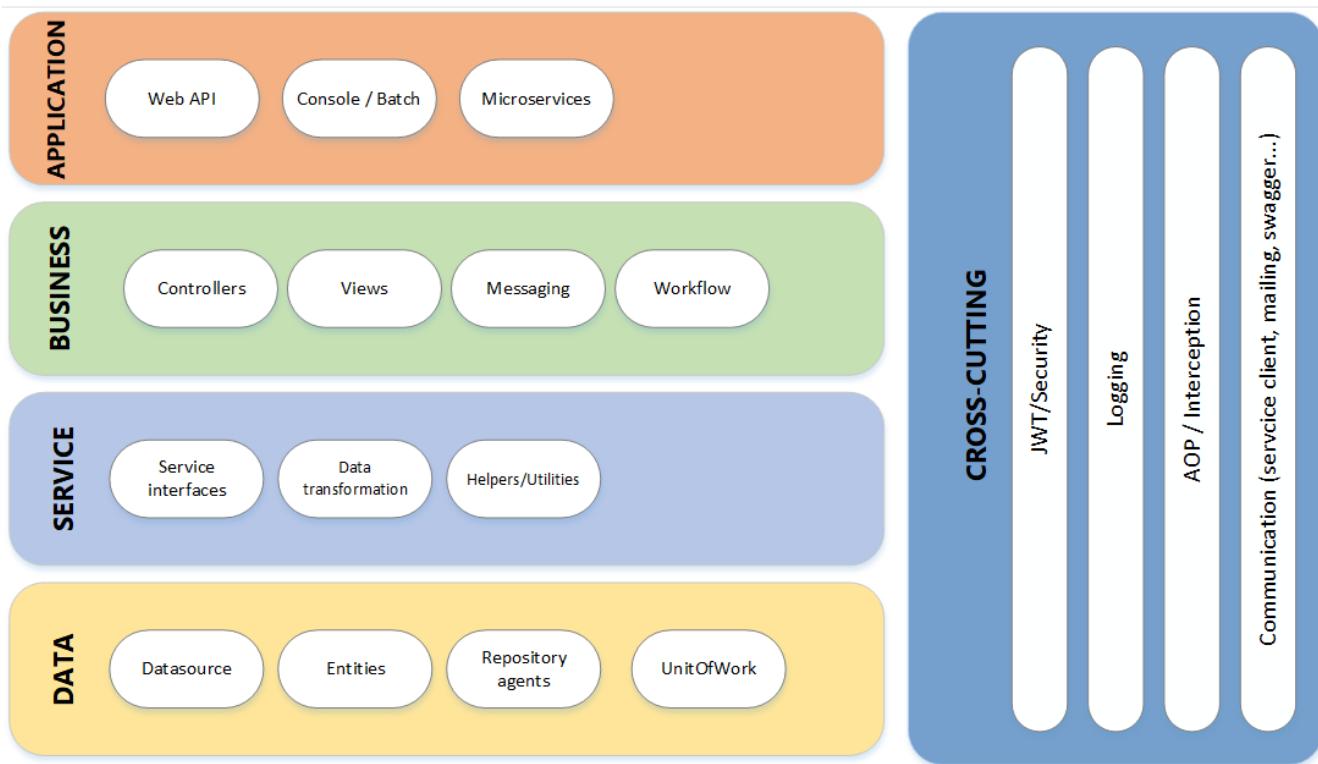


Figure 64. Architecture representation in deep

[[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 65. Current available functionality

[[architecture_guide.asciidoc_navy#software-stack#]] === Software stack

Table 3. 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

83.2. 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

83.2.1. 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.

83.2.2. 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.

83.2.3. 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 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
Property	Pascal case. Try to use get and set convention {get;set;}
Constant	Pascal case
Enum	Pascal case

Case	Convention
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

83.3. 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](#)

83.4. User guide



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.

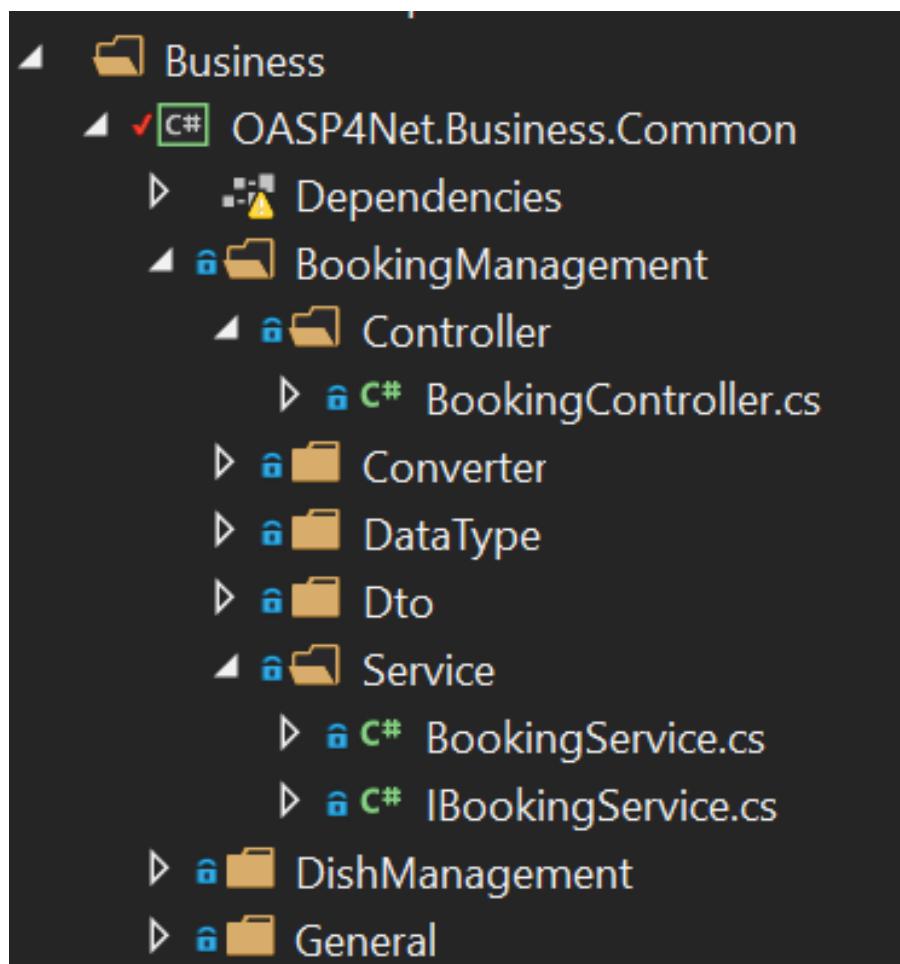


Figure 66. Screenshot of OASP4Net.Business.Common project in depth

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. Authentication fail</response>
/// <response code="403">Forbidden. Authorization error.</response>
/// <response code="500">Internal Server Error. The search process ended with
error.</response>
[HttpPost]
[Route("/mythaistar/services/rest/bookingmanagement/v1/booking/search")]
//[Authorize(Policy = "MTSWaiterPolicy")]
[AllowAnonymous]
[EnableCors("CorsPolicy")]
public async Task<IActionResult> BookingSearch([FromBody]BookingSearchDto
bookingSearchDto)
{

```

In order to be 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:

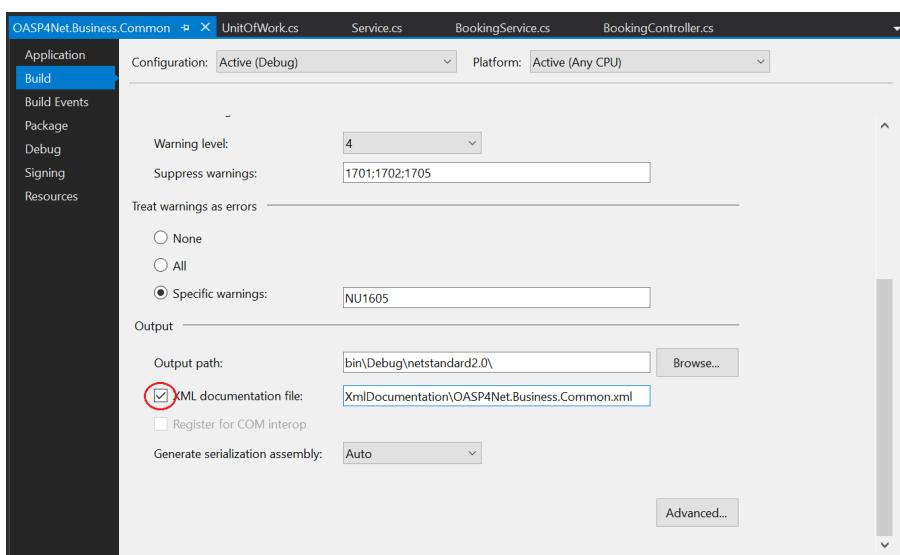


Figure 67. Project settings tab

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*.

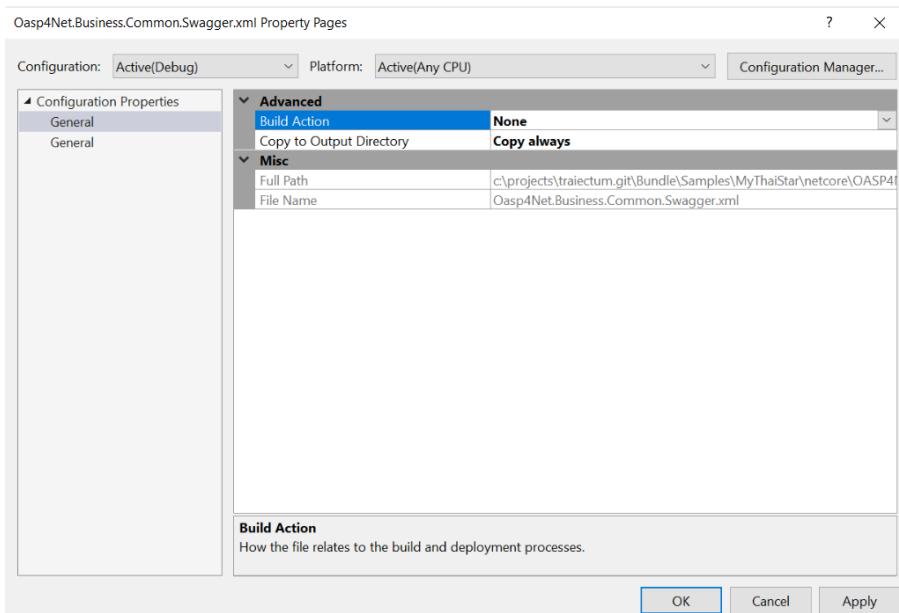


Figure 68. Swagger XML document file properties

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 querys via HTTP.

The screenshot shows the Seq application interface. The left pane displays a list of log entries from February 5, 2018, at 15:41:42.199 to 15:40:36.320. The right pane contains navigation and configuration options for signals and queries.

Date	Log Message
05 Feb 2018 15:41:42.199	Request finished in 1044.0852ms 200 text/plain; charset=utf-8
05 Feb 2018 15:41:42.167	Executing ObjectResult, writing value Microsoft.AspNetCore.Mvc.ControllerContext.
05 Feb 2018 15:41:41.620	Executed DbCommand (5ms) [Parameters=[], CommandType='Text', CommandTimeout='30'] SELECT [d.DishIngredie...
05 Feb 2018 15:41:41.614	Executed DbCommand (56ms) [Parameters=[], CommandType='Text', CommandTimeout='30'] SELECT [d.DishCateg...
05 Feb 2018 15:41:41.416	Executed DbCommand (258ms) [Parameters=[], CommandType='Text', CommandTimeout='30'] SELECT [d].[Id], [d].[D...
05 Feb 2018 15:41:41.157	Entity Framework Core 2.0.0-rtm-26452 initialized 'ModelContext' using provider 'Microsoft.EntityFrameworkCore.Sql...
05 Feb 2018 15:41:41.155	Request starting HTTP/1.1 POST http://10.0.2.2:8081/mythaistar/services/rest/dishmanagement/v1/dish/search a...
05 Feb 2018 15:40:46.651	Request finished in 1097.0648ms 200 text/plain; charset=utf-8
05 Feb 2018 15:40:46.643	Executing ObjectResult, writing value Microsoft.AspNetCore.Mvc.ControllerContext.
05 Feb 2018 15:40:46.056	Executed DbCommand (5ms) [Parameters=[], CommandType='Text', CommandTimeout='30'] SELECT [d.DishIngredie...
05 Feb 2018 15:40:46.050	Executed DbCommand (70ms) [Parameters=[], CommandType='Text', CommandTimeout='30'] SELECT [d.DishCateg...
05 Feb 2018 15:40:45.840	Executed DbCommand (250ms) [Parameters=[], CommandType='Text', CommandTimeout='30'] SELECT [d].[Id], [d].[D...
05 Feb 2018 15:40:45.588	Entity Framework Core 2.0.0-rtm-26452 initialized 'ModelContext' using provider 'Microsoft.EntityFrameworkCore.Sql...
05 Feb 2018 15:40:45.554	Request starting HTTP/1.1 POST http://10.0.2.2:8081/mythaistar/services/rest/dishmanagement/v1/dish/search a...
05 Feb 2018 15:40:36.339	Request finished in 1543.1441ms 200 text/plain; charset=utf-8
05 Feb 2018 15:40:36.320	Executing ObjectResult, writing value Microsoft.AspNetCore.Mvc.ControllerContext.

By default you can find the log information at *Logs* folder.

[[userguide.asciidoc_navy#jwt-module#]] === JWT module

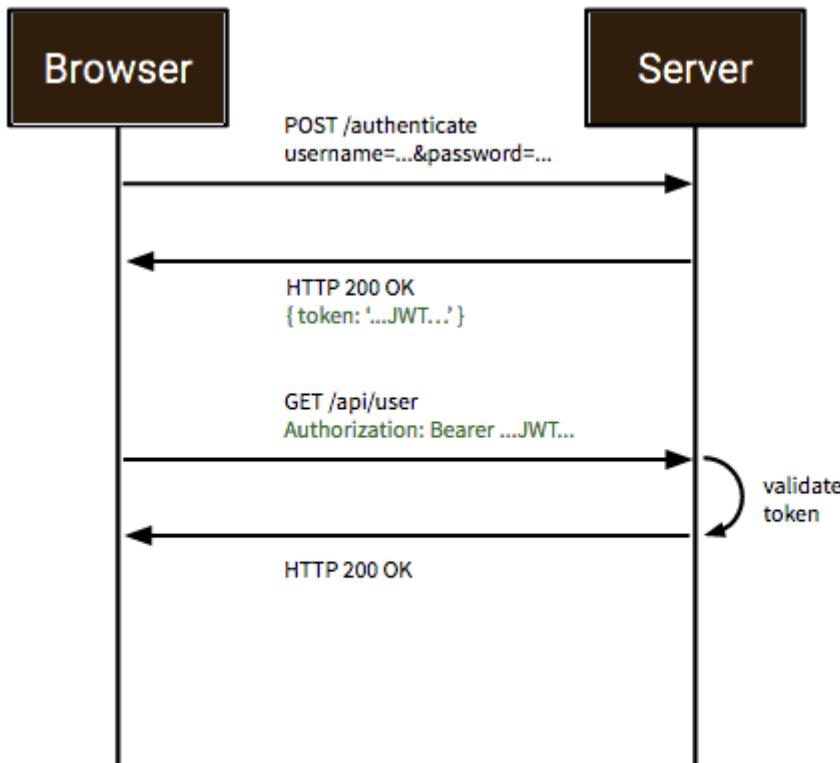
JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties allowing you to decode, verify and generate JWT.

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;
}
```

.1. Cheat Sheet

Operation	Example
Test	[Fact] public void Test() { }
Setup	public class TestFixture { public TestFixture() { ... } }
Teardown	public class TestFixture : IDisposable { public void Dispose() { ... } }

.2. 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*

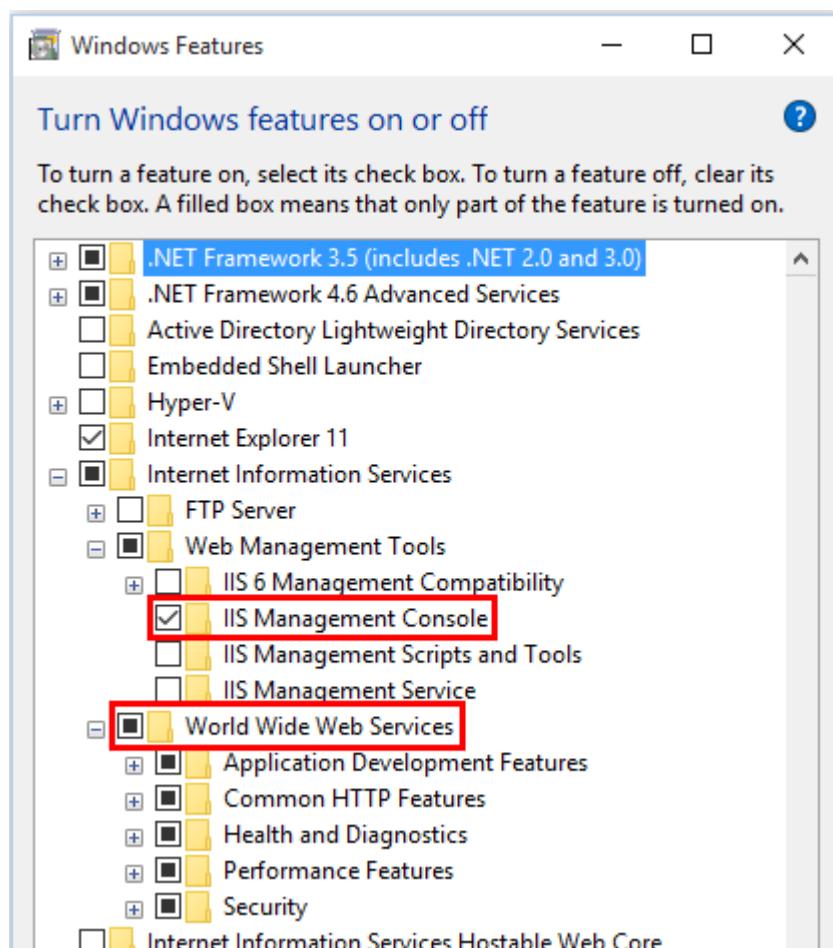
WebListener server will not work in a reverse-proxy configuration with IIS. You must use the [Kestrel server](#).

IIS configuration

Enable the Web Server (IIS) role and establish role services.

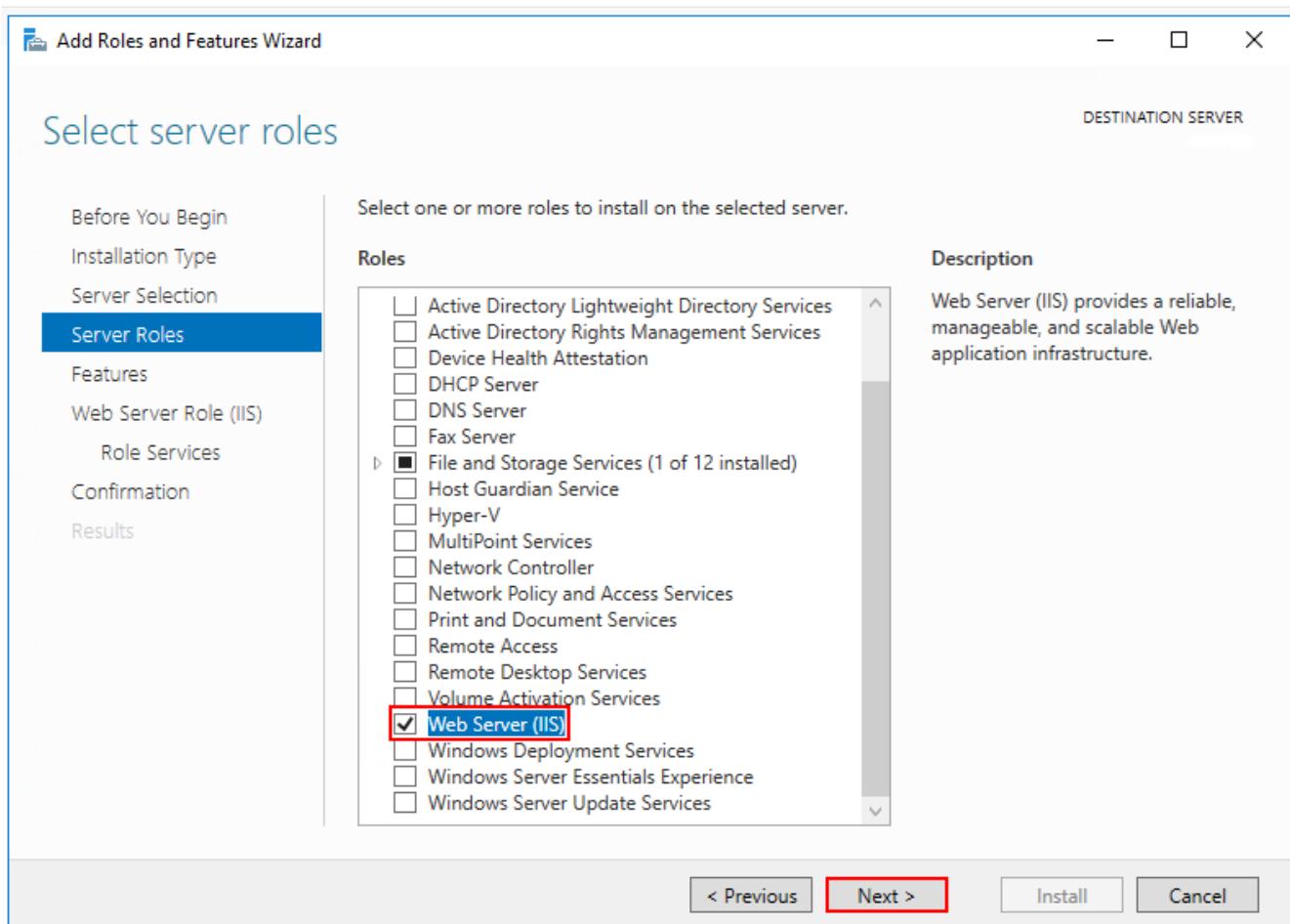
Windows desktop operating systems

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.

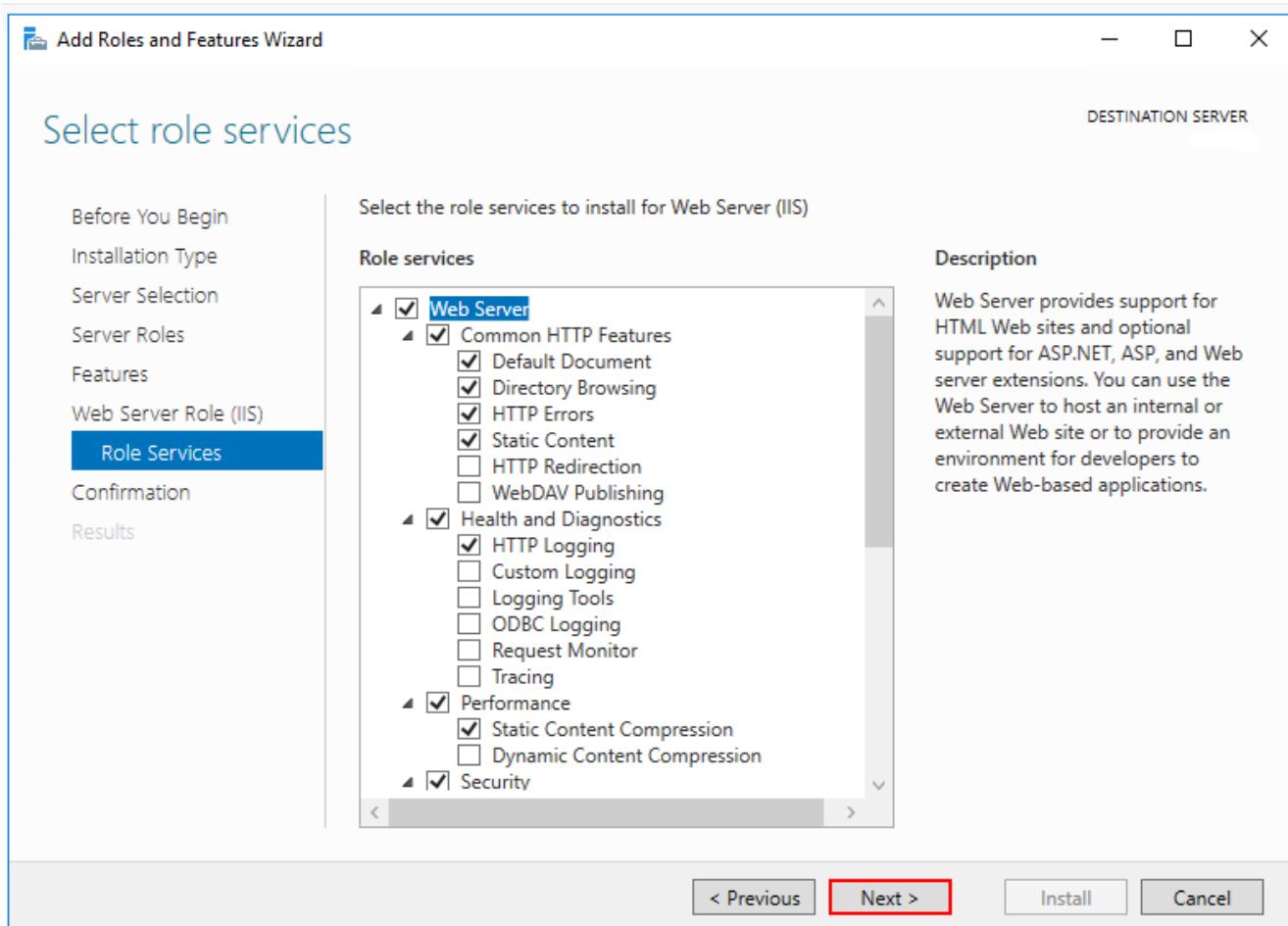


*Conceptually, the IIS configuration described in this document also applies to hosting ASP.NET Core applications on Nano Server IIS, but refer to [ASP.NET Core with IIS on Nano Server](#) for specific instructions.

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](#)

.3. Packages

Packages overview



OASP4Net is composed by a number of packages that increases the functionality and boosts time development. Each package has it's own configuration to make them work properly. In `appsettings.json` set up your environment. On `appsettings.{environment}.json` you can configure each component.

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 69. 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

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 a efficient way and not retrieving data each time you apply an expression.

- Example from My Thai Star .Net Core implementation:

```

public async Task<PaginationResult<Dish>> GetpagedDishListFromFilter(int currentPage,
int pageSize, bool isFav, decimal maxPrice, int minLikes, string searchBy, IList<long>
categoryIdList, long userId)
{
    var includeList = new
List<string>{"DishCategory", "DishCategory.IdCategoryNavigation",
"DishIngredient", "DishIngredient.IdIngredientNavigation", "IdImageNavigation"};

    //Here we create our predicate builder
    var dishPredicate = PredicateBuilder.True<Dish>();

    //Now we start applying the different 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 asynchrhonous 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 70. Swagger documentation

- Ensure that your XML files have the attribute copy always to true:

Figure 71. 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.

Required software

[Visual Studio Code](#)

[C# Extension for VS Code](#)

[.Net Core SDK](#)

[CORS in .Net Core](#)

.1. Templates

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 72. 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](#)

.1. Samples

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](#)

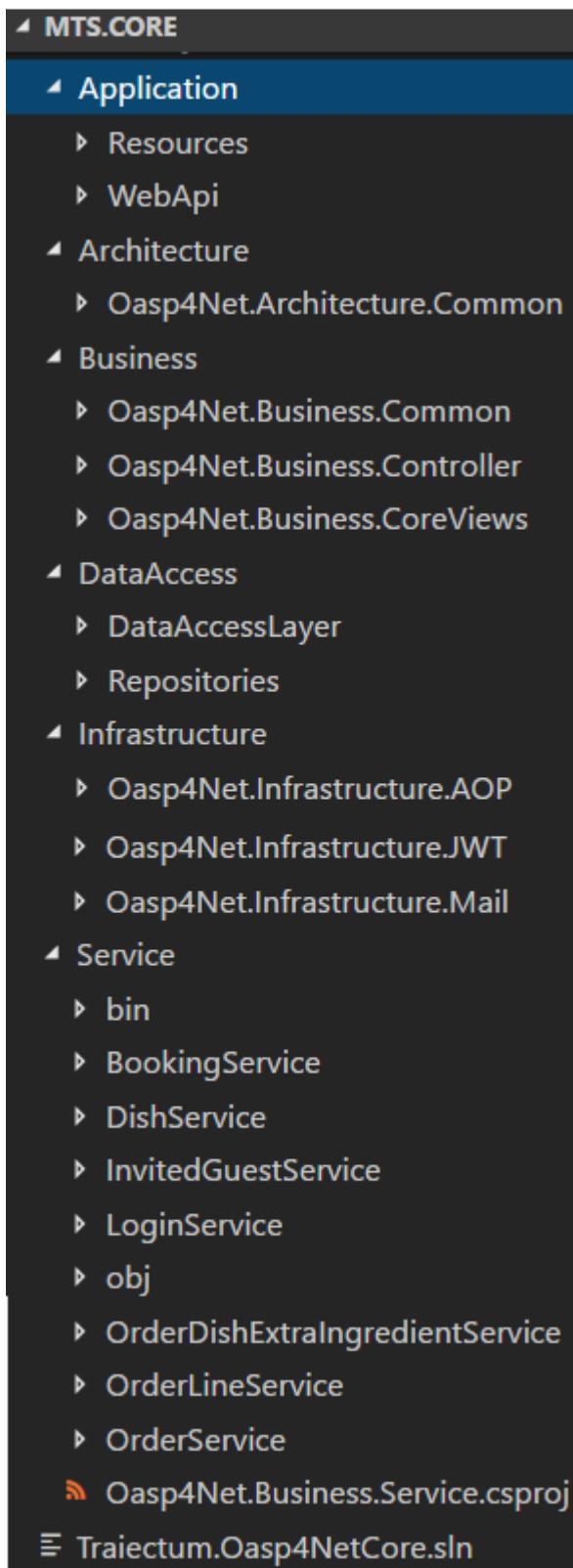
[[samples.asciidoc_navy#angular-client]] === Angular client

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

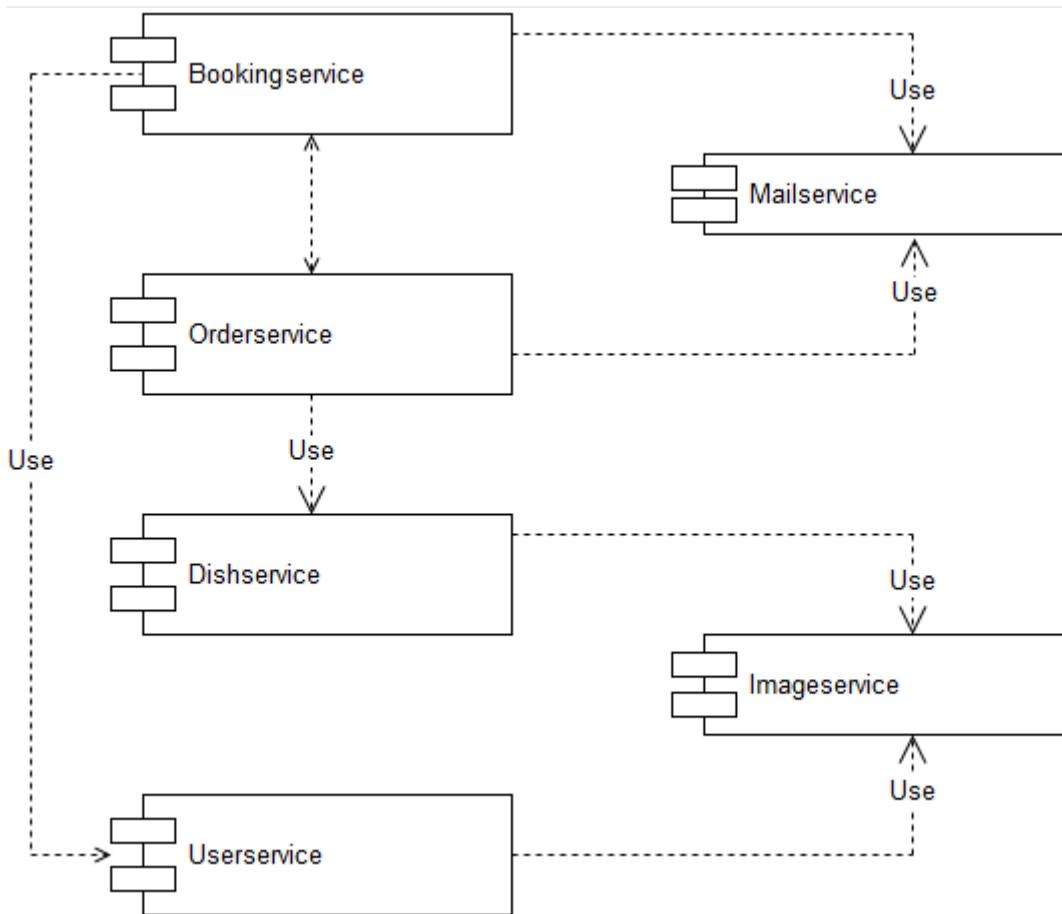
[[samples.asciidoc_navy#basic-architecture-details]] === Basic architecture details

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.

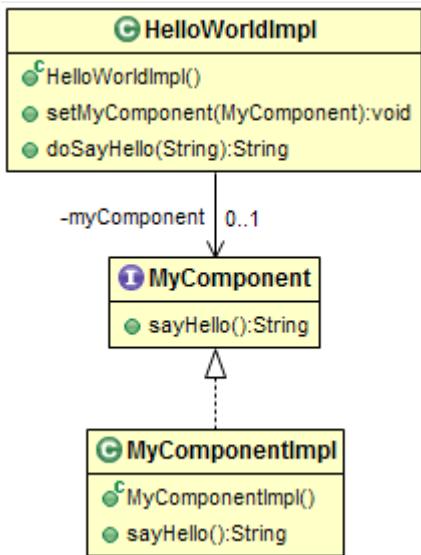
YYou can read more about _Authorization on:

[Authorization in ASP.NET Core](#)

[Claim based authorization](#)

[[samples.asciidoc_navy#dependency-injection#]] === Dependency injection

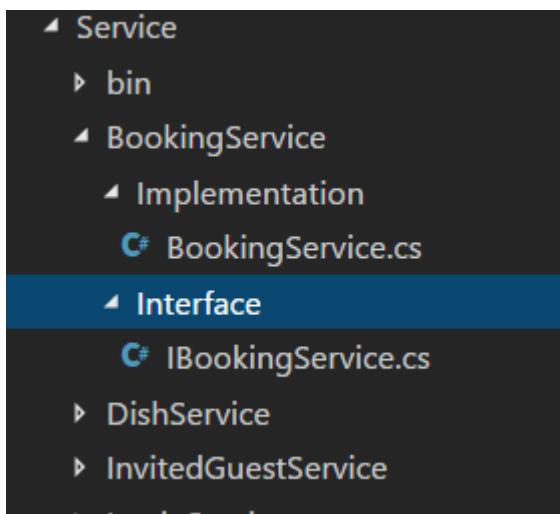
As it is explained in the [Microsoft documentation](#) we are going to implement the *dependency injection* pattern basing our solution on *.Net Core*.



- 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!



My Thai Star Email Service

Email

Show/Hide | List Operations | Expand Operations

POST /api/Email

[BASE URL: , API VERSION: v1]

Figure 73. 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](#)

Chapter 84. devonfw shop floor documentation

Unresolved directive in master.asciidoc - include::devonfw-shop-floor.wiki/master-devonfw-shop-floor.asciidoc[]

Chapter 85. cicdgen documentation

Unresolved directive in master.asciidoc - include::cicdgen.wiki/master-cicdgen.asciidoc[]

Chapter 86. devonfw testing with MrChecker documentation

Chapter 87. What is E2E Mr Checker Test Framework?

87.1. Home



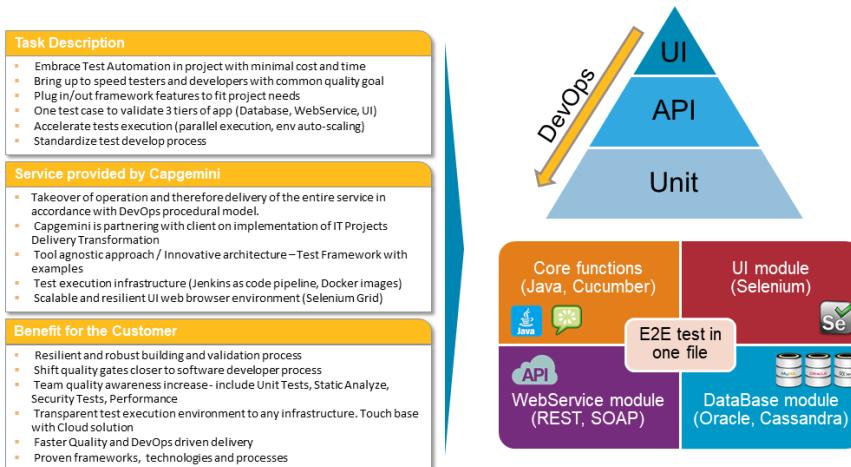
87.1.1. Contact

In case of any questions, please send mail to: DL PL E2E Test Framework <e2etestframework.pl@capgemini.com>

87.1.2. What is E2E Mr Checker Test Framework

Mr Checker Test Framework is the end to end test automation framework written in Java.

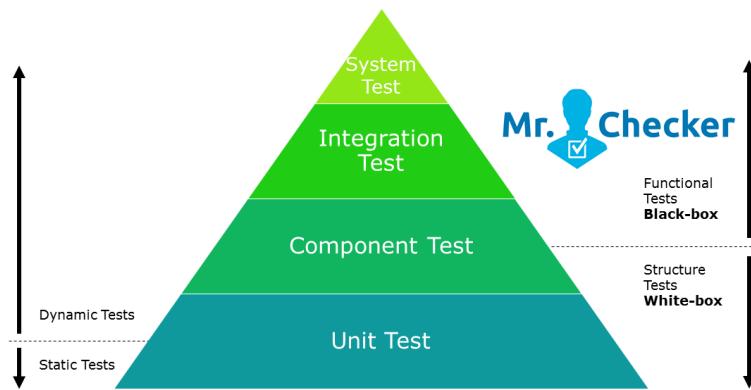
E2E Test Framework for DevOps & Smart Automation



87.1.3. 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

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).

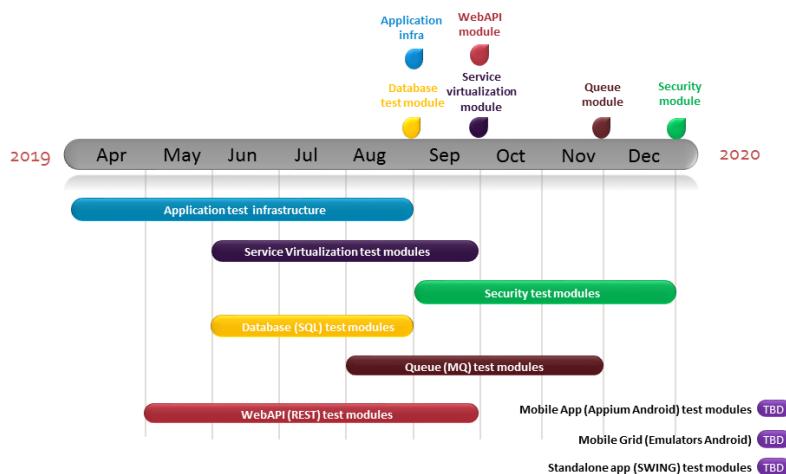
87.1.4. Benefits for the project

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.

87.1.5. Road map plan

Mr Checker E2E Framework road map



All slides: [link](#)

87.1.6. 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

Chapter 88. How to install Mr Checker?

88.1. 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.

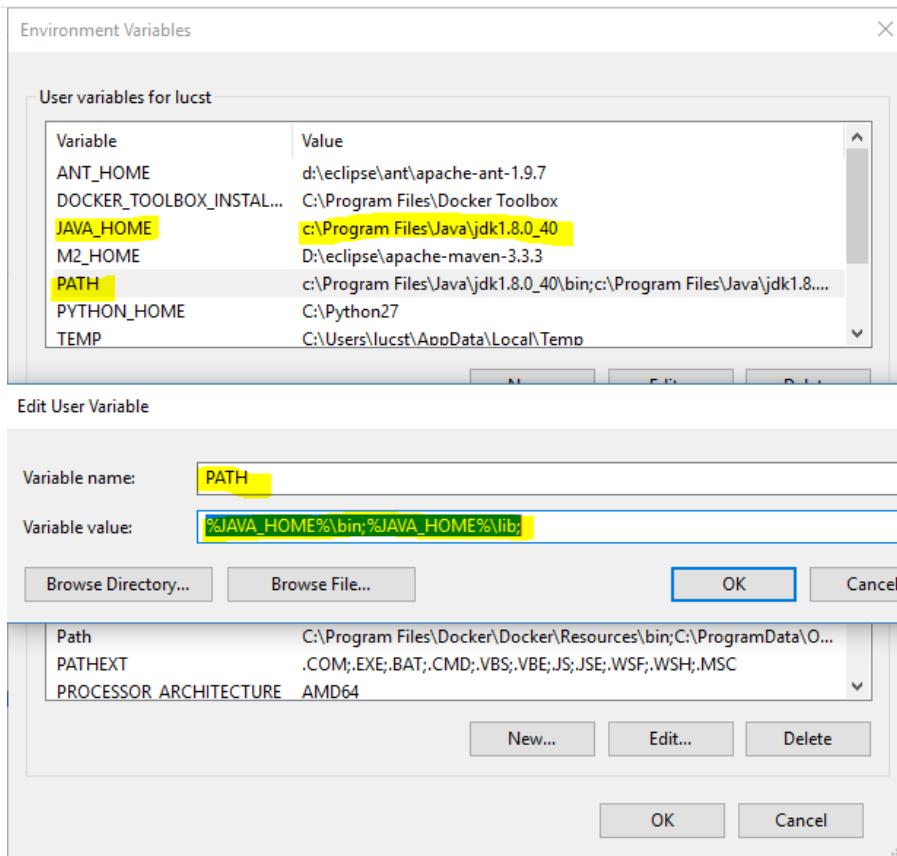
1. Install Java 1.8 JDK 64bit

- 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.		
<input checked="" type="radio"/> Accept License Agreement	<input type="radio"/> Decline License Agreement	
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:

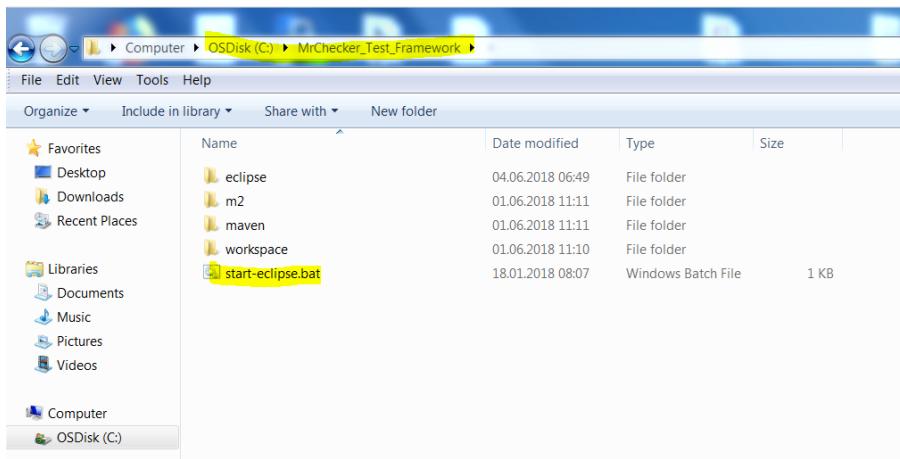
- **Easy out of the box installation** - Fast and easy solution - recommended for all users, who had previously not used any test automation environment. A drawback of these solutions is that it applies not for all Operation Systems
- **Out of the box installation** - with additional steps - when the first way is not working for you
- **Advanced installation** - Manual step by step installation containing all framework ingredients

Easy out of the box installation

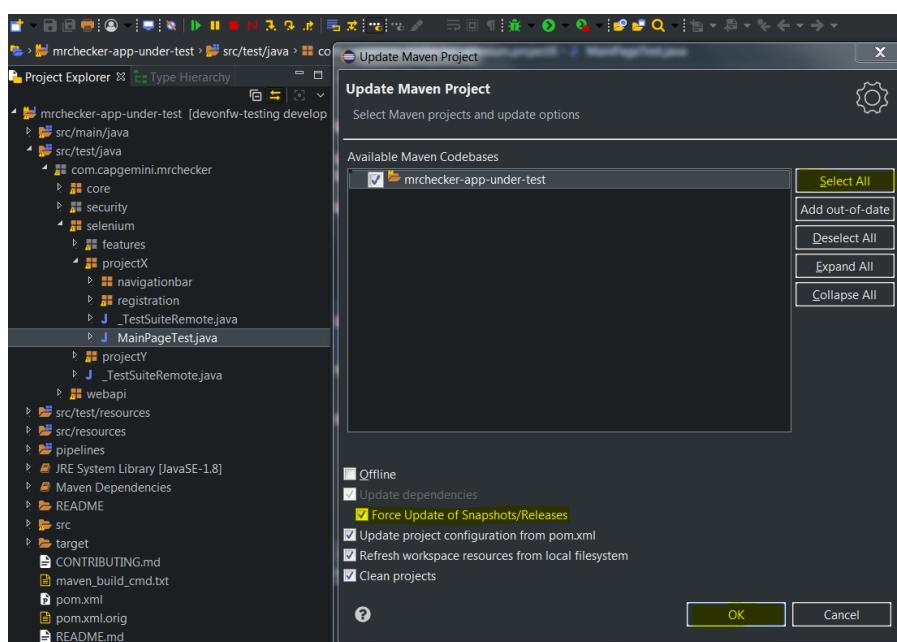
1. Click on the link [Ready to use MrChecker_Test_Environment](#) and download the package
2. Unzip downloaded MrChecker Test Framework to the folder C:\ on your PC - recommended tool: [7z](#) All necessary components, such as Eclipse, Java and Maven will be pre-installed for you. There is no need for 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:

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

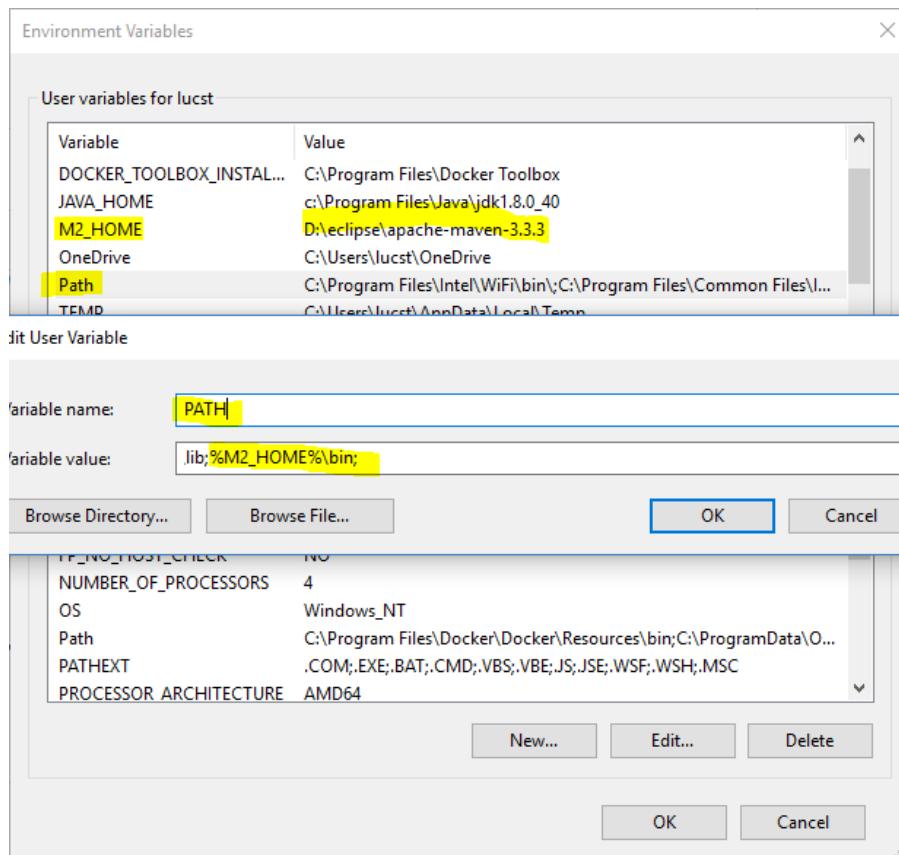
At this point, all test catalogues should be imported in Eclipse and ready to use.

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

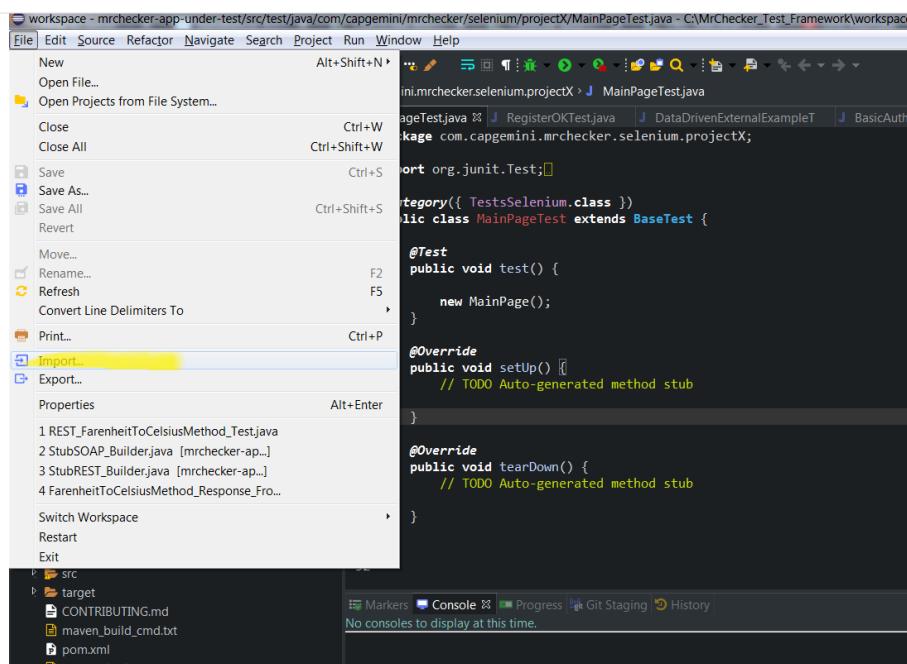


- Verify in command line:

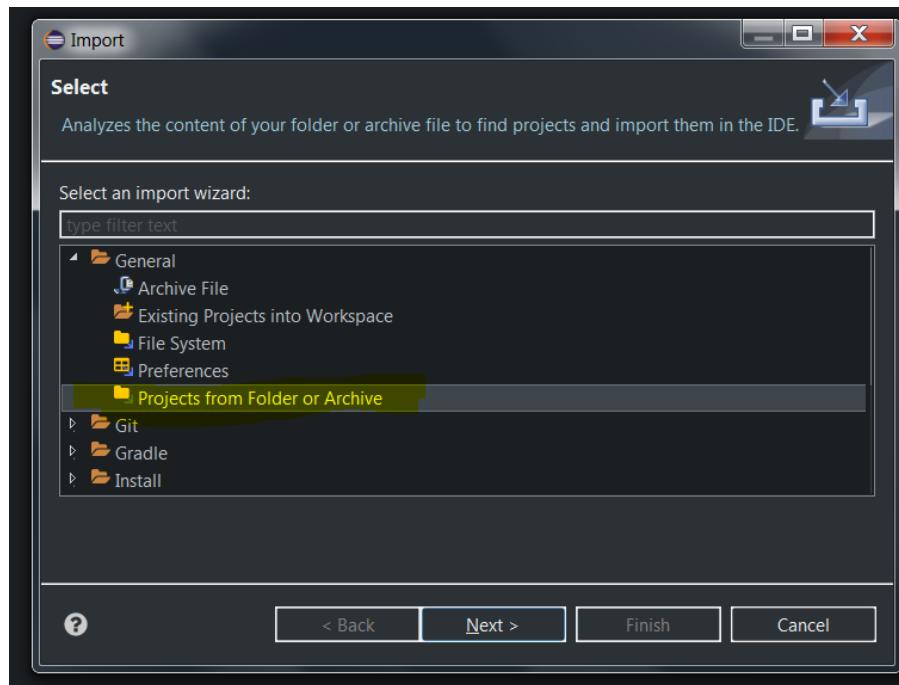
```
> mvn --version
```

2. Eclipse IDE

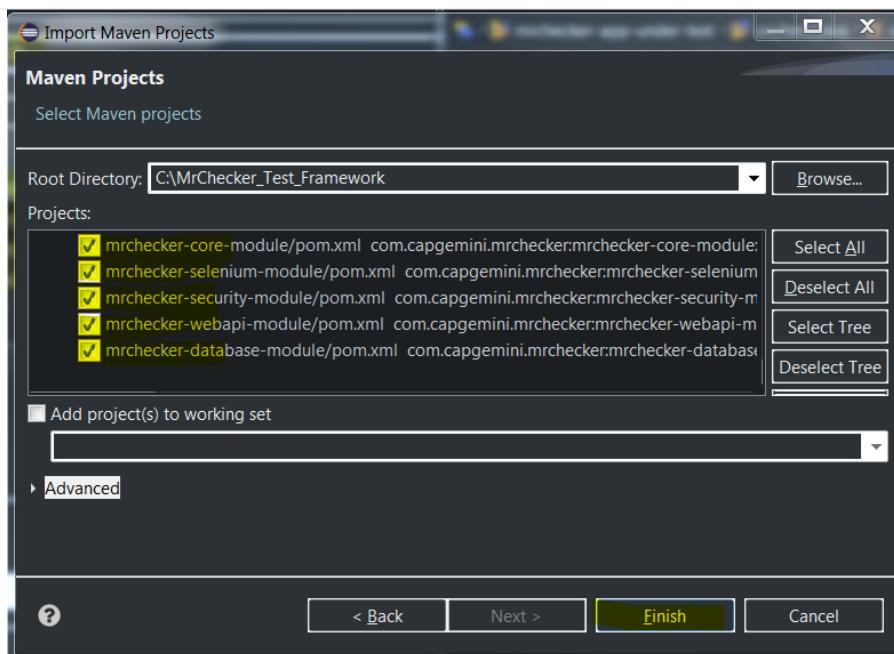
- Download and unzip [Eclipse](#)
3. Download Mr Checker Test Framework [source code](#)
 4. Import projects in Eclipse
 - Import:



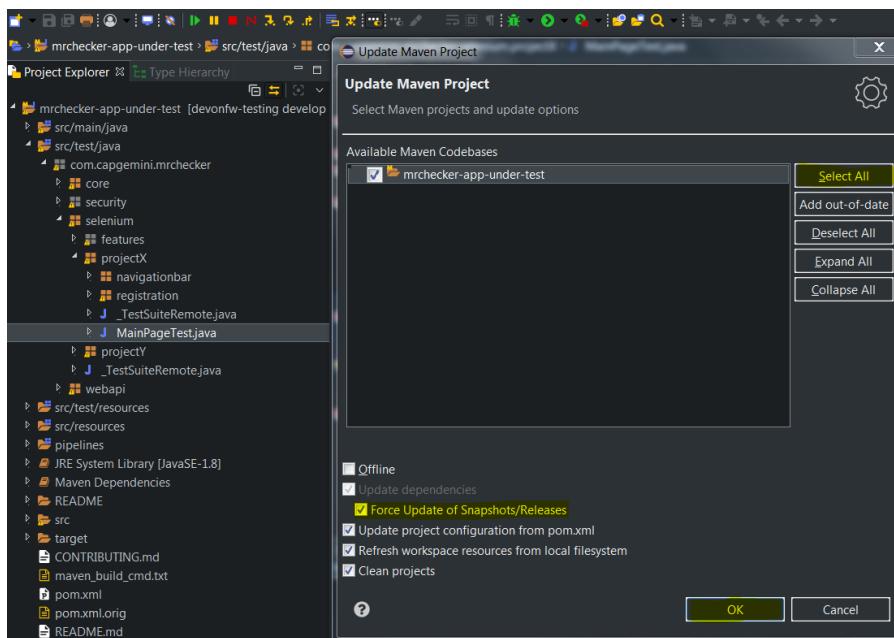
- Projects from folders:



- Open already created projects



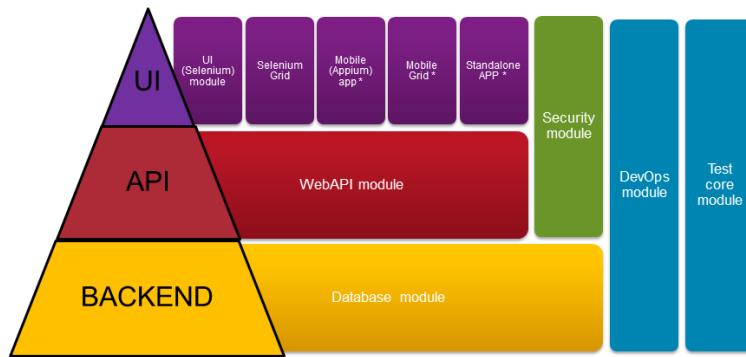
- Update project structure - *ALT + F5*



Chapter 89. Mr Checker Test Framework modules

This is the structure of Mr Checker Framework Modules:

E2E Test Framework modules



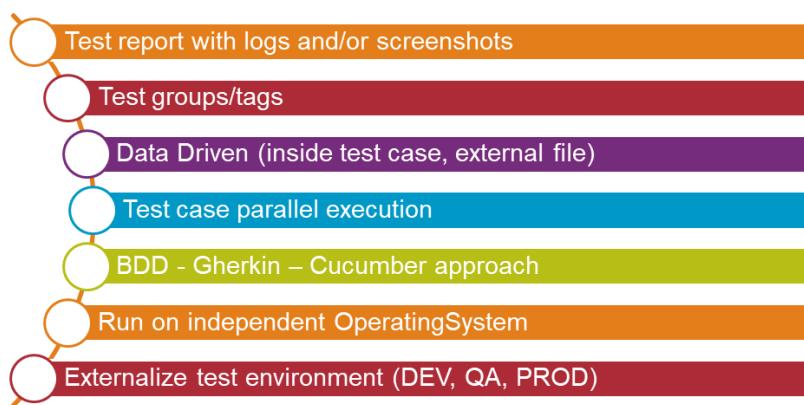
In this section, it is possible to find all information regarding the main modules of Mr Checker:

- [Core Test Module](#)
- [Selenium Test Module](#)
- [WebAPI Test Module](#)
- [Security Test Module](#)
- [Database Test Module](#)
- [Mobile Test Module](#)
- [Standalone Test Module](#)
- [DevOps Module](#)

89.1. Core Test Module

89.1.1. What is Core Test Module

Core functionality ingredients



89.1.2. 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

89.1.3. How to start?

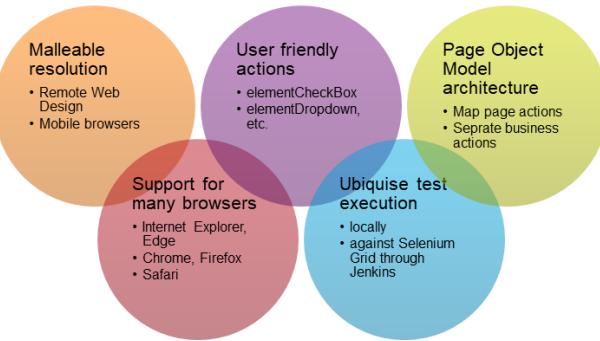
Read: [Framework Test Class](#)

Unresolved directive in devonfw-testing.wiki/master-devonfw-testing.asciidoc
include::framework-test-class.asciidoc[leveloffset=3]

89.2. Selenium Test Module

89.2.1. What is Mr Checker E2E Selenium Test Module

UI Selenium test module ingredients



89.2.2. Selenium Structure

- [What is Selenium](#)
- [What is WebDriver](#)
- [What is Page Object Model/Pattern](#)
- [List of web elements \(Button, Dropdown, Checkbox, Alert Popup, etc.\)](#)

89.2.3. 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](#)

89.2.4. How to start?

Read: [My first Selenium Test](#)

89.2.5. Selenium Best Practices

- [Table of best practices](#)

89.2.6. 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]

89.3. Security Test Module

89.3.1. 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.

89.3.2. 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.

89.3.3. Scope definition

Unresolved directive in devonfw-testing.wiki/master-devonfw-testing.asciidoc - include::DataBase-test-module.asciidoc[leveloffset=2]

89.4. Mobile Test Module

89.5. Standalone Test Module

89.6. DevOps Module

89.6.1. 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.

89.6.2. QA Team Goal

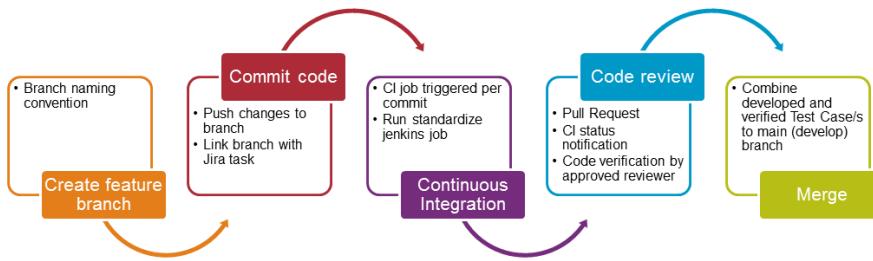
For QA engineers, it is essential to take care of the product code quality.

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.

89.6.3. Well rounded test case production process

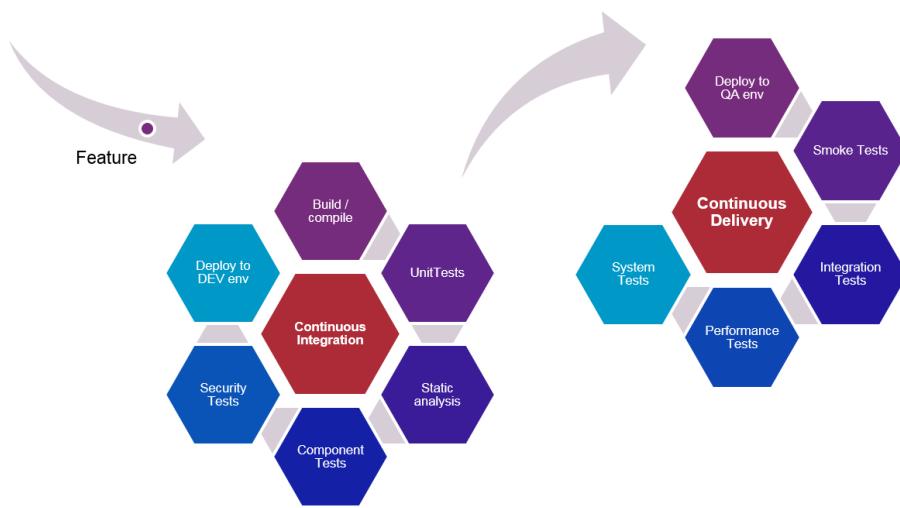
- How do we define top notch test cases development process in **E2E Mr Checker Test Framework**

Well defined Test Case develop process



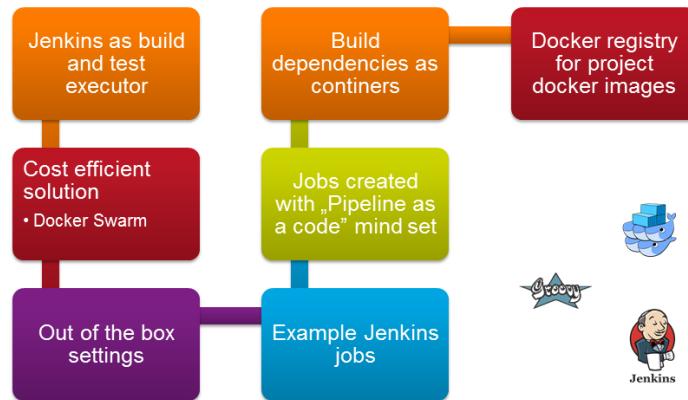
89.6.4. Continuous Integration (CI) and Continuous Delivery (CD)

- 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



89.6.5. What should you receive from this DevOps module

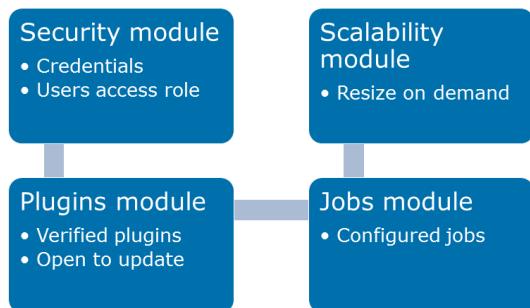
DevOps infrastructure ingredients



89.6.6. What will you gain with our DevOps module

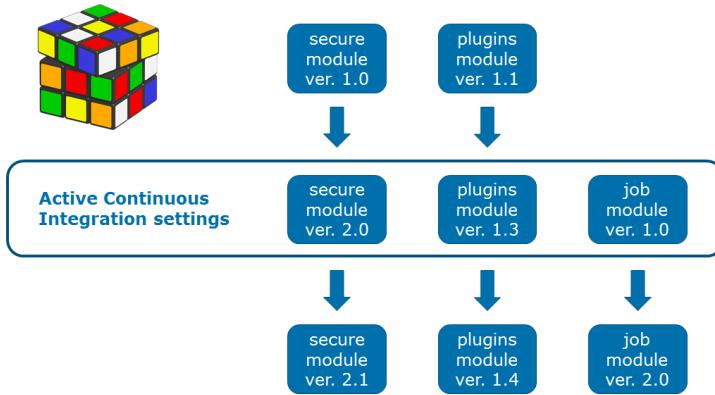
The CI procedure has been divided into transparent modules. This solution makes configuration and maintenance very easy because everyone is able to manage versions and customize the configuration independently for each module. A separate security module ensures the protection of your credentials and 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.

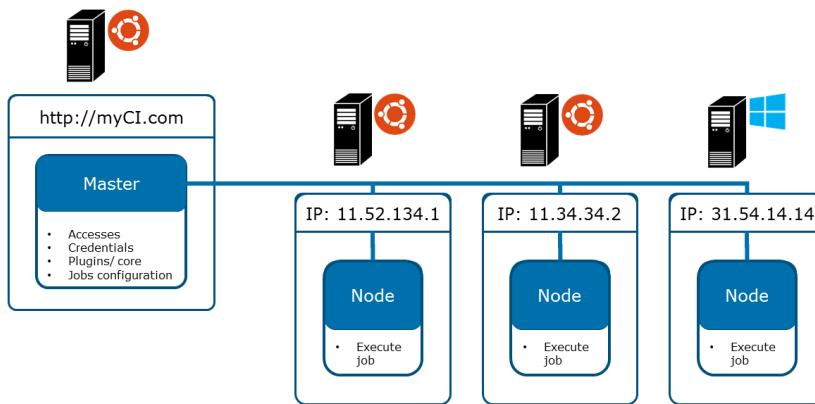
Setup your own proven CI modules



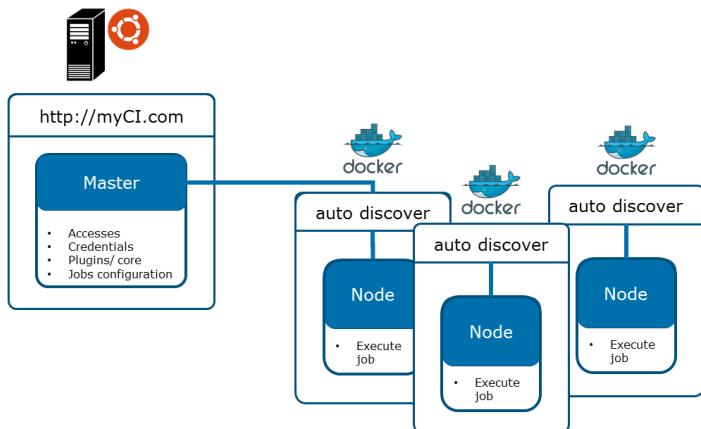
DevOps module supports a delivery model in which executors are made available to the user as needed. It has advantages such as:

- 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

89.6.7. 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](#)

89.6.8. 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

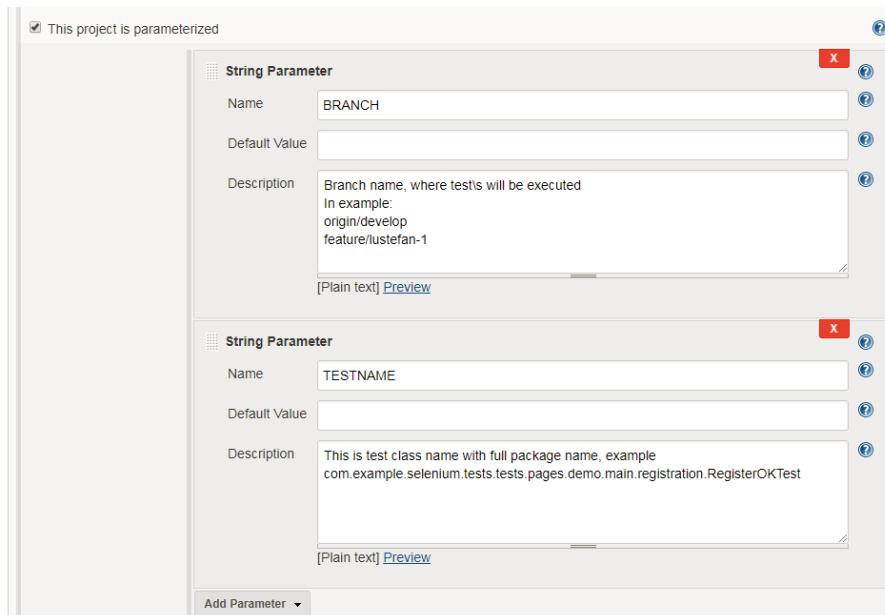
Jenkins is an Open Source Continuous Integration Tool. It allows the user to create automated build jobs which will run remotely on so called *Jenkins Slaves*. A build job can be triggered by several events, for example on new pull request on specified repositories or timed (e.g. at midnight).

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 following command:

```
clean install test-compile -DskipTests=true
```

A "Advanced..." button is located 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 following command:

```
site
```

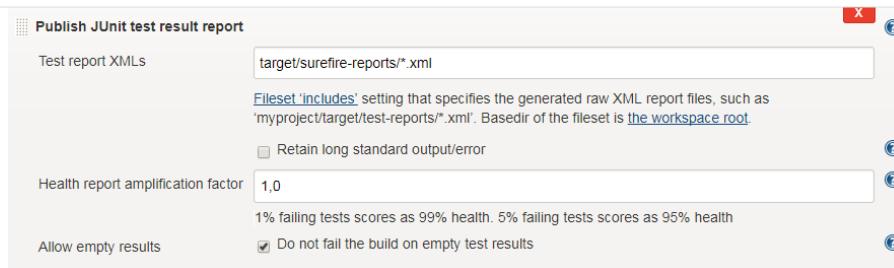
A "Advanced..." button is located at the bottom right.

Post Build Configuration

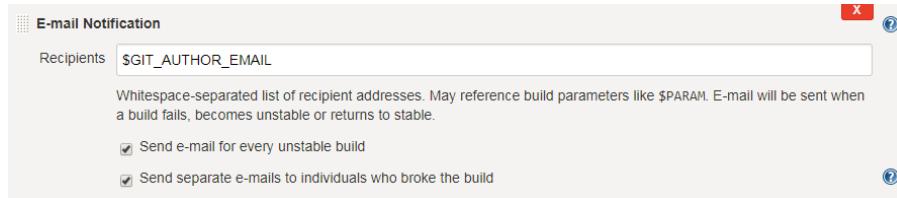
- At first, the results will be imported to the [Allure System](#)

The screenshot shows the Jenkins 'Allure Report' configuration step. The 'Results:' section has a 'Path' field containing 'target/allure-results'. There is an 'Add' button and a note about relative paths from the workspace. The 'Properties' section also has an 'Add' button. A "Advanced..." button is located 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.



- Finally, an E-Mail will be sent to the previously extracted author of the commit.



Using the Pre-Configured Custom Docker Image

If you are starting a new Jenkins instance for your tests, we'd suggest 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.

89.6.9. 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.

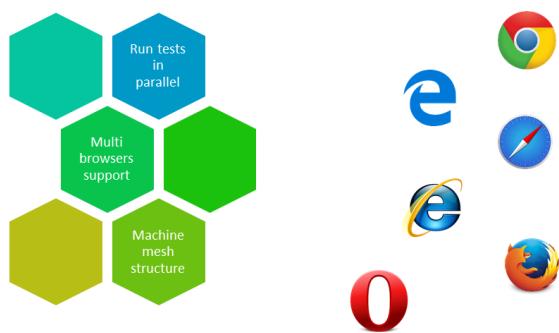
89.6.10. Selenium Grid

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, ...).

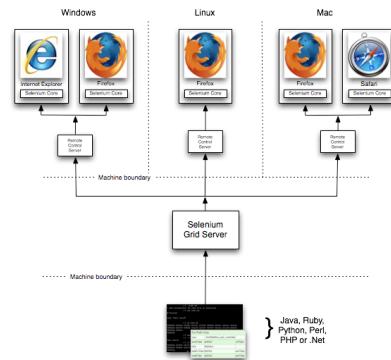
Selenium Grid – where we use it



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]]

==

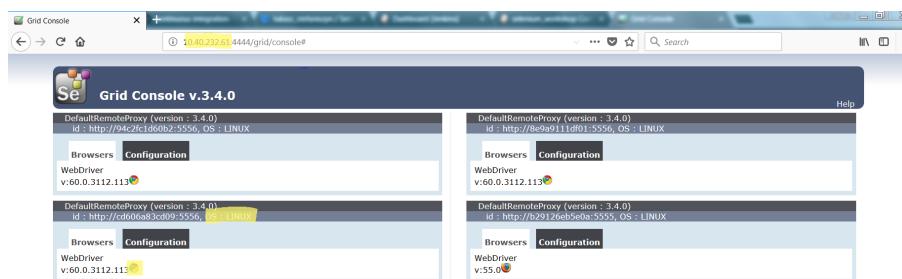
How to use Selenium Grid with E2E Mr Checker Test Frameworks

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



89.6.11. 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.

89.6.12. 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

89.6.13. 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"
```

89.6.14. 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)
```

Unresolved directive in devonfw-testing.wiki/master-devonfw-testing.asciidoc - include::How-to-build-this-DevOps-module.asciidoc[leveloffset=3]

Unresolved directive in devonfw-testing.wiki/master-devonfw-testing.asciidoc - include::Building-jobs-&running-builds.asciidoc[leveloffset=3]

Unresolved directive in devonfw-testing.wiki/master-devonfw-testing.asciidoc - include::Pipeline-structure.asciidoc[leveloffset=3]

Chapter 90. My Thai star documentation

Unresolved directive in master.asciidoc - include::my-thai-star.wiki/master-my-thai-star.asciidoc[]

90.1. Contributing Guide

90.1.1. Code Contributions

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](#).

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.

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.

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](#).

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**

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.

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 74. 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*.

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 code diff is shown for `samples/core/pom.xml`. Line 216 is highlighted with a green background and shows the addition of `<scope>provided</scope>`.

Below the code diff, hohwille adds a comment: 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 75. 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 summary of Travis CI build results for a pull request:

- 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 76. 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.

90.1.2. Contributor Covenant Code of Conduct

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.

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

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.

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.

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.

Attribution

This Code of Conduct is adapted from the [Contributor Covenant](https://www.contributor-covenant.org/version/1/4/code-of-conduct.html), version 1.4, available at <https://www.contributor-covenant.org/version/1/4/code-of-conduct.html>

90.1.3. Development Guidelines

- **Always ask before creating a pull request.** To avoid duplication efforts, it's 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.

90.1.4. Working with forked repositories

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.

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:

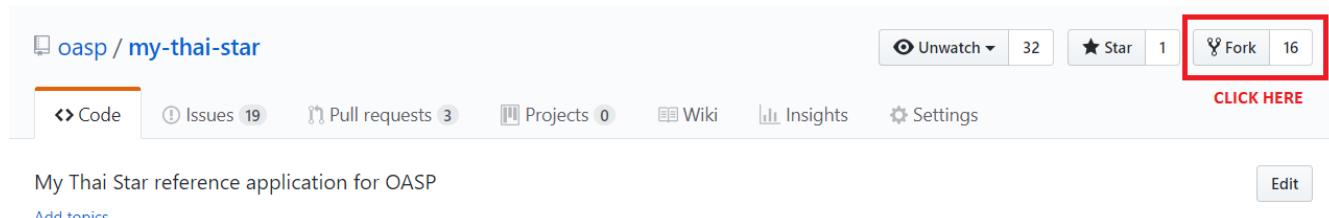
1. Fork the repository.

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!

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 red border and the text 'CLICK HERE' below it. Other buttons like 'Unwatch', 'Star', and 'Settings' are also visible. Below the header, there are navigation links for 'Code', 'Issues 19', 'Pull requests 3', 'Projects 0', 'Wiki', 'Insights', and 'Settings'. The main content area displays the repository's description: 'My Thai Star reference application for OASP' and an 'Edit' button. There is also a link to 'Add topics'.

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
```

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)
```

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(-)
```

- Push the changes to update your fork on GitHub, GitLab, Bitbucket, etc.

90.1.5. 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](#).

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~

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
...
...
```

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
- * . . .

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
| ... | ... | ...
|===
|...
```

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]

90.2. Release Notes

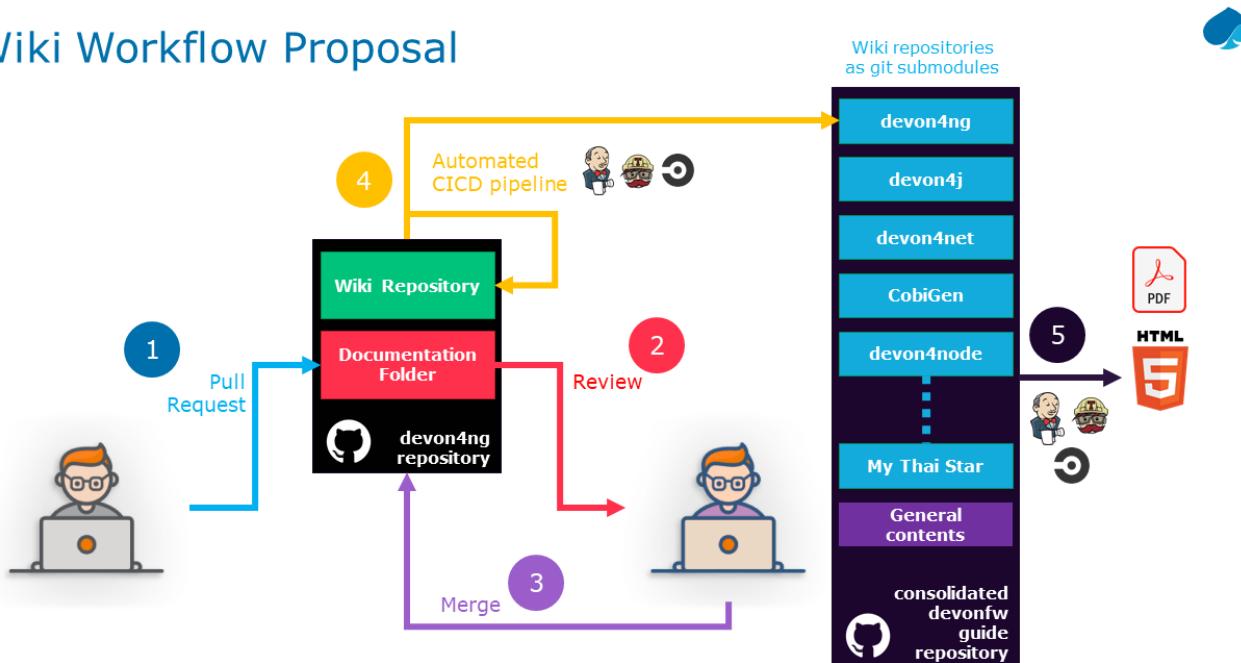
90.2.1. devonfw Release notes 3.1 “Goku”

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 and 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>.

Changes and new features

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

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

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

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](#).

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

devon4net

The following changes have been incorporated in devon4net:

- New circuit breaker component to communicate microservices via HTTP
- Resolved the update packages issue

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>

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.

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.

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>

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

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 Production Line

90.2.2. devonfw Release notes 3.0 “Fry”

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.

Changes and new features

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

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

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.

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.

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.

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, XContentOptionsHeader, 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.

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.

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>

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.

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.

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>

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>.

90.2.3. devonfw Release notes 2.4 “EVE”

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.

Changes and new features

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.

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

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.

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.

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.

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.

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>

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.

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.

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>

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>

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

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

90.2.4. devonfw Release notes 2.3 "Dash"

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.

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.

Changes and new features

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

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>

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

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

Documentation refactoring

The complete devonfw guide is restructured and refactored. Getting started guides are added for easy start with devonfw. Integration of the new Tutorial with the existing devonfw Guide whereby existing chapters of the previous tutorial were converted to Cookbook chapters. Asciidoc is used for devonfw guide PDF generation. See: <https://github.com/devonfw/devon-guide/wiki>

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.

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>

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

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>

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>

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>

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)

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>

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

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>

90.2.5. devonfw Release notes 2.2 "Courage"

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.

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>

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.

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>

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>

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>

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 the deprecated API

io.oasp.module.web.common.base.PropertiesWebApplicationContextInitializer

- Documentation of the usage of UserDetailsService of Spring Security

See: <https://github.com/oasp/oasp4j>

Wiki: <https://github.com/oasp/oasp4j/wiki>

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>

devonfw distribution based on Eclipse OOMPH

The new Eclipse devonfw distribution is now based on Eclipse OOMPH, which allows us, an any engagement, to create and manage the distribution more effectively by formalizing the setup instructions so they can be performed automatically (due to a blocking issue postponed to devonfw 2.2.1 which will be released a few weeks after 2.2.0)

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>

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>

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>

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>

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 Cloudfoundry

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>

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>

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*.

90.2.6. Release notes devonfw 2.1.1 "Balu"

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:

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.

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>

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](#)

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](#).

WebSphere & Wildfly deployment documentation

The new version of devonfw contains more elaborate and updated documentation about deployment on [WebSphere](#) and [Wildfly](#).

Version 2.1.1 Updates, fixes & some new features

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](#)

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.

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.)

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](#)

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.

Version 2.1 New features, improvements and updates

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.

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

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](#)

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.

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](#)

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](#)

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](#)

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](#)

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](#)

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](#)

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](#)

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](#)

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.

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](#)

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](#)

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](#)

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](#)

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](#)

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.

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.

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, kiran-vadla, 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!