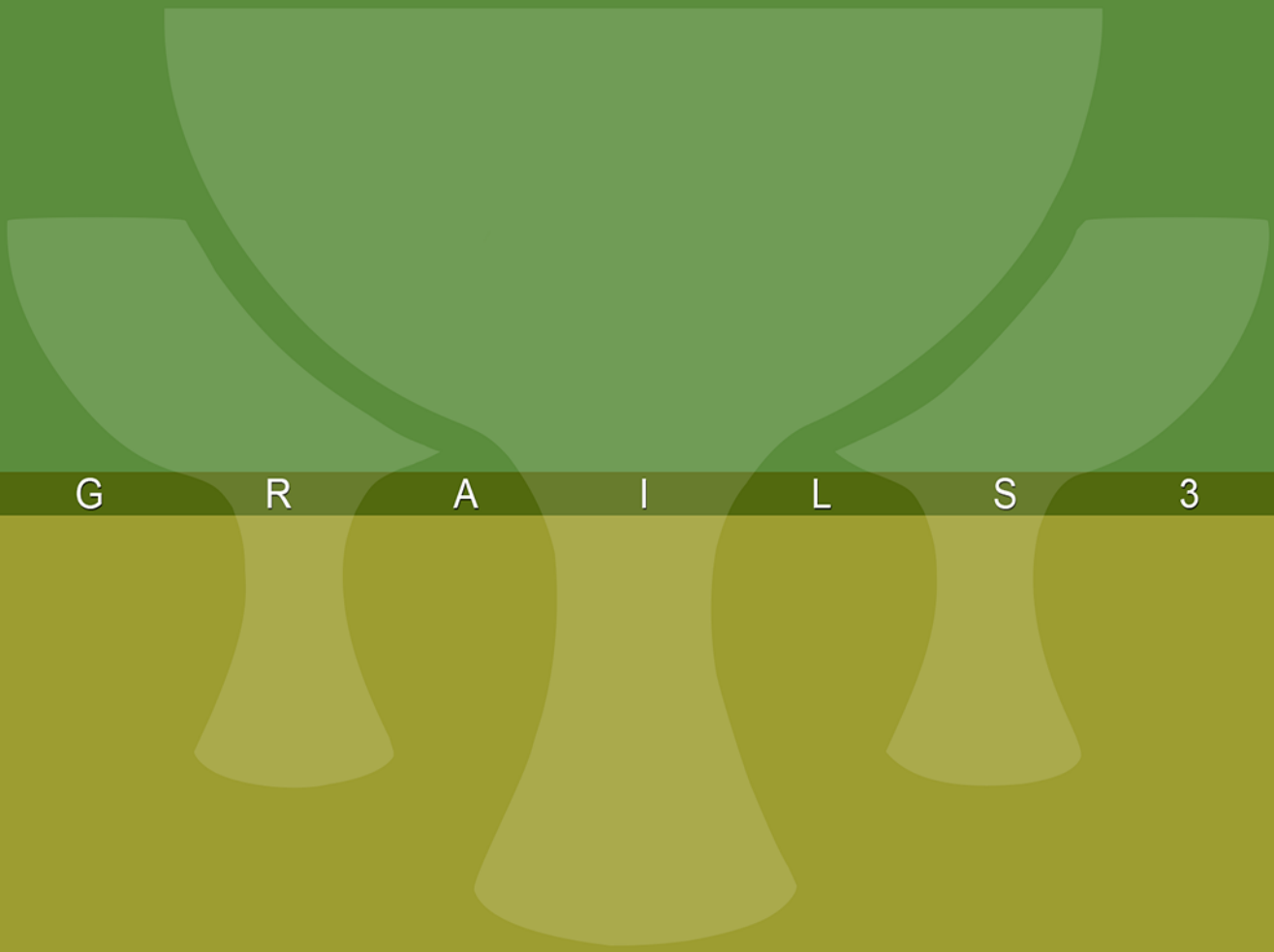
GRAILS 3

STEP BY STEP



G R A I L S 3

Greenfield applications made right with Grails 3

Grails 3 - Step By Step

Greenfield applications made right with Grails 3

Cristian Olaru

This book is for sale at http://leanpub.com/grails3book

This version was published on 2017-04-21

ISBN 978-973-0-24076-4

Tweet This Book!

Please help Cristian Olaru by spreading the word about this book on Twitter!

The suggested tweet for this book is:

Just bought “Grails 3 - Step by Step” book - https://leanpub.com/grails3book by @colaru #Grails #Java #GrailsThreeBook

The suggested hashtag for this book is #grailsthreebook.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#grailsthreebook

Also By Cristian Olaru

Ionic 2 - Step By Step

Contents

How this book is organized

We try to describe here how a complete greenfield application can be implemented with Grails 3 in a fast way using profiles and plugins - and we do this in the sample application that accompanies this book. You can find news about this book on its presentation site located here: www.grailsthreebook.com[1]. This book is not a replacement of the Grails 3 Reference Documentation[2] which is a complete description of Grails 3[3], made by the creators of the framework.

The source code used in this book is part of this sample application which is a free project hosted on GitHub on this location: https://github.com/colaru/mvp-application[4]. If a source code fragment is included in this book, then it is taken from this project. We use BDD or other specification by example techniques for describing the sample application specification - the leaving documentation for the project can be found here: http://serenity.itjobsboard.eu[5]. The application code is tested in a TDD style using automated unit, integration and functional tests.

The sample application is based on a multi project Gradle build so the application can be automatically built using Gradle - the build tool used by Grails 3. The web application is available here: https://www.itjobsboard.eu[6]. The admin application is available online here: https://admin.itjobsboard.eu[7] - users *admin*, *operator* or *customer* with password *Password123!*. The deployment is done automatically to AWS cloud in a Continuous Deployment style using Jenkins.

We use links to various external resources in this book (you will see them in the page's footers), because this book is intended to be in electronic format from the beginning. The reader can use these links for consulting external references in an easy way.

The book has two parts.

First part - Grails 3 Essentials

An introduction to application development with Grails 3; we describe the main application that will be implemented in the second part of the book

Chapter 1 is an introduction to the framework. You will find here some history of the framework and a lot of links to the Grails resources; it is presenting the classical three layered architecture for a web application provided by Grails 3

[1] https://grailsthreebook.com
[2] http://docs.grails.org/latest
[3] https://grails.org
[4] https://github.com/colaru/mvp-application
[5] http://serenity-mvp.s3-website.eu-central-1.amazonaws.com
[6] http://application.eu-central-1.elasticbeanstalk.com
[7] itjobsboard-env.eu-west-1.elasticbeanstalk.com

Chapter 2 describes how to start your work with Grails 3 framework and how to install all the tools for a free working environment - free as much as possible because you have to pay some money for the best tools on the market

Chapter 3 presents the project we want to implement as an example; it is presenting also the way we choose to work on implementing the sample application, based on BDD, TDD, CI and CD; it shows how the main application is split into multiple parts based on Grails 3 profiles

Second part - Practical example with Grails 3

A practical implementation of a greenfield application with Grails 3; the application is composed of multiple parts corresponding to various Grails 3 profiles

Chapter 4 describes the implementation of the application admin portal that will be used in Intranet by the site administrators, based on a classical Grails 3 Web profile

Chapter 5 describes the implementation of the application site exposed to the Internet to the customers, that is based on Grails 3 Angular profile

Chapter 6 describes a REST web API exposed with Grails 3 Rest profile and consumed by a mobile hybrid application created with Ionic 2 (published in Google Play Store, Apple App Store, and Windows Store)

Chapter 7 describes a Microservice developed with Grails 3 Rest API profile;

Introduction to Grails 3

Grails is a full stack Web framework, covering all the aspects of a modern web application development. Its plugins system is unique in the Java world - you can divide your application into complete functional modules, containing presentation and business logic. Because it runs in JVM, it has access to the entire Java ecosystem. Because is written in Groovy, it has access to all Groovy libraries and tools.

The main site for Grails 3 is grails.org[8]. Here is the definition of Grails on the official site:

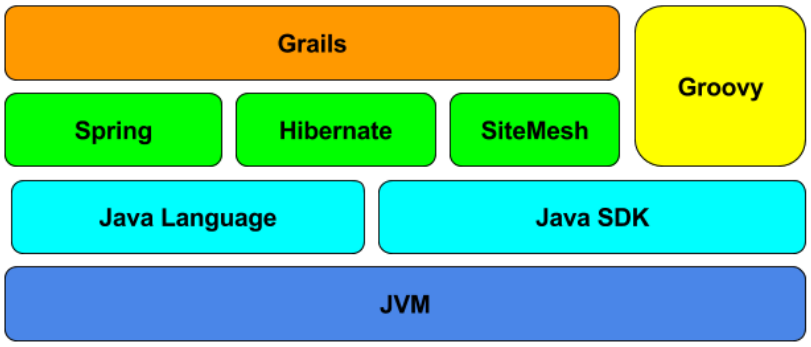
> Grails is a powerful web framework for the Java platform, aimed at multiplying developers' productivity, thanks to Convention-over-Configuration, sensible defaults, and opinionated APIs. It integrates smoothly with the JVM, allowing you to be immediately productive, whilst providing powerful features, including integrated ORM, Domain-Specific Languages, runtime and compile-time meta-programming, and Asynchronous programming.

If you have developed multiple applications over time, you have recognized a set of principles and good practices that can be reused. You don't want to start from scratch each time you build your new application. Grails has good technical principles that make it a good starter for your new applications. With Grails, you have an established architecture from the beginning, and you can implement new features in an easy way, using scaffolding and the plugins ecosystem. And this is more valuable when you have to get to the market first, before your competitors.

Some history

Grails 3 is based on a stack of other technologies Java[9], Spring[10], Groovy[11], Hibernate[12], Sitemesh[13]. We try here to show the history of these technologies and how they are used in Grails 3.

[8] https://grails.org/index.html
[9] https://www.oracle.com/java
[10] https://spring.io
[11] http://groovy-lang.org
[12] http://hibernate.org
[13] http://wiki.sitemesh.org



Technologies inside Grails

Java history

First it was Java, and was created by *Sun Microsystems*[14] and its father was *James Gosling*[15]. Java came with the idea of JVM - *Java Virtual Machine* that lets you *Write once, Run everywhere*. Everywhere means on any operating system. You write the code in a source file which is compiled to a byte code runnable by the JVM. Any operating system has an implementation for the Java Virtual Machine specification.

Because Sun was bought by Oracle, now Java is in Oracle's portfolio. Java is used by millions of developers (Oracle claims 10 million) and is the number one TIOBE[16] index of most used languages in the world.

Why was Java such a strong language from the beginning? Some clues:

- It was totally object-oriented from the beginning and let people think *everything is an object* (this is the name of a chapter from one of the best Java books Thinking in Java - TIJ[17]) written by Bruce Eckel[18].
- It eliminates the *pointers* and memory deallocation, letting this be handled by the JVM and the garbage collector; the GC is constantly improved by the Oracle team. G1 GC[19] is the most advanced GC these days.
- It has multithreading inside from the beginning.
- It has exceptions handling inside.
- It has modularity inside, represented by packages and the possibility of packaging libraries in *.jar* files that can be reused between projects; there are a lot of libraries on the market, and as proof, you can search a Maven global repo[20]

[14] https://en.wikipedia.org/wiki/Sun_Microsystems
[15] https://en.wikipedia.org/wiki/James_Gosling
[16] http://www.tiobe.com/tiobe_index
[17] http://mindview.net/Books/TIJ4
[18] https://twitter.com/bruceeckel
[19] http://www.oracle.com/technetwork/tutorials/tutorials-1876574.html
[20] https://mvnrepository.com/

Java EE history

After this was the *Java EE* (known in the past as J2EE or JEE) – an attempt to make it easy to create enterprise applications (compared to other 2 profiles *Java SE* for desktop applications and *Java ME* for mobile applications). The language and Java EE is specified by a committee named JCP[21] *Java Community Process*, which is responsible for creating *JSRs*, specifications implemented by various vendors. Any JSR has a reference implementation which is a proof that the specification can be implemented.

For example, the last specification for the Java Servlet is JSR 315: JavaTM Servlet 3.0 Specification[22] and the reference implementation for this standard is Oracle GlassFish 3.x (RI)[23]. Grails embeds by default inside, in DEV mode, a Tomcat[24] server and you can see what version of Java Servlet is implemented on each Tomcat version[25].

Here are the Java EE Full Platform Compatible Implementations[26] on different Java EE versions. The current version is Java EE 7, and here is the official tutorial[27]

Spring history

But the *Java EE* standards were too *de jure* for the Java community, compared to a lot of frameworks that emerged in the Java open source space, considered more *the facto standards* like: Struts for Web, Hibernate for persistence, etc. And then Spring Framework was born as a response to this *committee style* and *specification centric* style of driving the Java future - Rod Johnson[28] and others put the foundation, and Juergen Hoeller[29] is now responsible for the framework as part of the Pivotal portfolio. The framework introduced Dependency Injection and reaffirmed the power of POJOs (Plain Old Java Objects) - in contrast to EJBs (Enterprise Java Beans) which were at version 2 when Spring emerged - things have improved in EJB 3.

In time, these two technologies are evolving in parallel. For example, Java EE adopted the DI in its CDI, some implementations of JPA wrap Hibernate. Spring embraces annotations for configurations, instead of XMLs, as in Java EE. And Spring Framework has integration with some JSRs and is based on some technologies provided by Java EE.

The main documentation for Spring Framework is the reference documentation[30].

Groovy history

Java is an imperative style programming language and, compared with functional programming languages, *it suffers from accidental complexity* as Venkat Subramaniam[31] says.

[21] https://www.jcp.org/en/home/index
[22] https://jcp.org/en/jsr/detail?id=315
[23] https://glassfish.java.net
[24] http://tomcat.apache.org
[25] http://tomcat.apache.org/whichversion.html
[26] http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html
[27] https://docs.oracle.com/javaee/7/tutorial
[28] https://twitter.com/springrod
[29] https://spring.io/team/jhoeller
[30] http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle
[31] https://twitter.com/venkat_s

Groovy is a functional programing language that is bringing functional style to Java programming. It introduces the notion of *closure* which is the idea of function as a main citizen of your code. It is also a scripting language because it has the characteristic of incorporating a lot of semantics in a short syntax.

It was started by James Strachan[32], back in 2003, as a dynamic language for the JVM, inspired by Ruby, but closer to Java. Guillaume Laforge[33] become the project lead after James leaves the project, and he is still the lead today. Here is a history[34] of the most important moments of the language and the main contributors to the project.

The official Groovy site is groovy-lang.org[35]. Here is the definition of Groovy on the official site:

> Apache Groovy is a powerful, optionally typed and dynamic language, with static-typing and static compilation capabilities, for the Java platform aimed at improving developer productivity thanks to a concise, familiar and easy to learn syntax. It integrates smoothly with any Java program, and immediately delivers powerful features to your application, including scripting capabilities, Domain-Specific Language authoring, runtime and compile-time meta-programming and functional programming.

Some characteristics of Groovy:

- it was a way to get functional programming into JVM; it introduced *closure* to the Java world long before the Lambda Expressions in Java 8[36]
- it is compiled to Java bytecode that is running in JVM (there is a *groovyc* compiler that is compiling Groovy code to Java bytecode in the same way the *javac* compiler is doing with Java code)
- Groovy scriptlet can be embedded in Java code and the resulted mixture is still working after compilation
- is enriching the Java classes with other convenient methods resulting in a new API named GDK[37]

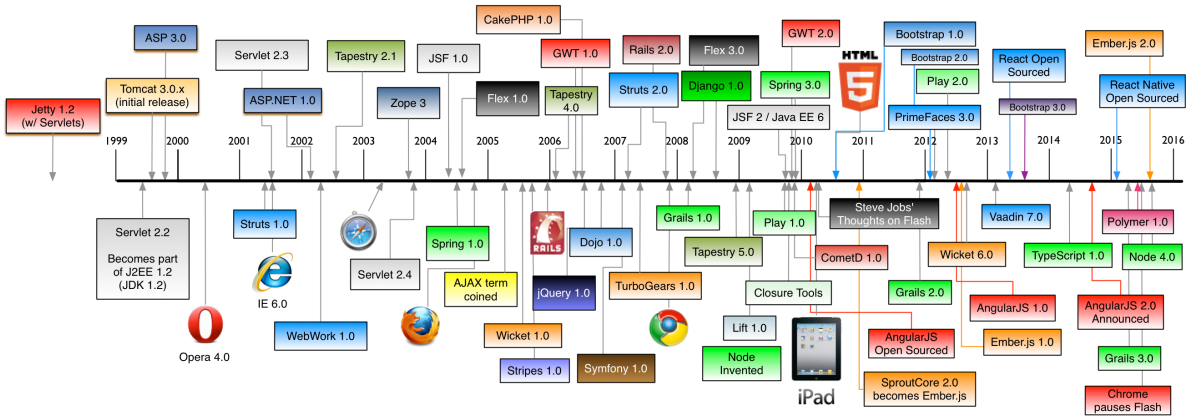
[32] https://twitter.com/jstrachan
[33] https://twitter.com/glaforge
[34] http://melix.github.io/blog/2015/02/who-is-groovy.html
[35] http://www.groovy-lang.org
[36] http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Lambda-QuickStart/index.html
[37] http://www.groovy-lang.org/gdk.html

Why not Scala (my opinion)

There is a famous quote from James Strachan blog[38]: *I can honestly say if someone had shown me the Programming in Scala book by by Martin Odersky, Lex Spoon & Bill Venners back in 2003 I'd probably have never created Groovy.* For me, as a Java developer, Scala it is a big elephant to eat (I don't want to learn a different syntax and so manny different new concepts). The same is with Play framework[39] (the main Scala Web framework) which is not as Grails, based on standard Java technologies like Servlets but on Netty[40]. And Scala has also a new build system SBT[41] that should be used. I personally prefer the Java ecosystem with the option of some Groovy scripting.

Grails history

This is a timeline created by Matt Raible[42] where he tries to describe the history of Web Frameworks[43].



A history of Web frameworks

As you can see in this diagram the Grails framework was born in 2008 with its first GA release. Grails 2 was released in 2011. And Grails 3 was first released in 2015.

Here is a top of Java Web frameworks published by RebelLabs[44] in 2017. Grails is placed on the 5th position immediately after SpringBoot (and Grails 3 has SpringBoot inside). Grails 3 is not a simple Web framework like the majority of the frameworks presented in this top but a full stack framework like Play which is the 7th position. JHipster which can be an alternative to the Grails 3 Angular(2) profile is the 10th position on the list.

[38] http://macstrac.blogspot.ro/2009/04/scala-as-long-term-replacement-for.html
[39] https://www.playframework.com
[40] http://netty.io
[41] http://www.scala-sbt.org
[42] https://twitter.com/mraible
[43] https://raw.githubusercontent.com/mraible/history-of-web-frameworks-timeline/master/history-of-web-frameworks-timeline.png
[44] https://zeroturnaround.com/rebellabs/java-web-frameworks-index-by-rebellabs

Java Web Frameworks Index | REBELLABS by ZEROTURNAROUND

Rank	Framework	Popularity
1	Spring MVC	32.00
2	JSF	22.30
3	GWT	9.84
4	Spring Boot	9.45
5	Grails	8.50
6	Struts	7.21
7	Play framework	6.00
8	Vaadin	2.75
9	Dropwizard	1.21
10	JHipster	0.74

Java Web Frameworks Index by RebelLabs in 2017

What makes Grails a good choice from the multitude of frameworks for building Web applications:

- Grails is a *full stack* web framework in the sense that you can create an entire Web application using it. It covers the web presentation, business domain and database persistence aspects of your application. Not alone, but integrating the right technologies for this: Java, Spring, Hibernate. And all of these are linked by Groovy language with is used for configuring your application - the DSL for configurations are Groovy based (now in Grails 3 you will have the option to use YAML also)
- conventions over configurations
  - for creating a service, it is enough to place a groovy file containing a class without state in */service* folder without annotating it with *@Service* or mark it in a configuration file like in Spring or annotate with *@Stateless* for a EJB 3 stateless session bean
  - for creating an *entity*, it is enough to place a POJO class in a */domain* folder and no *@Entity* annotation like in JPA or Hibernate or declaring it in configuration files
- scaffolding for generating a seed application or a variety of elements starting from domain objects; it can be static (generating the corresponding files in filesystem) or dynamic (generating just proxies in memory)
- environments inside from the beginning like TEST, DEV, PROD (or a custom one) letting you take different actions in runtime based on the given environment

- testing inside the framework (unit - in isolation using mocking, integration, functional) using JUnit[45], Spock[46] and Geb[47]
- static resources served in an optimized way (*asset pipeline* plugin replacing *resources* plugin)
- asynchronous features based on plugins in first versions and directly inside in version 3 using Reactor[48]
- dynamic reload of classes and resources in runtime based on spring-loaded[49] technology and *on the fly reloading* in Tomcat (dev mode); no server restarts are needed in development mode or products like JRebel
- and many more...

Also, the history of the companies that stay before the framework is interesting. First it was supported by G2One Inc a company founded by the Groovy and Grails project leads, Guillaume Laforge and Graeme Rocher. It was acquired by SpringSource (first Interface 21) which was on background of Spring Framework. SpringSource was acquired by VMWare and transferred to the Pivotal portfolio. In 2015, the end of support from Pivotal for Grails and Groovy was announced and Grails is now supported by OCI and Groovy was adopted by Apache Foundation.

G2One Inc[50] -> SpringSource[51] -> VMware[52] -> Pivotal[53] -> OCI[54]

We present here a list of the main contributors to the Grails framework and the plugins ecosystem, Groovy or other related tools:

- Graeme Roche[55] - the father of framework with more than 10 years of development
- Jeff Scott Brown[56] - Cofounder, implementer of Rest API and Grails books writer
- Guillaume Laforge[57] - Groovy project lead
- Burt Beckwith[58] - Security guy (Spring security and other security plugins)
- Craig Burke[59] - Asset Pipeline suite of plugins
- Alvaro Sanchez[60] - Rest security plugin
- Marco Vermeulen[61] - SdkMan

Here is the contributors list for Grails core Git repository[62] Here is the reference documentation[63]

[45] http://junit.org
[46] http://spockframework.org
[47] http://www.gebish.org
[48] https://projectreactor.io
[49] https://github.com/spring-projects/spring-loaded
[50] https://www.crunchbase.com/organization/g2one#/entity
[51] https://www.crunchbase.com/organization/springsource#/entity
[52] https://www.crunchbase.com/organization/vmware#/entity
[53] https://www.crunchbase.com/organization/pivotal#/entity
[54] https://www.ociweb.com/products/grails/
[55] https://twitter.com/graemerocher
[56] https://twitter.com/jeffscottbrown
[57] https://twitter.com/glaforge
[58] https://twitter.com/burtbeckwith
[59] https://twitter.com/craigburke1
[60] https://twitter.com/alvaro_sanchez
[61] https://twitter.com/marc0der
[62] https://github.com/grails/grails-core/graphs/contributors
[63] http://docs.grails.org/latest/guide/single.html

Grails application architecture

A classical layered architecture

If we are talking about the architecture of a Grails 3 application, we should express this in architectural design patterns. And the right place to find architectural design patterns is this book Patterns Of Enterprise Application Architecture[64] written by Martin Fowler[65]. His catalog of design patterns[66] is available online. We will use a set of patterns from this book but other patterns are used inside the Grails integrated frameworks.

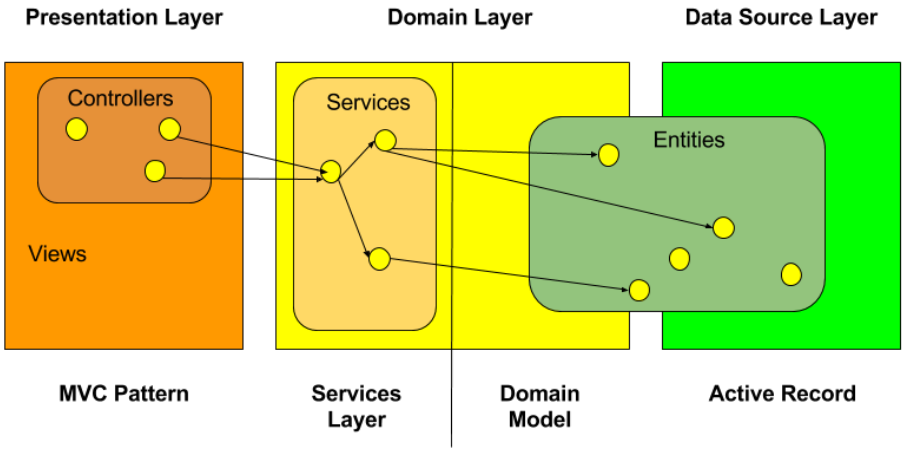
First of all, a Grails application is a classical three layered architecture composed from presentation, domain and data source layers.

Grails layered architecture

In a Grails application, the layers are represented by:

- Presentation Layer - is a classical Model View Controller - MVC[67] represented by Spring MVC framework[68]. The Spring MVC framework is wrapped by Grails which has some specific elements. In Grails, the *C* are the Grails *controllers*, the *V* are the Grails *GSP views* (rendered from *layouts*, *templates* and *tag libraries*), and the *M* are the domain *entities* (the entities are part of the domain layer but are reused in this layer). Also, the Spring MVC framework is based on Servlet API, JSPs, tag libraries and is responsible for interaction with the user via HTTP 1 (or 2)
- Domain Layer - Is split between a Services Layer[69] which are Spring singletons beans and a Domain Model[70] of *entities* which are the mappers to the database
- Data Source Layer - represented by GORM framework and domain entities which are classical Active Record[71]. Gorm offers access to Sql databases and is based on Hibernate which is based also on Java JDBC API and uses SQL for queries.

[64] http://martinfowler.com/books/eaa.html
[65] http://martinfowler.com
[66] http://martinfowler.com/eaaCatalog
[67] http://martinfowler.com/eaaCatalog/modelViewController.html
[68] http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html
[69] http://martinfowler.com/eaaCatalog/serviceLayer.html
[70] http://martinfowler.com/eaaCatalog/domainModel.html
[71] http://martinfowler.com/eaaCatalog/activeRecord.html



Grails three layered architecture

Because it is a technology stack, we have to deal with a lot of knowledge and abstraction:

Layer	**Presentation Layer**	**Domain Layer**	**Data Source Layer**
Patterns	MVC	Service Layer & Domain Model	Active Record
Frameworks and libraries	Spring MVC	Spring Beans, DI, Transactions	GORM (Hibernate and NoSql)
Java technologies	Servlets, JSP, tag libraries	POJO	JDBC
Basic technologies	HTML and HTTP	-	SQL

Some observations:

- The domain/persistence entities are used in the entire Grails application from presentation to the persistence with different roles; they are primary persistence objects having state and persistence methods (CRUD operations but also finders generated dynamically by the Grails framework) but you can choose to enrich them with other functionality because they are part of the domain layer
- Dependency injection[72] (which is coming from Spring) is used for injecting services in other services and in controllers
- The services layer acts as a façade or as an API to the layer in front of it; typically, on this layer we will offer the ACID transactional support
- Controllers from the MVC are responsible just for managing the user interaction flow and will not contain any logic; the place for logic is in the services (and eventually in the domain entities)

[72] http://www.martinfowler.com/articles/injection.html

New technologies inside Grails 3

Grails 3 is a rewrite of Grails 2 and now is based on Spring Boot and not just on Spring framework. The old build system (Gant scripts) are replaced with Gradle with is already used by Spring Boot. This migration has a downside - all the plugins have to be migrated to the new Grails version. The good thing is that the most important plugins were already been migrated by the Grails community. And a set of plugins are not needed anymore because they can be replaced with Gradle plugins and Spring starters.

Spring Boot

You can access the reference documentation for SpringBoot[73].

Some advantages from having Spring Boot as your Grails 3 heart:

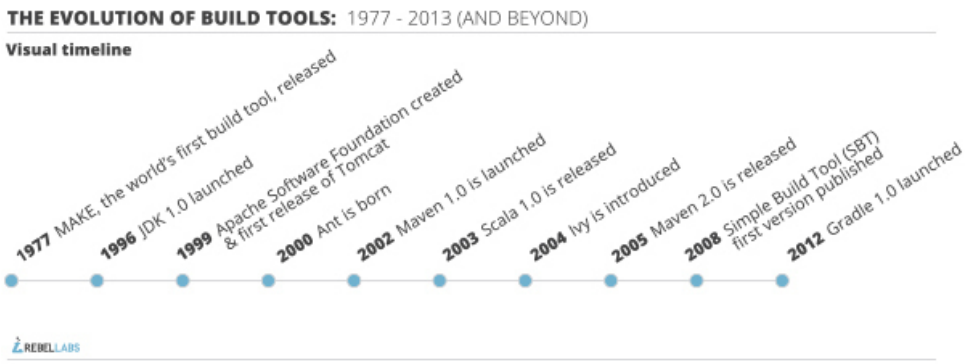
- All Spring Boot starters[74] (starter is the SpringBoot plugin system) are available for your application
- In SpringBoot[75] you can have both Java and Groovy source files
- Some new technologies like websockets and reactive (Reactor project) are available now to your Grails application via Spring Boot

Gradle

In Grails 3, the old Gant[76] scripts were replaced with Gradle build tool. You can access the Gradle main site[77]. The Gradle user guide[78] is also available. Gradle is the third generation build tool for Java after Ant[79] and Maven[80] and has inside all the accumulated experience from its predecessors.

We present here a diagram from a report published by ZeroTurnaround's RebelLabs which provided a timeline of build tools for Java ecosystem from 1977 through 2013[81]

[73] http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle
[74] https://github.com/spring-projects/spring-boot/tree/master/spring-boot-starters
[75] https://projects.spring.io/spring-boot
[76] https://gant.github.io/
[77] https://gradle.org
[78] https://docs.gradle.org/current/userguide/userguide.html
[79] http://ant.apache.org
[80] https://maven.apache.org
[81] https://zeroturnaround.com/rebellabs/java-build-tools-part-2-a-decision-makers-comparison-of-maven-gradle-and-ant-ivy



A evolution of build tool timeline

As an example of its importance, Gradle was chosen by Google as the de facto build tool for Android projects. We try to enumerate here just a part of the advantages that come with Gradle:

- All Gradle plugins[82] are now available for you and can now be used in a variety of aspects of building
- The main scripting language in Grails is (still) Groovy, the same scripting language used in Grails. In this way, it will be simple for you to improve your build system whenever you like.
- Gradle is now offering the possibility to create *multi projects builds* for Grails 3

Gorm

Starting with Grails 3 the database persistence part of the framework named Gorm[83] was extracted completely outside the framework by the Grails team and was redesigned to be a wrapper over both Sql and NoSql databases. Also, starting with Gorm 4, this persistence framework can be used outside of Grails applications (in other Spring Boot applications for example).

In the new Gorm with Hibernate integration (which was the default persistence mechanism in Grails 1 and 2), you can still have support to the main SQL databases on market: MySql[84], Oracle[85], Postgresql[86], MsSql Server[87], etc. Also the framework is now offering support for other persistence technologies like the NoSql databases: MongoDB[88], Neo4j[89], Cassandra[90], Redis[91]. And the trend of Gorm is to use non-blocking drivers and to offer as reactive an approach as possible.

[82] https://plugins.gradle.org
[83] http://gorm.grails.org/latest
[84] https://www.mysql.com
[85] https://www.oracle.com/database/index.html
[86] https://www.postgresql.org
[87] https://www.microsoft.com/en-us/sql-server
[88] https://www.mongodb.com
[89] https://neo4j.com
[90] http://cassandra.apache.org
[91] http://redis.io

LogBack

LogBack is the most modern logging framework (the successor of Log4J) and it comes to Grails 3 because was chosen as the logging framework for Spring Boot. You can access the LogBack main site[92].

Modularity aspects of Grails

Plugins

Plugins were a main feature of Grails from the beginning of the framework. Here is the list of Grails 3 plugins[93]. Even the framework itself in his core is a bunch of essential internal plugins. In fact, Grails is an aggregator for its plugins. If you look in a *web profile* of a Grails application you can see the default Grails plugins that are declared for this profile (will be different plugins for different profiles)

```
dependencies {
    ....
    compile "org.grails:grails-core"
    compile "org.grails:grails-dependencies"
    compile "org.grails:grails-web-boot"
    compile "org.grails.plugins:cache"
    compile "org.grails.plugins:scaffolding"
    compile "org.grails.plugins:hibernate5"
    console "org.grails:grails-console"
    profile "org.grails.profiles:web"
    runtime "com.bertramlabs.plugins:asset-pipeline-grails:2.11.6"
    testCompile "org.grails:grails-plugin-testing"
    testCompile "org.grails.plugins:geb"
    ....
}
```

Here we try to enumerate some plugins and let you understand how many technologies can be accessible in such a simple way:

- internal plugins
  - Hibernate plugin[94] is the plugin with the GORM implementation for Hibernate 5 persistence framework[95]
  - Asset Pipeline[96] it is responsible for rendering in an optimal way (unification in one file, compression, minification, and cache-digests) for static resource in Web pages;

[92] http://logback.qos.ch
[93] https://grails.org/plugins.html
[94] http://plugins.grails.org/plugin/hibernate5
[95] http://hibernate.org
[96] http://plugins.grails.org/plugin/asset-pipeline-grails

it is a replacement for the old Resources plugin[97] from Grails <2.x and there are other plugins[98] responsible for other aspects of serving resources: compile of CSS from LESS and SASS, trans-piling of Javascript from CoffeeScript

- external plugins
  - Spring Security Core Plugin[99] is the plugin responsible for the security of the Grails applications; it is based on Spring Security[100] library and it has a lot of companion plugins for other aspects of security - for example Spring Security REST Plugin[101] for REST API security
  - Quartz plugin for Grails[102] is the plugin responsible with scheduling jobs based on a well known Java library Quartz[103]; you can use it with another plugin Quartz Monitor Grails Plugin[104] responsible for monitoring the list of jobs
  - Mail plugin[105] is the plugin that can be used to send emails via a configurable SMTP server
  - Elastic Search plugin[106] will let you implement search for your site; your data is indexed using Elastic Search[107] library
  - Audit Logging plugin[108] can be used for auditing your application activity

About plugin's benefits:

- should be declared in configuration files and are resolved as dependencies in a Maven, Ivy, Gradle style
- expose Grails constructs like *services*, *controllers*, *views*, *entities*, et cetera
- come with configurations that must be added to main app configuration file
- easy to integrate Java, Groovy and even client side JavaScript libraries into Grails applications

Profiles

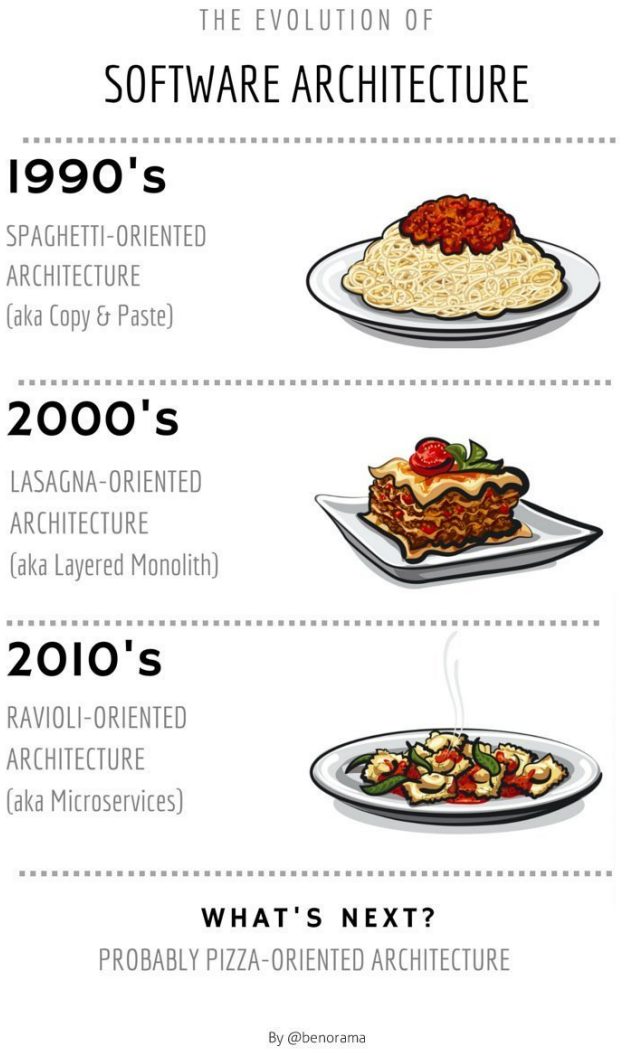
Profiles were introduced in Grails 3 and now we can create different types of applications, not just the main type that was available till now, based on a server side HTML rendered interface using technologies like Gsp (based on a MVC framework on the server side). Now we can create a Grails 3 application that is just exposing a web service API without a classical server side rendered Web interface. Also, we can create a rich client directly from the server side - using scaffolding in the same way we have done till now for the classical type.

This is in accord with the global trend of IT industry these days which is trying to break the monolith application into small pieces, introducing the concept of Miroservices[109]. In these days,

[97] https://grails.org/plugin/resources
[98] http://plugins.grails.org/q/asset%20pipeline
[99] http://plugins.grails.org/plugin/spring-security-core
[100] http://projects.spring.io/spring-security
[101] http://plugins.grails.org/plugin/spring-security-rest
[102] http://plugins.grails.org/plugin/quartz
[103] http://www.quartz-scheduler.org
[104] http://plugins.grails.org/plugin/org.grails.plugins:quartz-monitor
[105] http://plugins.grails.org/plugin/mail
[106] http://plugins.grails.org/plugin/elasticsearch
[107] https://www.elastic.co
[108] http://plugins.grails.org/plugin/audit-logging
[109] http://martinfowler.com/articles/microservices.html

Java EE is creating a new Microprofile[110] for microservices in addition to their Full Profile and Web Profile.

You can see very clear the evolution of software architecture in the last decades in a tweet from Benoit Hediard[111]:



Software architecture evolution

Another reason for this profiles movement is the need to expose just Web services and not a HTML based Web interface. And this is more valuable in the context of the mobile application and the *mobile first*[112] approach in developing applications. That's why a new set of server side rendering technologies were introduced in Grails 3 like Grails Views[113] that are rendering JSON and XML from domain objects.

Here you have the complete list of Grails 3 profiles[114]. Here is the Git official repository for Grails

[110] http://microprofile.io
[111] https://twitter.com/benorama
[112] http://blogs.atlassian.com/2012/01/modern-principles-in-web-development
[113] http://views.grails.org/latest/
[114] https://github.com/grails-profiles

3 profiles[115].

These are the profiles introduced in Grails 3:

- angular - A profile for creating applications using AngularJS
- rest-api - Profile for REST API applications
- base - The base profile extended by other profiles
- angular2 - A profile for creating Grails applications with Angular 2
- plugin - Profile for plugins designed to work across all profiles
- profile - A profile for creating new Grails profiles
- react - A profile for creating Grails applications with a React frontend
- rest-api-plugin - Profile for REST API plugins
- web - Profile for Web applications
- web-plugin - Profile for Plugins designed for Web applications
- webpack - A profile for creating applications with node-based frontends using webpack

Some details about the most important profiles that will be used in the application built in this book:

- web - is a profile for creating classical Grails applications with a server side MVC framework in the same way it was in Grails 1 and 2; the presentation web interface is rendered on the server side
- rest-api - is a profile for creating applications that are exposing REST APIs; don't have a Web presentation and no MVC is needed
- angular(2) - is a profile for creating applications with a rich client based on AngularJS[116](1 or 2 versions) Javascript framework where MVC is located on client side (JavaScript) and the communication with the server side is via REST

[115] https://github.com/grails/grails-profile-repository/tree/master/profiles
[116] https://angularjs.org

The Mobile Application - Rest API Profile

About this chapter content

First, we will install Ionic 2[117] and all the development tools for Android, iOs and Windows Phone. We will generate our client app using Ionic CLI[118], and we will describe the anatomy of an Ionic 2 application. We will improve and document our exposed REST API, and we will add security using JWS tokens. We will deploy our app to all the tree known mobile app stores.

Here are the primary resources for this application:

- the source code for the Ionic 2 client is stored in GitHub: https://github.com/colaru/mvp-application/tree/master/app-web/ionic-client
- the REST API documentation: https://app.swaggerhub.com/api/colaru/ItJobsBoard/1.0.0
- mobile app in GooglePlay Store: https://play.google.com/store[119]
- mobile app in Apple AppStore: http://www.apple.com/itunes[120]
- mobile app in Microsoft Store: https://www.microsoft.com/en-us/store/top-free/apps/mobile[121]

Introduction to Ionic 2 and Hybrid Mobile Applications

There are a big number of smartphones and tablets on the market, each having a given operating system and is produced by a given manufacturer. We live in the years when the number of mobile phones overcomes the number of desktop and laptop computers, and the difference is still growing (Google is reporting that there are more searches from mobile now than from desktops).

There are some stores today where you can publish your mobile application based on its operating system. There are some signs[122] that in a number of devices Android is the winner in the market (~70%), followed by iOS (~20%). Windows Phone has just a few percents but they choose to invest in mobile in the last time and your application can be also available to other Windows 10 devices like PC, tablet, Xbox due to UWP[123] (Windows 10 is around 25% from desktop OS market). Here is the list of major mobile operating systems on the market:

[117] http://blog.ionic.io/announcing-ionic-2-0-0-final
[118] http://ionicframework.com/docs/v2/cli
[119] https://play.google.com/store/apps/details?id=eu.itjobsboard.mobile.app
[120] https://itunes.apple.com/us/app/itjobsboard/id1209609917
[121] https://www.microsoft.com/en-us/store/p/itjobsboard/9nj1zw45vlw7
[122] http://www.idc.com/promo/smartphone-market-share/os
[123] https://docs.microsoft.com/en-us/windows/uwp/get-started/whats-a-uwp

Devices	OS	Store	Language
Android phones and tablets	Google Android	PlayStore	Android(Java)
iPhone and iPad	Apple iOs	AppStore	Objective-C or Swift
Microsoft phones and tablets	Windows Phone	WindowsStore	C#, Visual Basic

Google is the owner of Android OS[124] and the devices are manufactured by many others like Samsung, Motorola, etc. There is a reference device Google Pixel[125] series manufactured by Google. Till now it was Nexus, but the manufacturer was not Google. The programming language is Java (Android is derived from Java), and the development tool is Android Studio (based on IntelliJ Idea now, in the past was Eclipse) which can run on Linux, Windows or Mac operation systems.

Apple is the owner of the iOs operating system[126] and is also the manufacturer of the iPhones[127] and iPads[128]. The programming language in Objective C[129] with the addition of a more scripting language named Swift[130]. The development tool is named XCode[131], and you need a Mac OS system to develop your application.

Microsoft is the owner of Windows Phone[132] operating system and is manufacturing itself some physical devices - based on acquired Nokia[133] company. Starting with version 10, the Windows and Windows Phone are sharing the same store and is possible to publish a mobile app and will be available on mobile and desktop at the same time due to Universal Windows Platform - UWP[134]. You can program in C# or other .Net language. The development tool is Microsoft Visio Studio[135] available on Windows OS.

It is hard to develop applications for all the stores because you have to know a multitude of languages Android(Java) for Java, Objective-C or Swift for iOs, C# or Visual Basic for Windows. A solution for this (write once publish everywhere) is to create hybrid applications using tools like Ionic[136].

Ionic is a tool for creating hybrid mobile applications combining some other tools: Apache Cordova[137] and AngularJS 2[138]. An Ionic application is a web HTML5 AngularJS application (composed from HTML, JS, CSS) that is running in a browser on the device WebView[139]. It has access to the native platform via JavaScript calls offered by Cordova framework[140]. Ionic 2 is

[124] https://www.android.com
[125] https://madeby.google.com/phone
[126] http://www.apple.com/ios
[127] http://www.apple.com/iphone
[128] http://www.apple.com/ipad
[129] https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html
[130] https://developer.apple.com/swift/
[131] https://developer.apple.com/xcode
[132] https://www.microsoft.com/en/mobile/windows10
[133] https://www.microsoft.com/en/mobile/phones/lumia
[134] https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide
[135] https://www.visualstudio.com
[136] https://ionicframework.com
[137] https://cordova.apache.org
[138] https://angular.io
[139] https://cordova.apache.org/docs/en/latest/guide/overview/#webview
[140] https://cordova.apache.org/docs/en/latest/guide/overview

a rewrite of Ionic 1 with AngularJS 2 inside (AngularJS 2 is also using TypeScript[141] instead of JavaScript). We will use this second version of the Ionic framework in this book.

Install Ionic 2

First, we will install Ionic 2[142] on our machine, and for this, we need NodeJS[143]. Download (LTS - Long Term Support version) and install it on your machine. After installation you can verify:

```
npm version
```

You will get something similar with this:

```
{ npm: '3.10.10',
  ares: '1.10.1-DEV',
  http_parser: '2.7.0',
  icu: '57.1',
  modules: '48',
  node: '6.9.5',
  openssl: '1.0.2k',
  uv: '1.9.1',
  v8: '5.1.281.89',
  zlib: '1.2.8' }
```

We have NodeJS 6.9.5 and NPM 3.10.10. We need NPM[144] for managing JavaScript libraries (similar with yum or apt package managers in Linux). Next, we install *ionic* and *cordova* using NPM:

```
npm install -g ionic cordova
```

You can

```
ionic -version
cordova --version
```

I get 2.0.0 for Ionic and 6.5.0 for Cordova.

You have to install Android platform and iOS platform on your machine (we hope you have a Mac OS to be able to install both). This because you have to publish your application in store, you need the binary for each platform *.apk* for Android *.ipa* for iOS.

For Android, you can follow this Cordova Android Platform Guide[145]. If you already have Java JDK installed you have to install Android Studio[146] (you will be familiar with it if you are already using IntelliJ Idea) which is coming with:

[141] https://www.typescriptlang.org/index.html
[142] http://ionicframework.com/docs/v2/intro/installation
[143] https://nodejs.org/en
[144] https://www.npmjs.com
[145] https://cordova.apache.org/docs/en/latest/guide/platforms/android
[146] https://developer.android.com/studio/install.html?pkg=studio

- SDK Manager - will let you install the Android SDK (at a given level)
- ADV Manager - will let you configure the Android emulators
- Android Device Monitor- will let you see your devices logs

For iOS, you can follow this Cordova iOS Platform Guide[147]. You have to install the XCode first which is coming with the SDK, iOs and iPad emulators and all you need to deploy your app in production.

For Windows 10 follow this Cordova Windows Platform Guide[148]. You have to install at least Visual Studio Community Edition 2015/2017 which is free. You will get the Windows phone SDK, the emulator and all you need to deploy your application in production.

You can use a virtualization solution (VirtualBox for example) if you are on Mac or Linux for running the Windows OS - you can get images available for free for one month[149]. If you want a Linux box you have OsBoxes[150].

Generate client app

Now you are prepared to start your first application using Ionic 2 and its Command-Line-Interface - CLI[151]. We will generate a *sidemenu* starter application - we will have a side menu. There are two other alternatives *blank* and *tabs*. We will locate our client in *app-web* module near our static HTML client we made.

```
cd ~/MVPS/mvp-application/app-web
ionic start --v2 ionic-client sidemenu
```

We used the –v2 flag to create a Ionic 2 project (otherwise, the old Ionic 1 project is generated). The confirmation:

```
Creating Ionic app in folder ~/MVPS/mvp-application/app-web/ionic-client based on sidem\
enu project
Downloading: https://github.com/driftyco/ionic2-app-base/archive/master.zip
[==============================] 100% 0.0s
Downloading: https://github.com/driftyco/ionic2-starter-sidemenu/archive/master.zip
[==============================] 100% 0.0s
Installing npm packages...

Adding initial native plugins
[==============================] 100% 0.0s

Adding in iOS application by default

```

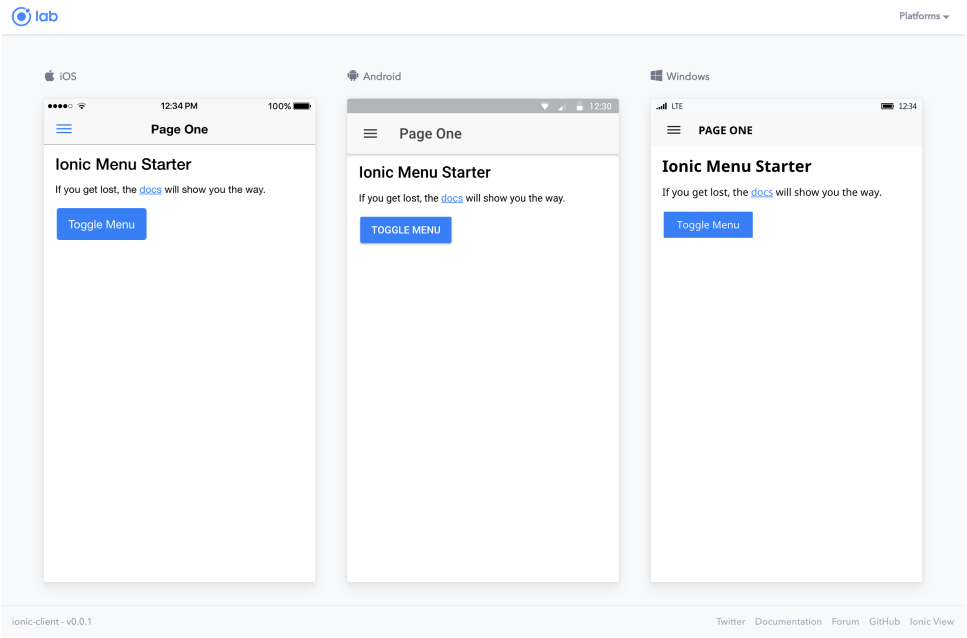
[147] https://cordova.apache.org/docs/en/latest/guide/platforms/ios
[148] https://cordova.apache.org/docs/en/latest/guide/platforms/win8/index.html
[149] https://developer.microsoft.com/en-us/windows/downloads/virtual-machines
[150] http://www.osboxes.org
[151] http://ionicframework.com/docs/v2/cli

```
Saving your Ionic app state of platforms and plugins
Saved platform
Saved plugins
Saved package.json

 Your Ionic app is ready to go! 
.....
```

Now build it and run it:

```
ionic build
ionic serve --lab
```

We will see the app as it looks on all three platforms:



Ionic generated client

Now, if we look at the structure of generated app:

```
tree ionic-client -L 2 | pbcopy
```

We will see this folders structure:

```
ionic-client
├── config.xml
├── hooks
│   └── README.md
├── ionic.config.json
├── node_modules
│   ├── @angular
│   ├── @ionic
│   ├── @types
├── package.json
├── platforms
│   ├── ios
│   └── platforms.json
├── plugins
│   ├── cordova-plugin-console
│   ├── cordova-plugin-device
│   ├── cordova-plugin-splashscreen
│   ├── cordova-plugin-statusbar
│   ├── cordova-plugin-whitelist
│   ├── fetch.json
│   ├── ionic-plugin-keyboard
│   └── ios.json
├── resources
│   ├── android
│   ├── icon.png
│   ├── ios
│   └── splash.png
├── src
│   ├── app
│   ├── assets
│   ├── declarations.d.ts
│   ├── index.html
│   ├── manifest.json
│   ├── pages
│   ├── service-worker.js
│   └── theme
├── tsconfig.json
├── tslint.json
└── www
    ├── assets
    ├── build
    ├── index.html
    ├── manifest.json
    └── service-worker.js
```

This is a classical Ionic 2 application folders structure. We describe here the main folders and files:

- config.xml - is the main configuration file for an Ionic application; you describe your app

metadata, required device access rights, preferences and used Cordova plugins.
- /plugins - is the place where all Cordova plugins are located
- /www - here are the application place after build
- /src - is where the Angular 2 application is located

Angular 2

We will investigate the generated Ionic application from the Angular 2 point of view. And we will make some changes - we will prepare the app for our needs. A significant change from previews version of Angular is the use of TypeScript in replacement for JavaScript - you can see the *.ts* extensions for script files. There are other changes from an Ionic 1, but the main concepts are the same - a migration guide[152] is available.

Let see the structure of the app:

```
tree ~/MVPS/mvp-application/app-web/ionic-client/src
```

We will see:

```
├── app
│   ├── app.component.ts
│   ├── app.html
│   ├── app.module.ts
│   ├── app.scss
│   └── main.ts
├── assets
│   └── icon
│       └── favicon.ico
├── declarations.d.ts
├── index.html
├── manifest.json
├── pages
│   ├── page1
│   │   ├── page1.html
│   │   ├── page1.scss
│   │   └── page1.ts
│   └── page2
│       ├── page2.html
│       ├── page2.scss
│       └── page2.ts
├── service-worker.js
└── theme
    └── variables.scss
```

We see that we have 2 pages. For each page, we have an entry in the *pages* folder. Let's create other pages using Ionic CLI[153].

First, let's create a new *index* page:

[152] https://ionicframework.com/docs/v2/intro/migration
[153] https://ionicframework.com/docs/v2/cli/generate

```
ionic g page index
```

Then a page for showing the jobs:

```
ionic g page jobs
```

And we will delete the default created pages (Page1 and Page2):

```
tree ~/MVPS/mvp-application/app-web/ionic-client/src/pages
```

We will see:

```
├── index
│   ├── index.html
│   ├── index.scss
│   └── index.ts
└── jobs
    ├── jobs.html
    ├── jobs.scss
    └── jobs.ts
```

Let's create a service for communicating with the server:

```
ionic g provider jobsData
```

The service will be created on *providers* folder and will be named *jobs-data.ts*:

```
nano ~/MVPS/mvp-application/app-web/ionic-client/src/providers/jobs-data.ts
```

We will add the code for listing the jobs from REST API:

```
import {Injectable} from '@angular/core';
import {Http} from '@angular/http';
import 'rxjs/add/operator/map';
import { GlobalVariable } from '../app/global';

@Injectable()
export class JobsData {

  data;

  constructor(public http: Http) {
    console.log('Enter JobsData Provider');
  }

  listJobs() {
    return this.http.get(GlobalVariable.BASE_API_URL + "/job").map((res) => res.json())
  }
}
```

We will inject this service in our page component and we will call the *listJobs* method:

```
sublime ~/MVPS/mvp-application/app-web/ionic-client/src/pages/jobs/jobs.ts
```

The code:

```
import { Component } from '@angular/core';
import { NavController, NavParams } from 'ionic-angular';
import { JobsData } from "../../providers/jobs-data";
import { GlobalVariable } from '../../app/global';

@Component({
  selector: 'page-jobs',
  templateUrl: 'jobs.html'
})
export class JobsPage {

  selectedItem: any;
  icons: string[];
  items: any;
  contextPath: string;

  constructor(public navCtrl: NavController, public navParams: NavParams, public jobsDa\
ta: JobsData) {
    // If we navigated to this page, we will have an item available as a nav param
    this.selectedItem = navParams.get('item');
    this.contextPath = GlobalVariable.BASE_API_URL

    jobsData.listJobs().subscribe(data => {
      this.items = data;
      console.log("Data: " + JSON.stringify(this.items, null, 2))
    });
  }

  itemTapped(event, item) {
    this.navCtrl.push(JobsPage, {
      item: item
    });
  }
}
```

And we will adapt the page template to show the jobs as cards[154]:

```
nano ~/MVPS/mvp-application/app-web/ionic-client/src/pages/jobs/jobs.html
```

And here is the code:

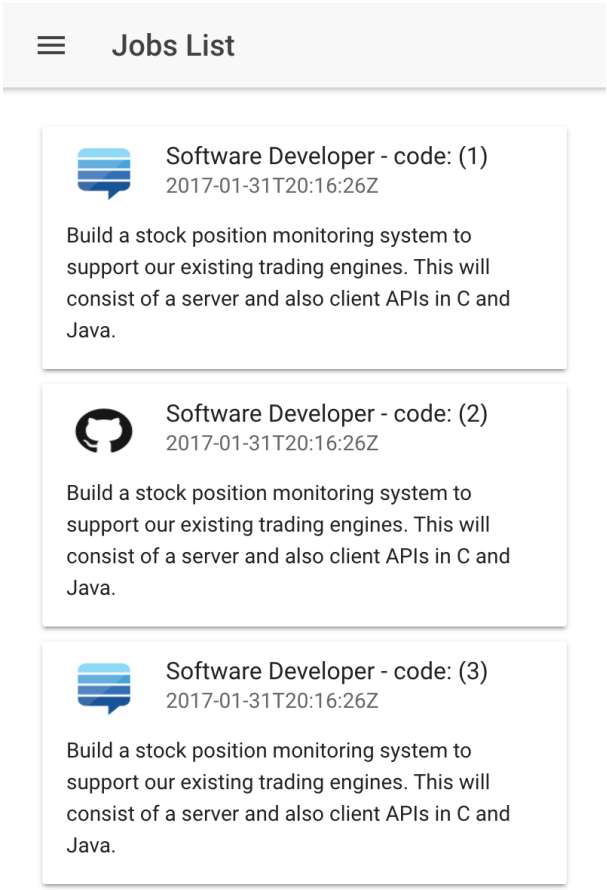
[154] https://ionicframework.com/docs/v2/components/#cards

```
<ion-header>
  <ion-navbar>
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>Jobs List</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding class="cards-bg social-cards">
  <ion-card *ngFor="let item of items" (click)="itemTapped($event, item)">

    <button ion-item >
      <ion-avatar item-left>
        <img style="width: 50px; length: 50px" src="{{contextPath}}/renderLogo?id={{ite\
m.publisher.id}}"
            alt="{{item.publisher.name}}"
            class="img-responsive">
      </ion-avatar>
      <h2>{{item.title}}</h2>
      <p>{{item.dateCreated}}</p>
    </button>

    <ion-card-content>
      <p>{{item.description}}</p>
    </ion-card-content>
  </ion-card>

  <div *ngIf="selectedItem" padding>
    You navigated here from <b>{{selectedItem.title}}</b>
  </div>
</ion-content>
```

The result is here:

```
ionic serve
```

The jobs page on our web page:

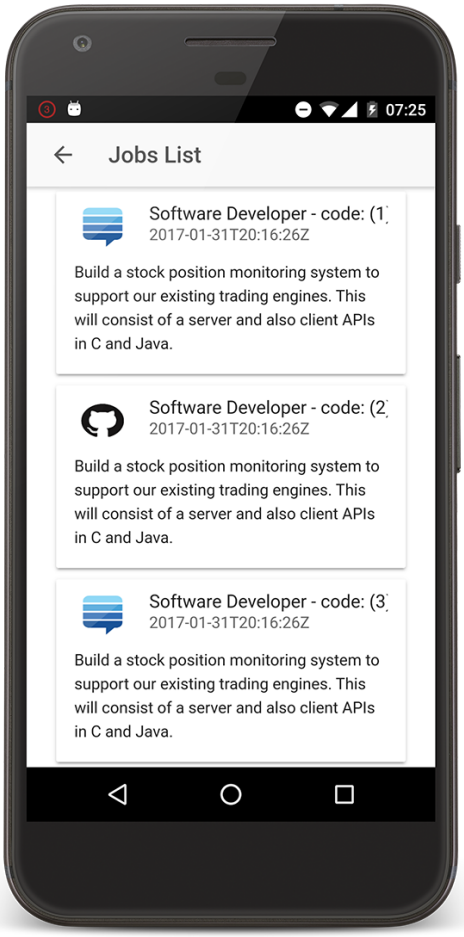


List of jobs on Web page

The result is here:

```
1 ionic run android
```

The jobs page on our device:



List of jobs on Android device

Cordova

Apache Cordova[155] is a part of Ionic Framework and is a JavaScript framework that can be used to make a native call from a browser to the native platform, independent of its type: Android, iOs or Windows Phone. It is exposing a unified API that can be used on all three platforms (or other supported platforms). First, it was named PhoneGap[156] and it was acquired by Adobe, and became Apache project under the name Apache Cordova. Here is a discussion[157] about the distinctions between the two.

First of all, the main configurations for the mobile application are in *config.xml* file. Lets change the name of the application and other metadata:

[155] https://cordova.apache.org
[156] http://phonegap.com
[157] http://phonegap.com/blog/2012/03/19/phonegap-cordova-and-whate28099s-in-a-name

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<widget id="eu.itjobsboard.mobile" version="0.0.1"
xmlns="http://www.w3.org/ns/widgets"
xmlns:cdv="http://cordova.apache.org/ns/1.0">
  <name>ItJobsBoard</name>
  <description>It jobs board application</description>
  <author email="admin@itjobsboard.eu"
  href="http://itjobsboard.eu">It Jobs Board Team
  </author>
.....
</widget>
```

We will discuss two aspects of Cordova: plugins and platforms.

Cordova, it is a plugin-able system, and there is a list of plugins[158] available on the market (mostly free). You can see the list of plugins already installed on your application:

```
ionic plugin list
```

You will see:

```
cordova-plugin-console 1.0.5 "Console"
cordova-plugin-device 1.1.4 "Device"
cordova-plugin-splashscreen 4.0.1 "Splashscreen"
cordova-plugin-statusbar 2.2.1 "StatusBar"
cordova-plugin-whitelist 1.3.1 "Whitelist"
ionic-plugin-keyboard 2.2.1 "Keyboard"
```

They are located on *plugins* folder:

```
tree ~/MVPS/mvp-application/app-web/ionic-client/plugins -L 1
```

You will see:

```
plugins
├── android.json
├── cordova-plugin-console
├── cordova-plugin-device
├── cordova-plugin-splashscreen
├── cordova-plugin-statusbar
├── cordova-plugin-whitelist
├── fetch.json
├── ionic-plugin-keyboard
└── ios.json
```

And are declared on *config.xml* file:

[158] https://cordova.apache.org/plugins

```
cat ~/MVPS/mvp-application/app-web/ionic-client/config.xml
```

You will find the list of declared plugins:

```
<plugin name="ionic-plugin-keyboard" spec="~2.2.1"/>
<plugin name="cordova-plugin-whitelist" spec="1.3.1"/>
<plugin name="cordova-plugin-console" spec="1.0.5"/>
<plugin name="cordova-plugin-statusbar" spec="2.2.1"/>
<plugin name="cordova-plugin-device" spec="1.1.4"/>
<plugin name="cordova-plugin-splashscreen" spec="~4.0.1"/>
```

Each time you want a plugin you have to declare it here in *config.xml*.

Cordova let you install platforms - various operating systems where you can publish your application. The *ios* platform was added the first time I generated the application. We can add Android in this way:

```
ionic platform app android
```

If you want to see a list of platforms, use this command:

```
ionic platforms list
```

And the response:

```
Installed platforms:
  android 6.1.2
  ios 4.1.1
Available platforms:
  amazon-fireos ~3.6.3 (deprecated)
  blackberry10 ~3.8.0
  browser ~4.1.0
  firefoxos ~3.6.3
  osx ~4.0.1
  webos ~3.7.0
```

The platform content is stored in *platforms* folder:

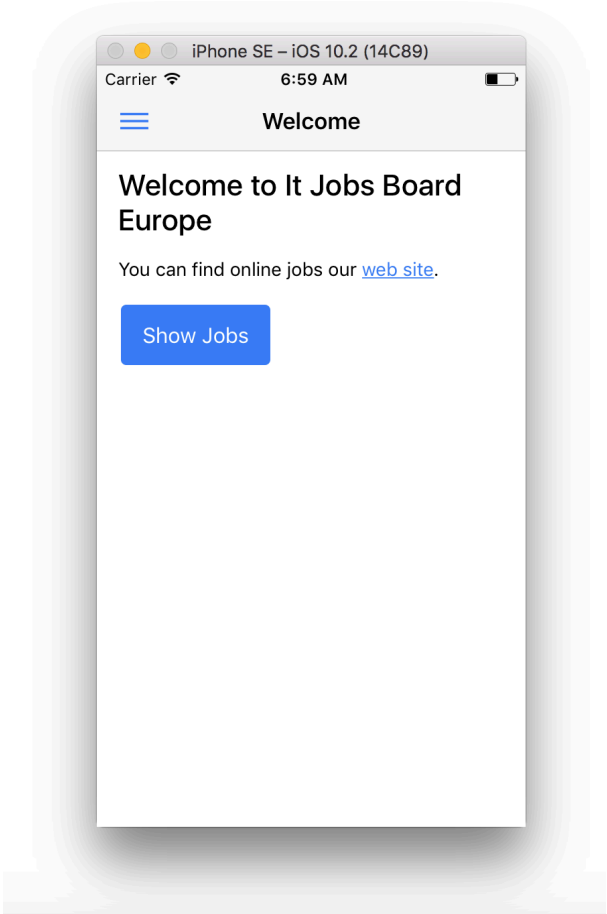
```
tree ~/MVPS/mvp-application/app-web/ionic-client/platforms -L 1
```

You will see the installed platforms:

```
platforms
├── android
├── ios
└── platforms.json
```

Now we can start to work on a given platform. We can use the *iOs* emulator to play our application:

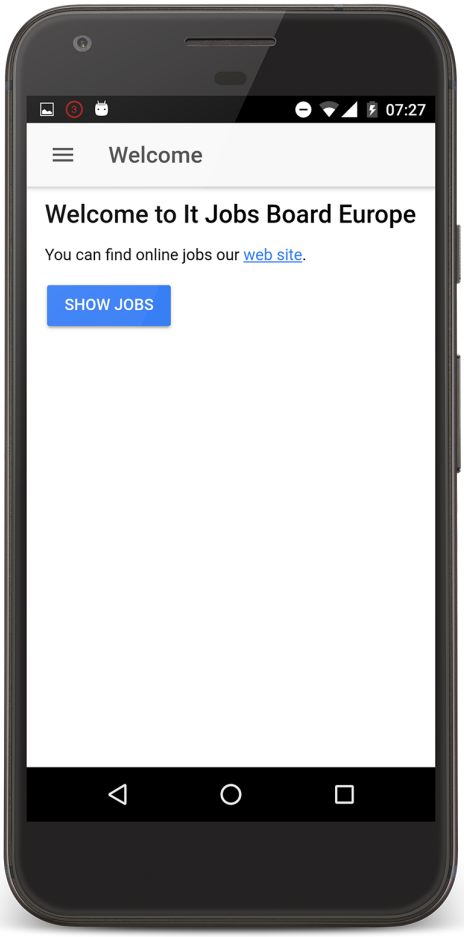
```
ionic emulate ios
```



The application running on iOS emulator

Or we can run our application on an Android device connected by USB to our computer (it has to be set in developer mode):

```
ionic run android
```



The application running on an Android device

If you want to see the list of available commands, run:

```
1 ionic --list
```

Ionic Native

Ionic Native[159] is a collection of the best free Cordova plugins already integrated into Ionic 2 - you still have to declare and install the plugin, but you have the API exposed in Angular 2 style. The Ionic Native script is already installed in any new generated Ionic 2 application. It is a replacement for ngCordova[160] from Ionic 1. We will enumerate just a few plugins (part of Ionic Native) that can be used in your commercial application just for illustrating the possible native features that can be accessed via plugins:

[159] http://ionicframework.com/docs/v2/native
[160] http://ngcordova.com

Plugin name	*Description*
AdMob[161]	Plugin for Google Ads, including AdMob / DFP (double-click for publisher) and mediations to other Ad networks.
App Rate[162]	The AppRate plugin makes it easy to prompt the user to rate your app, either now, later, or never.
File[163]	This plugin implements a File API allowing read/write access to files residing on the device.
Geolocation[164]	This plugin provides information about the device's location, such as latitude and longitude.
Google Analitycs[165]	This plugin connects to Google's native Universal Analytics SDK
InApp Purchase[166]	A lightweight Cordova plugin for in app purchases on iOS/Android.
InAppBrowser[167]	Launches in app Browser
Splashscreen[168]	This plugin displays and hides a splash screen during application launch.
SMS[169]	Send SMS from your application

You can use all the Cordova plugins[170] even if they are not part of Ionic Native. You can also use commercial plugins - Ionic is offering a marketplace on Ionic Market[171] for this.

We will investigate 2 plugins here. First, we will change the default splash-screen (and the application icon) showing the *Splashscreen plugin* already used in the generated app. And we will introduce *InAppBrowser plugin* for displaying our external links in our application (not into a different window in the system browser).

So, the *Splashscreen* plugin is declared in *config.xml* file and is used in *app.component.ts* where the splash-screen is hidden after the application is started:

[161] http://ionicframework.com/docs/v2/native/admob
[162] http://ionicframework.com/docs/v2/native/app-rate
[163] http://ionicframework.com/docs/v2/native/file
[164] http://ionicframework.com/docs/v2/native/geolocation
[165] http://ionicframework.com/docs/v2/native/google-analytics
[166] http://ionicframework.com/docs/v2/native/inapppurchase
[167] http://ionicframework.com/docs/v2/native/inappbrowser
[168] http://ionicframework.com/docs/v2/native/splashscreen
[169] http://ionicframework.com/docs/v2/native/sms
[170] https://cordova.apache.org/plugins
[171] https://market.ionic.io/plugins

```
import { StatusBar, Splashscreen } from 'ionic-native';
.....
initializeApp() {
  this.platform.ready().then(() => {
    // Okay, so the platform is ready and our plugins are available.
    // Here you can do any higher level native things you might need.
    StatusBar.styleDefault();
    Splashscreen.hide();
  });
}
```

All that we will do is to replace the default splash-screen image and icon[172] generated by Ionic. These images are stored in *resources* folder:

```
tree ~/MVPS/mvp-application/app-web/ionic-client/resources -L 2
```

The files:

```
├── android
│   ├── icon
│   └── splash
├── icon.png
├── ios
│   ├── icon
│   └── splash
└── splash.png
```

The icon is *icon.png* and the splash-screen is *splash.png*. From these 2 files we can generate all the resources files for our platforms (various devices with different screens resolutions) using the *resources* command (after we modified the images using a tool like Gimp[173] or Photoshop[174]):

```
ionic resources
```

With the result (also the *config.xml* will be changed to reflect the newly generated files):

```
Ionic icon and splash screen resources generator
 uploading icon.png...
 uploading splash.png...
icon.png (1024x1024) upload complete
splash.png (2208x2208) upload complete
.......
```

Next we will add the *InAppBrowser plugin* entering a new line in *config.xml*:

[172] http://ionicframework.com/docs/cli/icon-splashscreen.html
[173] https://www.gimp.org
[174] http://www.adobe.com/products/photoshop.html

```
sublime ~/MVPS/mvp-application/app-web/ionic-client/config.xml
```

Add the plugin declaration in the config file (we installed it and declare it at the same time - see the *–save* option):

```
ionic plugin add cordova-plugin-inappbrowser --save
```

The confirmation:

```
Fetching plugin "cordova-plugin-inappbrowser@1.6.1" via npm
Installing "cordova-plugin-inappbrowser" for android
Installing "cordova-plugin-inappbrowser" for ios
Saved plugin info for "cordova-plugin-inappbrowser" to config.xml
```

We will change the method called when a job posted is tapped in *jobs.ts*:

```
itemTapped(event, item) {
    // we use _self instead of _system
    let browser = new InAppBrowser(item.jobUrl, '_self');
}
```

See this call from *jobs.html* where the method is called:

```
<ion-card *ngFor="let item of items" (click)="itemTapped($event, item)">
```

Rest API

In this subchapter, we will improve our REST API. We will first try to clarify what is a RET API; we will document our API using Swagger, we will add token based security for our REST API

From verbs to nouns

The way the applications can communicate in a distributed way is an old subject. First was the idea of RPC - Remote procedure calls[175] - remote calls between programs running on different machines which are exposing procedures/methods using binary data for messages transportation.

Cordoba[176] was such an initiative for communicating between different languages. Java RMI[177] it was an initiative for communicating between JVMs running on different physical machines.

The next evolution step were Web Services using HTTP protocol for communication and XML or JSON (text and not binary data) for messages. SOAP[178] was a first success in the Web Services market was but is still have to be used - is based on XML messages and still has a procedural style of exposing an API. That's way other types of Web Services emerged like REST.

REST - Representational State Transfer[179] principles were enunciated first time by https://twitter.com/fielding[180]

[175] https://en.wikipedia.org/wiki/Remote_procedure_call
[176] https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture
[177] https://en.wikipedia.org/wiki/Java_remote_method_invocation
[178] https://en.wikipedia.org/wiki/SOAP
[179] https://en.wikipedia.org/wiki/Representational_state_transfer
[180] https://twitter.com/fielding

in his 2000 Ph.D. dissertation Architectural Styles and the Design of Network-based Software Architectures[181]

REST is built around HTTP standard: * you expose resources to the world (nouns) * we don't have other methods to operate on the resources than the HTTP commands (GET, POST, PUT, DELETE, ...) (verbs) * the HTTP error codes are used for errors (2XX, 3XX, 4XX, ...) * you have a unique intelligible URI to operate with a resource * texts are used for messages (JSON is preferred when we can process it from JavaScript on client side)

Swagger specification

One of the interesting parts of SOAP is the WSDL - Web Services Description Language[182] specification. WSDL specification is a contract describing the exposed API that can be used in many ways. Can be used to generate the client or the server. It can also be itself generated by the server side. It is the central point of a SOAP API.

In the case of REST there were some tries to define such an interface (WADL, RAML) but finally, we have a winner on the market. It is Open API[183] that is a try to standardize a well-known schema model globally from Swagger[184]. Will let you define your API in JSON or YAML. Swagger let you not just to specify your API but also to use this specification for documenting your API - you can easily generate a documentation of your API for your users, and they can test it in real-time.

Let's document our REST API. There are some Grails 2 plugins for generating the Swager spec via annotations and show the spec in Swagger-UI, but none of them is working in Grails 3. Swaggydoc plugin[185] has an update for Grails 3 but is not working - the project is no longer maintained.

The solution, in this case, is to use the tools that Swagger site is offering. They are offering an online Swagger editor[186] for editing our Swagger specs in JSON or YMAL. We have the chance to generate clients or servers in a multitude of programming languages from Swagger specifications using Swagger Codegen[187]. We also have an option to store a Swagger specification online and to view it on Swagger-UI on Swagger Hub[188]. We created a hosted Swagger specification for the version *1.0.0* of our project:

```
1 open https://app.swaggerhub.com/api/colaru/ItJobsBoard/1.0.0
```

Swagger YAML content:

[181] http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
[182] https://en.wikipedia.org/wiki/Web_Services_Description_Language
[183] https://github.com/OAI/OpenAPI-Specification
[184] http://swagger.io
[185] https://grails.org/plugin/swaggydoc
[186] http://swagger.io/swagger-editor
[187] http://swagger.io/swagger-codegen
[188] https://app.swaggerhub.com

```yaml
swagger: '2.0'
info:
  description: It Jobs Board Europe API
  version: "1.0.0"
  title: itjobsboard.eu exposed API
  contact:
    email: admin@itjobsboard.eu
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html

tags:
- name: admins
  description: Secured Admin-only calls
- name: customers
  description: Operations available to regular developers
schemes:
- https
paths:
  /jobs:
    get:
      tags:
      - customers
      summary: list jobs
      operationId: list
      description: |
        List all available jobs in the system.
      produces:
      - application/json
      parameters:
      - in: query
        name: limit
        description: maximum number of records to return
        type: integer
        format: int32
        minimum: 0
        maximum: 50
      responses:
        200:
          description: search results matching criteria
          schema:
            type: array
            items:
              $ref: '#/definitions/Job'
        400:
          description: bad input parameter
definitions:
  Job:
    type: object
```

```
    required:
    - id
    - active
    - applyInstructions
    - contactEmail
    - description
    - jobUrl
    - title
    properties:
      id:
        type: number
        example: 1
      active:
        type: boolean
        example: true
      applyInstructions:
        type: string
        example: admin@stackoverflow.com
      contactEmail:
        type: string
        example: admin@stackoverflow.com
      dateCreated:
        type: string
        format: int32
        example: 2017-01-31T20:16:26Z
      description:
        type: string
        example: Build a stock position monitoring system to support our existing tradi\
ng engines. This will consist of a server and also client APIs in C and Java.
      expirationDate:
        type: string
        format: int32
        example: 2017-01-31T20:16:26Z
      jobUrl:
        type: string
        example: https://stackoverflow.com/jobs/132418/software-developer-silver-fern-i\
nvestments
      lastUpdated:
        type: string
        format: int32
        example: 2017-01-31T20:16:26Z
      publisher:
        $ref: "#/definitions/Publisher"
      remote:
        type: boolean
        example: true
      salaryEstimate:
        type: string
        example: 1000000 per year
```

```
    tags:
      type: array
      items:
        $ref: "#/definitions/Type"
    title:
      type: string
      example: Software Developer
    type:
      $ref: "#/definitions/Type"
  Publisher:
    type: object
    required:
    - id
    properties:
      id:
        type: number
        example: 1
  Type:
    type: object
    required:
    - id
    properties:
      id:
        type: number
        example: 1
  Tag:
    type: object
    required:
    - id
    properties:
      id:
        type: number
        example: 1

host: itjobsboard.eu
basePath: /api/v1.0/
```

REST Security on the server side

We will introduce security for our Ionic application based on Spring Security for Grails plugin[189]. In this case, we use *JWT cryptographic tokens - JSON Web Tokens*[190] instead of classical Java Web session based security used in the admin application. This is a full stateless style of offering security to application specific to B2C applications - the clients are not able to store a secret, and in this case, the token is often expired and it is transported via HTTPS[191].

[189] http://plugins.grails.org/plugin/grails/spring-security-rest

[190] https://jwt.io/introduction

[191] https://stormpath.com/blog/build-secure-user-interfaces-using-jwts

The creator of the Rest Security plugin created a workshop to accommodate with REST security Creating applications with Grails, Angular, and Spring Security[192].

The simple way to implement REST security in our application is to create at the beginning your Angular(2) profile Web application with rest security enabled - *–features security*. This will generate your client and server already configured for security. In our case, the *web-app* is already generated without security. We have to do this by hand.

Now it is a good time for versioning our REST API - a good practice because the clients will become dependent of our API and we have to migrate to new versions of API without breaking the clients. For guidelines about how to version an API see the Semantic Versioning specification[193]. We will create a first version *1.0.0* for our API, and we will specify two new URLs for our REST API:

```
nano ~/MVPS/mvp-application/app-web/grails-app/controllers/app/web/UrlMappings.groovy
```

We will expose two resources - jobs and featured jobs. We want *the list of featured jobs* to be seen by everybody, but we want the entire *list of jobs* to be visible just for authenticated users (it is not a really security case here but is enough to illustrate the security concepts):

```
"/api/v1.0/jobs"(controller: "job", action: "listAll")
"/api/v1.0/featuredJobs"(controller: "job", action: "listFeatured")
```

We create two new controllers in *JobController*:

```
nano ~/MVPS/mvp-application/app-web/grails-app/controllers/app/admin/jobsboard/JobContr\
oller.groovy
```

And the controllers:

```
@Secured(["ROLE_CUSTOMER"])
def listAll(Integer max) {
    params.max = Math.min(max ?: 10, 100)
    respond Job.list(params), model:[jobCount: Job.count()]
}

@Secured(["permitAll"])
def listFeatured(Integer max) {
    params.max = 1
    params.order = "desc"
    params.sort = "title"
    respond Job.list(params), model:[jobCount: Job.count()]
}
```

Now we will protect these resources accordingly using JWT tokens security. For this, first we have to change the *security core plugin* with *rest security plugin* (all the core features are still part of this REST plugin):

[192] http://alvarosanchez.github.io/grails-angularjs-springsecurity-workshop
[193] http://semver.org

```
sublime ~/MVPS/mvp-application/app-web/build.gradle
```

We will change to:

```
//    compile 'org.grails.plugins:spring-security-core:3.1.1'
compile "org.grails.plugins:spring-security-rest:2.0.0.M2"
```

If we restart the app we will have the confirmation of the installation for this REST plugin (and the core plugin is still there):

```
Configuring Spring Security Core ...
... finished configuring Spring Security Core


Configuring Spring Security REST 2.0.0.M2...
... finished configuring Spring Security REST
```

Now we have to configure the security filters for the newly added plugin. We have to add the stateless filters for our API path */api/*** (we keep state-full filters for this path */ *). *Also because we will have *permitAll* access to some of our API resources we will enable *enableAnonymousAccess = true* for our version *1.0* of our API and set stateless filters to it:

```
sublime ~/MVPS/mvp-application/app-web/grails-app/conf/application.groovy
```

Now the security settings for filters chains will look like this:

```
grails {
    plugin {
        springsecurity {
            filterChain {
                chainMap = [
                        //Stateless chain
                        [pattern: '/api/v1.0/**',    filters: 'anonymousAuthenticationFilter,restTokenVal\
idationFilter,restExceptionTranslationFilter,filterInvocationInterceptor'],
                        [pattern: '/api/**',        filters: 'JOINED_FILTERS,-anonymousAuthenticationFil\
ter,-exceptionTranslationFilter,-authenticationProcessingFilter,-securityContextPersist\
enceFilter,-rememberMeAuthenticationFilter'],

                        [pattern: '/assets/**',      filters: 'none'],
                        [pattern: '/**/js/**',       filters: 'none'],
                        [pattern: '/**/css/**',      filters: 'none'],
                        [pattern: '/**/images/**',   filters: 'none'],
                        [pattern: '/**/fonts/**',    filters: 'none'],
                        [pattern: '/**/favicon.ico', filters: 'none'],

                        //Stateful chain
```

```
                        [pattern: '/**',        filters: 'JOINED_FILTERS,-restTokenValidationFilter,\
-restExceptionTranslationFilter']                ]
        }

        //Other Spring Security settings
        //...

        rest {
            token {
                validation {
                    enableAnonymousAccess = true
                }
            }
        }
    }
}
```

Now when we make a call to */api/login* with a valid user and password, we will get a response containing a valid JWT token. We can use this token to access resources under */api/v1.0/jobs* path just if we are in the *ROLE_CUSTOMER*. And */api/v1.0/featuredJobs* even if we don't have an access token.

REST Security on the client side

On the client's side, we have to introduce a new library named angular2-jwt[194] responsible for JWT token management. First, we have to install it with NPM:

```
cd ~/MVPS/mvp-application/app-web/ionic-client
npm install angular2-jwt
```

We have to change our *index page* page and added two forms, one for *login* and another for *signup*, each submitting the user credentials to the server *(ngSubmit)="login(loginCreds.value)"* and *(ngSubmit)="signup(signupCreds.value)"*. We show a *div* containing this two forms in case the user is not authenticated *!auth.authenticated()* and a *div* containing welcome info and a logout button in case of authenticated user *auth.authenticated()*:

```
nano ~/MVPS/mvp-application/app-web/ionic-client/src/pages/index/index.html
```

The new content:

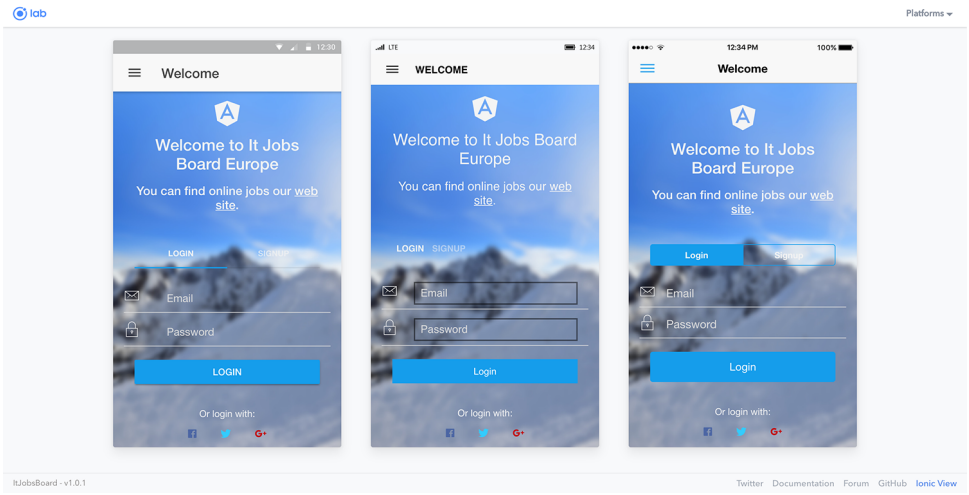
[194] https://github.com/auth0/angular2-jwt

```
<ion-header>
  <ion-navbar>
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>
    <ion-title>Welcome</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding class="login auth-page">

  <!-- Logo -->
  <div text-center padding>
    <h1 color="light" >
      <i class="ion-social-angular"></i>
      <ion-icon name="logo-angular"></ion-icon>
    </h1>
    <h2 color="light" >Welcome to It Jobs Board Europe</h2>
    <h5 color="light" >You can find online jobs our <a href="http://itjobsboard.eu">web\
 site</a>.</h5>
  </div>

  <div *ngIf="!auth.authenticated()">
    <div padding>
      <ion-segment [(ngModel)]="authType">
        <ion-segment-button value="login">
          <b color="light">Login</b>
        </ion-segment-button>
        <ion-segment-button value="signup">
          <b color="light">Signup</b>
        </ion-segment-button>
      </ion-segment>
    </div>

    <div [ngSwitch]="authType">

      <!-- Login form -->
      <form *ngSwitchCase="'login'" #loginCreds="ngForm" (ngSubmit)="login(loginCreds.v\
alue)">

        <ion-item>
          <ion-icon item-left name="ios-mail-outline"></ion-icon>
          <ion-input type="text" name="username" placeholder="Email" ngModel></ion-inpu\
t>
        </ion-item>

        <ion-item>
          <ion-icon item-left name="ios-lock-outline"></ion-icon>
          <ion-input type="password" name="password" placeholder="Password" ngModel></i\
```

```
50  on-input>
51        </ion-item>
52
53        <div padding>
54          <button block type="submit" ion-button secondary>Login</button>
55        </div>
56
57        <!-- Social login -->
58        <div padding text-center>
59          <span color="light" >Or login with:</span>
60          <p>
61            <a padding class="facebook-color">
62              <ion-icon name="logo-facebook"></ion-icon>
63            </a>
64            <a padding class="twitter-color">
65              <ion-icon name="logo-twitter"></ion-icon>
66            </a>
67            <a padding class="googleplus-color">
68              <ion-icon name="logo-googleplus"></ion-icon>
69            </a>
70          </p>
71        </div>
72      </form>
73
74
75      <form *ngSwitchCase="'signup'" #signupCreds="ngForm" (ngSubmit)="signup(signupCre\
76  ds.value)">
77        <ion-item>
78          <ion-icon item-left name="ios-mail-outline"></ion-icon>
79          <ion-input type="text" name="username" placeholder="Email" ngModel></ion-inpu\
80  t>
81        </ion-item>
82
83        <ion-item>
84          <ion-icon item-left name="ios-lock-outline"></ion-icon>
85          <ion-input type="password" name="password" placeholder="Password" ngModel></i\
86  on-input>
87        </ion-item>
88
89        <div padding>
90          <button block type="submit" ion-button secondary>Signup</button>
91        </div>
92      </form>
93    </div>
94
95    <div padding style="text-align: center">
96      <p *ngIf="error" class="error"><ion-badge color="danger" item-right>{{ error }}</\
97  ion-badge></p>
98    </div>
```

```
  </div>

  <div *ngIf="auth.authenticated()">
    <div padding>
      <h1>Hello {{ user }}</h1>
      <button ion-button secondary block (click)="logout()">Logout</button>
    </div>
  </div>
</ion-content>
```

Now the index page with some LAF improuvements will look like this:



Application login page

And in the controller part we will implement the calls for user login (based on the introduced angular2-jwt plugin):

```
nano ~/MVPS/mvp-application/app-web/ionic-client/src/pages/index/index.ts
```

New content:

```
import { Component } from '@angular/core';
import { NavController, NavParams } from 'ionic-angular';
import {AuthService} from "../../providers/auth";
import {Headers, Http} from "@angular/http";
import {JwtHelper} from "angular2-jwt";
import {GlobalVariable} from "../../app/global";

@Component({
  selector: 'page-index',
  templateUrl: 'index.html'
})
export class IndexPage {

```

```
14    LOGIN_URL: string = GlobalVariable.BASE_API_URL + "/api/login";
15    SIGNUP_URL: string = GlobalVariable.BASE_API_URL + "/api/signup";
16
17    auth: AuthService;
18    // When the page loads, we want the Login segment to be selected
19    authType: string = "login";
20    // We need to set the content type for the server
21    contentHeader: Headers = new Headers({"Content-Type": "application/json"});
22    error: string;
23    jwtHelper: JwtHelper = new JwtHelper();
24    user: string;
25    loginCreds: any = {}
26
27    constructor(public navCtrl: NavController, public navParams: NavParams, private http:\
28   Http) {
29      this.auth = AuthService;
30      let token = localStorage.getItem('id_token');
31      if(token) {
32        this.user = this.jwtHelper.decodeToken(token).sub;
33      }
34    }
35
36    login(credentials) {
37      // alert(JSON.stringify(credentials))
38      // credentials = {username:"admin", password: "Password123!"}
39      if(credentials.username == "" || credentials.password == "") {
40        this.error = "User name and password should be provided!"
41      } else {
42        this.http.post(this.LOGIN_URL, JSON.stringify(credentials), { headers: this.conte\
43  ntHeader })
44          .map(res => res.json())
45          .subscribe(
46            data => this.authSuccess(data.access_token),
47            err => {
48              console.log(err)
49              if(err.status == 401) {
50                this.error = "Invalid password and/or username.";
51              }
52              if(err.status == 400) {
53                this.error = "Invalid call to the server.";
54              }
55            }
56        );
57      }
58    }
59
60    signup(credentials) {
61      this.http.post(this.SIGNUP_URL, JSON.stringify(credentials), { headers: this.conten\
62  tHeader })
```

```
63      .map(res => res.json())
64      .subscribe(
65        data => this.authSuccess(data.access_token),
66        err => {
67          console.log(err)
68          if(err.status == 401) {
69            this.error = "Invalid password and/or username.";
70          }
71          if(err.status == 400) {
72            this.error = "Invalid call to the server.";
73          }
74        }
75      );
76    }
77
78    logout() {
79      localStorage.removeItem('id_token');
80      this.user = null;
81    }
82
83    authSuccess(token) {
84      this.error = null;
85      localStorage.setItem('id_token', token);
86      var tokenData = this.jwtHelper.decodeToken(token);
87      // alert(JSON.stringify(tokenData))
88      this.user = tokenData.sub
89    }
90
91    ionViewDidLoad() {
92      console.log('ionViewDidLoad IndexPage');
93    }
94  }
```

The *LOGIN_URL* is the endpoint provided by the *Grails Security plugin* for exchanging the user and password into a valid token. We define a method named *login* based on *http* service which is sending the user and password and parse the response with a *authSuccess* method to get the token and store it in the browser local storage[195]. In the case of an error, an error message is shown in view page via the *error* variable. A *logout* method can be used - it removes the token from local storage.

The class *JwtHelper* is used to parse the token and get the username. Here is the content of a JWT token:

[195] https://www.w3schools.com/html/html5_webstorage.asp

```
{
  "principal":"H4sIAAAAAAAAAJVSPO/bQBR/ThNRgVSgEkgdYAE25Eh0zMTfqpUbUNMsIIEu9sM9ON+ZuzM\
kS5WJDhlS0SJV7Vfgm7RLPwBqh67MXfvOEBxYUG+y3/38+/d8cQUVo+F5rBkXxk9FFnPpm1RzGRsMM81tx88M6g\
htjniRA5s0gevjlcALoMQjC0+DA3bMqoLJuLrZOsDQ1toalpSObxj3NUvwROlD/5Y7VBrvCBTU3tcSjGzDJAtDl\
UlbV3K9nXKN0TZMFLNAhYduNBXSDUrLmTDD0BGUrCUwCmCMZfadIlWOxsL4tdnMclFtoK0F8DhlxpC7e0ka1ll3\
986mpARH8B7K7dSjQ90tOKjvePxVJQSl5kqa+aZMVMT3uRMn/u7M2c/+t26zBECdLD78TTF/tgLd77t/Z/OivdD\
C9JD1AlZrp+RmsmB+q9EpX37Z+nR+9WHnESk7xMb/72N++aa5zqpKUqaZVUM7ItqTsnsm8pWHyQdb6PgNnqQC6Y\
+SFqNbiYKY4pa1EoO+LYy+2QzW95bXXr+su9cKixIuSfVJHtptyw8U7ar35+OP/twvYngFlWMmMqTOJwpQPUtaq\
E8vzmfGPv/u5QkGf/M/CjCs3REDAAA="",
  "sub":"customer",
  "roles":[
    "ROLE_CUSTOMER"
  ],
  "exp":1487790029,
  "iat":1487786429
}
```

After a successful login a response is received containing a valid token and also a refresh token that can be used for accessing pages secured with role *ROLE_ADMIN* (we shorted the tokens for space reasons):

```
{
  "username":"customer",
  "roles":[
    "ROLE_CUSTOMER"
  ],
  "token_type":"Bearer",
  "access_token":"...Wok2O_KnDmb5C6K3gzjM-0q2zgotwd26Hhbl3_-q-yw",
  "expires_in":3600,
  "refresh_token":"...GZi-6FOtgsZeXA5GPYs5BG_zkSzduTO8WbnmStQulLg"
}
```

Now we can use two methods in our *Jobs* page for retrieving the jobs that will be shown. First is *listJobs()* if the user is authenticated and the second is *listFeaturedJobs()* if the user is not authenticated - *AuthService.authenticated()* the call is used for checking this.

```
import { Component } from '@angular/core';
import { NavController, NavParams } from 'ionic-angular';
import { JobsData } from "../../providers/jobs-data";
import { GlobalVariable } from '../../app/global';

import { InAppBrowser } from 'ionic-native';
import {AuthService} from "../../providers/auth";

@Component({
  selector: 'page-jobs',
  templateUrl: 'jobs.html'
```

```
12  })
13  export class JobsPage {
14
15    selectedItem: any;
16    icons: string[];
17    items: any;
18    contextPath: string;
19
20    constructor(public navCtrl: NavController, public navParams: NavParams, public jobsDa\
21  ta: JobsData) {
22      // If we navigated to this page, we will have an item available as a nav param
23      this.selectedItem = navParams.get('item');
24      this.contextPath = GlobalVariable.BASE_API_URL
25
26      if(AuthService.authenticated()) {
27        this.listJobs()
28      } else {
29        this.listFeaturedJobs()
30      }
31    }
32
33    listJobs() {
34      this.jobsData.listJobs().subscribe(data => {
35        this.items = data;
36        console.log("Jobs: " + JSON.stringify(this.items, null, 2))
37      });
38    }
39
40    listFeaturedJobs() {
41      this.jobsData.listFeaturedJobs().subscribe(data => {
42          this.items = data;
43          console.log("Featured jobs: " + JSON.stringify(this.items, null, 2))
44        },
45        err => {
46          console.log(err)
47        });
48    }
49
50    itemTapped(event, item) {
51      // we use _self instead of _system
52      new InAppBrowser(item.jobUrl, '_self');
53    }
54  }
```

Finally, the *JobData* provider has two different methods. One named *listJobs* that is using the *authHttp* provided by *angular2-jwt* (see the imports) - this call is injecting automatically the header *Authorization:Bearer token...* for the call taking the token from local storage. The second method *listFeaturedJobs()* is using the usual *http* service from Angular.

```
import {Injectable} from '@angular/core';
import {Http} from '@angular/http';
import 'rxjs/add/operator/map';
import { GlobalVariable } from '../app/global';
import {AuthHttp} from 'angular2-jwt';

@Injectable()
export class JobsData {

  constructor(public http: Http, public authHttp: AuthHttp) {
    console.log('Enter JobsData Provider');
  }

  listJobs() {
    return this.authHttp.get(GlobalVariable.SECURED_API_URL + "/jobs").map((res) => res\
.json())
  }

  listFeaturedJobs() {
     return this.http.get(GlobalVariable.SECURED_API_URL + "/featuredJobs").map((res) \
=> res.json())
  }
}
```

We can test that the */avi/v1.0/jobs* is not accessible without a valid token:

```
open http://localhost:8080/api/v1.0/featuredJobs
```

We will get:

```
{"message":"401 UNAUTHORIZED. The request has not been applied because it lacks valid a\
uthentication credentials for the target resource.","error":401}
```

To the app store

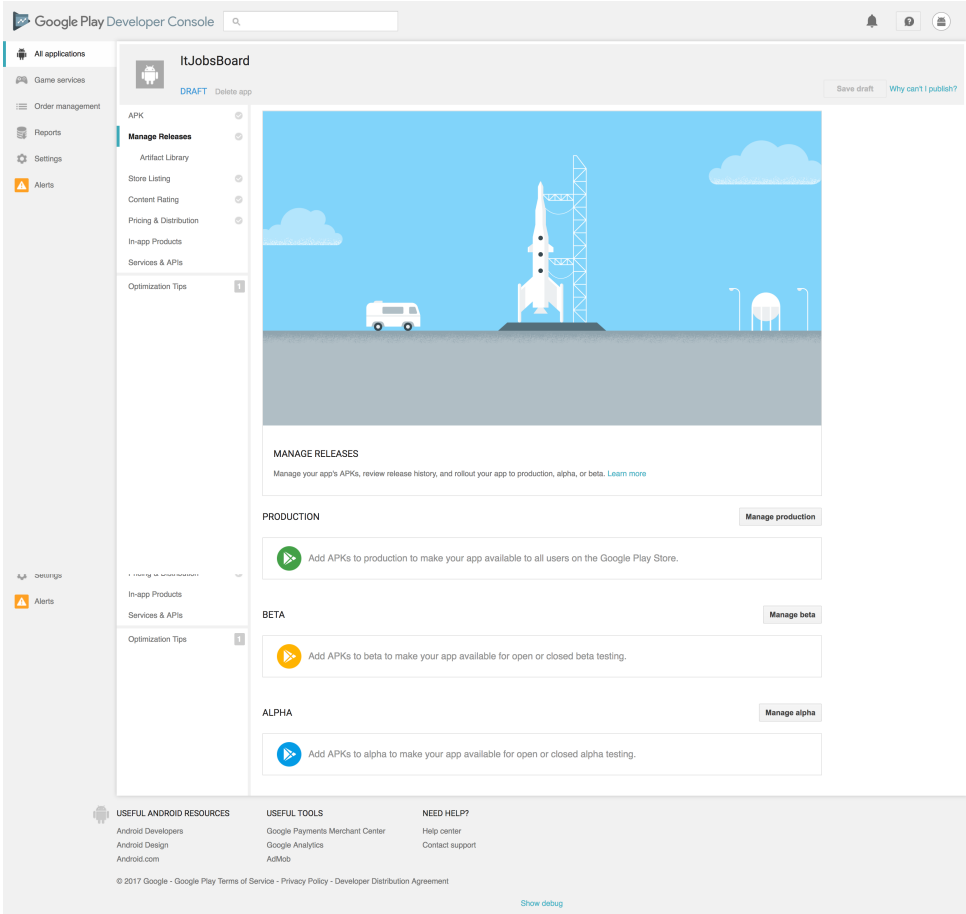
Publishing to the app store is simple in the case of Google Play store because there are no so many reviews. But will be mode difficult with the Apple AppStore and Windows Store. You have to improve you application usability to be accepted in the store. You can use a professional template from *Ionic Market*[196] an official Ionic market for starters, plugins, and themes. Now we will describe the steps that should de follow to publish the app in all three stores.

[196] https://market.ionic.io

Google Play Store

The Android store is Google Play[197] - your application is easily accepted (in hours) on this store. There are many free applications in this store, and their users are in general not paying for apps. Will cost you $25 per life for publishing in this store and Google takes 30% from revenue. The user receives 70% from revenue.

First, you have to create a Play Developer account[198], and there you have to create an application and edit your *Store Listing*. *Title* and *Full Description* are important for ASO[199] - be aware that around 60% of users are coming from store searches. In *Manage Releases* you will upload your *.apk* files.



Google Play Store

For creating an APK from your Ionic sources you have to use this script:

```
1 nano ~/MVPS/mvp-application/app-web/ionic-client/build.sh
```

The script:

[197] https://play.google.com/store
[198] https://developer.android.com/distribute/googleplay/developer-console.html
[199] https://blog.kissmetrics.com/app-store-optimization/

```
#!/bin/bash

echo "###################################"
project_path=~/MVPS/mvp-application/app-web/ionic-client
android_sdk_path=~/Library/Android/sdk/build-tools/22.0.1
android_adb_path=~/Library/Android/sdk/platform-tools
app_name=ItJobsBoard.apk

echo "###################################"
echo $project_path
echo "###################################"
echo $android_sdk_path

cd $project_path
cordova build --release android
cd $project_path

#apk
cp $project_path/platforms/android/build/outputs/apk/android-release-unsigned.apk .
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore jobsboard.keystore and\
roid-release-unsigned.apk jobsboard -storepass Password123!
echo $android_sdk_path/zipalign
$android_sdk_path/zipalign -vf 4 android-release-unsigned.apk android-$app_name

$android_adb_path/adb -d install -r $project_path/android-$app_name
```

As you can see the *apk* is encrypted. You have to create a *keystore* using this command (*keytool* is in *bin* folder of your JDK installation):

```
keytool -genkey -v -keystore jobsboard.keystore -keyalg RSA -keysize 2048 -validity 100\
00 -alias jobsboard
```

You have to execute it:

```
cd ~/MVPS/mvp-application/app-web/ionic-client/
./build.sh
```

You will find the APK file in your *Ionic* app folder: *ionic-client/android-ItJobsBoard.apk*. You have to upload it and publish the app to the store.

Apple App Store

The store is Apple Store[200] - your application is verified in detail and takes around one week to be published. You can make more money from this store, and in general, there are more payable applications than free. You pay $99 per year for publishing[201] on this store.

Build the *iOs* project:

[200] http://www.apple.com/itunes/charts/free-apps

[201] https://developer.apple.com/

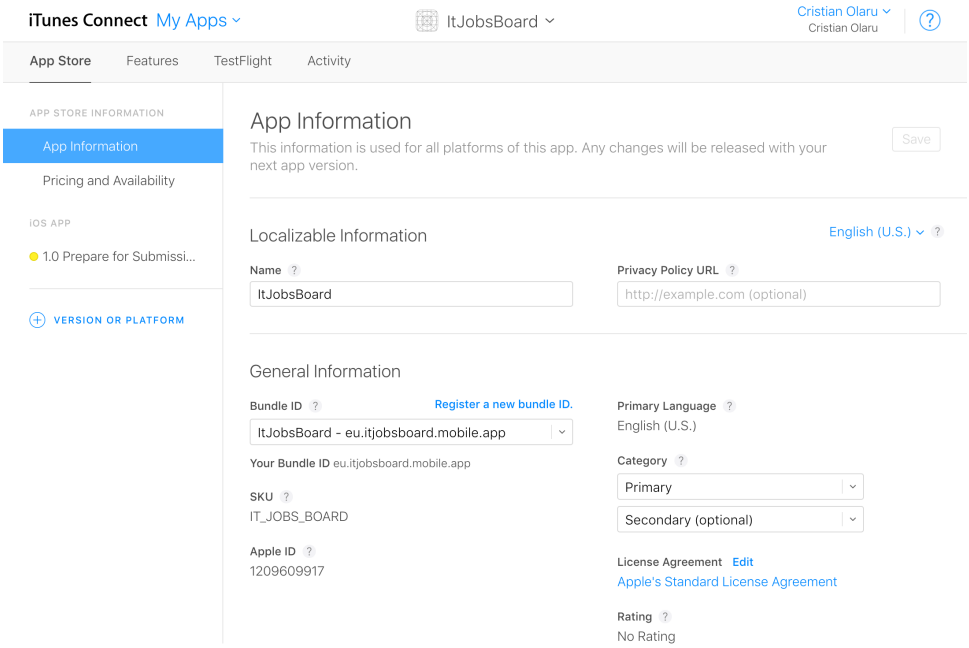
```
ionic build ios
```

First open the project in XCode pointing to the generated *.xcodeproj* file (from */plathorms/ios* folder):

```
~/MVPS/mvp-application/app-web/ionic-client/platforms/ios/ItJobsBoard.xcodeproj
```

You can first test the application by running the application in the XCode provided emulator using *Product/Run*.

For publishing, you need to have an Apple Id[202]. Then you have to go to your Apple developer account in the area named Certificates, Identifiers & Profiles[203]. There you have to create an application id - in our case *eu.itjobsboard.mobile.app* associated with an application named *ItJobsBoard*. Then, you have to go to your iTunes Connect account[204] where you have to request a new application using *New App* using the application id created in the previews step. There you have to complete *App Information* and other application data before submitting it for approval.



Apple AppStore

Also, you have to upload the binary file, and this can be done in two ways. The first option is to use *Product/Archive* option from *XCode*. Second is using a distinct application named *Application Loader*.

Windows App Store

The store for Windows applications is Windows Store[205] and is not so rich in applications like the previous stores. You have to pay $40 for starting publishing[206] in this store. The advantage of

[202] https://appleid.apple.com
[203] https://developer.apple.com/account/ios/certificate
[204] https://itunesconnect.apple.com
[205] https://www.microsoft.com/en-US/store/apps/windows
[206] https://developer.microsoft.com/en-us/windows

this store is that it is a unified store for all Windows 10 applications and you can publish your application to all Windows 10 devices.

For building your application you need a Windows 10 machine with Visual Studio 2015 or 2017 installed (you can get one for free usage for 1 month or you can buy one forever Windows 10 preinstalled with Visual Studio Community Edition[207]). Then you have to get the project from GitHub on your Windows 10 machine (lets say you are doing this in a *C:MVPS* folder):

```
cd C:\MVPS
git clone https://github.com/colaru/mvp-application.git
```

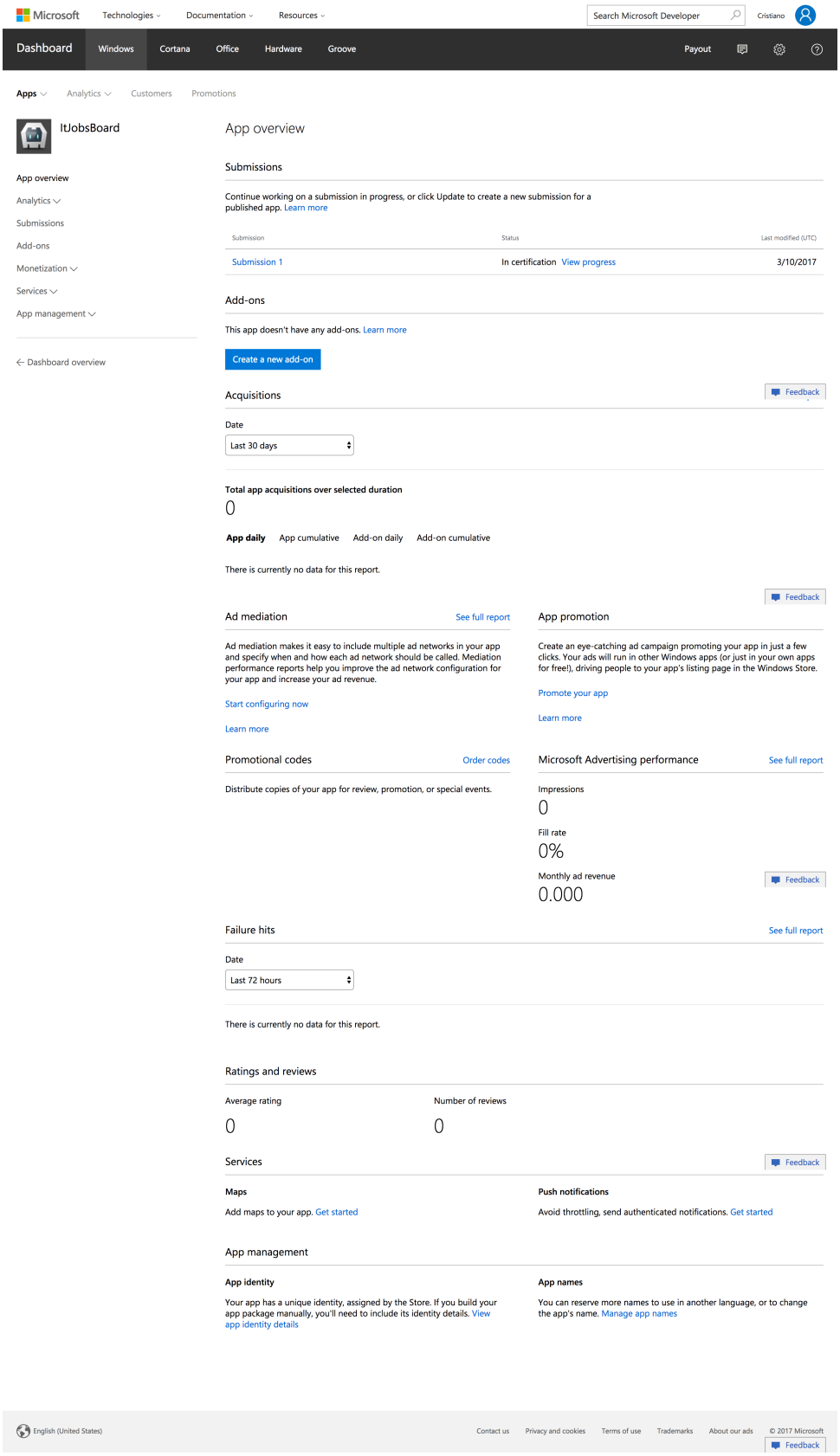
Now you will be able to build our project:

```
cd C:\MVPS\mvp-application\app-web\ionic-client
ionic platform add windows
ionic build windows
```

Now you can open in Visual Studio the Windows 10 project located in *C:MVPS\mvp-application\app-web\ionic-client\platforms\windows* and you can start publishing it to the Windows Store. You can create the deployment artifacts in *Project/Store/Create App Packages* and there you can choose to link to an existing application definition from your Windows Developer Center account[208] or to create a new one. The package will be signed automatically and you have to upload it manually into the store using *Project/Store/Upload App Packages*.

You don't have support in Ionic 2 for generating icons and splash screen resources like in iOs and Android using *ionic resources* command but you can generate all the icons and splash screens using *Properties/Visual Assets* option from Visual Studio. Also Windows 10 resources can be generated using this online tool[209].

[207] https://developer.microsoft.com/en-us/windows/downloads/virtual-machines
[208] https://developer.microsoft.com/en-us/windows
[209] http://wat-docs.azurewebsites.net/Tools



Windows Store

To do list

There are a set of things we didn't manage to implement in the sample application. We will try to implement these features in the near future, but can be a good exercise for the reader to implement them:

- Implement refresh token using *angular2-jwt* following this Ionic 2 sample[210]
- Implement users signup on client and server side
- Implement a scenario where the customer can save jobs posts after login and the saves are kept on the server side after user logoff

[210] https://auth0.com/docs/quickstart/native/ionic2