المعهد الوطني للعلوم التطبيقية و التكنولوجيا

Insat

Academic Year : 2017 - 2018

Software Architectures Class

JHipster Project

Workshop Report

Prepared by :

Ichrak MARS – Karim SAYEH

Introduction to Jhipster:

JHipster is one of those open-source projects you stumble upon and immediately think, "Of course!" It combines three very successful frameworks in web development: Bootstrap, Angular, and Spring Boot. Bootstrap was one of the first dominant web-component frameworks. Its largest appeal was that it only required a bit of HTML and it worked! Bootstrap showed many in the Java community how to develop components for the web. It leveled the playing field in HTML/CSS development, much like Apple's Human Interface Guidelines did for iOS apps.
At its core, JHipster is a *Yeoman* generator. Yeoman is a code generator that you run with a '*yo*' command to generate complete applications or useful pieces of an application. Yeoman generators promote what the Yeoman team calls the "*Yeoman workflow*". This is an opinionated client-side stack of tools that can help developers quickly build beautiful web applications. It takes care of providing everything needed to get working without the normal pains associated with a manual setup.

In this tutorial, we will be illustrating two demos:

1. Creating a Monolithic Application
2. Creating a Microservice Application

First Demo:

1. Goal

In this tutorial, we will create a simple application using the JHipster platform.

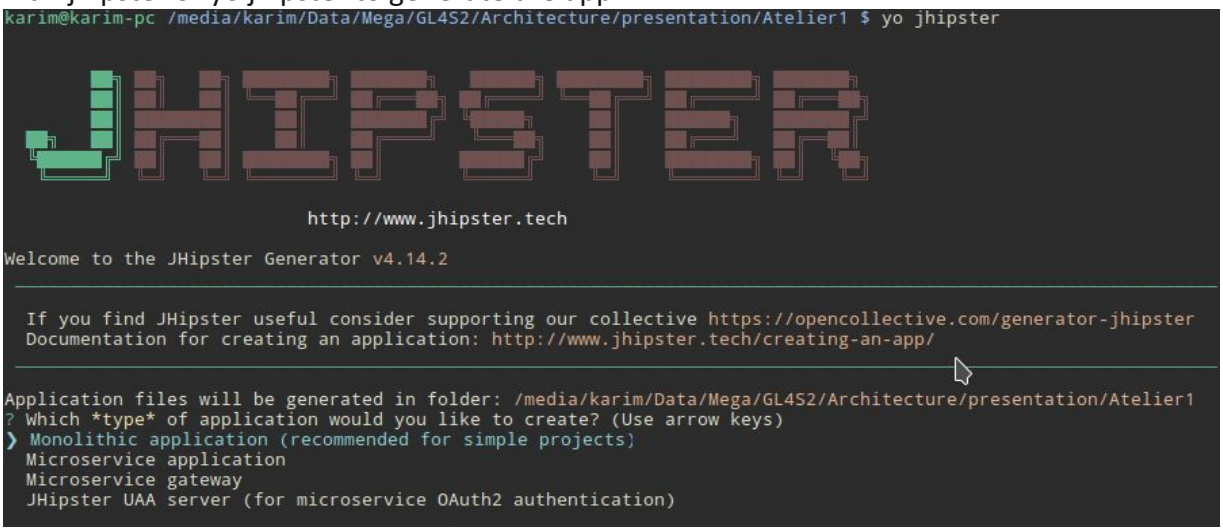
2. Requirements

Before we start, you need to setup the environment so make sure to install:

- Java
- Node.js
- Git
- Yarn: **npm install -g yarn**
- Yeoman: **npm install -g yo**
- Jhipster generator: **npm install -g generator-jhipster**

3. Creating an application

- First of all, we make an empty directory,
- Go to the directory,
- Run jhipster or yo jhipster to generate the app.



The type of the application that we will work with, is a Monolithic application

```
? Which *type* of application would you like to create? Monolithic application (recommended for simple projects)
? What is the base name of your application? (Atelier1)
```

By default, the base name of the application is the same as the name of the current folder, so we will just hit enter.

- choose the default java package name

```
? What is the base name of your application? Atelier1
? What is your default Java package name? tn.insat.jhipster
```

We will use the JHipster Registry to monitor the application

```
? Do you want to use the JHipster Registry to configure, monitor and scale your application
  No
❯ Yes
```

JHipster will not ask us which type of authentication we would like to use, because when selecting the JHipster Registry above, we can only use JWT authentication.

In this tutorial we will be working with Relational databases so go ahead and choose >SQL

```
? Which *type* of database would you like to use? (Use arrow keys)
❯ SQL (H2, MySQL, MariaDB, PostgreSQL, Oracle, MSSQL)
  MongoDB
  Cassandra
  [BETA] Couchbase
```

- choose your production database, in our case it's MySQL

```
? Which *type* of database would you like to use? SQL (H2, MySQL, MariaDB, PostgreSQL, Oracle, MSSQL)
? Which *production* database would you like to use? (Use arrow keys)
❯ MySQL
  MariaDB
  PostgreSQL
  Oracle (Please follow our documentation to use the Oracle proprietary driver)
  Microsoft SQL Server
```

- choose your development database, once again we will use MySQL

```
? Which *development* database would you like to use?
  H2 with disk-based persistence
  H2 with in-memory persistence
❯ MySQL
```

We will work with the Spring cache abstraction and the Ehcache implementation

```
? Do you want to use the Spring cache abstraction? (Use arrow keys)
❯ Yes, with the Ehcache implementation (local cache, for a single node)
  Yes, with the Hazelcast implementation (distributed cache, for multiple nodes)
  [BETA] Yes, with the Infinispan (hybrid cache, for multiple nodes)
  No (when using an SQL database, this will also disable the Hibernate L2 cache)
```

- type "y" to use Hibernate 2nd level cache

```
? Do you want to use Hibernate 2nd level cache? (Y/n) y
```

- choose maven as the build tool

```
? Would you like to use Maven or Gradle for building the backend? (Use arrow keys)
❯ Maven
  Gradle
```

For this tutorial will use social login & Elasticsearch technologies, hit "space" when selecting the technologies and "enter" to confirm

```
? Which other technologies would you like to use?
 ◉ Social login (Google, Facebook, Twitter)
❯◉ Search engine using Elasticsearch
 ◯ WebSockets using Spring Websocket
 ◯ API first development using swagger-codegen
 ◯ Asynchronous messages using Apache Kafka
```

- choose Angular 5 for the client side

```
? Which *Framework* would you like to use for the client? (Use arrow keys)
❯ Angular 5
  AngularJS 1.x
```

- enable SASS

```
? Would you like to enable *SASS* support using the LibSass stylesheet preprocessor? (y/N) y
```

- enable internationalization support

```
? Would you like to enable internationalization support? (Y/n) y
```

- choose the native language of your application

```
? Please choose the native language of the application (Use arrow keys)
❯ English
  Estonian
  Farsi
  French
  Galician
  German
  Greek
(Move up and down to reveal more choices)
```

- choose additional languages, we chose arabic & french

```
? Please choose additional languages to install
 ◯ Tamil
 ◯ Thai
 ◯ Ukrainian
❯◯ Vietnamese
 ◉ Arabic (Libya)
 ◯ Armenian
 ◯ Catalan
(Move up and down to reveal more choices)
```

- we will try out all of these testing frameworks

```
? Besides JUnit and Karma, which testing frameworks would you like to use?
❯◉ Gatling
 ◉ Cucumber
 ◉ Protractor
```

- we will install other generators from the JHipster marketplace so go ahead and type “y”

```
? Would you like to install other generators from the JHipster Marketplace? (y/N) y
? Which other modules would you like to use?
 ◯ (generator-jhipster-primeng-2.0.42) Generate PrimeNG Components
 ◉ (generator-jhipster-ionic-3.1.2) A JHipster Module that generates an Ionic Client
 ◯ (generator-jhipster-db-helper-1.0.0) A JHipster module for already existing databases
❯◯ (generator-jhipster-docker-2.4.0) Additional Docker support: Docker Hub, Local SMTP Server, NGinx
 ◯ (generator-jhipster-pages-2.0.4) create pages : client only (static), or client and server side.
oad data from server, Save data to server, form, table...
 ◉ (generator-jhipster-elasticsearch-reindexer-1.2.1) Generates a service that reindexes all database
 rows for each of your entities
 ◉ (generator-jhipster-ci-1.0.0) JHipster module, Continuous Integration support in your JHipster app
lication
(Move up and down to reveal more choices)
```

The creation of the application will take a while so in the meantime you can:

- create a database for the application, by default the database name should be equal to the application name, in our case “Atelier1”, but you can change the datasource url in the application-dev.yml config file if you want to use another database.
- verify your database credentials in src/main/resources/config/application-dev.yml.

```yaml
application-dev.yml
spring:
    profiles:
        active: dev
        include: swagger
    devtools:
        restart:
            enabled: true
        livereload:
            enabled: false # we use gulp + BrowserSync for livereload
    jackson:
        serialization.indent_output: true
    datasource:
        type: com.zaxxer.hikari.HikariDataSource
        url: jdbc:mysql://localhost:3306/Atelier1?useUnicode=true&characterEncoding=utf8&useSSL=false
        username: root
        password:
        hikari:
            data-source-properties:
                cachePrepStmts: true
                prepStmtCacheSize: 250
                prepStmtCacheSqlLimit: 2048
                useServerPrepStmts: true
    jpa:
        database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
        database: MYSQL
        show-sql: true
        properties:
            hibernate.id.new_generator_mappings: true
            hibernate.cache.use_second_level_cache: true
```

In this same file you can also configure the mail of the application, the server port, enable ssl, etc … but keep in mind that when you run the application in the production mode you have to make the changes in application-prod.yml

After a while, these success messages should be displayed, and now we can run our app

```
Done in 62.83s.
If you find JHipster useful consider supporting our collective https://opencolle

Server application generated successfully.

Run your Spring Boot application:
 ./mvnw

Client application generated successfully.

Start your Webpack development server with:
 yarn start

Application successfully committed to Git.
```

4. Running the application

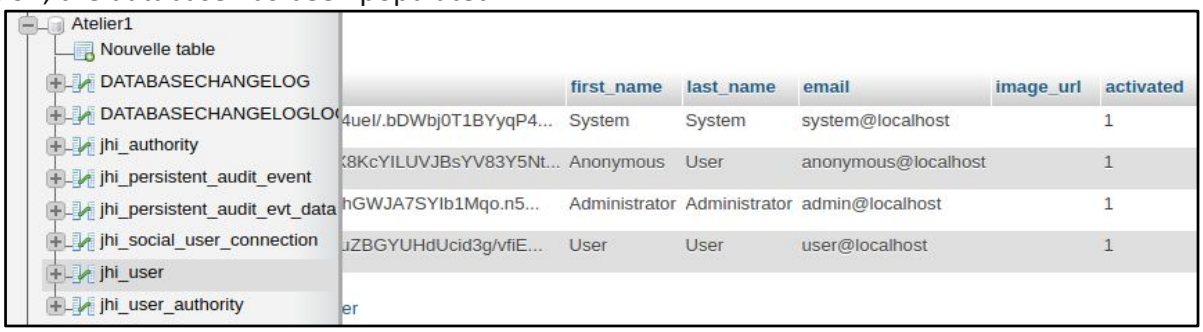
Use. /mvnw to run the server that includes a compiled version of the angular application. However, if you want to run a webpack development server for the angular application you need to run: npm start.

```
21:56:44.332 [main] DEBUG org.springframework.boot.devtools.restart.ChangeableUrls - Matching URLs
ntation/Atelier1/target/classes/]

:: JHipster ⓗ  :: Running Spring Boot 1.5.11.RELEASE ::
:: http://www.jhipster.tech ::

2018-04-06 21:56:51.449  WARN 26167 --- [  restartedMain] c.c.c.ConfigServicePropertySourceLocator
://localhost:8761/config/Atelier1/dev/master": Connexion refusée (Connection refused); nested exce
sed)
2018-04-06 21:56:51.496  INFO 26167 --- [  restartedMain] tn.insat.jhipster.Atelier1App
2018-04-06 21:56:57.878 DEBUG 26167 --- [  restartedMain] t.i.jhipster.config.AsyncConfiguration
2018-04-06 21:57:00.076 DEBUG 26167 --- [  restartedMain] c.ehcache.core.Ehcache-usersByLogin
2018-04-06 21:57:00.126 DEBUG 26167 --- [  restartedMain] c.ehcache.core.Ehcache-usersByEmail
```

Our first JHipster app is up and running

```
2018-04-06 21:58:56.991 DEBUG 26167 --- [  restartedMain] i.g.j.c.apidoc.SwaggerConfigurat
2018-04-06 21:58:57.018 DEBUG 26167 --- [  restartedMain] i.g.j.c.apidoc.SwaggerConfigurat
2018-04-06 21:59:06.333  INFO 26167 --- [  restartedMain] tn.insat.jhipster.Atelier1App
2018-04-06 21:59:06.334  INFO 26167 --- [  restartedMain] tn.insat.jhipster.Atelier1App
----------------------------------------------------------
        Application 'Atelier1' is running! Access URLs:
        Local:          http://localhost:8081
        External:       http://127.0.1.1:8081
        Profile(s):     [swagger, dev]
----------------------------------------------------------
```

In addition, the database has been populated :

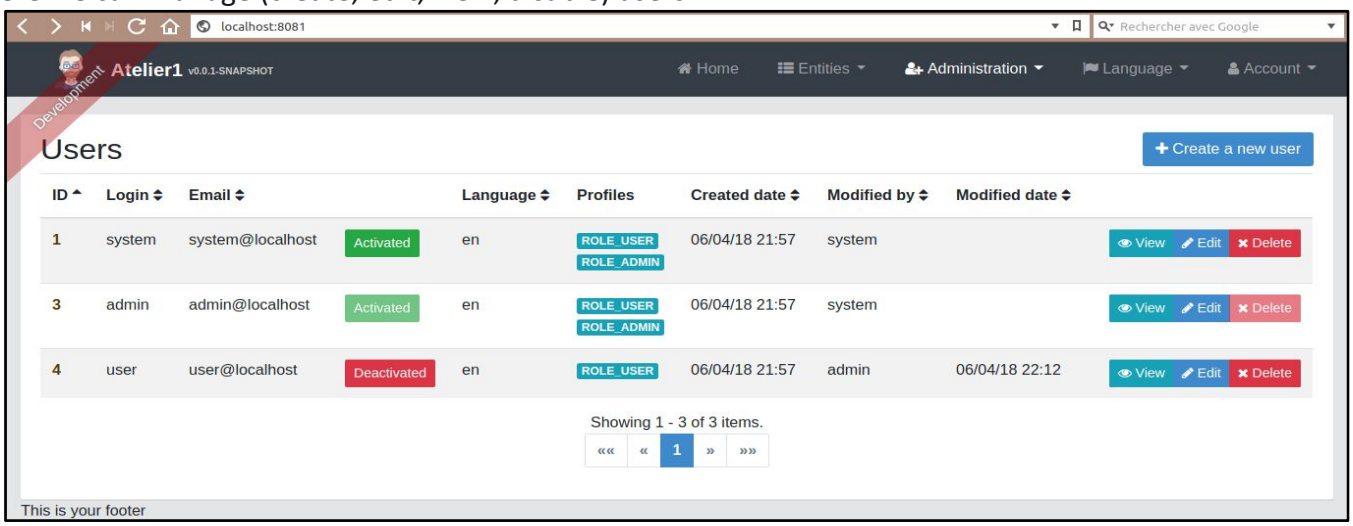


let’s see how it looks :

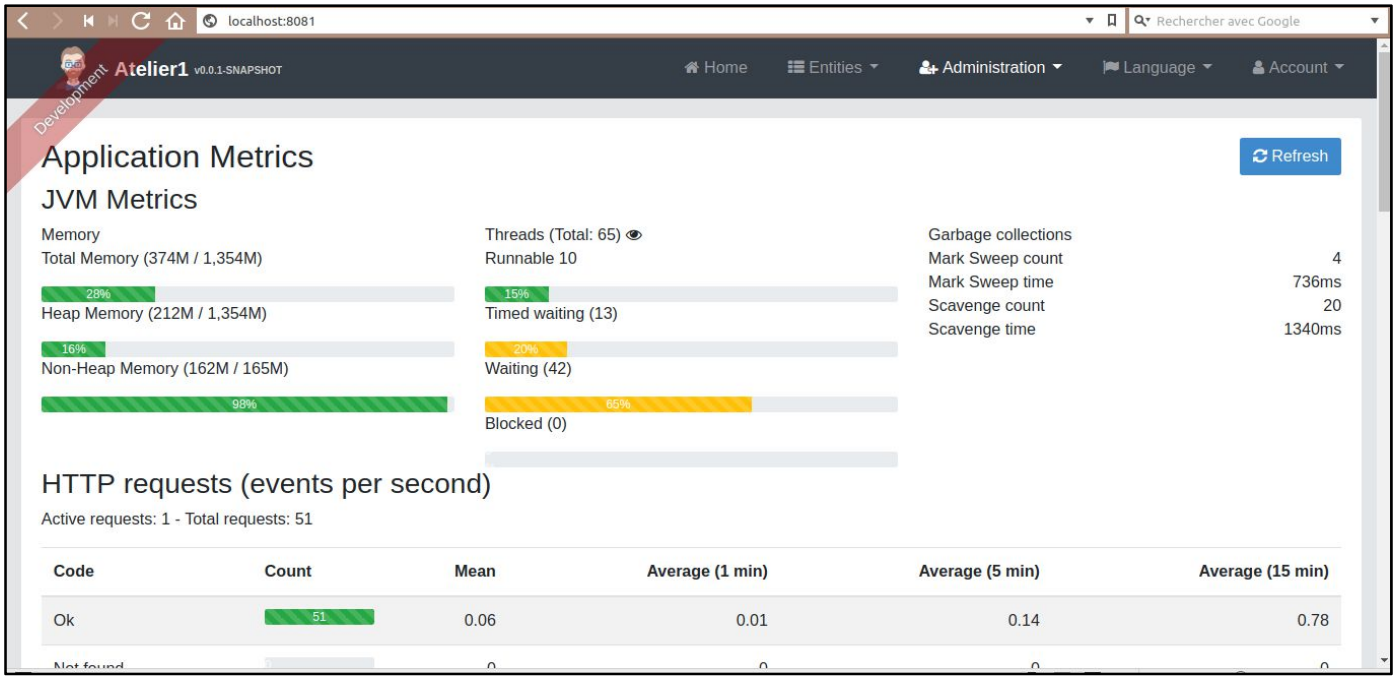
And let’s login as ‘*admin:admin*’

And we're successfully logged in as the administrator

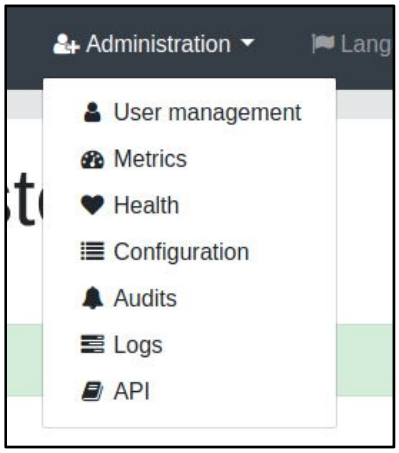
Here we can manage (create, edit, view, disable) users



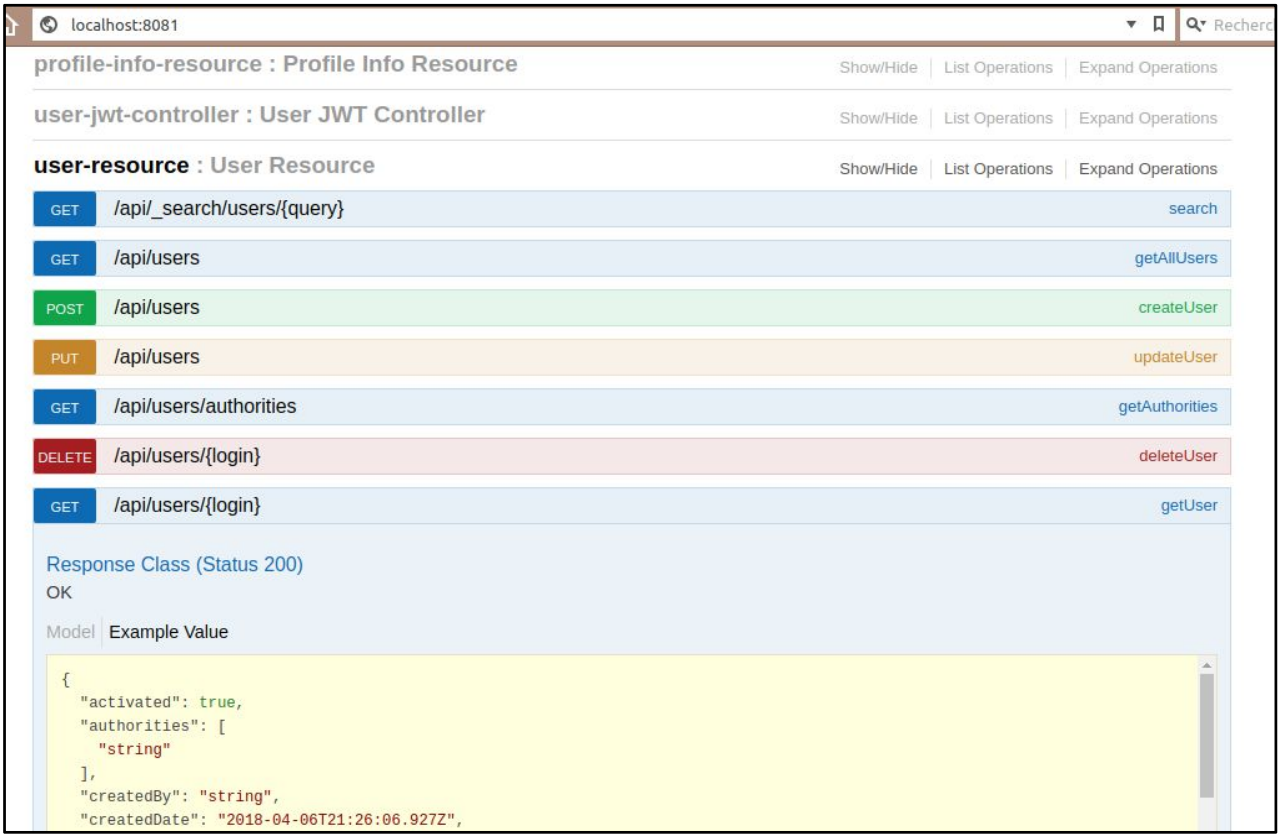
view application metrics



And much more

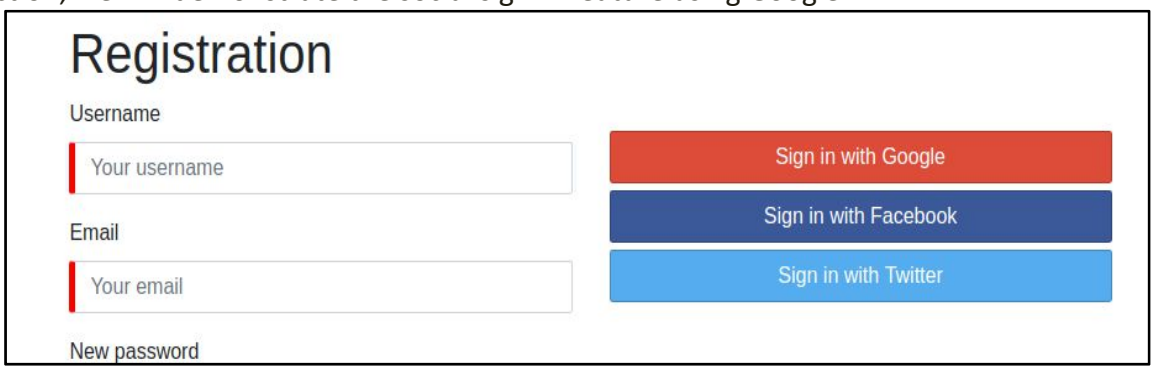


The API section is made with “swagger” that provides a complete documentation of the API and the possibility to test requests:

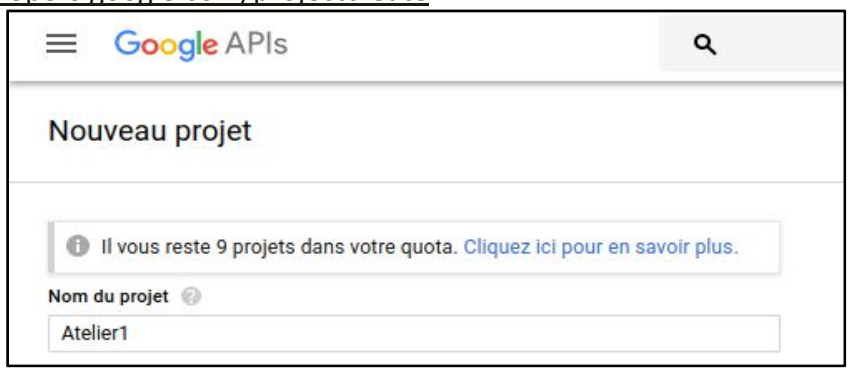


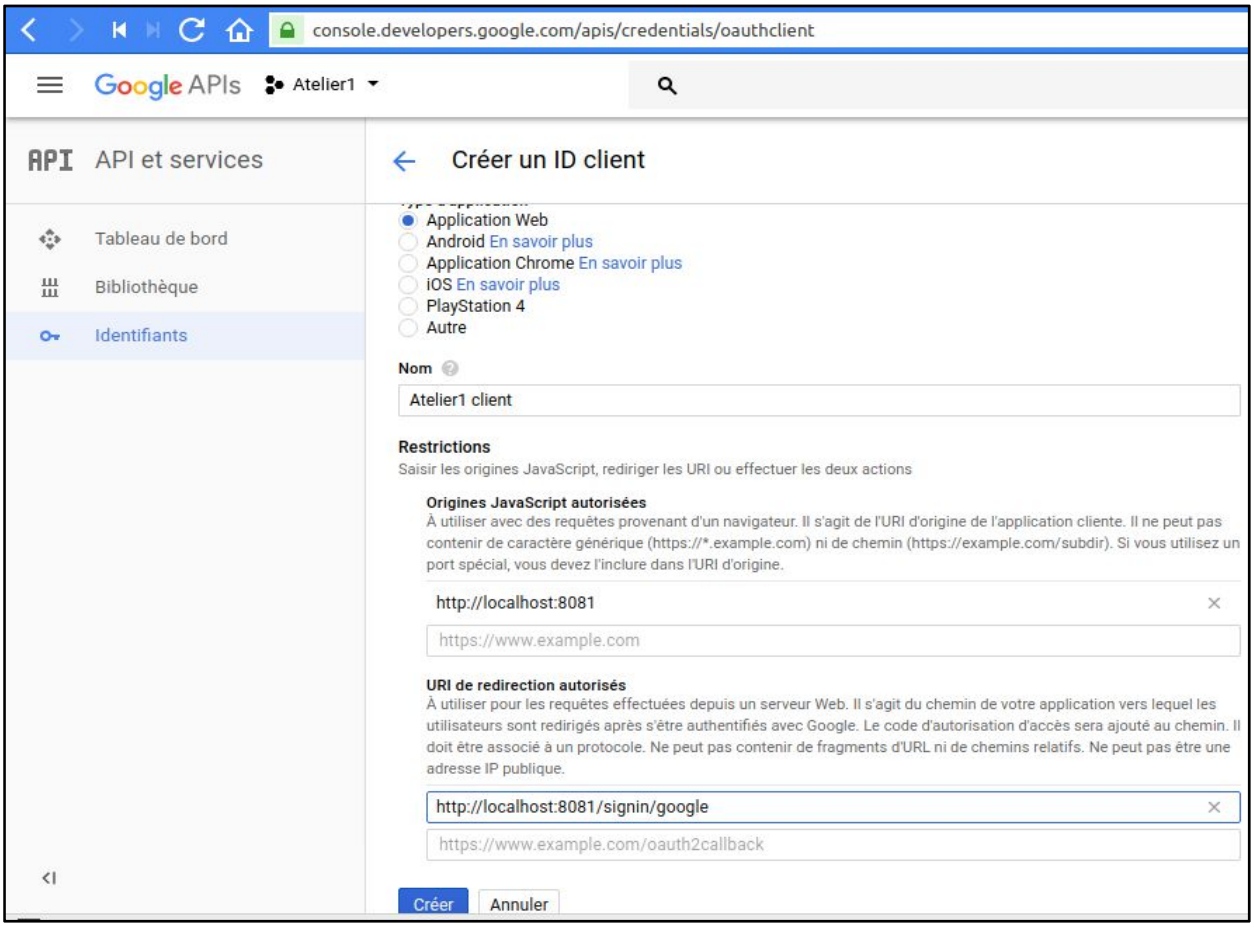
5. Sign in

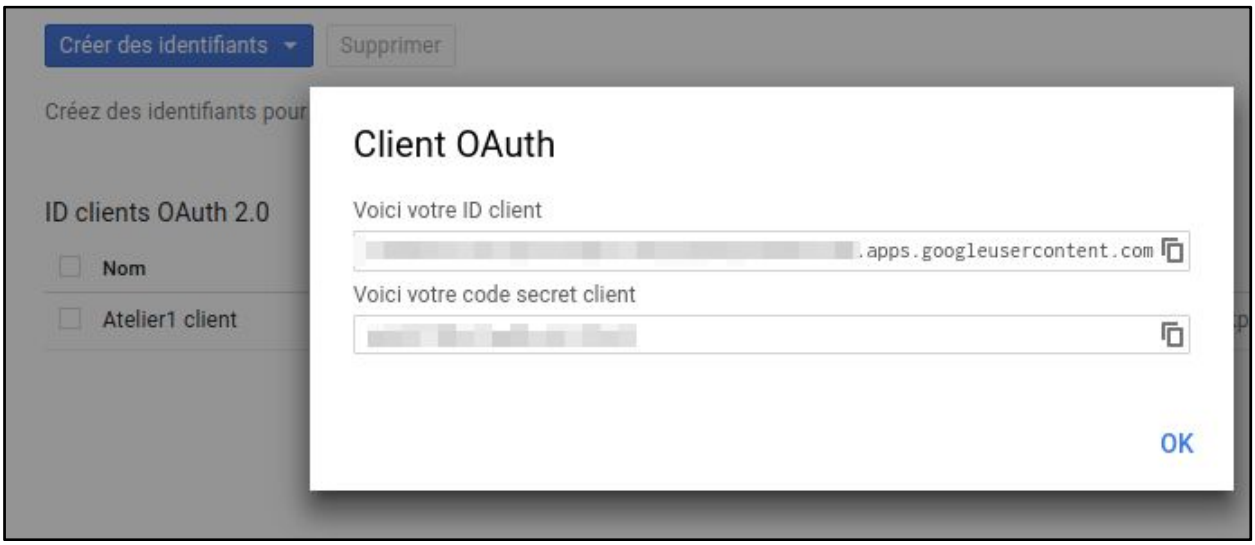
In this section, we will demonstrate the social sign in feature using Google



First, you need to create a new project in the google console
https://console.developers.google.com/projectcreate



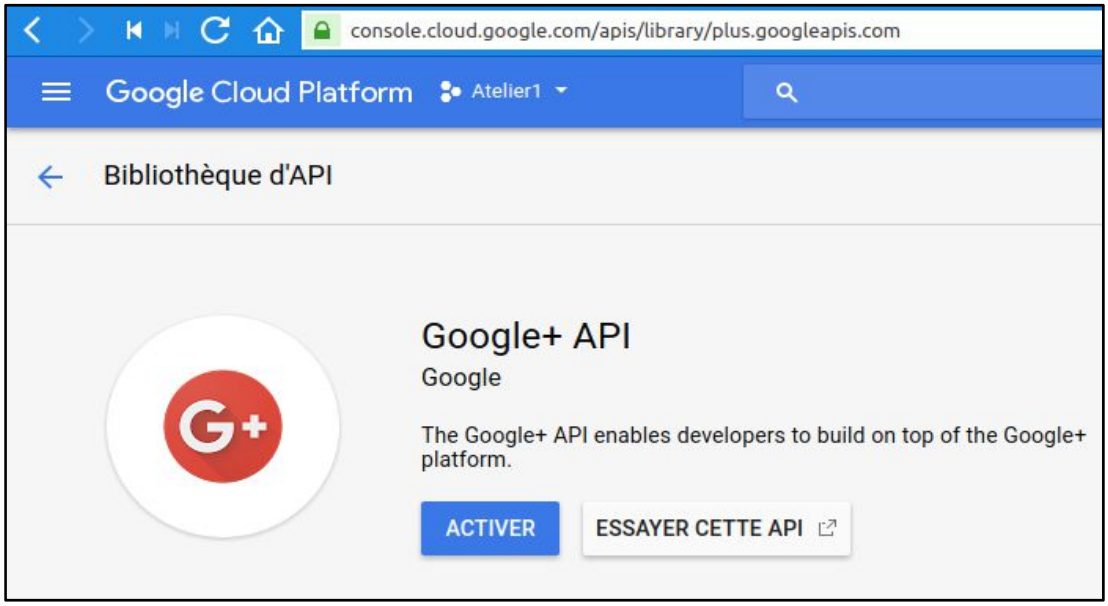




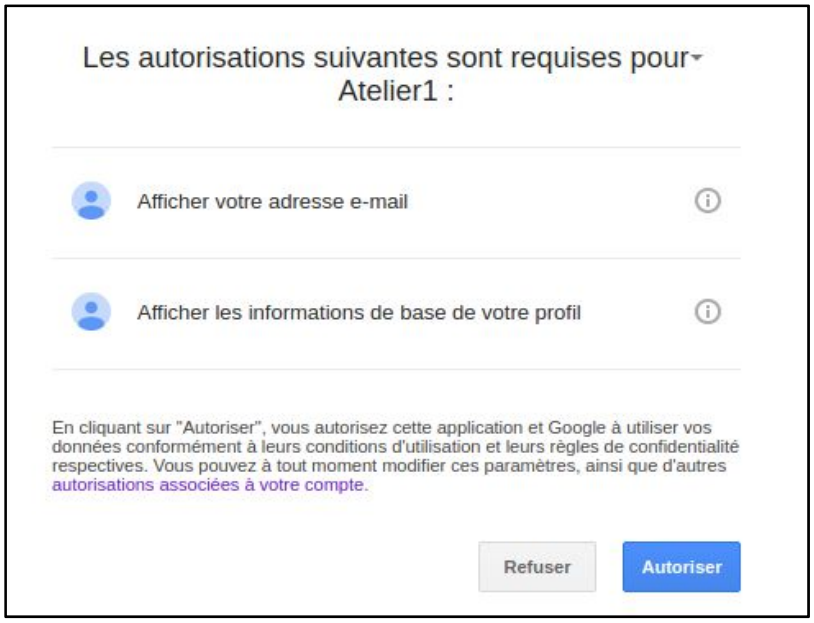
Now you need to copy the client id and the secret in the src/main/resources/config/application.yml file

```
application.yml
social:
    # see https://developers.google.com/+/web/signin/server-side-flow#step_1_create_a_clie
    google:
        client-id: [redacted].apps.googleusercontent.com
        client-secret: [redacted]

    # see https://developers.facebook.com/docs/facebook-login/v2.2
    facebook:
        client-id: xxx
        client-secret: xxx

    # see https://apps.twitter.com/app/
    twitter:
```

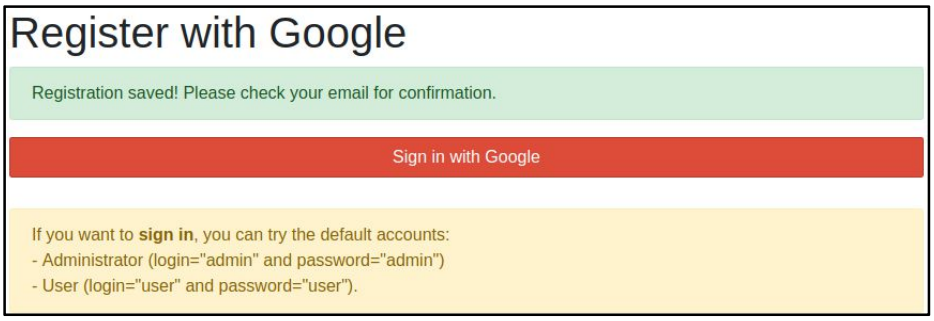
And don't forget to enable google+ API



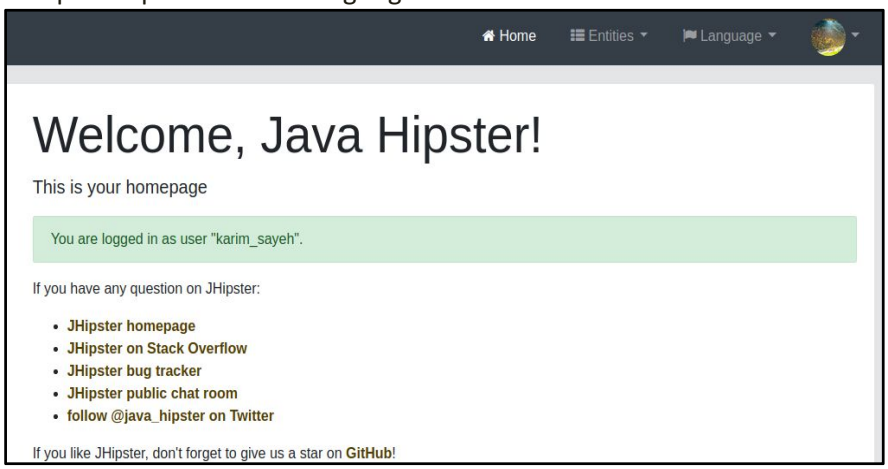
Then, restart the application
When clicking on "sign in with google", we are redirected to the google authorization page:



The registration was successful :



When we click on “sign in with google”, we can access the app, and as we can see, the application brought the first name, last name, and the profile picture from the google+ account

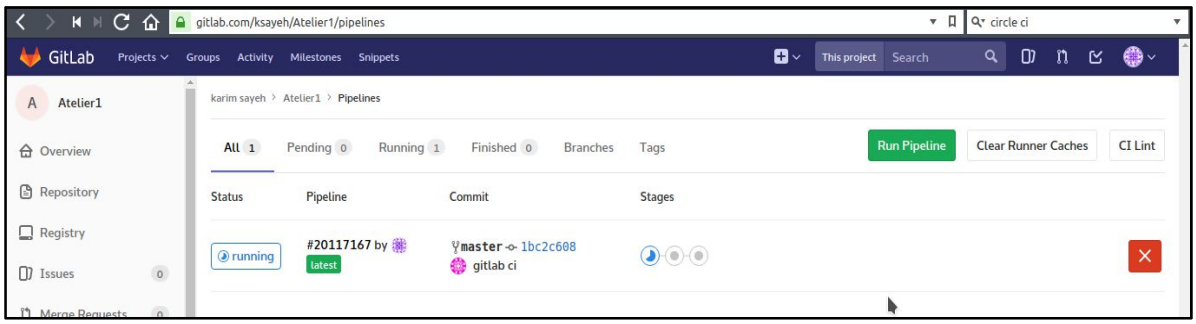


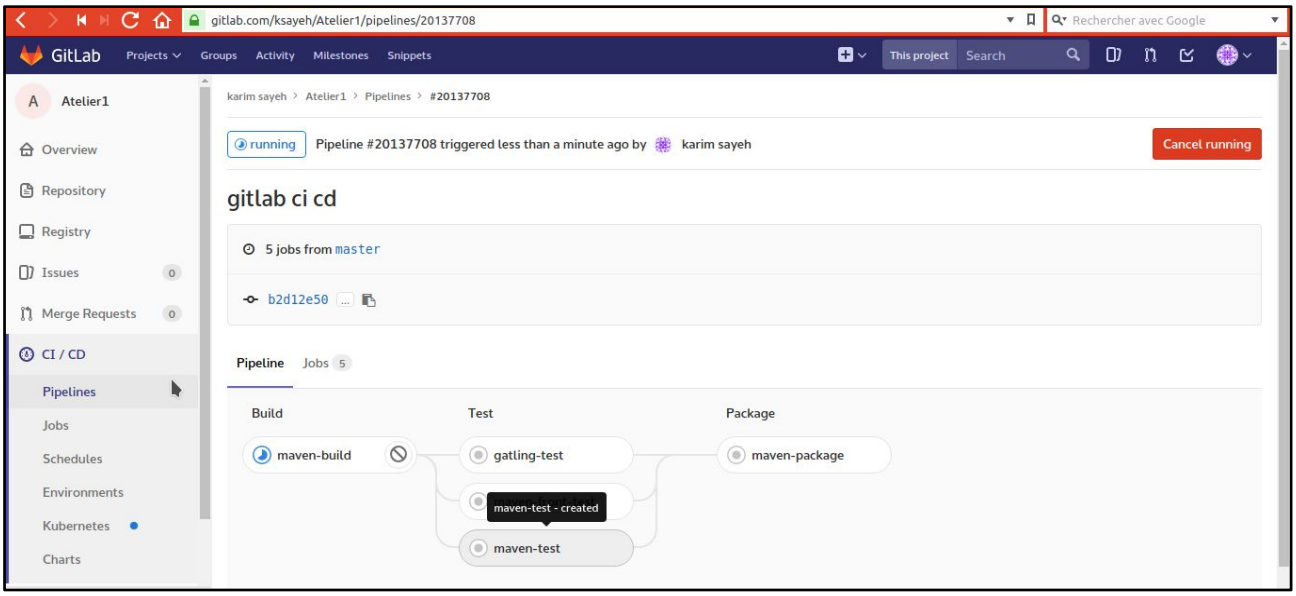
6. Continuous Integration

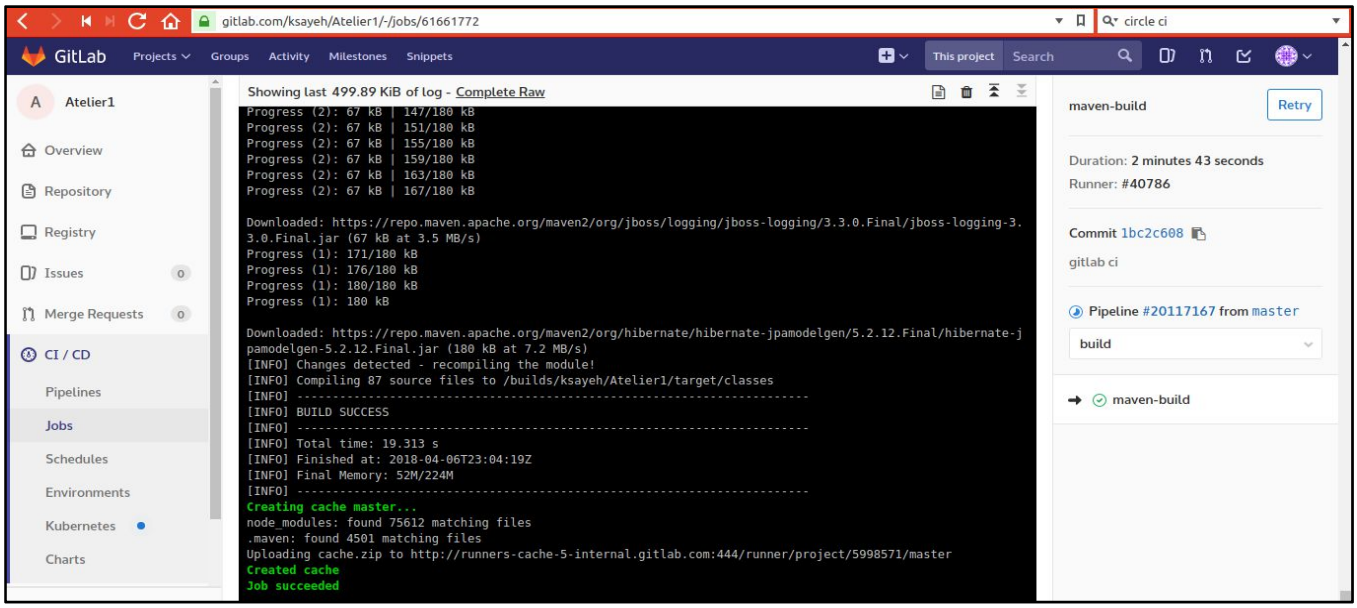
We will use GitLab CI/CD system to set up continuous integration for our project: jhipster ci-cd

```
karim@karim-pc /media/karim/Data/Mega/GL4S2/Architecture/presentation/Atelier1 $ jhipster ci-cd
Using JHipster version installed locally in current project's node_modules
Executing jhipster:ci-cd
Options:
Welcome to the JHipster CI/CD Sub-Generator
? What CI/CD pipeline do you want to generate?
 ◯ Jenkins pipeline
 ◯ Travis CI
❯◉ GitLab CI
 ◯ CircleCI
karim@karim-pc /media/karim/Data/Mega/GL4S2/Architecture/presentation/Atelier1 $ jhipster ci-cd
Using JHipster version installed locally in current project's node_modules
Executing jhipster:ci-cd
Options:
Welcome to the JHipster CI/CD Sub-Generator
? What CI/CD pipeline do you want to generate? GitLab CI
? In GitLab CI, perform the build in a docker container (hint: GitLab.com uses Docker container)? Yes
? Deploy to heroku?
   create .gitlab-ci.yml
Congratulations, JHipster execution is complete!
```

And don’t forget to create a Gitlab Repository and add it as a remote to the project: git remote add origin https://gitlab.com/ksayeh/Atelier1.git. Each commit pushed to GitLab will trigger the defined pipeline in .gitlab-ci.yml





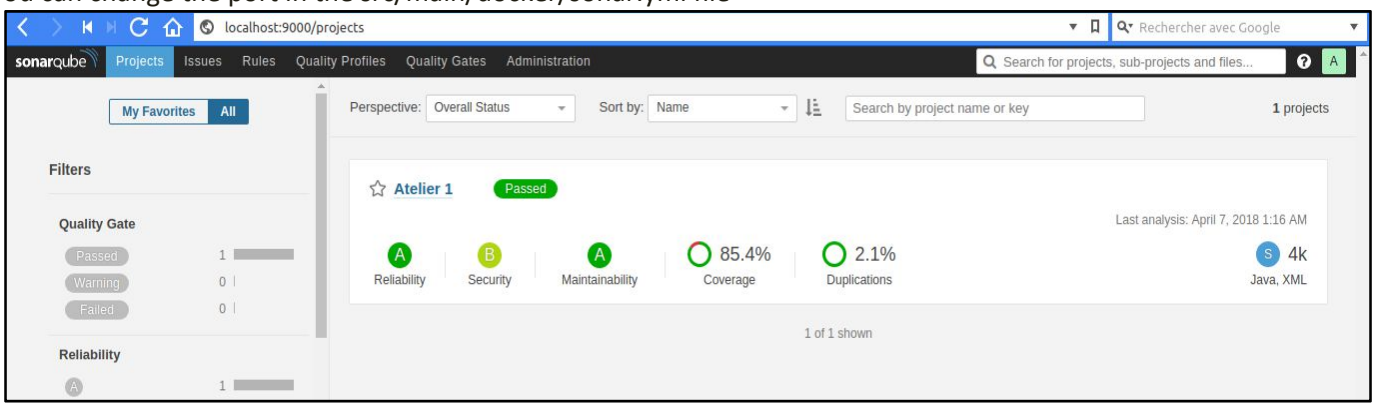


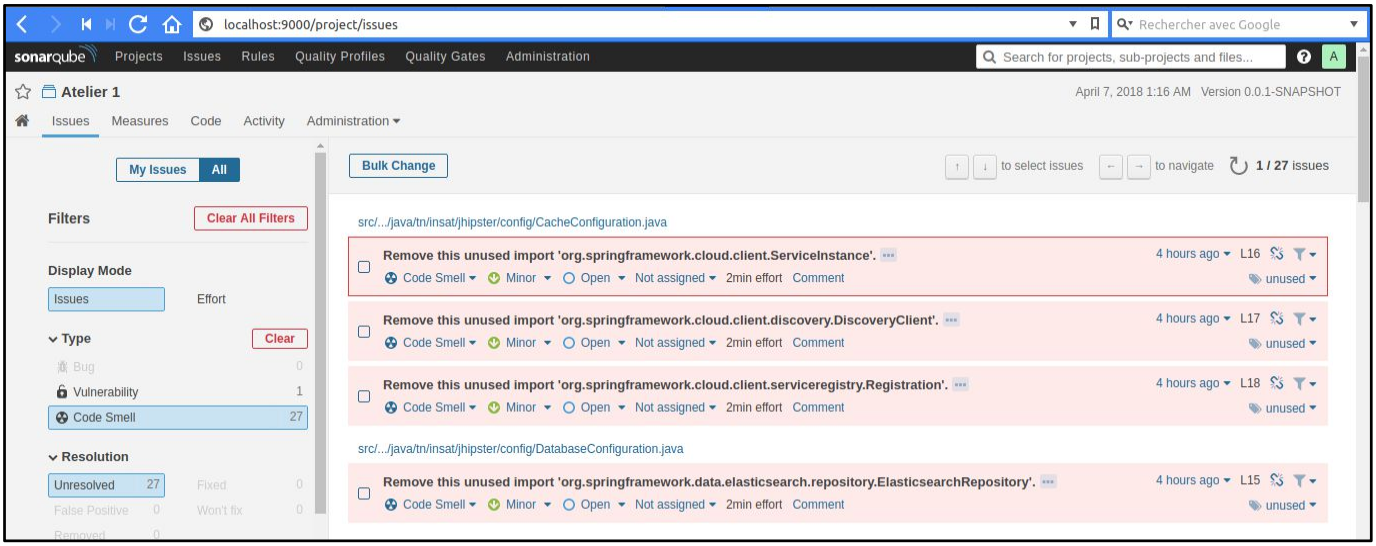
7. Code Quality

Code quality metrics are provide by sonar, which is already configured for you.
Sonar will run inside a docker container so we need to install “Docker” and the “docker compose” tool
sudo apt install docker-compose
run the container: sudo docker-compose -f src/main/docker/sonar.yml up -d
Perform sonar analysis: ./mvnw clean test sonar:sonar

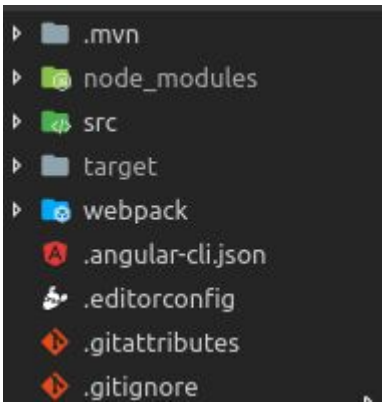
```
[INFO] Analysis report generated in 2356ms, dir size=864 KB
[INFO] Analysis reports compressed in 5557ms, zip size=389 KB
[INFO] Analysis report uploaded in 1504ms
[INFO] ANALYSIS SUCCESSFUL, you can browse http://localhost:9000/dashboard/index/tn.insat.jhipster:atelier-1
[INFO] Note that you will be able to access the updated dashboard once the server has processed the submitte
[INFO] More about the report processing at http://localhost:9000/api/ce/task?id=AWKdd3Eh-O4CZkzdxoz7
[INFO] Task total time: 1:02.672 s
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 06:45 min
[INFO] Finished at: 2018-04-07T01:17:55+01:00
[INFO] Final Memory: 87M/659M
[INFO] ------------------------------------------------------------------------
```

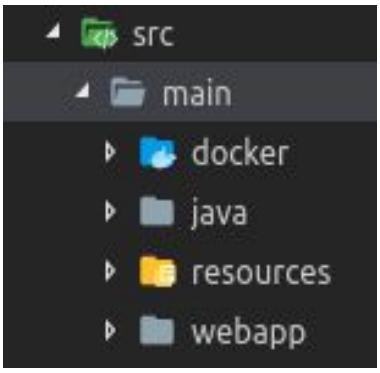
Visit *localhost:9000* to see the results.
You can change the port in the src/main/docker/sonar.yml file

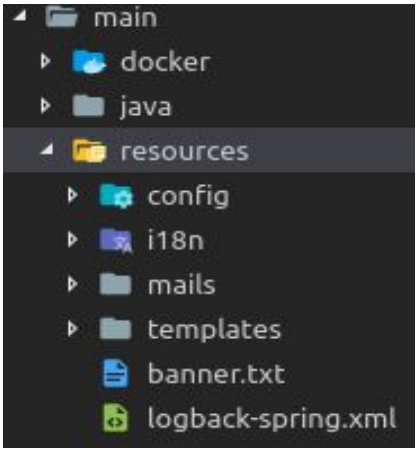


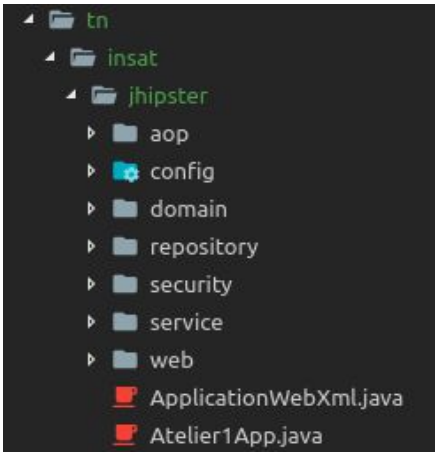


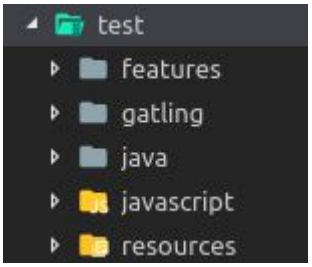
8. Project Structure





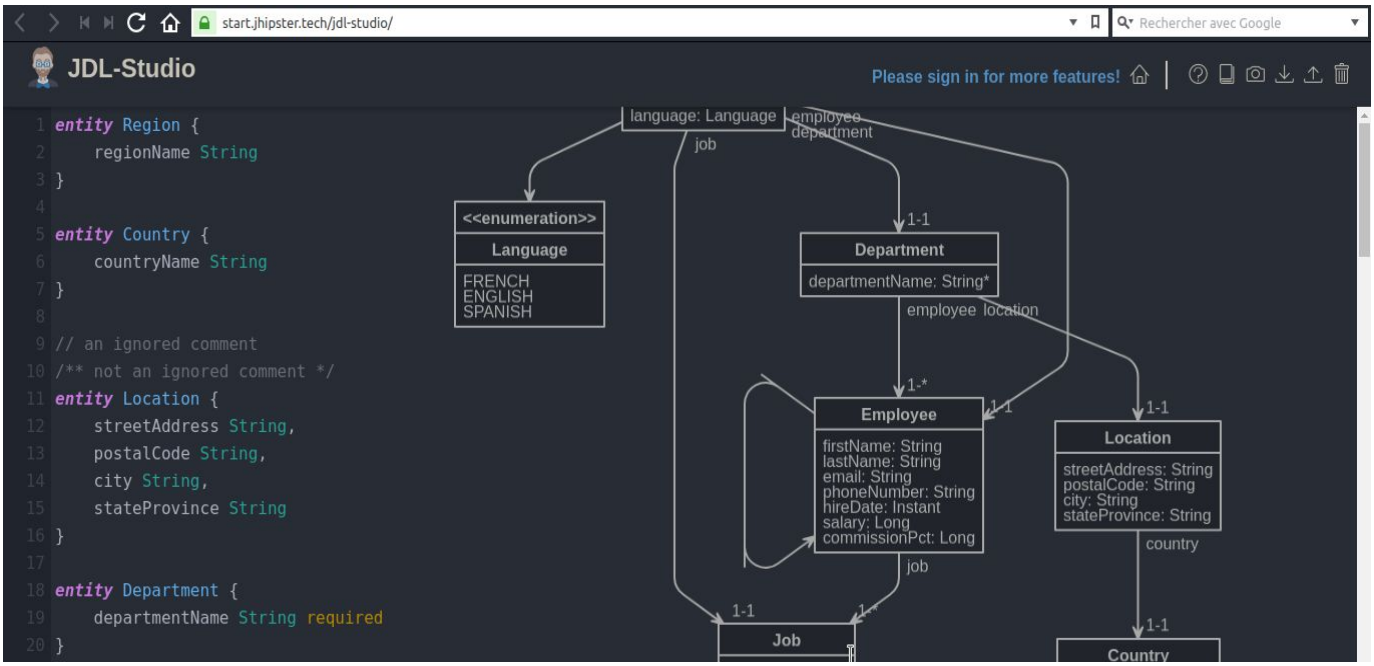




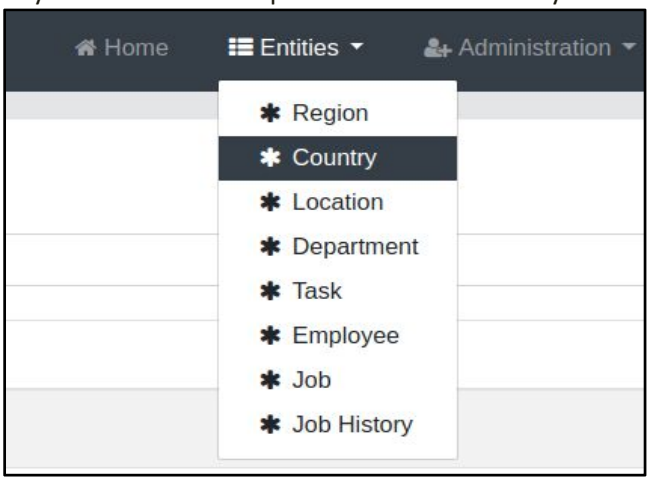


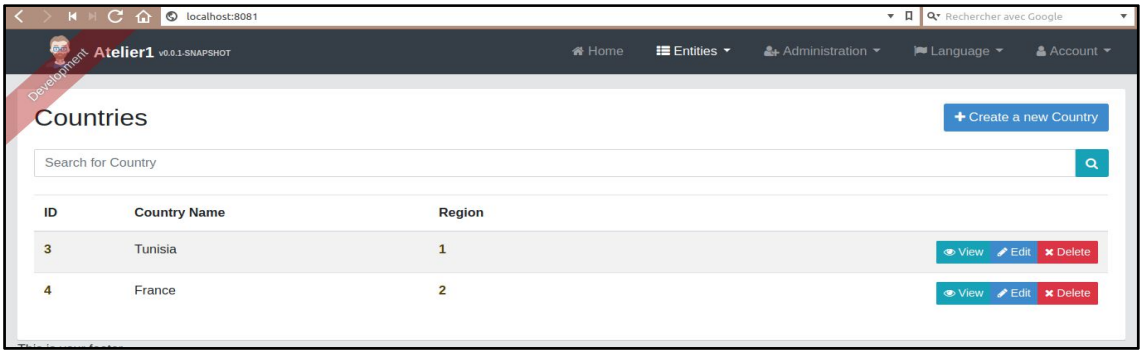
9. Creating entities

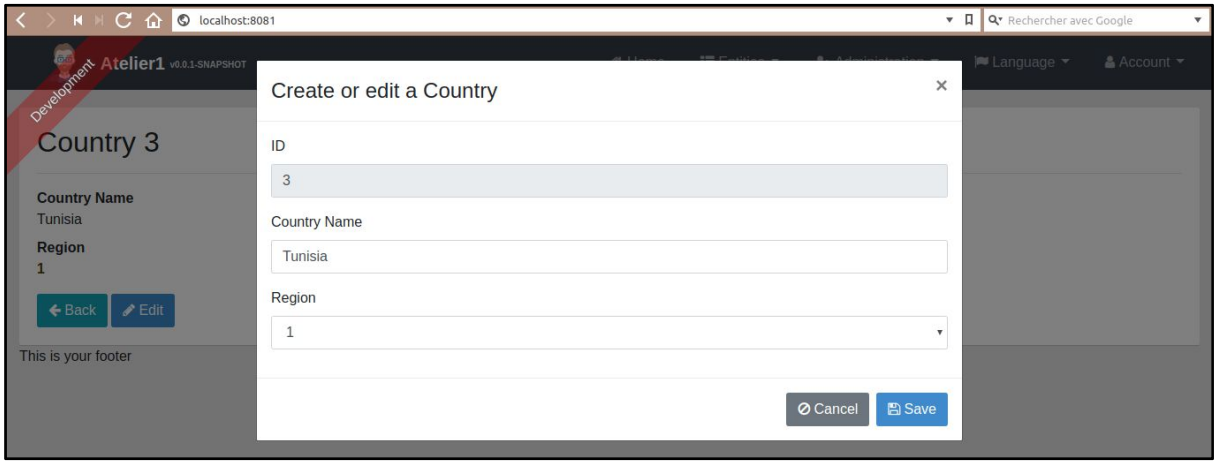
To create entities and manage relations we will use JDL Studio: an online tool to create entities and relationships using the domain-specific language JDL.
When visiting https://start.jhipster.tech/jdl-studio/ we will find a domain definition example using JDL, so we're going to use it in our demo :

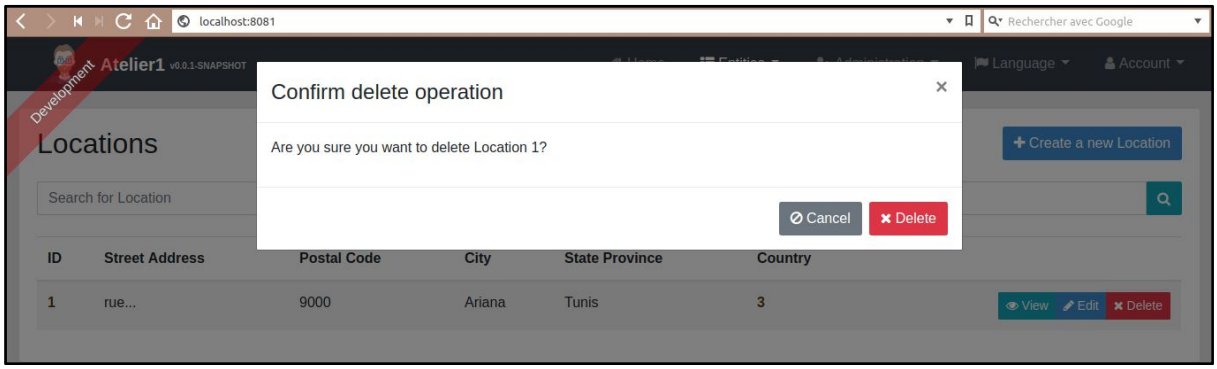


All you have to do is download the JDL file (in the top right corner you'll find the download button) , and then run: jhipster import-jdl your-jdl-file.jh to generate the entities, repositories, services, tests and update the client app
These are the results, we have fully functional CRUD operations for each entity









10. Mobile Client: Ionic

First, you have to install ionic: *npm i -g ionic*

We will use the ionic jhipster generator module: *npm install -g generator-jhipster-ionic*

Run: *yo jhipster-ionic* and specify the backend folder

```
karim@karim-pc /media/karim/Data/Mega/GL4S2/Architecture/presentation $ yo jhipster-ionic

Welcome to the Ionic Module for JHipster! v3.1.2

? What do you want to name your Ionic application? Atelier1Mobile
? Enter the directory where your JHipster app is located: Atelier1

Creating Ionic app with command: ionic start Atelier1Mobile oktadeveloper/jhipster
✔ Creating directory ./Atelier1Mobile - done!
✔ Looking up starter - done!
```

As the tool suggests, make sure to run the backend before running ionic

```
Hipster Ionic App created successfully!

Run the following commands (in separate terminal windows) to see everything working:

    cd Atelier1 && ./mvnw
    cd Atelier1Mobile && ionic serve
```

But first we're going to update the api port of the backend from 8080 to 8081, in src/providers/api/api.ts

```
api.ts
@Injectable()
export class Api {
  public static API_URL: string = 'http://localhost:8081/api'

  constructor(public http: HttpClient) {
  }
```

Before generating pages for our entities, we need to set the angularJSSuffix in the .jhipster/Entity.json in the backend project to ""

```
    "angularJSSuffix": ""
```

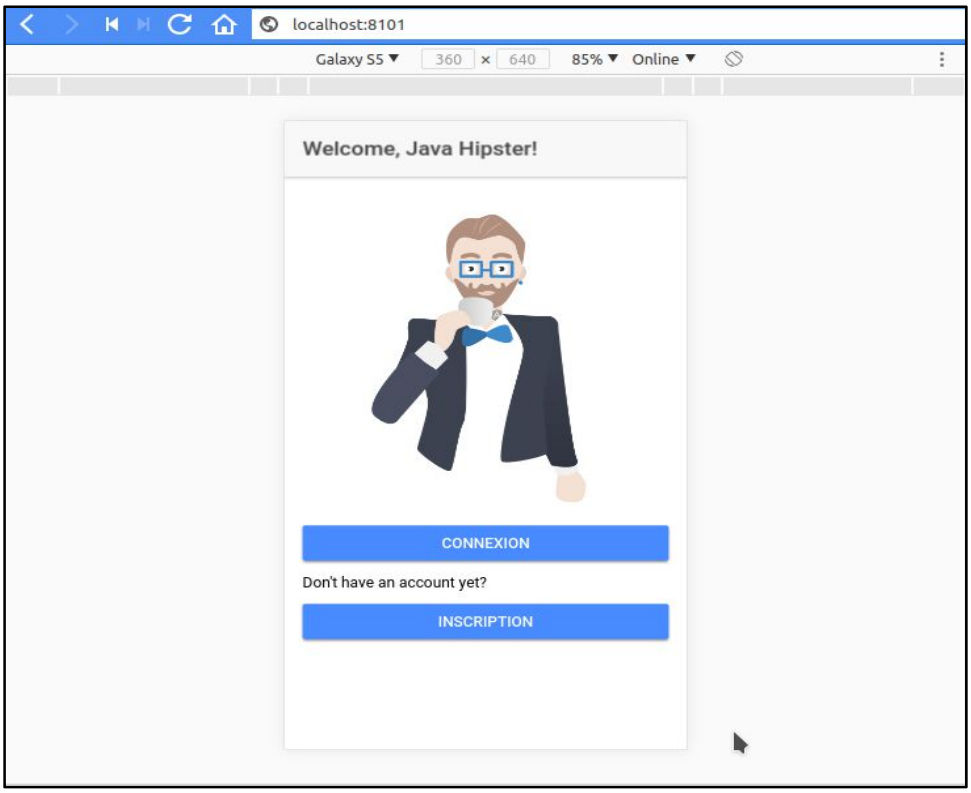
Instead of "mySuffix" used in the example that we found in JDL Studio

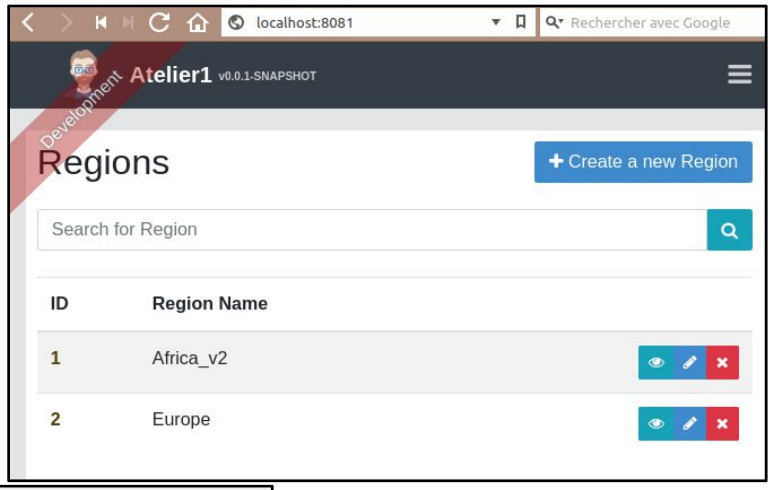
```
// Set an angular suffix
angularSuffix * with mySuffix
```

Generate an entity:yo jhipster-ionic:entity Region

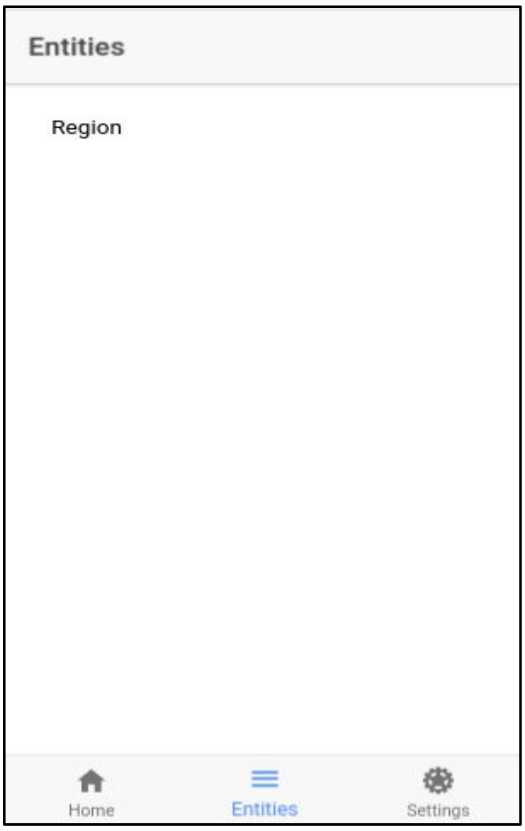
```
le $ yo jhipster-ionic:entity Region

The entity Region is being created.

? Do you want to generate this entity from an existing app? Yes
? Enter the path to your app's root directory: ../Atelier1

Found the .jhipster/Region.json configuration file, entity can be automatically ge
nerated!

? Do you want to update the entity? This will replace the existing files for this
entity, all your custom code will be overwritten (Use arrow keys)
❯ Yes, re generate the entity
```

run *ionic : ionic serve*


localhost:8101
Galaxy S5
360
640
85%
Online
Welcome, Java Hipster!
CONNEXION
Don't have an account yet?
INSCRIPTION

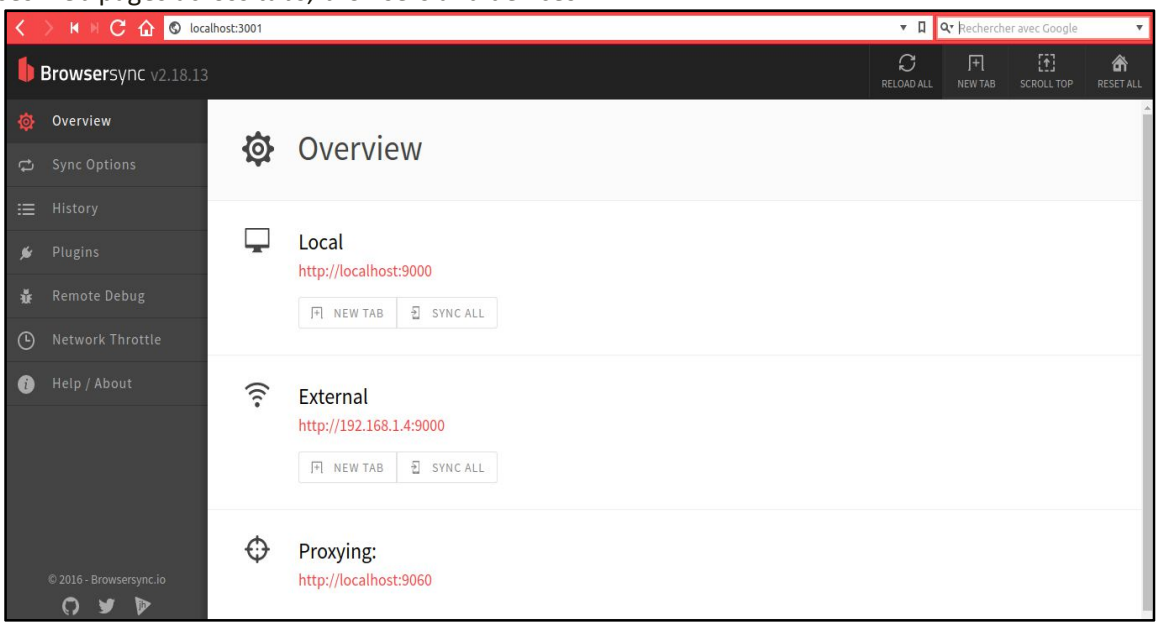

localhost:8081
Rechercher avec Google
Development
Atelier1 v0.0.1-SNAPSHOT
Regions
Create a new Region
Search for Region
ID
Region Name
1
Africa_v2
2
Europe

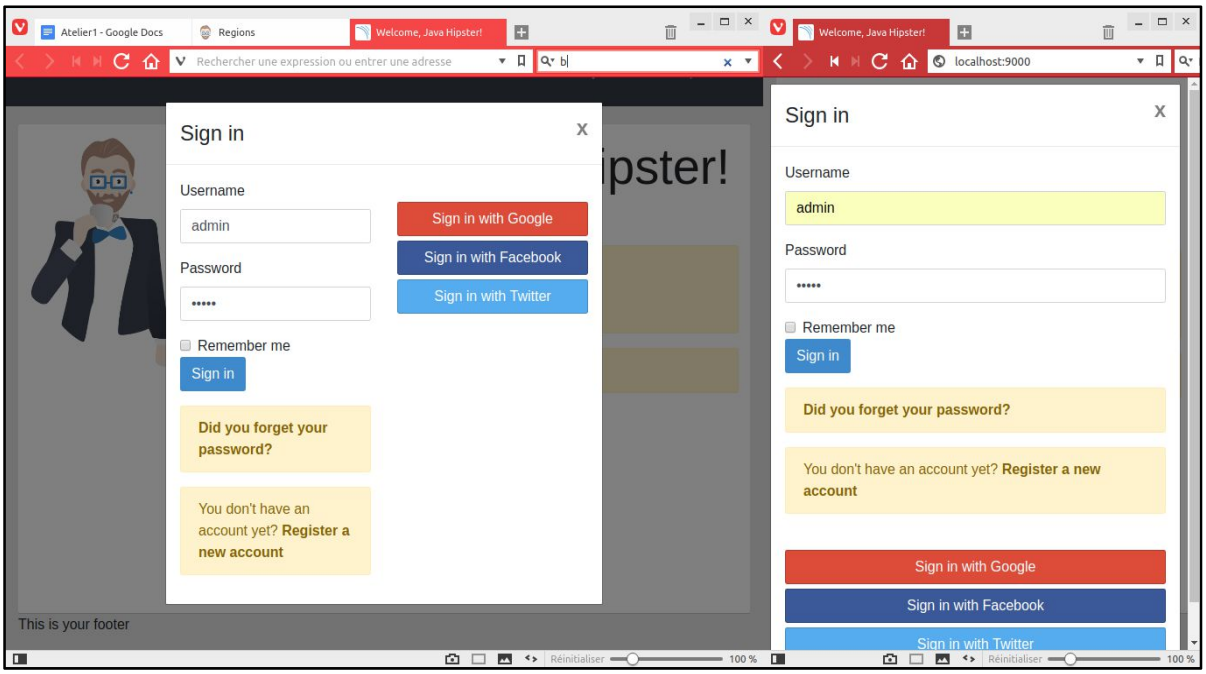

Connexion
Username
user
Mot de passe
Remember Me
CONNEXION



11. Webpack dev server:

To run a developpement server for angular with hot reloading and other features all you need to do is run : npm start. You can access the app in localhost:9000. This will also start *Browsersync* in localhost:3001, a test tool that synchronises web pages across tabs, browsers and devices.





Second Demo:

1. Goal:

In our first demo, we illustrated how to create and bootstrap a monolithic application, which is relatively easy to maintain.

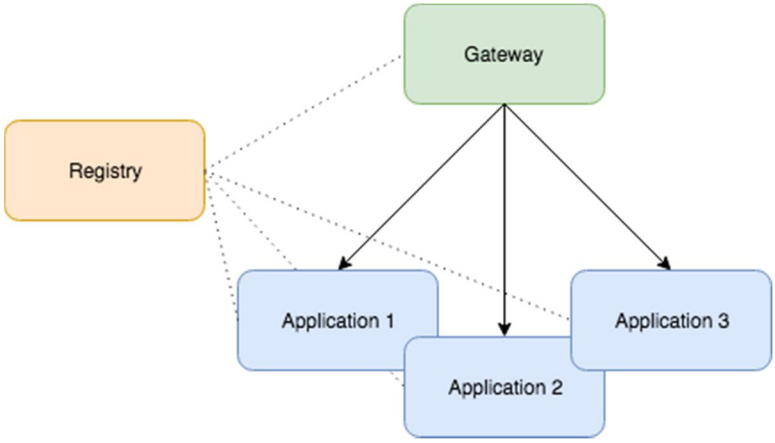
Our microservice system, on the other hand, will separate the front-end the back-end, which, in turn, can also be split into several small applications, each dealing with a subset of the full application domain. Naturally, as with all microservice implementations, this solves some problems but also introduces some complexity, such as dealing with component registry and security.

JHipster will take care of most difficulties of managing the microservice applications, with the help of modern open-source tools like *Netflix's Eureka Server* and *Hashicorp's Consul*.
There are, of course, some things to consider here, like how big or complicated our domain is, how critical is our application and what levels of availability do we want need to have, are we going to host our services on different servers and locations, etc. The goal of these tools is of course to make these permutations possible and easy to manage.

2. JHipster Microservice Components

When working on a Microservice architecture with JHipster, we'll need to build and deploy at least three different projects: a JHipster Registry, a Microservice Gateway, and at least one Microservice Application.

- **The JHipster Registry** is an essential piece of the microservice architecture. It ties all the other components together and enables them to communicate with each other.
- **The Microservice Application** contains the back-end code. Once running it will expose the API for the domain it is concerned with. A Microservice Architecture may be composed of many microservice applications, each containing a few related entities and business rules.
- **And the Microservice Gateway** has all the front-end (Angular) code and will consume the API created by the whole group of micro service applications:



3. Requirements:

We will need to have these tools installed:

1. Install Java 8 from Oracle
2. Install Git from https://git-scm.com
3. Install Node.js from http://nodejs.org (I used Node 6.11.0 to write this article)
4. Install Yarn using the Yarn installation instructions
5. Run the following command to install Yeoman *'yarn global add yo'*
6. Run the following command to install JHipster *'yarn global add generator-jhipster'*

4. Creating a Microservice Project

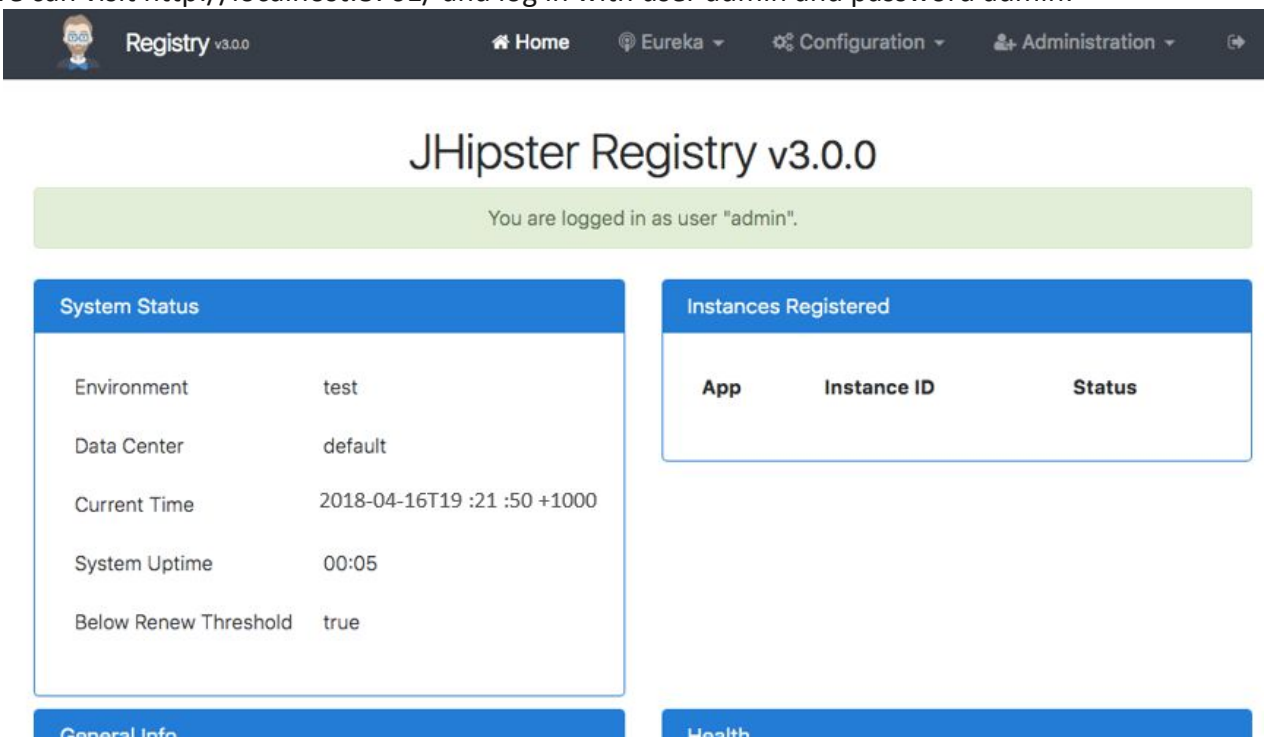
Now let's install the three core components of our microservice project.

4.1. Installing JHipster Registry

Since the JHipster Registry is a standard JHipster, so we just need to download and run it. There is no need to modify it:

```
git clone https://github.com/jhipster/jhipster-registry
cd jhipster-registry && ./mvnw
```

This will clone the jhipster-registry project from GitHub and start the application. Once it successfully starts up, we can visit http://localhost:8761/ and log in with user admin and password admin:



3.2. Installing a Microservice Application

Here is where we start to build the actual features of our project. In this example, we'll create a simple Microservice Application that manages cars. So first, we'll create the application, and then we'll add an entity to it:

```
# create a directory for the app and cd to it
mkdir car-app && cd car-app
# run the jhipster wizard
yo jhipster
```

Once the wizard starts, let's follow the instructions to create a Microservice type application named carapp. Some other relevant parameters are:

- port: 8081
- package: com.car.app
- authentication: JWT
- service discovery: JHipster Registry

The screenshot bellow shows the complete set of options:

```
? (1/15) Which *type* of application would you like to create? Microservice application
? (2/15) What is the base name of your application? carsapp
? (3/15) Would you like to install other generators from the JHipster Marketplace? No
? (4/15) As you are running in a microservice architecture, on which port would like your server to run? It should be uni
que to avoid port conflicts. 8081
? (5/15) What is your default Java package name? com.cars.app
? (6/15) Which *type* of authentication would you like to use? JWT authentication (stateless, with a token)
? (7/15) Which Service Discovery and Configuration solution would you like to use? JHipster Registry (using Eureka and Sp
ring Cloud Config)
? (8/15) Which *type* of database would you like to use? SQL (H2, MySQL, MariaDB, PostgreSQL, Oracle)
? (9/15) Which *production* database would you like to use? MySQL
? (10/15) Which *development* database would you like to use? H2 with disk-based persistence
? (11/15) Do you want to use Hibernate 2nd level cache? Yes, with HazelCast (distributed cache, for multiple nodes)
? (12/15) Would you like to use Maven or Gradle for building the backend? Maven
? (13/15) Which other technologies would you like to use? (Press <space> to select, <a> to toggle all, <i> to inverse sel
ection)
? (14/15) Would you like to enable internationalization support? No
? (15/15) Besides JUnit and Karma, which testing frameworks would you like to use?
```

Now we'll add a car entity to our application:

```
# runs entity creation wizard
yo jhipster:entity car
```

The entity creation wizard will start. We should follow the instructions to create an entity named the car with three fields: make, model, and price.

Once that's finished, our first Microservice application is complete. If we have a look at the generated code, we'll notice that there's no javascript, HTML, CSS or any front-end code. Those will all be produced once the Microservice Gateway is created. Also, check out the README file for important information about the project and useful commands.

To finish up, let's run our newly created component:

```
./mvnw
```

P.S: Before running the above command, we should ensure that the jhipster-registry component is up and running. Otherwise, we'll get an error.

If everything went according to plan, our car-app would start, and the jhipster-registry log will tell us that the app was successfully registered:

```
Registered instance CARAPP/carapp:746e7525dffa737747dcdcee55ab59f3
  with status UP (replication=true)
```

3.3. Installing a Microservice Gateway

Now the front-end bit. We'll create a Microservice Gateway and indicate to it that we have an entity on an existent component for which we want to create the front-end code:

```
# Create a directory for our gateway app
mkdir gateway-app && cd gateway-app
# Runs the JHipster wizard
yo jhipster
```

Let's follow the instructions to create an application of type Microservice gateway. We'll name the application gateway, and select the following options for the other parameters:

- port: 8080
- package: com.gateway
- auth: JWT
- service discovery: JHipster Registry

Here is a summary of the complete set of parameters:

```
? (1/15) Which *type* of application would you like to create? Microservice gateway
? (2/15) What is the base name of your application? gateway
? (3/15) Would you like to install other generators from the JHipster Marketplace? No
? (4/15) As you are running in a microservice architecture, on which port would like your server to run? It should be uni
que to avoid port conflicts. 8080
? (5/15) What is your default Java package name? com.gateway
? (6/15) Which *type* of authentication would you like to use? JWT authentication (stateless, with a token)
? (7/15) Which Service Discovery and Configuration solution would you like to use? JHipster Registry (using Eureka and Sp
ring Cloud Config)
? (8/15) Which *type* of database would you like to use? SQL (H2, MySQL, MariaDB, PostgreSQL, Oracle, MSSQL)
? (9/15) Which *production* database would you like to use? MySQL
? (10/15) Which *development* database would you like to use? H2 with disk-based persistence
? (11/15) Do you want to use Hibernate 2nd level cache? Yes, with HazelCast (distributed cache, for multiple nodes)
? (12/15) Would you like to use Maven or Gradle for building the backend? Maven
? (13/15) Which other technologies would you like to use? (Press <space> to select, <a> to toggle all, <i> to inverse sel
ection)
? (14/15) Which *Framework* would you like to use for the client? AngularJS 1.x
? (15/15) Would you like to use the LibSass stylesheet preprocessor for your CSS? No
? (16/15) Would you like to enable internationalization support? No
? (17/15) Besides JUnit and Karma, which testing frameworks would you like to use? (Press <space> to select, <a> to toggl
e all, <i> to inverse selection)
```

Let's move on to entity creation:

```
# Runs entity creation wizard
yo jhipster:entity car
```

When asked if we want to generate from an existent microservice, choose Yes, then type in the relative path to the car-app root directory (ex.: ../car-app). Finally, when asked if we want to update the entity, choose *Yes*, regenerate the entity.

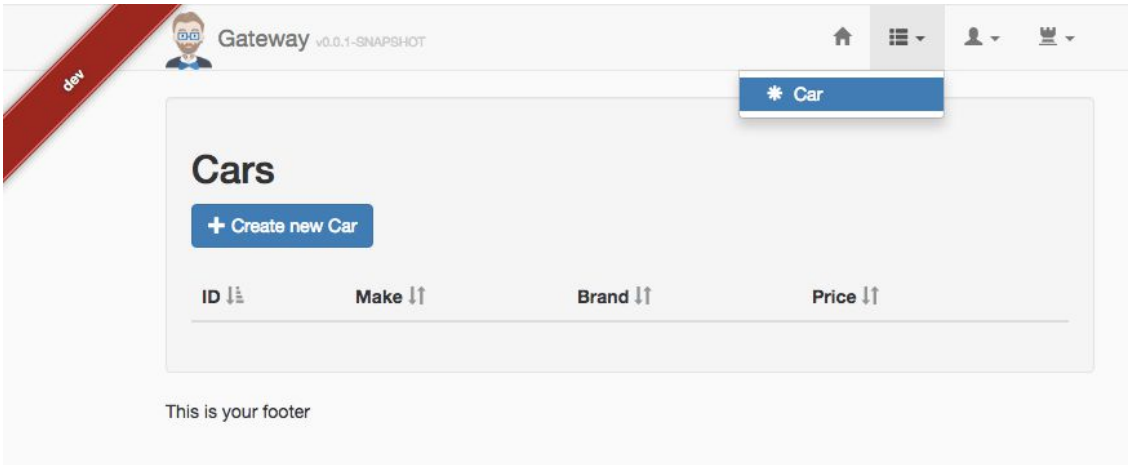
JHipster will find the Car.json file which is part of the existent Microservice Application we've created earlier and will use the metadata contained in that file to create all the necessary UI code for that entity:

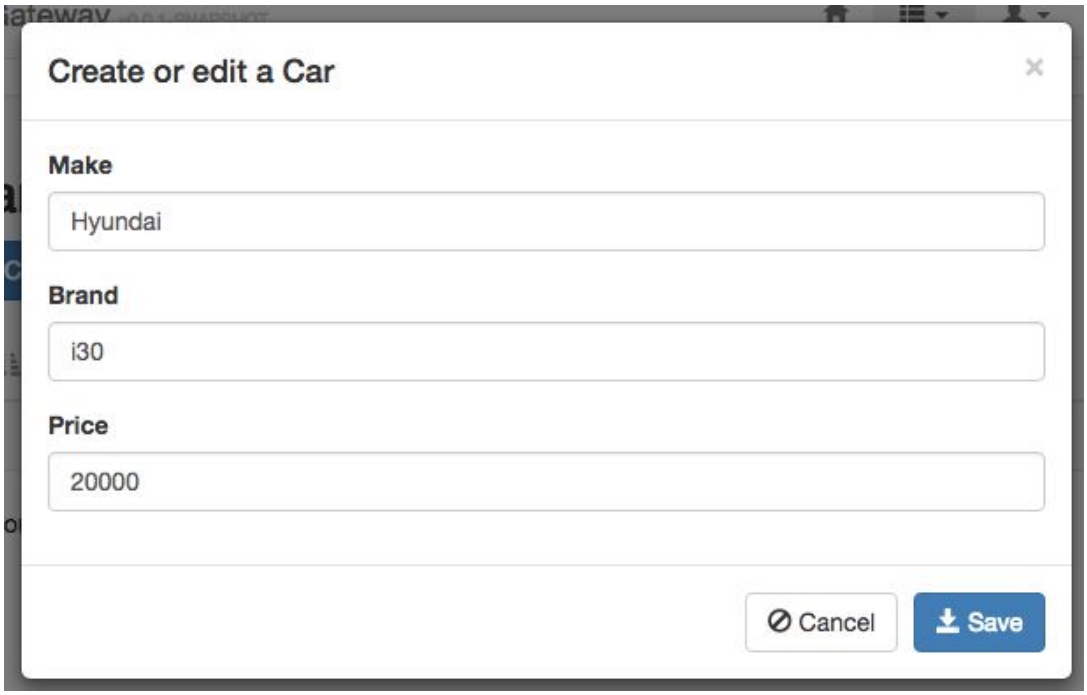
```
Found the .jhipster/Car.json configuration file, entity can be automatically generated!
```

Time to run the gateway-app and test if everything is working:

```
# Starts up the gateway-app component
./mvnw
```

Let's now navigate to *http://localhost:8080/* and log in with user admin and password admin. On the top menu, we should see an item Car that will take us to the car list page. All good!





3.4. Creating a Second Microservice Application

Next, let's take our system one step further and **create a second component of type Microservice Application**. This new component will manage car dealers, so we'll add an entity called the dealer to it.

Let's create a new directory, navigate to it and run the yo jhipster command:

```
mkdir dealer-app && cd dealer-app
yo jhipster
```

After that, we type in dealerapp as the application's name and choose port 8082 for it to run (it is critical that this is a different port than the ones we're using for the jhipster-registry and car-app).

For the other parameters, we can choose any option we want. Remember this is a separate microservice so that it can use different database types, cache strategy, and tests than the car-app component.

Let's add a couple of fields to our dealer entity. For example name and address:

```
# Runs the create entity wizard
yo jhipster:entity dealer
```

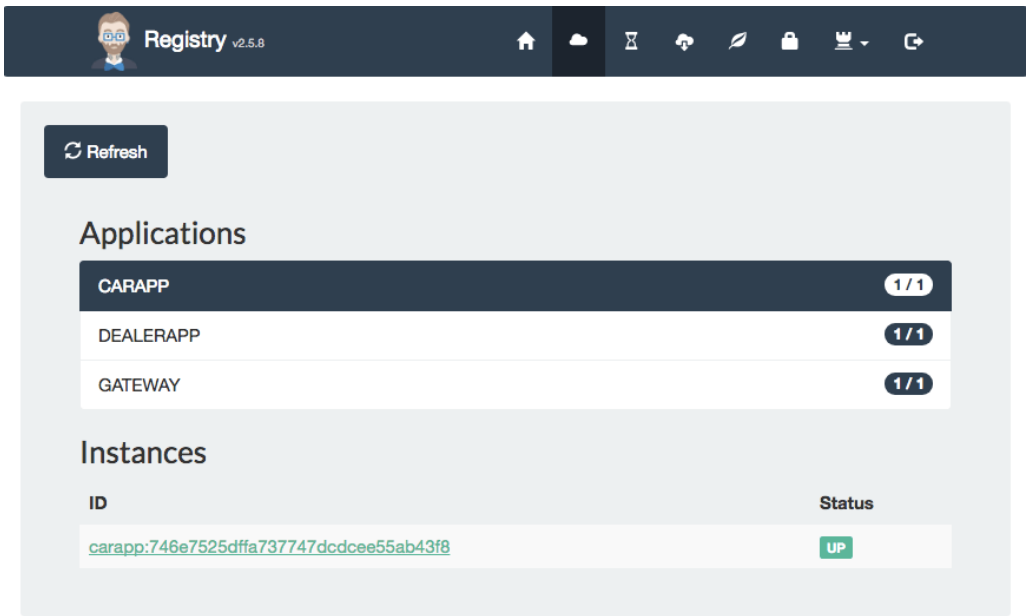
P.S: We shouldn't forget to navigate to gateway-app and tell it to generate the front-end code for the dealer entity:

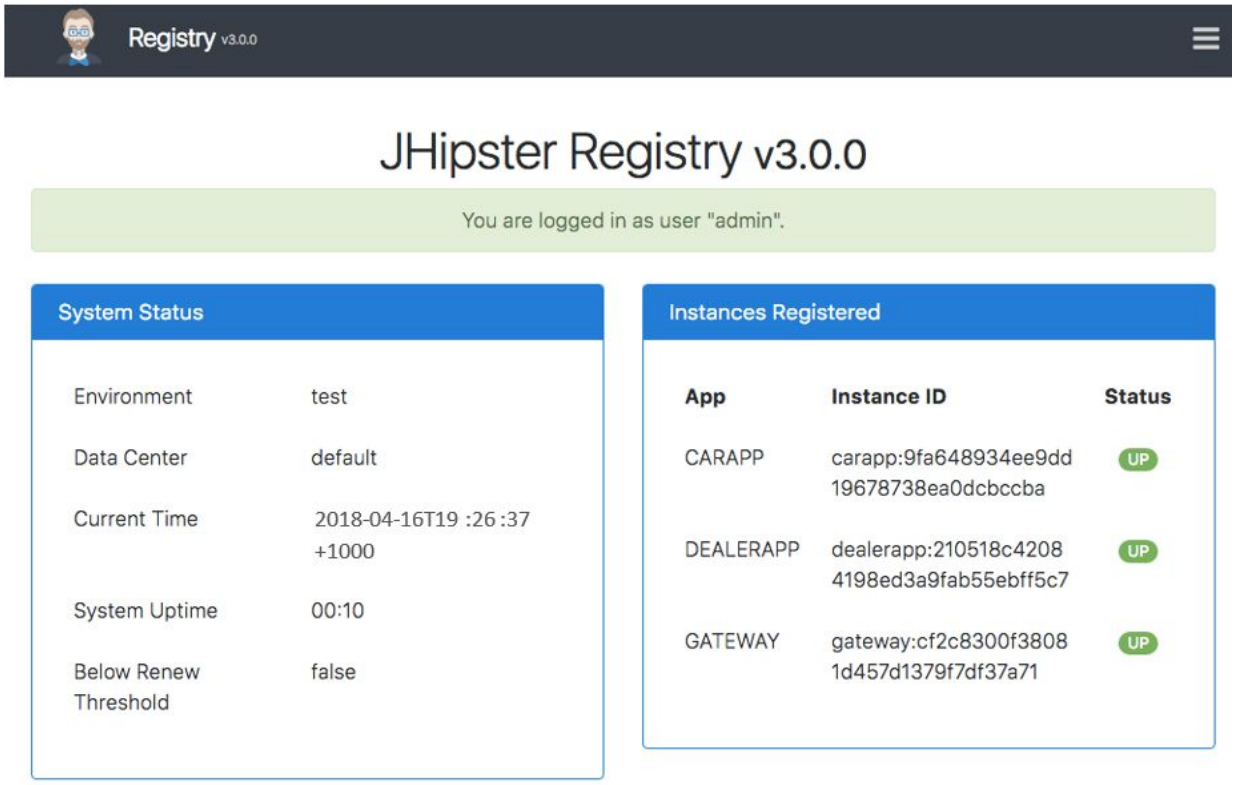
```
# Navigate to the gateway-app root directory
cd ../gateway-app
# Runs the create entity wizard
yo jhipster:entity dealer
```

Finally, run './mvnw' on the dealer-app root directory to start up that component.

Next, we can visit our gateway application at *http://localhost:8080* and refresh the page to see the newly created menu item for the Dealer entity.

Before we wrap up, let's have a look at the jhipster-registry application again at *http://localhost:8761/*. Click on the Applications menu item to check that all of our three components were successfully identified and registered:





That's it! We've created a sophisticated architecture comprised of one Gateway app with all the front-end code backed by two microservices in just a few minutes.

Starting a Microservice Architecture project with JHipster is quite easy; we only need to create as many Microservice Applications as we need and one Microservice Gateway and we are ready to go.