Create and deploy a basic JHipster application to Heroku

A tutorial for beginners by David Garcerán.

Student: David Garcerán García / LinkedIn: https://linkedin.com/in/davidgarceran

Teacher: Alfredo Rueda Unsaín / LinkedIn: https://linkedin.com/in/alfredorueda

Course: Development of Cross-Platform Applications

Study Centre: Stucom http://www.stucom.com/

Index

1. Introduction

1.1. Basics

This is an easy step-to-step tutorial to create an app based on Angular and Spring, and deploy it with Heroku. To make it as easy as possible we're going to start a JHipster application with PostgreSQL. JHipster is a generator that uses Yeoman (http://yeoman.io/) to create a basic application based on Spring (back-end) + Angular (front-end).

1.2. What do I need?

To know what you need before you start, you'll have to go to this address and follow all the steps to install JHipster in your computer: http://jhipster.github.io/installation/

Also you'll need Heroku Toolbelt to deploy the app: https://toolbelt.heroku.com/

You can use as IDE whatever you're familiar with. In my case I'll use IntelliJ IDEA 15, a really professional and competitive one, created by JetBrains. But don't worry, you can do all these things with another one. In fact all you need is your command console.

As a database here we will use PostgreSQL, so you need it installed on your computer. They provide you all the installers and (at least, that's in the case of Windows) a basic graphic console to manage your databases called pgAdmin III. Everything is available for all the platforms:

http://www.postgresql.org/download

http://www.pgadmin.org/

1.3. The application

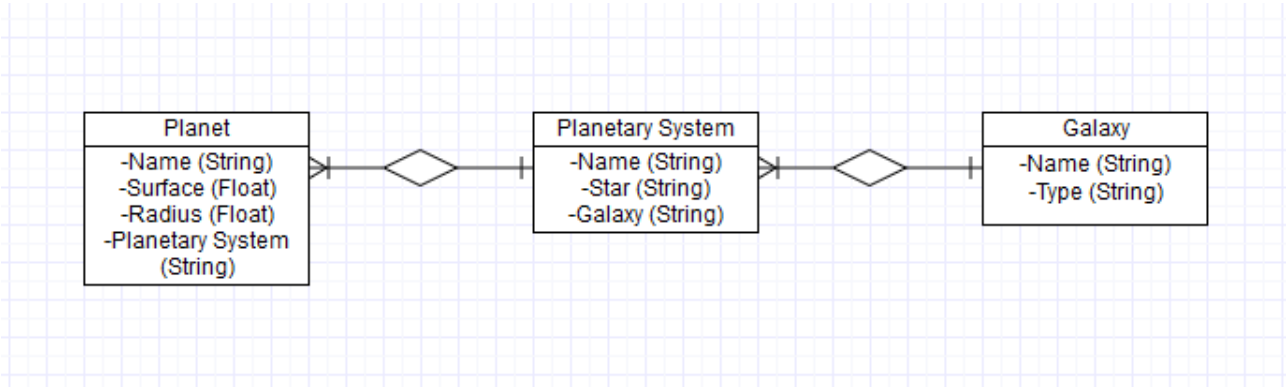

Image 1.1.: Entity relationship diagram of our application

The application we're going to create is a database that will be able to manage planets, planetary systems and galaxies. A planet has a planetary system, and a planetary system is part of a galaxy, so our main entities are pretty clear and already designed (*Image 1.1.*). All we need to do is to start to work on it!

2. Starting to work with JHipster

2.1. First steps with JHipster

Your environment should be already working. I recommend you to check all the versions you are using of all the utilities we need like Yeoman, Node or Bower and make sure they are up to date, because if they aren't, it will cause many problems. I also recommend to not to forget to install Maven, even if you use an IDE like IntelliJ IDEA 15 that integrates everything and where you can run everything. If you don't install it you won't be able to follow this tutorial because Heroku will ask for Maven to deploy your app. It's not much fun to change the PATH or set the JAVA_HOME, but you have to deal with that, and it's pretty easy to find how to do it.

Let's open the `cmd` and go look for the path where you want to save your application. I recommend you to make a folder for it, that's what I did:

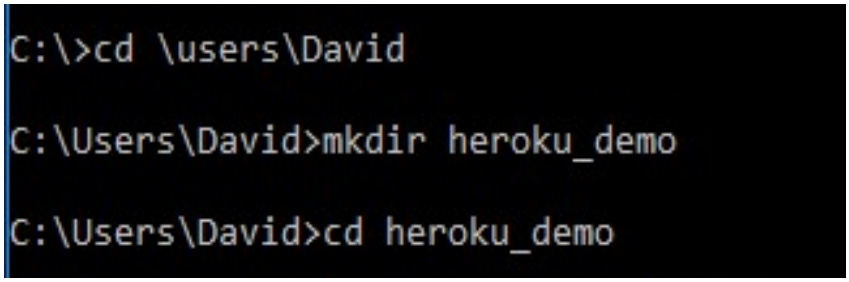

Image 2.1

You have now a beautiful folder that contains absolutely nothing, so let's fill it with all the code we need to start to work. All the JHipster applications starts with the next command: `yo jhipster`

Image 2.2: Starting a JHipster application

That command (*Image 2.2*) opens the menu of a generator that will ask you some questions. You'll have to use the same ones as on the *Image 2.4.*, because they will be exactly what you need for this application.

The JHipster generator have many options we're not going to use here, so if you like this simple app you can generate some others and experiment with all the possibilities.

If you want more information about the questions and options of the JHipster generator, check out this address:

http://jhipster.github.io/creating-an-app/

Image 2.3.:Welcome and version

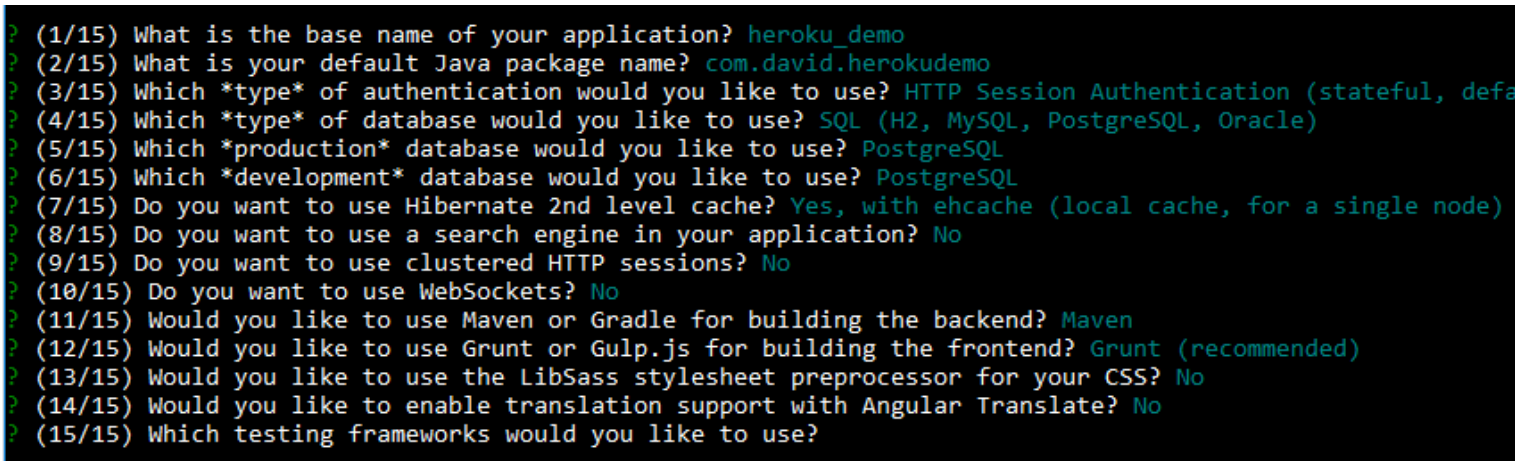

Image 2.4.: Options of our JHipster application

The most interesting options we're going to use with these questions are the ones related to a database. Heroku works with PostgreSQL so it's really important to use it as our database system. You can change that later, but for now we will keep it like that.

Also we're not going to use testing frameworks (question 15), you can deactivate the one that is active by default (Gatling) with the space bar. We're not going to use LibSass or Angular Translate either, so we are going to have the most basic application possible ready for everyone to be used.

Then, after the last question, JHipster creates a beautiful application already prepared with a back-end and a front-end.

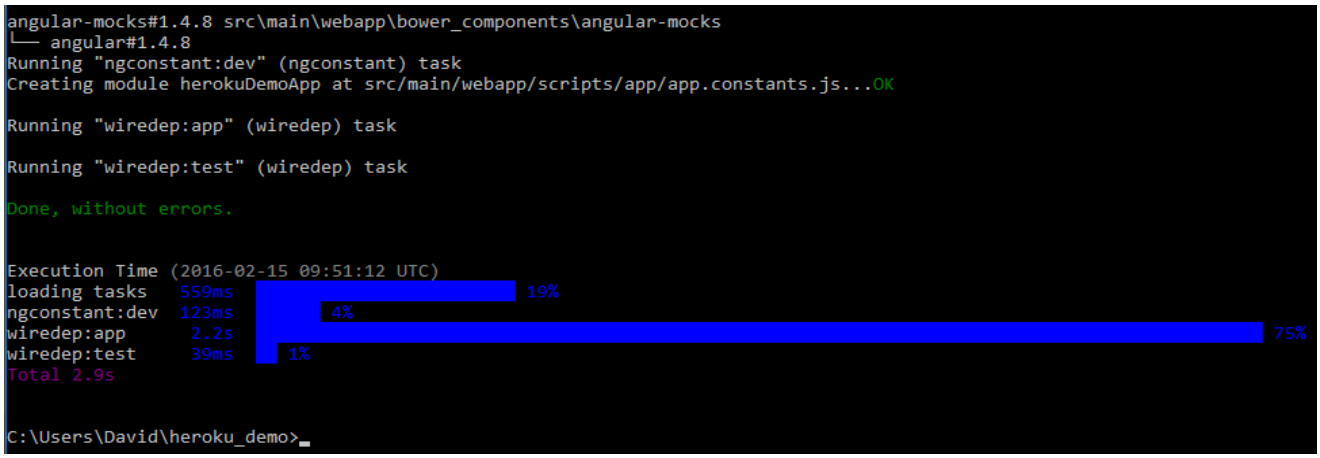

Image 2.5.: Job finished

2.2. Creating the entities

The entities of this application are, as we have mentioned before (*Image 1.1*): planet, planetary system and galaxy. I know it's a bit disappointing to not see anything of your application yet, but you have to be patient. In fact, if you would set up the database properly you could see what you have created on the steps before, but you wouldn't be able to do anything with what you have right now. So let's start! The first entity that we will create will be ‘galaxy’. I choose this one because it's the only one without a relationship, so we don't need a reference to create our entity.

The command to create an entity is yo jhipster:entity <name of the entity>.

After that, JHipster will ask you some questions to create every field of your entity. You have to choose the name of the field, then the type of data that the user will enter there, then you have to choose to add restrictions or not. Once you finish with the fields, it will ask you if you want to add relationships with other entities. In our case, we will add the relationships only on ‘planet’ (*Image 2.8.1* and *Image 2.8.2*) and ‘planetary_system’ (*Image 2.7*).

WARNING: I've created a couple of fields that are redundant and repetitive. The fields you don't need are “galaxy” inside the entity “planetary_system”, and the field “system” inside the entity “planet”. Why don't you need them and they're redundant? Because I forgot that the relationships made after them are also a field that accomplish everything I wanted: select a galaxy and a planetary system that would come from the list of the ones created. I realized this problem a bit late, but I'll show you where you can fix it on the Annex II. By the way, I'm aware that on the next update of JHipster (version 3.0) you'll be able to modify entities so you won't need that.

```
C:\Users\David\heroku_demo>yo jhipster:entity galaxy
The entity galaxy is being created.
Generating field #1
? Do you want to add a field to your entity? Yes
? What is the name of your field? name
? What is the type of your field? String
? Do you want to add validation rules to your field? No
================Galaxy================
name (String)
Generating field #2
? Do you want to add a field to your entity? Yes
? What is the name of your field? type
? What is the type of your field? String
? Do you want to add validation rules to your field? No
================Galaxy================
name (String)
type (String)
Generating field #3
? Do you want to add a field to your entity? No
================Galaxy================
name (String)
type (String)
Generating relationships with other entities
? Do you want to add a relationship to another entity? No
==========Galaxy==========
name (String)
type (String)
------------------
? Do you want to use a Data Transfer Object (DTO)? No, use the entity directly
? Do you want to use separate service class for your business logic? No, the REST controller should use the
? Do you want pagination on your entity? Yes, with pagination links
Everything is configured, generating the entity...
```

Image 2.6.: Entity galaxy

```
C:\Users\David\heroku_demo>yo jhipster:entity planetary_system
The entity planetary_system is being created.
Generating field #1
? Do you want to add a field to your entity? Yes
? What is the name of your field? name
? What is the type of your field? String
? Do you want to add validation rules to your field? No
================Planetary_system================
name (String)
Generating field #2
? Do you want to add a field to your entity? Yes
? What is the name of your field? star
? What is the type of your field? String
? Do you want to add validation rules to your field? No
================Planetary_system================
name (String)
star (String)
Generating field #3
? Do you want to add a field to your entity? Yes
? What is the name of your field? galaxy
? What is the type of your field? String
? Do you want to add validation rules to your field? No
================Planetary_system================
name (String)
star (String)
galaxy (String)
Generating field #4
? Do you want to add a field to your entity? No
================Planetary_system================
name (String)
star (String)
galaxy (String)
Generating relationships with other entities
? Do you want to add a relationship to another entity? Yes
? What is the name of the other entity? galaxy
? What is the name of the relationship? galaxy_relationship
? What is the type of the relationship? many-to-one
? When you display this relationship with AngularJS, which field from 'galaxy' do you want to use? name
==========Planetary_system=============
name (String)
star (String)
galaxy (String)
------------------
galaxy_relationship - galaxy (many-to-one)
Generating relationships with other entities
? Do you want to add a relationship to another entity? No
==========Planetary_system=============
name (String)
star (String)
galaxy (String)
------------------
galaxy_relationship - galaxy (many-to-one)
? Do you want to use a Data Transfer Object (DTO)? No, use the entity directly
? Do you want to use separate service class for your business logic? No, the REST controller should use
? Do you want pagination on your entity? Yes, with pagination links
Everything is configured, generating the entity...
```

Image 2.7.: Entity planetary_system

```
C:\Users\David\heroku_demo>yo jhipster:entity planet
The entity planet is being created.
Generating field #1
? Do you want to add a field to your entity? Yes
? What is the name of your field? name
? What is the type of your field? String
? Do you want to add validation rules to your field? No
================Planet================
name (String)
Generating field #2
? Do you want to add a field to your entity? Yes
? What is the name of your field? surface
? What is the type of your field? Float
? Do you want to add validation rules to your field? No
================Planet================
name (String)
surface (Float)
Generating field #3
? Do you want to add a field to your entity? Yes
? What is the name of your field? radius
? What is the type of your field? Float
? Do you want to add validation rules to your field? No
================Planet================
name (String)
surface (Float)
radius (Float)
Generating field #4
? Do you want to add a field to your entity? Yes
? What is the name of your field? system
? What is the type of your field? String
? Do you want to add validation rules to your field? No
================Planet================
name (String)
surface (Float)
radius (Float)
system (String)
Generating field #5
? Do you want to add a field to your entity? No
================Planet================
name (String)
surface (Float)
radius (Float)
system (String)
Generating relationships with other entities
? Do you want to add a relationship to another entity? Yes
? What is the name of the other entity? planetary_system
? What is the name of the relationship? planetary_system
? What is the type of the relationship? many-to-one
? When you display this relationship with AngularJS, which field from 'planetary_system' do you want to use? name
==========Planet==========
name (String)
surface (Float)
radius (Float)
system (String)
------------------
planetary_system - planetary_system (many-to-one)
Generating relationships with other entities
? Do you want to add a relationship to another entity? No
==========Planet==========
```

Image 2.8.1.: Entity planet 1

```
? Do you want to add a relationship to another entity? No
==========Planet==========
name (String)
surface (Float)
radius (Float)
system (String)
------------------
planetary_system - planetary_system (many-to-one)
? Do you want to use a Data Transfer Object (DTO)? No, use the entity directly
? Do you want to use separate service class for your business logic? No, the REST controller should use the repository directly
? Do you want pagination on your entity? Yes, with pagination links
Everything is configured, generating the entity...
```

Image 2.8.2.: Entity planet 2

3. Checking if everything works on localhost

3.1. Importing the project to IntelliJ IDEA 15

You've been working hard and now you have a complete app ready to work! Good job! Let's run it on localhost! I'm going to use IntelliJ IDEA 15, but as I said on the first part of this tutorial, you can find many more ways to edit these files (another IDE's) and to run the application.

Follow the next steps:

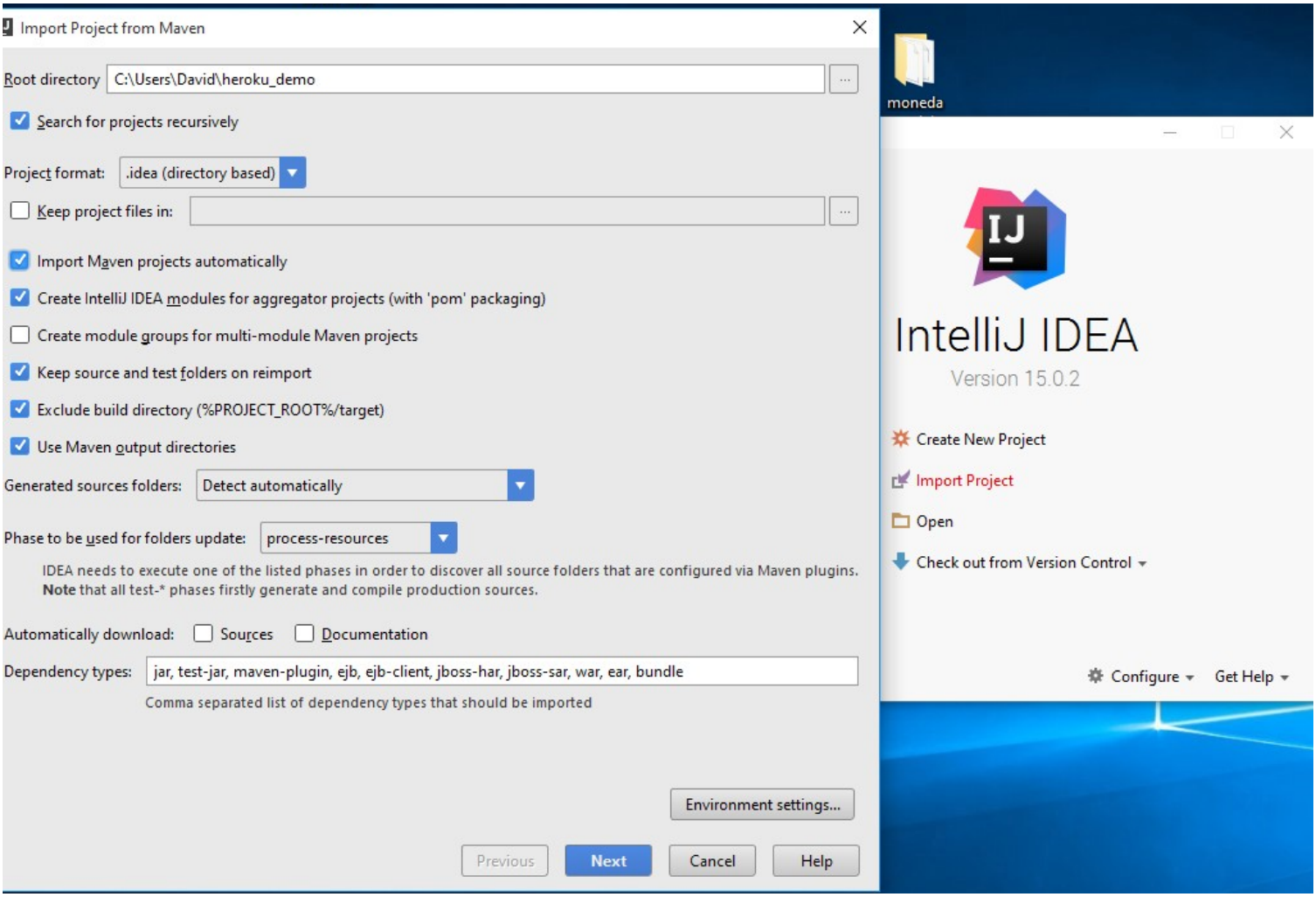

Image 3.1.

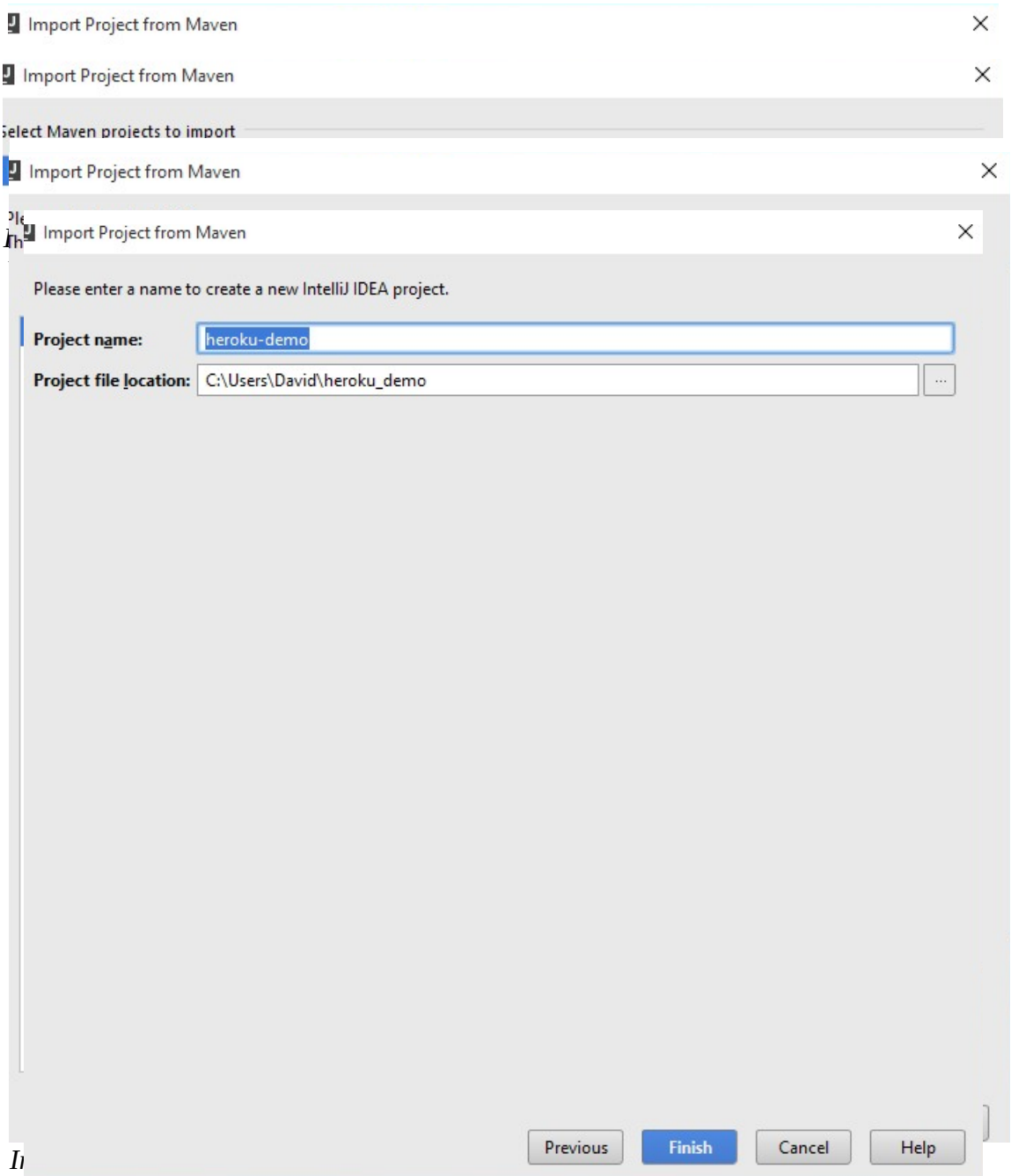

Image 3.5.

3.2. Prepare the database: pgAdmin III

Now you should have your project perfectly imported to IntelliJ IDEA 15. What do we have to do now? The answer is: prepare our database.

As I said on the first chapter of the tutorial, we're going to use pgAdmin III, because it's an easy tool for using databases and learn about their structure with a graphical interface. Also it's free and works in all the platforms.

Open the pgAdmin III. When you installed PostgreSQL, you registered an user with a password. In my case the name of the admin user was "postgres", which is the default name. That's very important because you have to add a user to every database in order to connect with it. Let's check out our panel and set up our database.

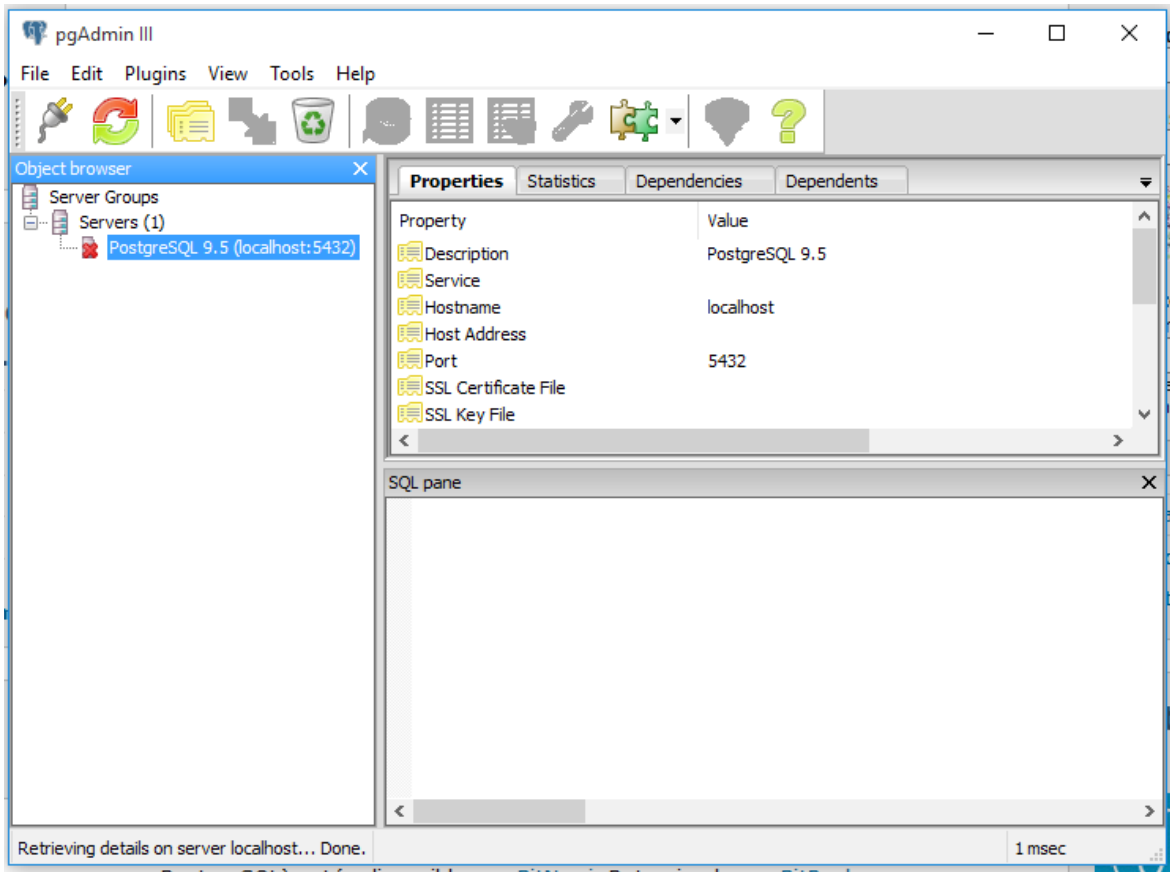

Image 3.6.

There on the left you have your servers. In my case I only have that one: the localhost one. Right click over it and select “connect” button (*Image 3.7.*), and then insert your password.

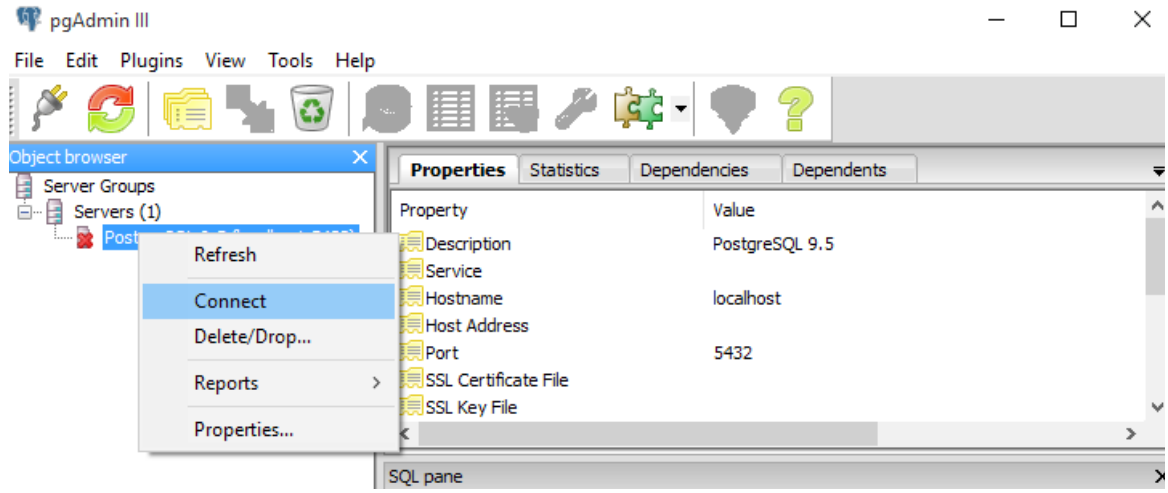

Image 3.7.

After that you'll see how your panel gets filled with some new things, but what you need now is the “Databases” part. Right click over it and select “New database...” (*Image 3.8.*).

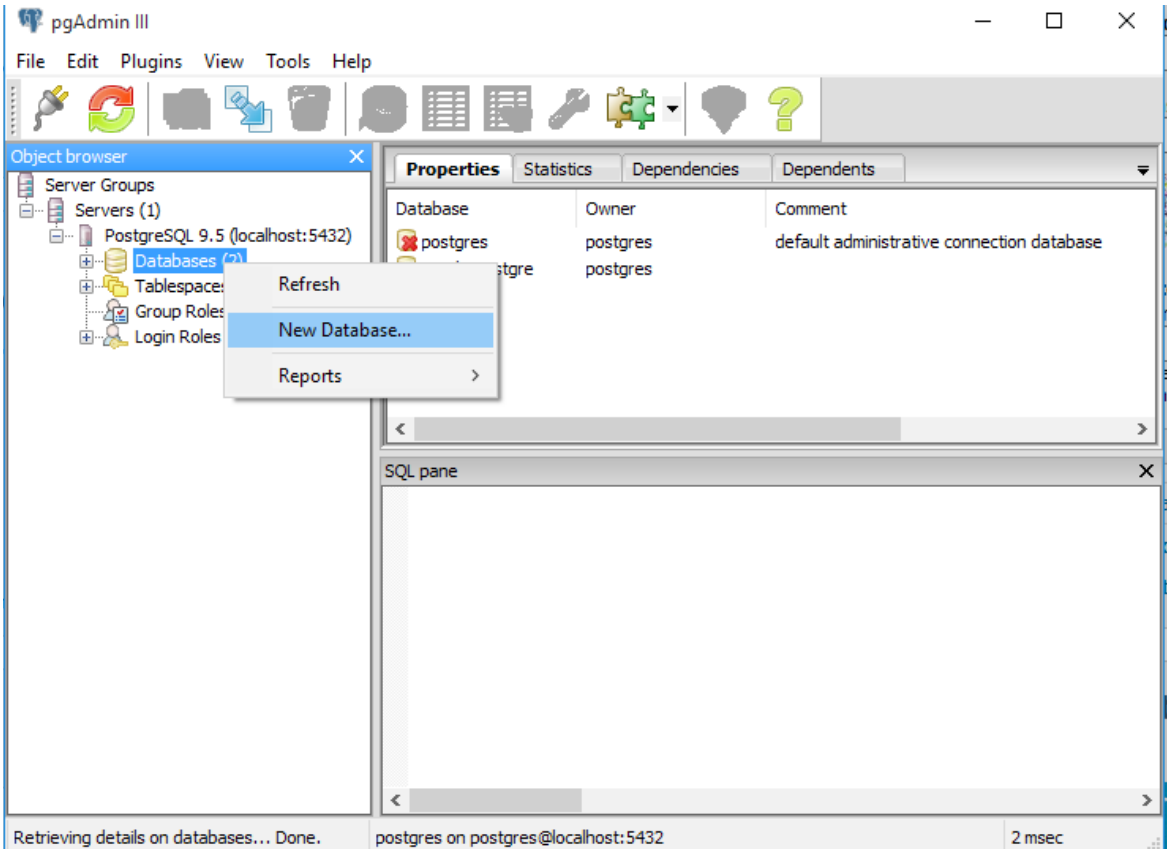

Image 3.8.

Make sure you have a user added to your database (*Image 3.9*) and create it with the name “planets”. The rest of the options will keep their default values.

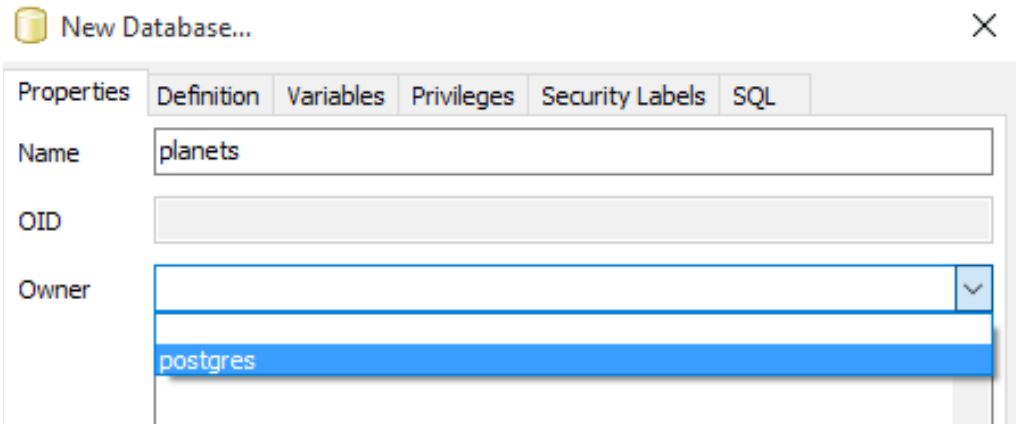

Image 3.9.: That's my admin user, yours can be named another way

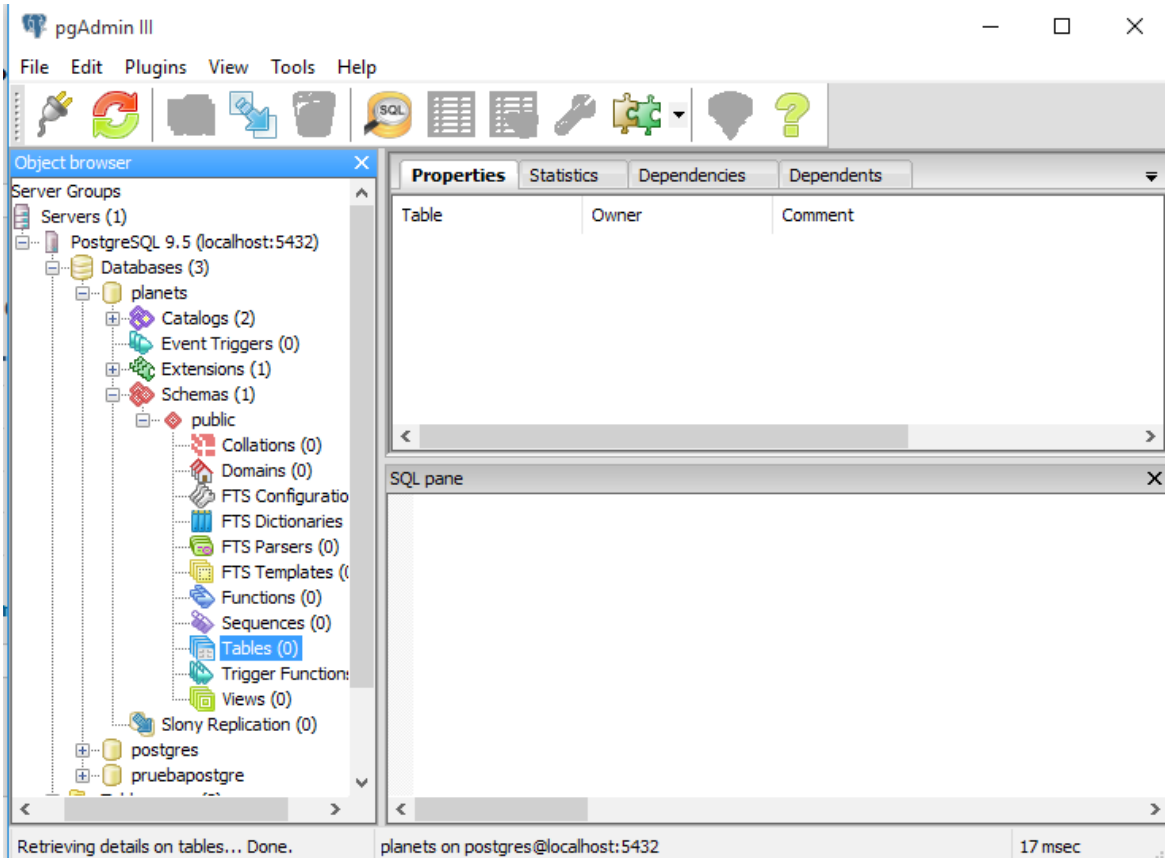

Image 3.10.

There you have it, your “planets” database should be connected and running. Take a look where your tables are located (*Image 3.10*): it's empty now but it could be filled with weird tables in just a few minutes. If you think something is not entering in your database you can always check it there!

3.3. Prepare the database: adapting the code

Now that we have our PostgreSQL server running with our own database, we can change the code of the Java files that JHipster have created in the folder where we loaded it on the first step, so we can let our app connect with our database.

That's the step before we run our application and we see how JHipster works, so be patient!

Let's take a look to IntelliJ IDEA 15 first (*Image 3.11*.).

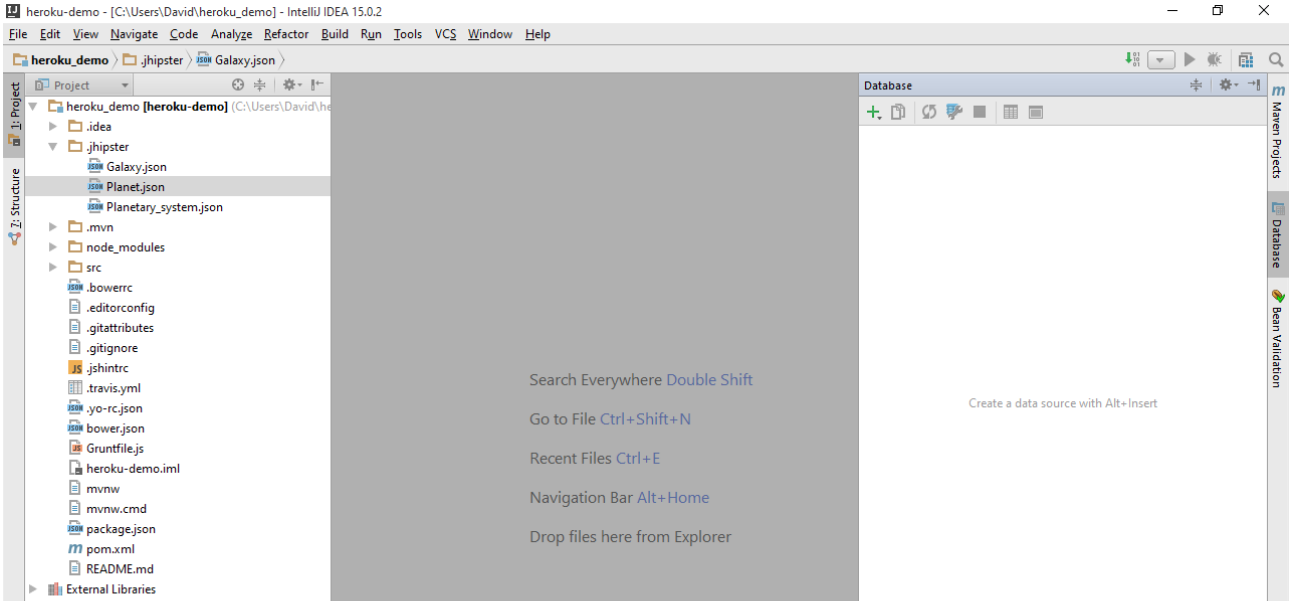

Image 3.11.

On the left we have all our folders. The structures of our entities are in the json files inside the .jhipster directory. On the right we have “Maven Projects”, “Databases” and “Bean Validations”. Click on “Databases” and a panel will be opened. Click then on the green plus symbol in the top left of the panel and a menu with the option of PostgreSQL inside the Data Source part will be opened as well (*Image 3.12*).

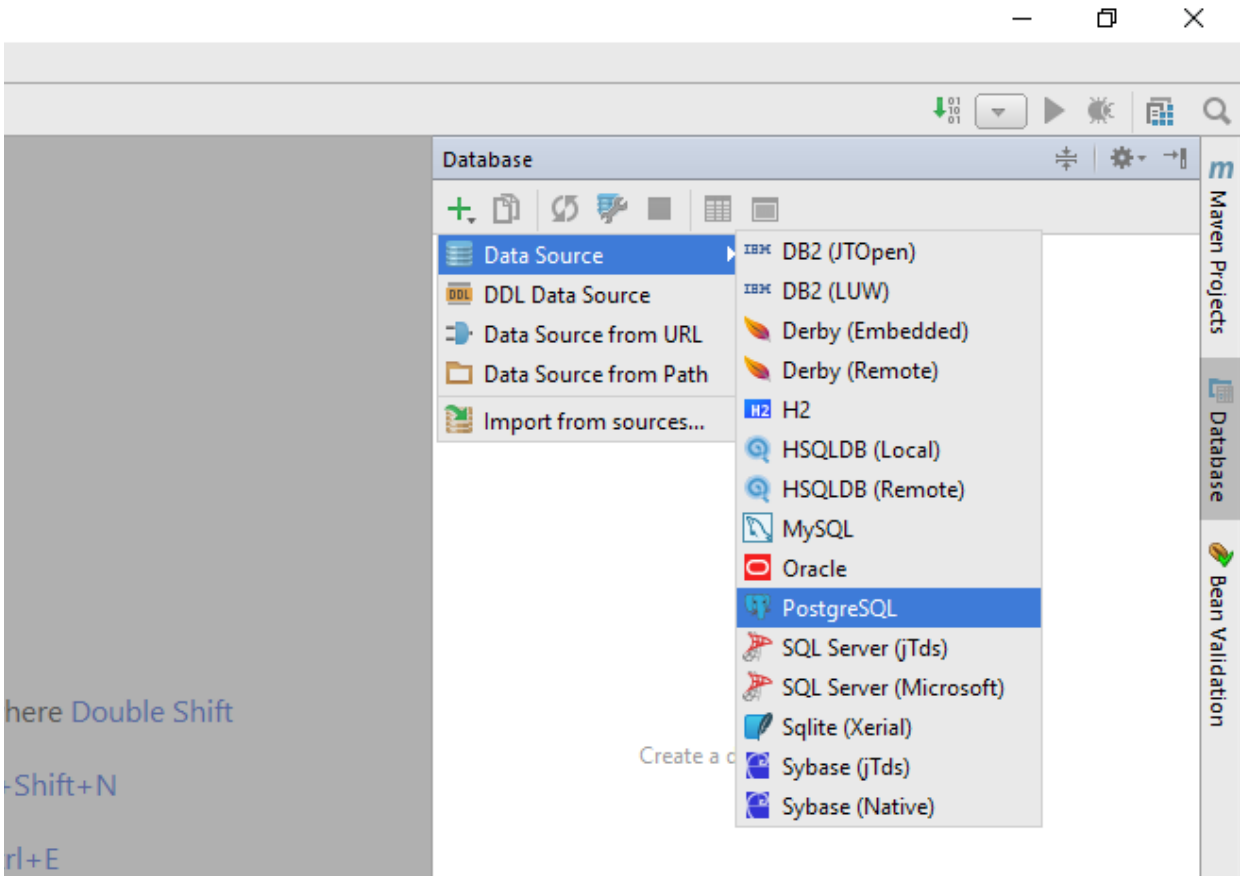

Image 3.12.

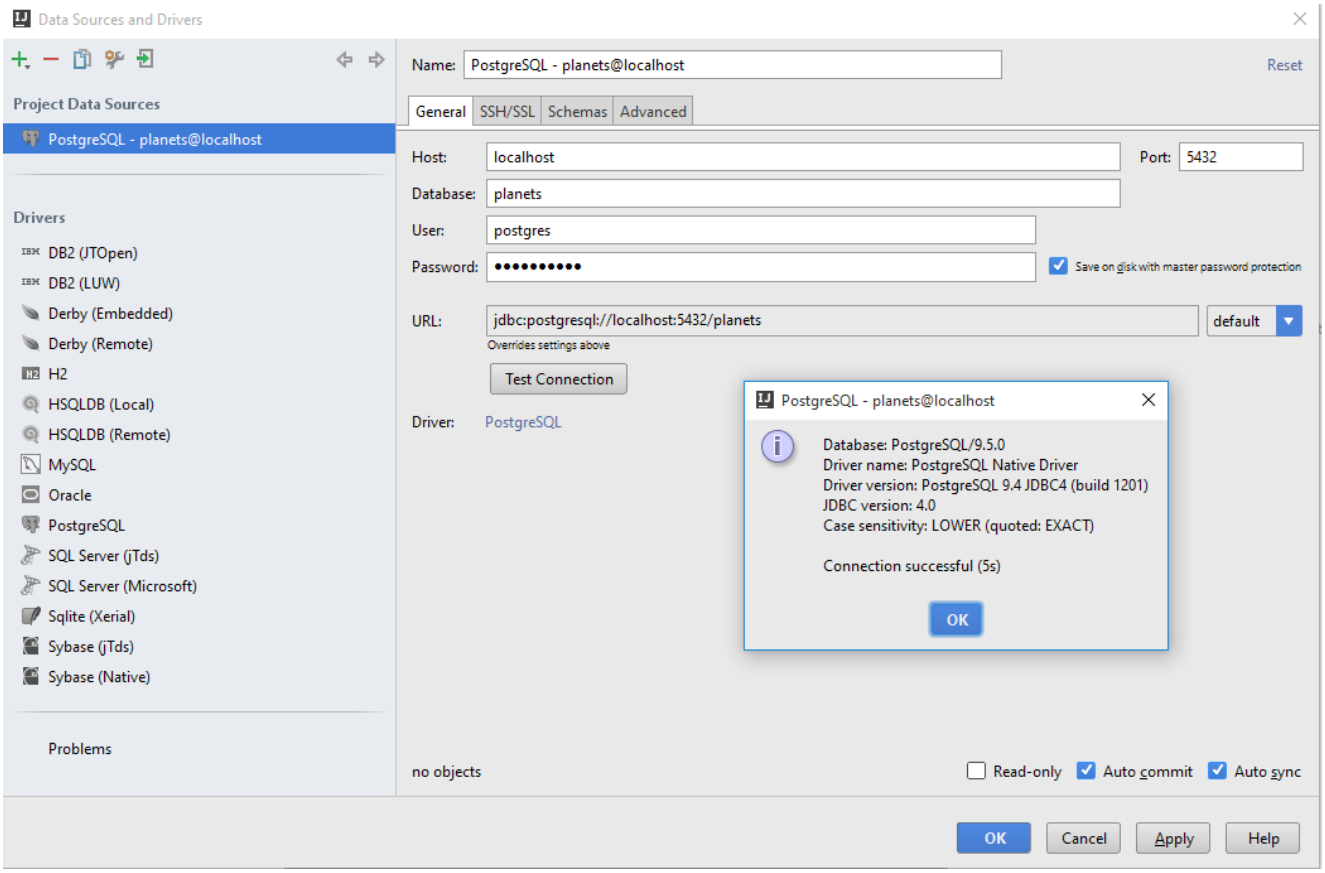

Image 3.13.

Now you have to fill all the fields on the “General” tab and you have to click on “test connection”. The output should be something like what you can see on *Image 3.13*. The host must be localhost because that's what we're trying to achieve, and the port should be the default one by PostgreSQL (5432).

This is technically unnecessary to run our application but it's really useful to test that everything is working. Now let's paste the URL of your database, the username, password and name of the database on the file called application-dev.yml. As you can see (Image 3.14.) the file is already prepared to be filled with the information from your database.

On the screenshot of my application-dev.yml you can see I don't use quotation marks for strings like the url, but if for example your username or your password starts with a symbol, you'll have to use them. Don't forget about it because it can make your app crash before it starts.

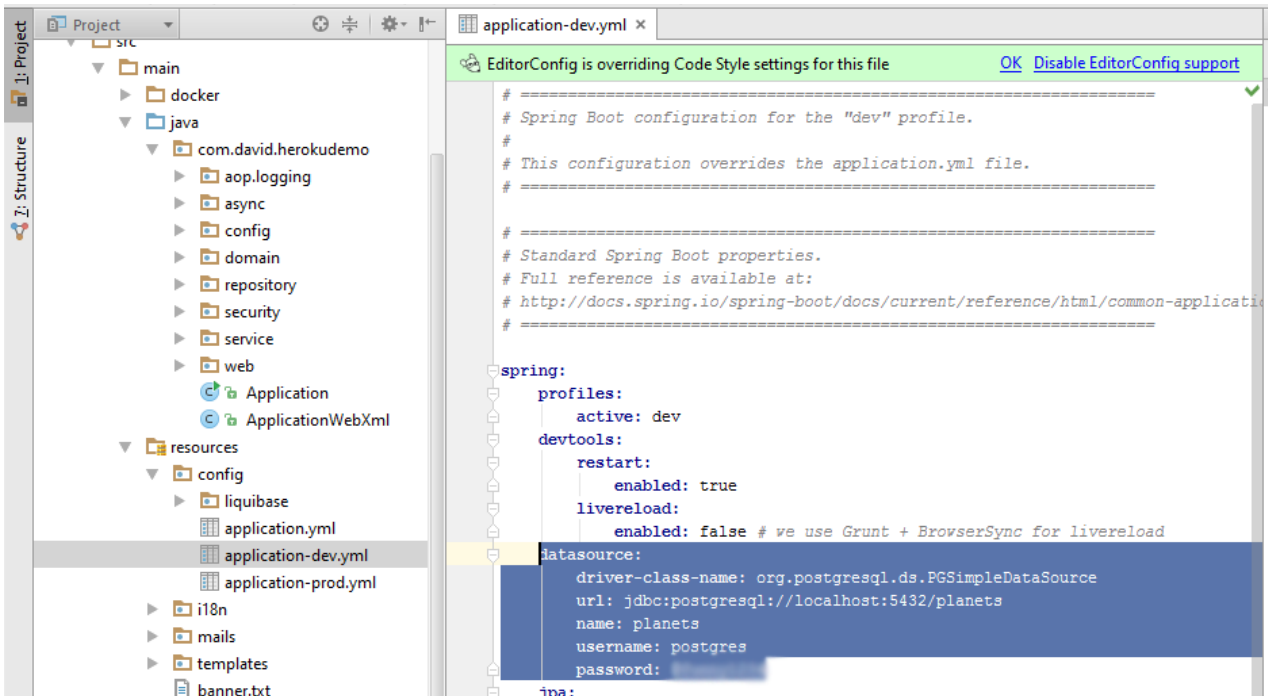

Image 3.14.

And that's all, now let's see what happens if we run it! The main of your application it's located in src>main>java>[com.name.app]>Application.java

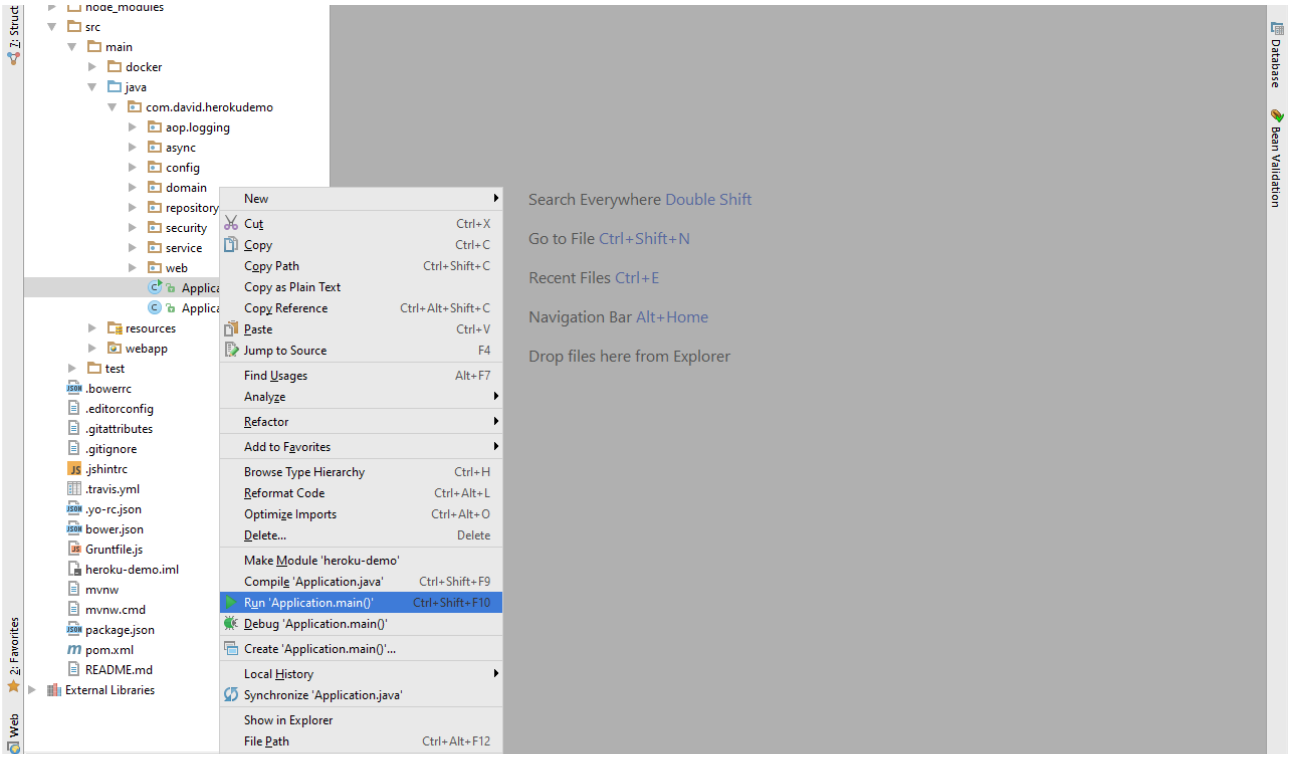

Image 3.15.

Image 3.16.

And there it is, your JHipster finally running on localhost and connected to your PostgreSQL database. Enter http://localhost:8080/ in your browser and start to play with it!

Friendly reminder: you can sign in there as an administrator with the username “admin” and the password “admin”. If you can sign in it means that you're accessing to the database properly.

If you want to know a bit more about the dashboard JHipster have created there for you, go to the Annex I of this tutorial.

4. Deploy to Heroku

4.1. Deploying the app

Now that everything is working, you have to upload your creation and share it with the world. Sometimes localhost isn't enough, so Heroku helps you providing a small space on their servers, and a small amount of energy to run your applications. It's not much but it's enough for us.

Before you deploy it, you have to log in through the console using the next command:

`heroku login`

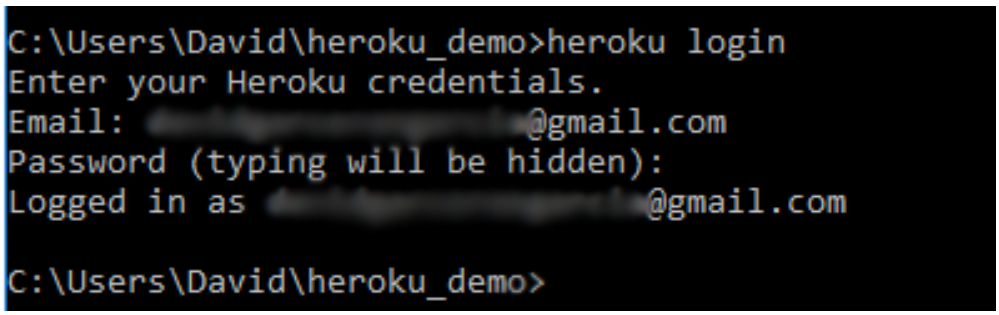

Image 4.1.

And after you insert your e-mail and your password, go to the folder where you have your project (inside the command console, of course) and type the other important command:

`yo jhipster:heroku`

Select your configuration and it will be uploaded to the heroku subdomain you insert when it asks for the name to deploy as (*Image 4.2*). If the name is already in use it will give you a random one.

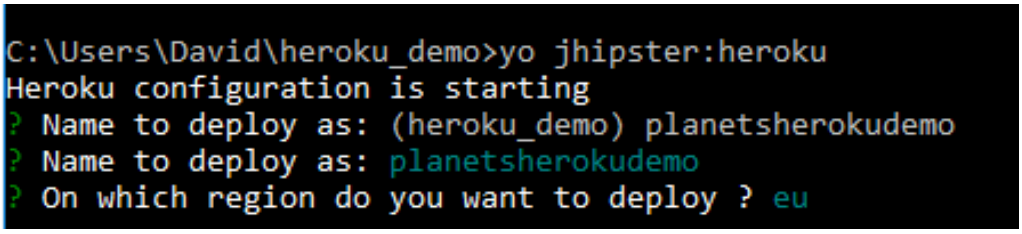

Image 4.2

When it finishes you can use `heroku open` to open the browser with the address of your app in Heroku, and if it something goes wrong after uploading it, you can check it on `heroku logs`. If you use the command `heroku logs –tail` you'll see the logs as they generate.

4.2. Adding new contents

I'm sure you'll want to change something on your applications, like adding new code or updating it.

How can we do that? It's easy, first you have to run the command

`mvn package -Pprod -DskipTests`

that will prepare your package to be uploaded to Heroku, and then you have to upload it with

```
heroku deploy:jar --jar target/*.war
```

that will deploy the new version of your application.

4.5. What's in my database?

Enter on your Heroku panel in the website (https://dashboard.heroku.com/), look for the name of your application and click on something like the *Image 4.3.*

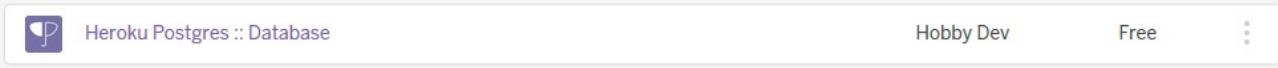

Image 4.3.

Clicking there you'll access to all the information of your database (*Image 4.4.*).

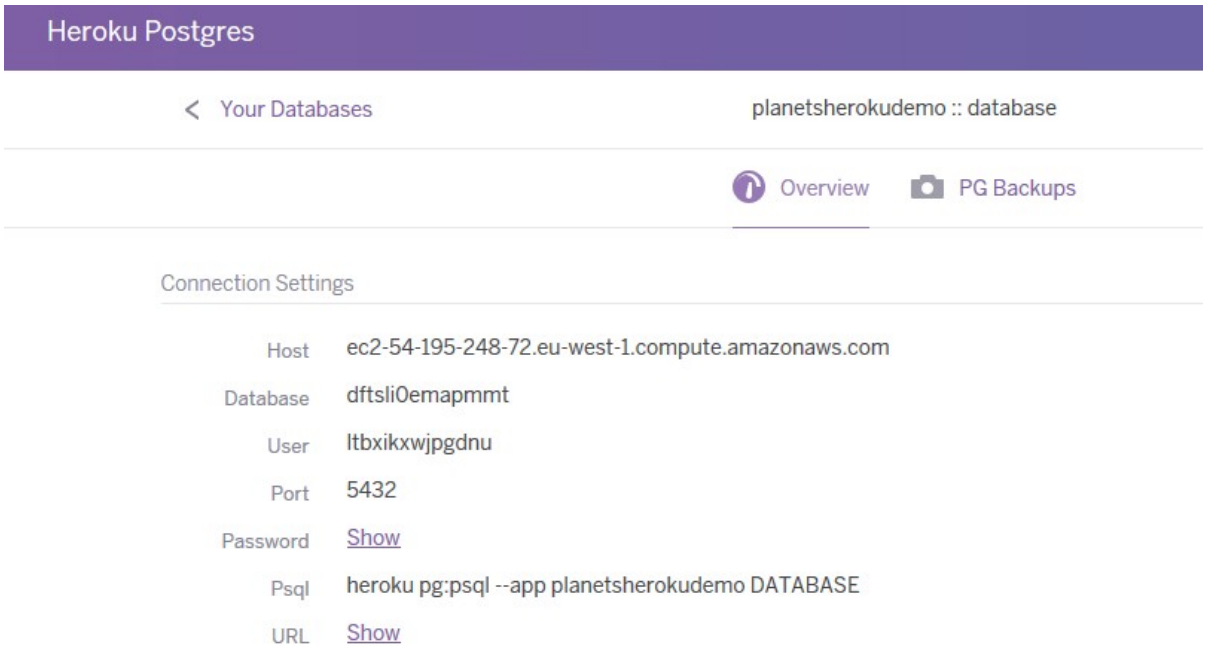

Image 4.4.

So you can add a connection to your pgAdmin III panel! It's pretty easy to do that, just go to File > Add Server… and fill the fields as you can see on the next images.

Check out where I write the name of my database (*Image 4.6.*): on the “Maintenance DB” field. And don't forget about adding the SSL (*Image 4.7.*)!

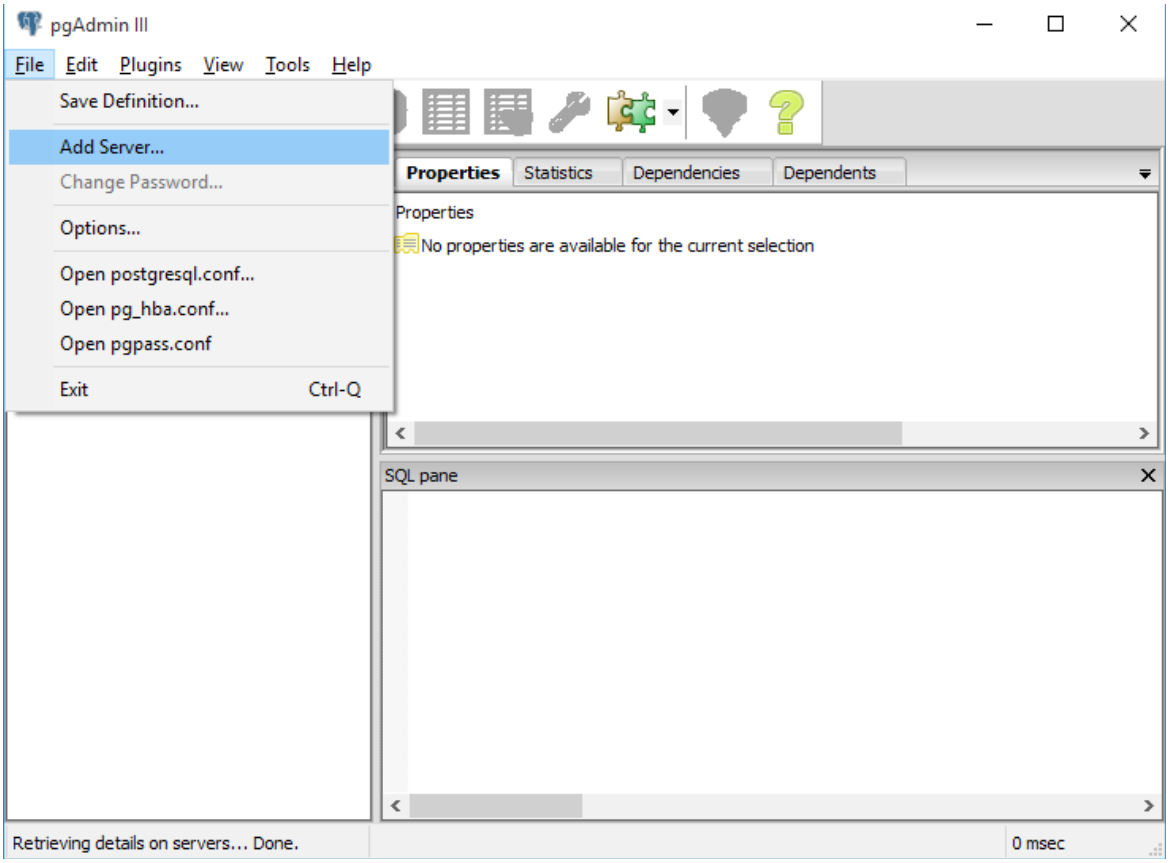
pgAdmin III
File
Edit
Plugins
View
Tools
Help
Save Definition...
Add Server...
Change Password...
Options...
Open postgresql.conf...
Open pg_hba.conf...
Open pgpass.conf
Exit
Ctrl-Q
Properties
Statistics
Dependencies
Dependents
Properties
No properties are available for the current selection
SQL pane
Retrieving details on servers... Done.
0 msec

Image 4.5.

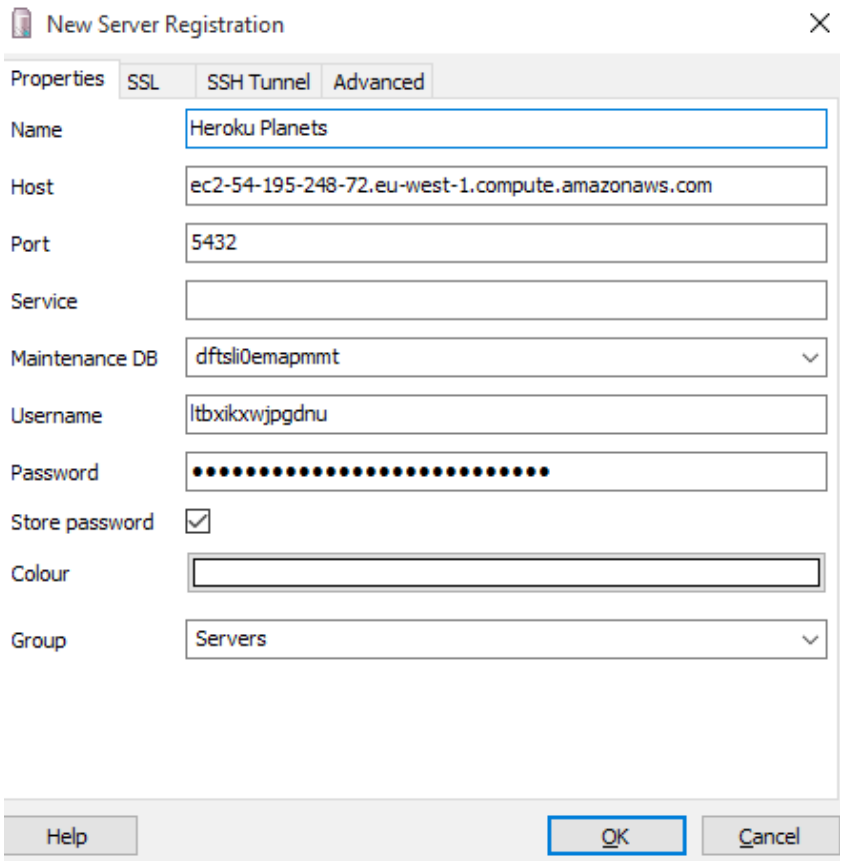
New Server Registration
Properties
SSL
SSH Tunnel
Advanced
Name
Heroku Planets
Host
ec2-54-195-248-72.eu-west-1.compute.amazonaws.com
Port
5432
Service
Maintenance DB
dftsli0emapmmt
Username
ltbxikxwjpgdnu
Password
Store password
Colour
Group
Servers
Help
OK
Cancel

Image 4.6.

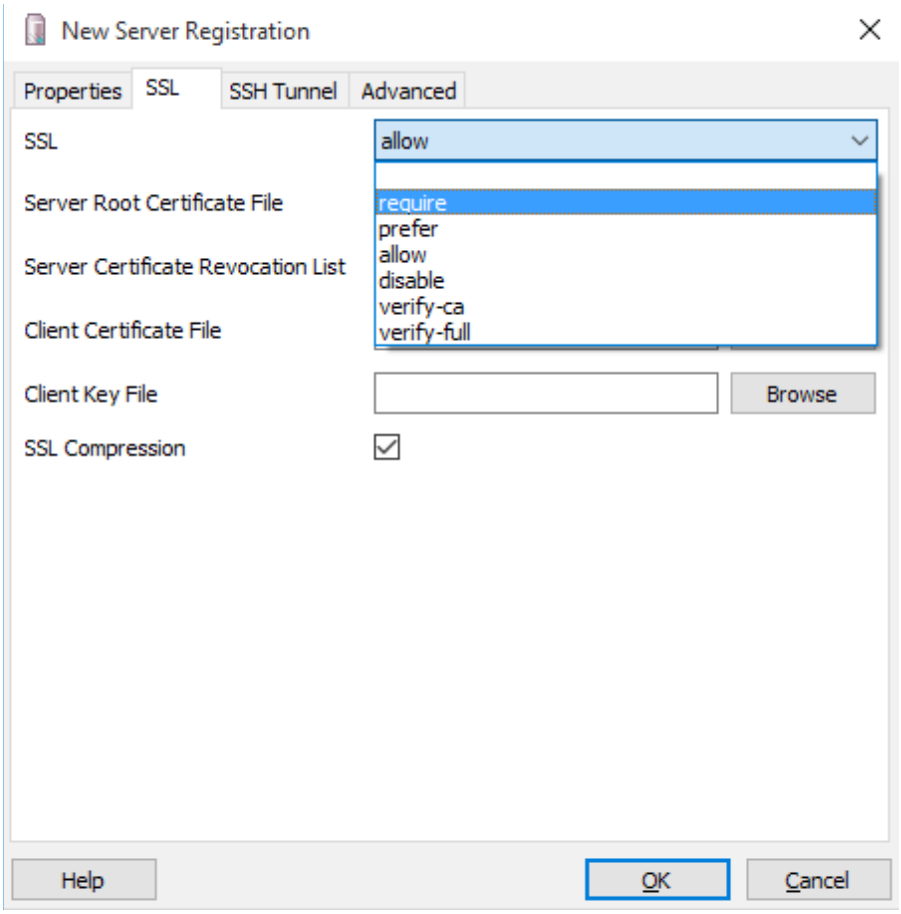

Image 4.7.

If you have followed these steps you should have your database connected on pgAdmin III, but there's a problem: you're seeing all the databases of that server, so let's fix it.

Disconnect the server (*Image 4.8.*), go to the properties of that server and on the advanced properties tab, in the field called “DB Restriction” you have to add the name of your database with single quotes (*Image 4.9.*).

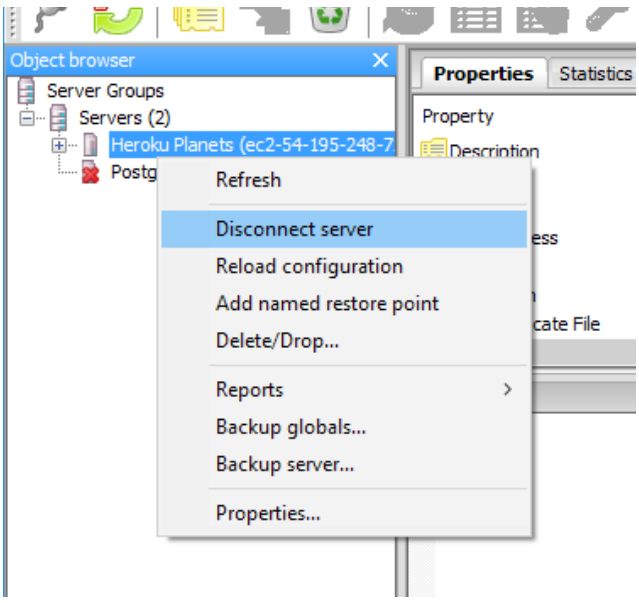

Image 4.8.

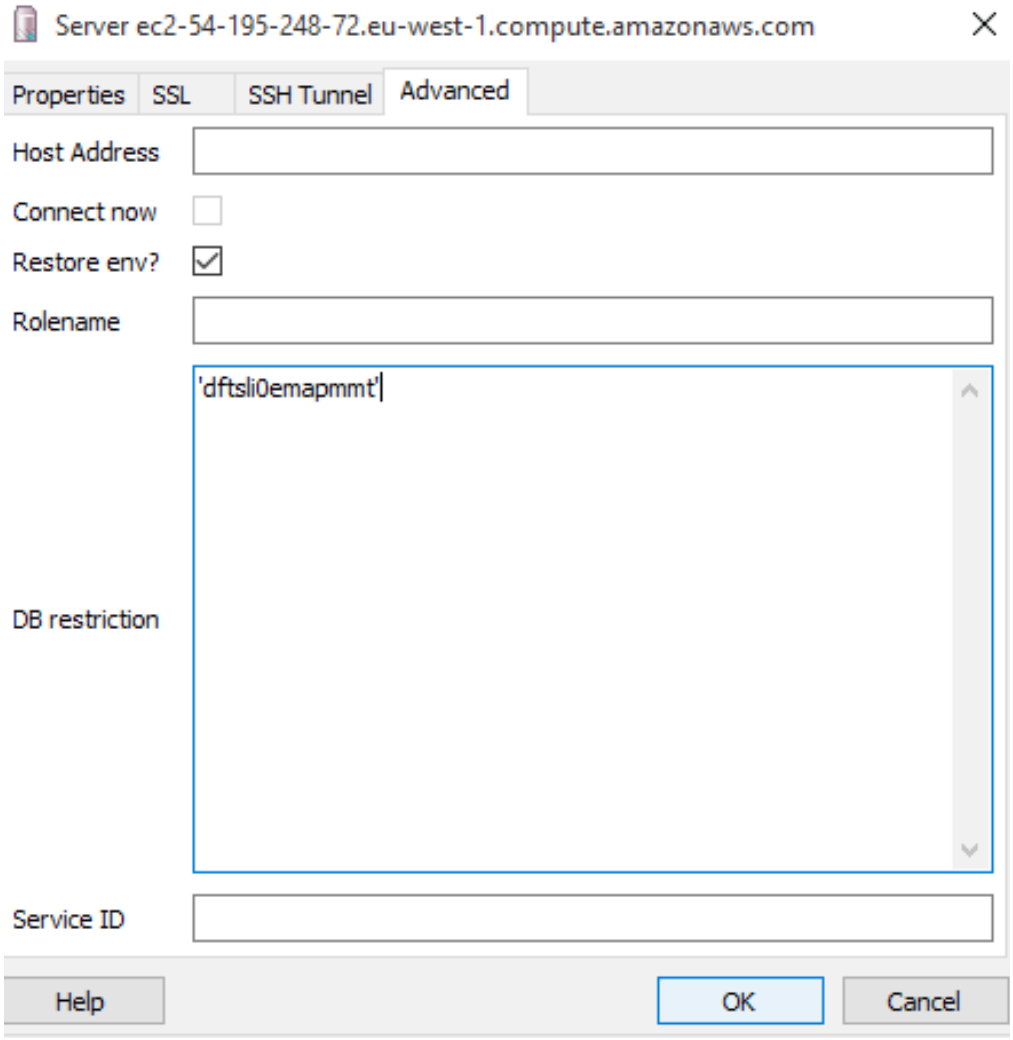

Image 4.9.

Et voilà! You're ready to see what have we done here!

4.4. The live demo

You can see and use the live demo of this tutorial on http://planetsherokudemo.herokuapp.com/.

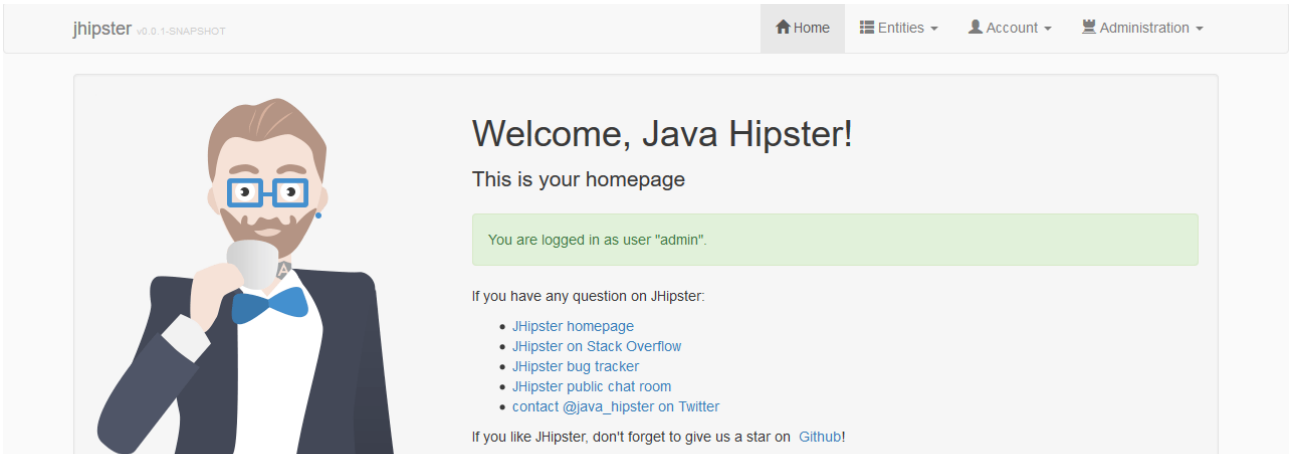

Image 4.10.: The first page of the app

5. ANNEX I: Introduction to the JHipster dashboard

Once we enter in our JHipster application, we discover that it has created our front-end with Angular and Bootstrap, and it is really useful, more than what you can imagine. You can add data to your entities, you can edit, delete, check the relationships, etc. in a very visual way. It's so easy to manage everything, so I'm sure you'll learn very quickly. Let's start by the log in!

JHipster provides you with two different users to try two different kinds of permissions: admin and user. We will use admin because we want to see the complete system. Log in with the user “admin” and the password “admin” and add a galaxy to the database!

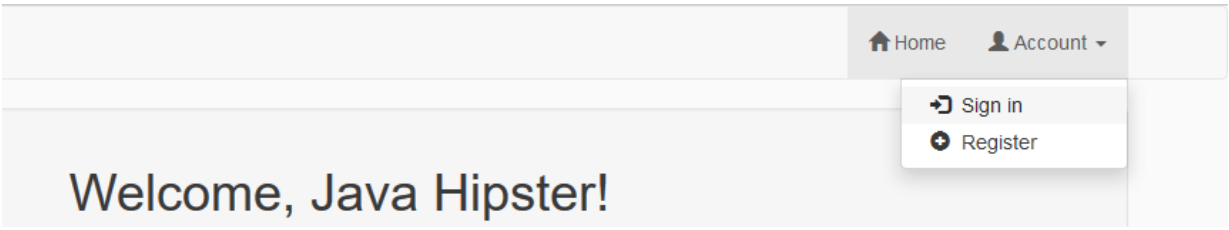

Image 5.1.

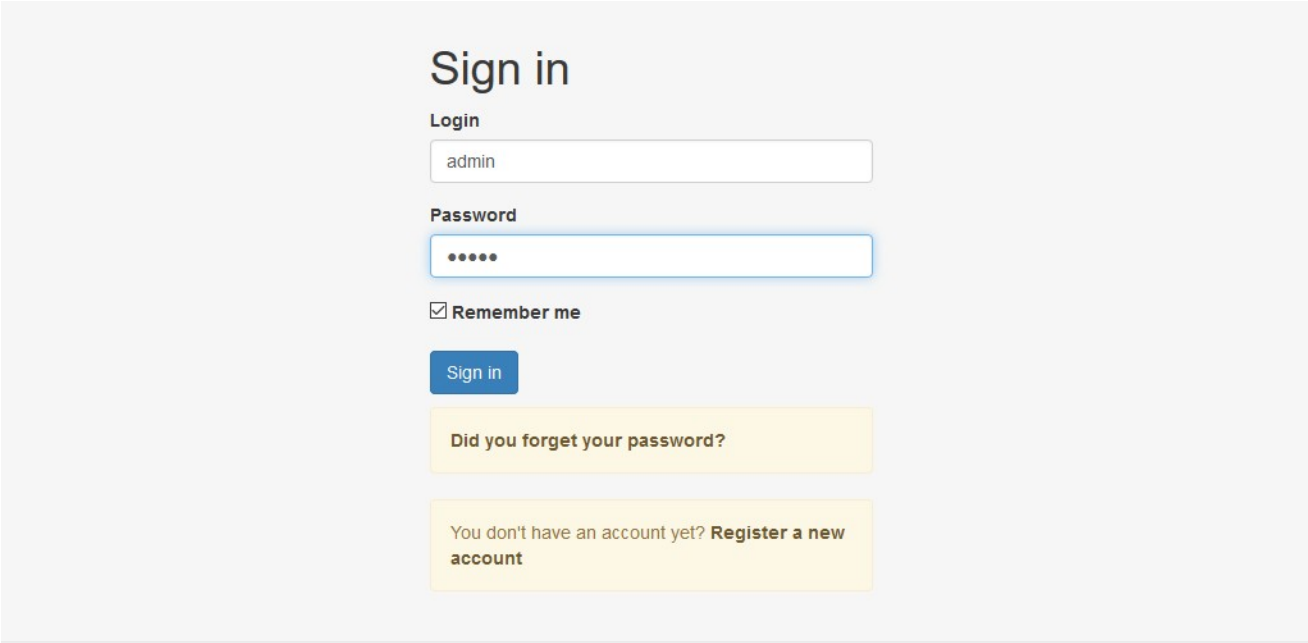

Image 5.2.

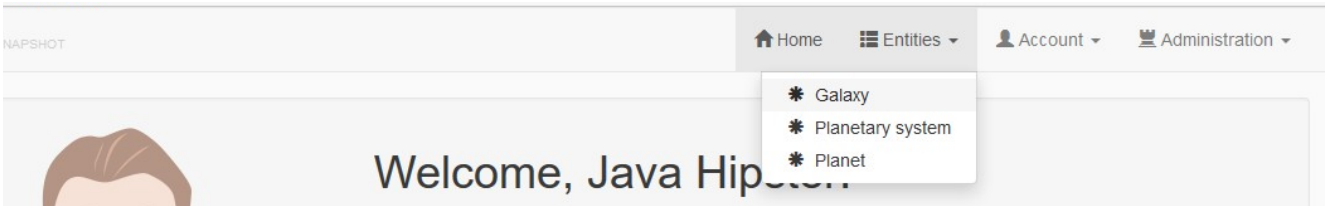

Image 5.3.

Image 5.4.

Your database is completely empty, so you have to add one galaxy. And yes, we all know that the plural of “galaxy” is “galaxies”, not “galaxys”, but it's obvious that JHipster just added an “s” to the end of the word. You can change that on the files it has created. In fact you can change every string that you see on the HTML part! Check out the Annex II to see where are these files.

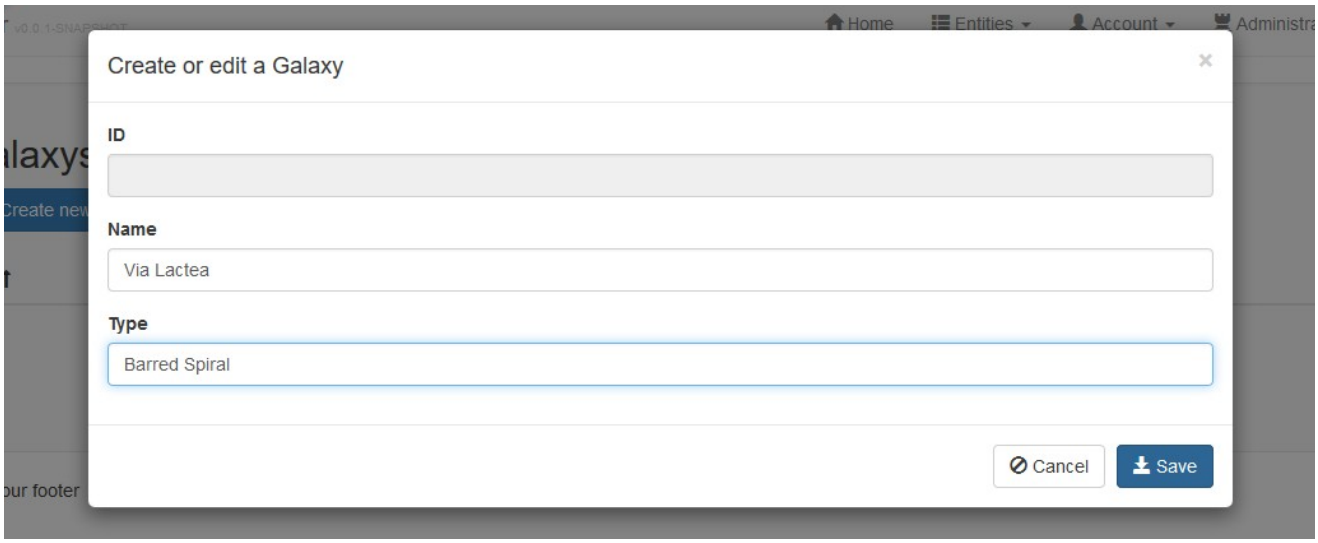

Image 5.5.

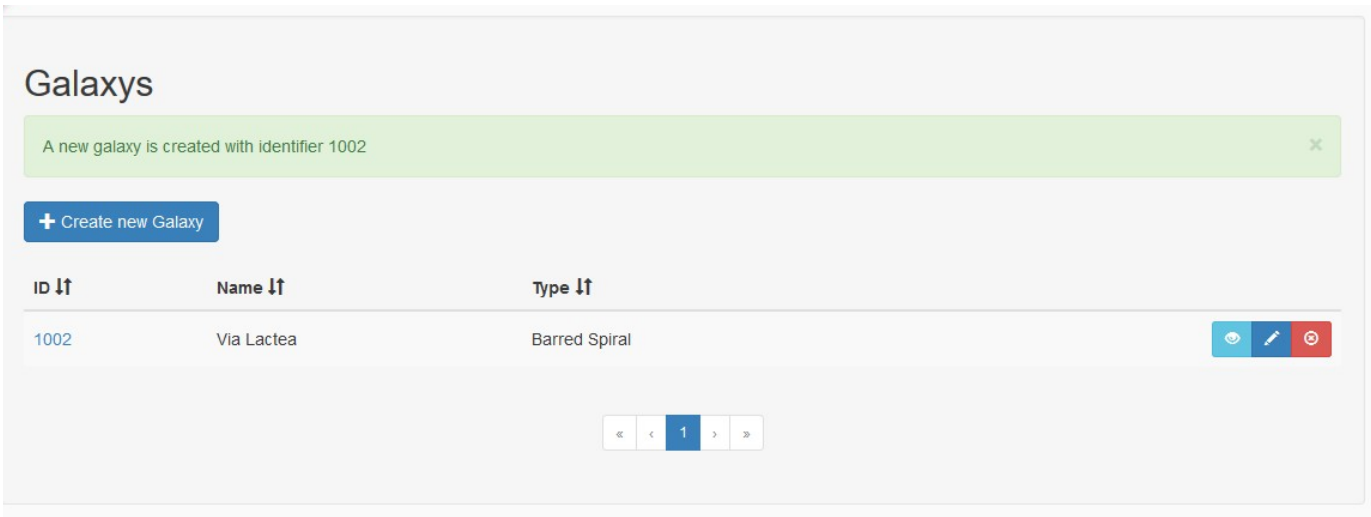

Image 5.6.

Now that you have a galaxy in the system, we can create a planetary system the same way, but do you remember the relationship we added? We asked JHipster to create a relationship between Galaxy and Planetary System, and we said that Angular should take the field “name” of Galaxy, right? So here it is! (*Image 5.7.*)

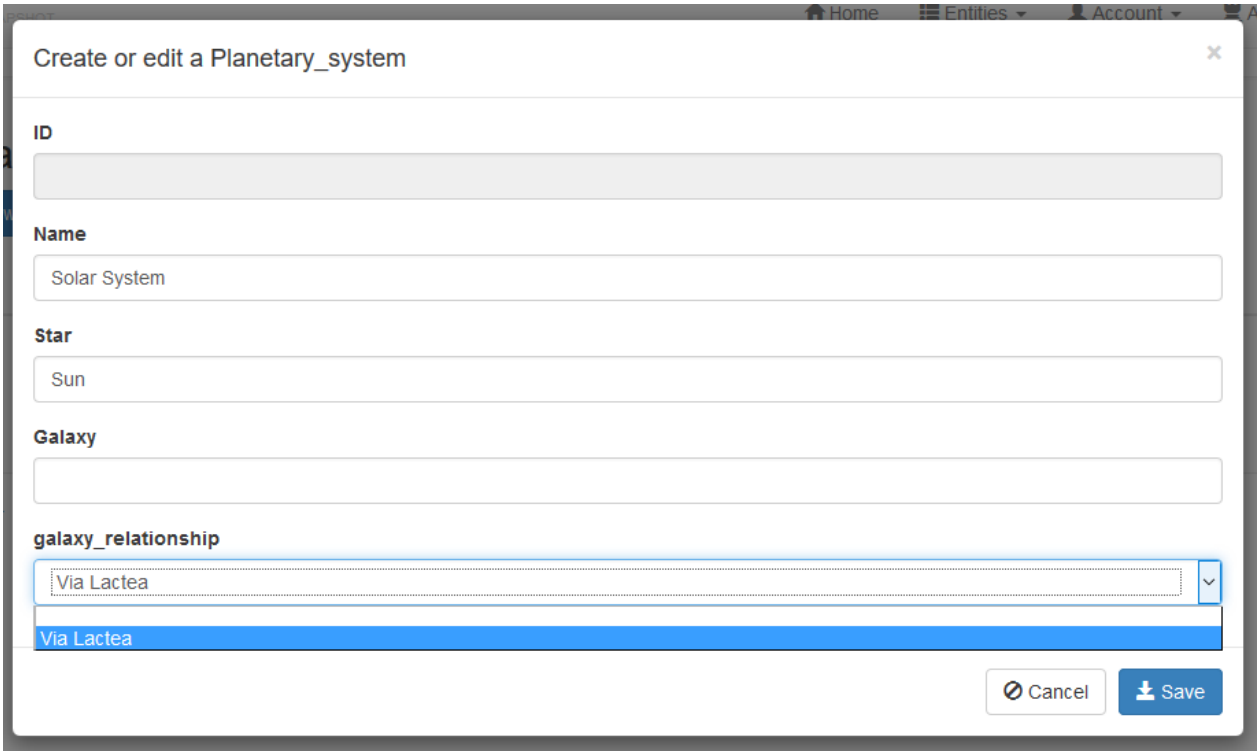

Image 5.7.

As you can imagine, it's the same for planets:

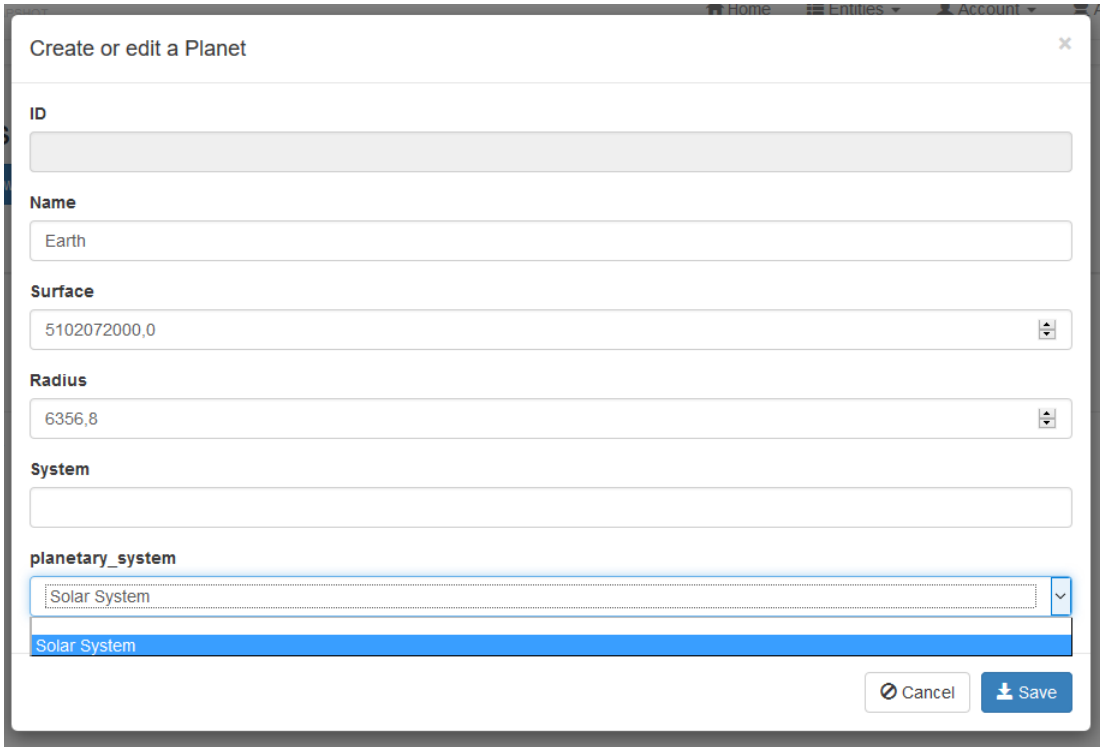

Image 5.8.

6. ANNEX II: Changing the front-end

In the part 2.2 of this tutorial I introduced you to a typical error that you'll have in case you don't pay attention to what you're doing, especially being a beginner. As you can see on the screenshots that follow that warning, I was creating a field that was unnecessary. It has added some code I didn't need, so I had to delete it. Let's find out how to do it with the entity “Planetary System”, I'm sure you can do the same for the other ones if you have the same problem:

On

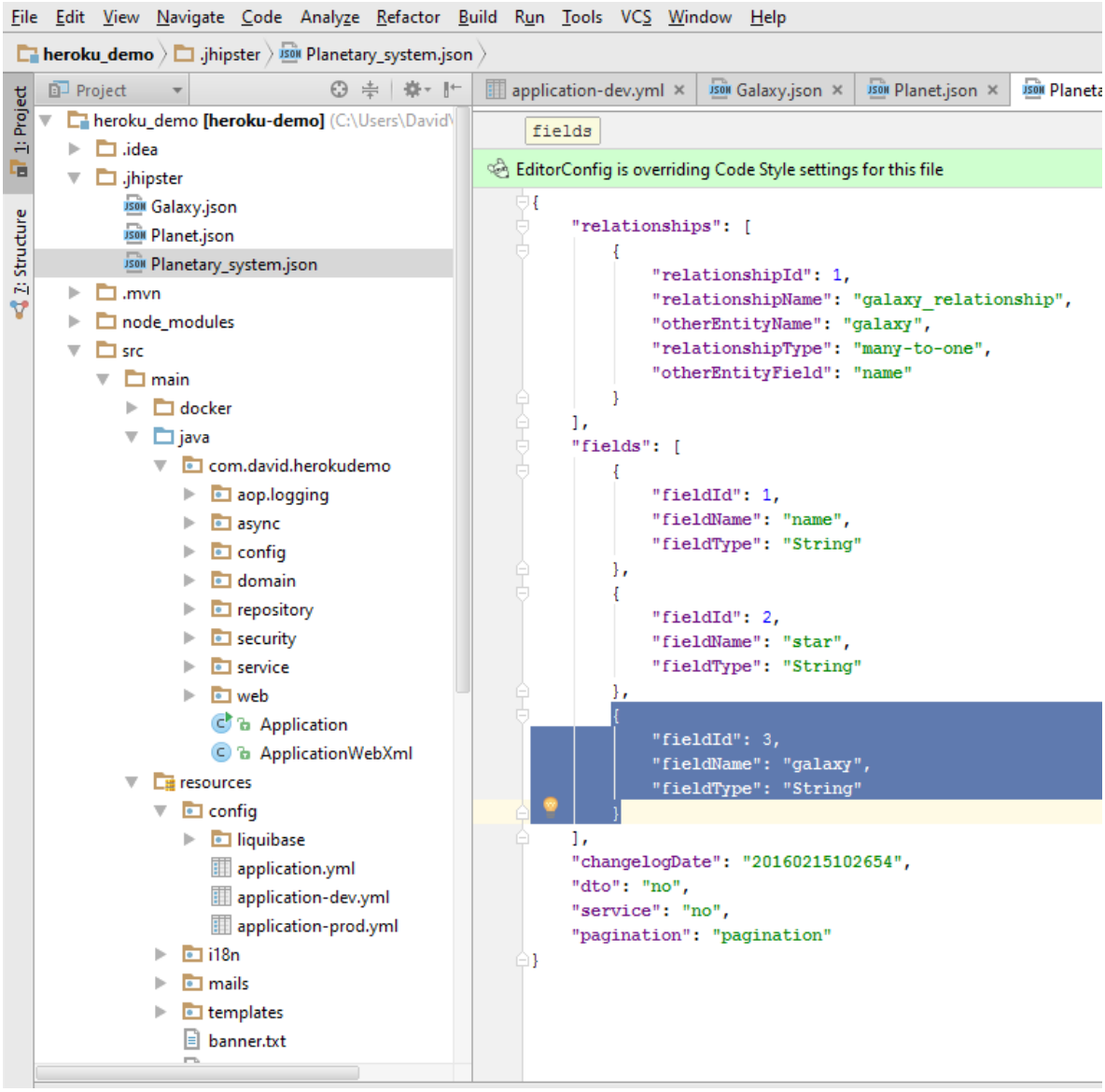

Image 6.1.

the .jhipster folder is where the entities are created and saved. If you want to modify something from them, go there.

I'm deleting the highlighted part: the field with the id 3. On the top of the file you can see that we already have a field created by the relationship that makes the one I'm deleting completely unnecessary.

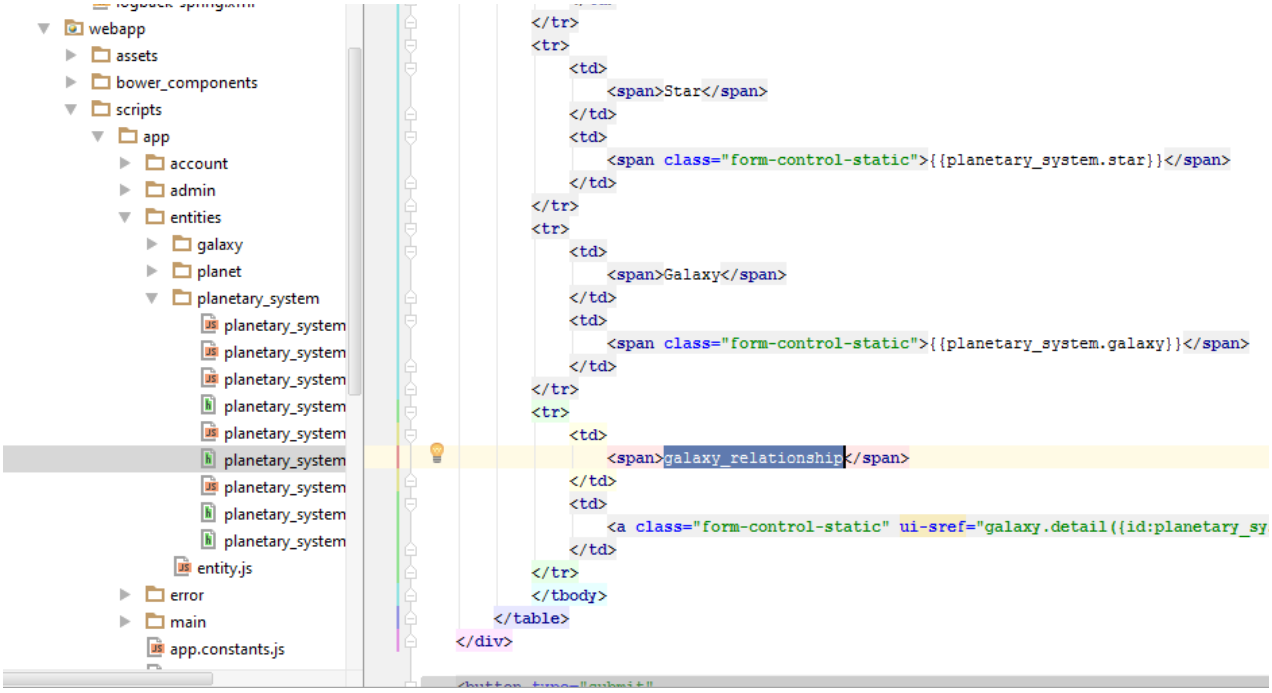

Image 6.2.

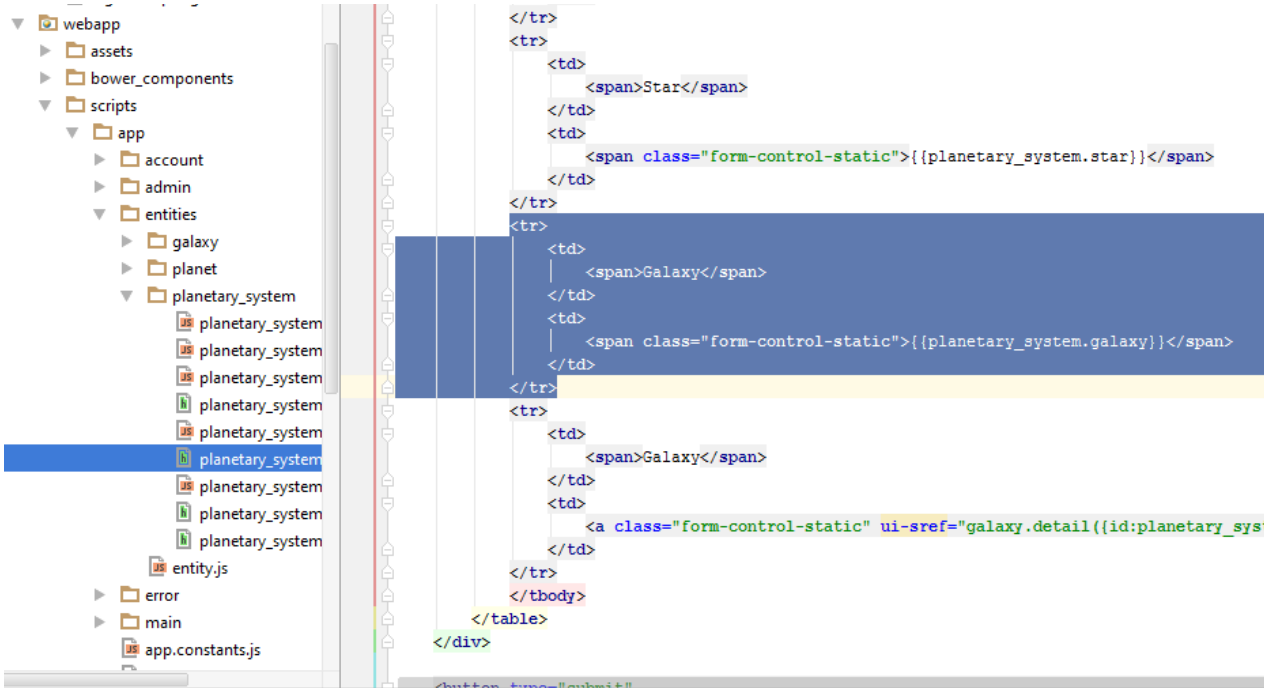

Image 6.3.

Once we change the JSON file, we have to change the HTML so that it's what we can see on the website of our app: the front-end. JHipster creates a lot of folders and files to organize that part. We have to go to the *nameProject/src/main/webapp/scripts/app/entities/nameEntity*. In the file *<nameEntity>-detail.html* I'm going to change the name of the title of the column on the table that is on the HTML (*Image 6.2.*) and that shows you the relationship that we have created before (just because I think that “Galaxy” looks better than “galaxy_relationship”).

We are also going to delete the field that we already deleted on the JSON file, so it doesn't show it

anymore on that page (*Image 6.3.*).

We're going to do the same for the next pages: *<nameEntity>-dialog.html* and *<nameEntity>.html*

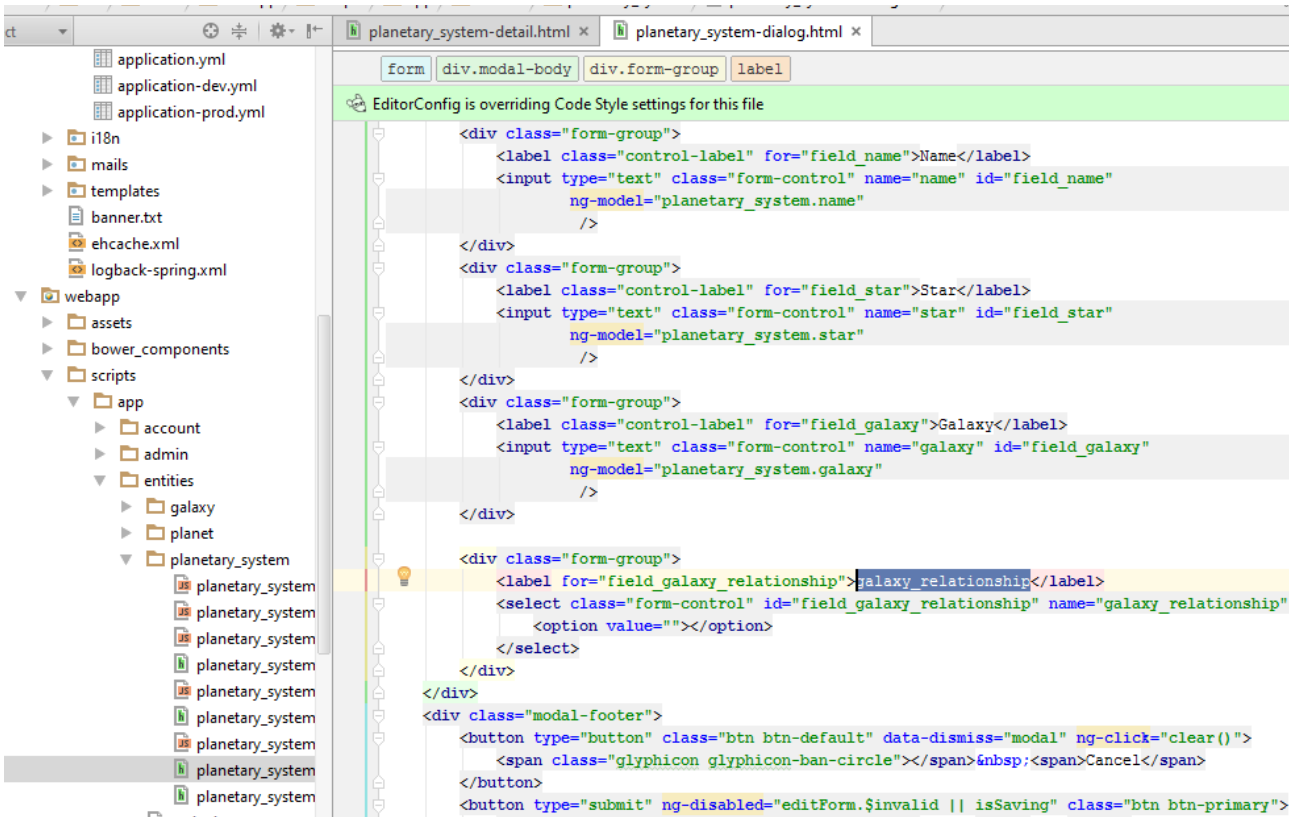

Image 6.4.: Changing a label

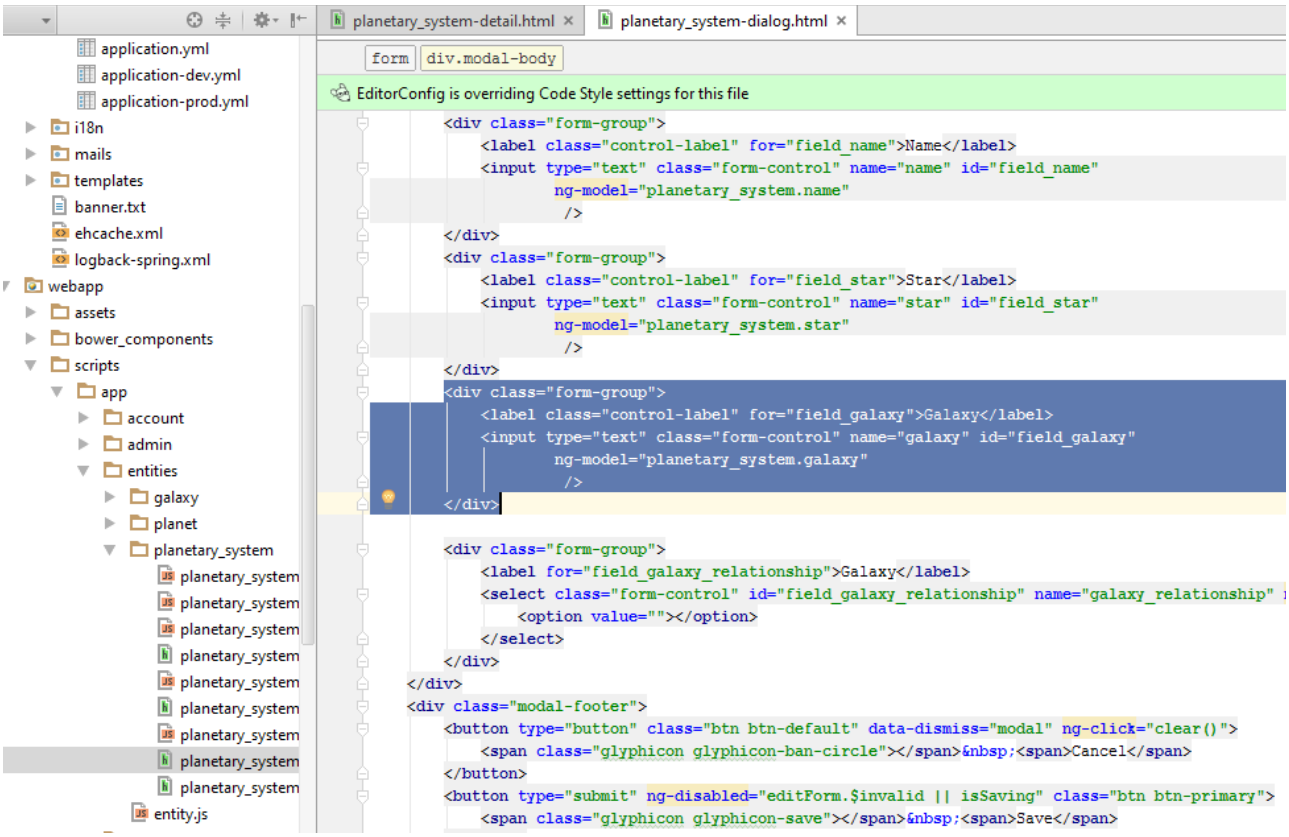

Image 6.5.: Deleting unnecessary code

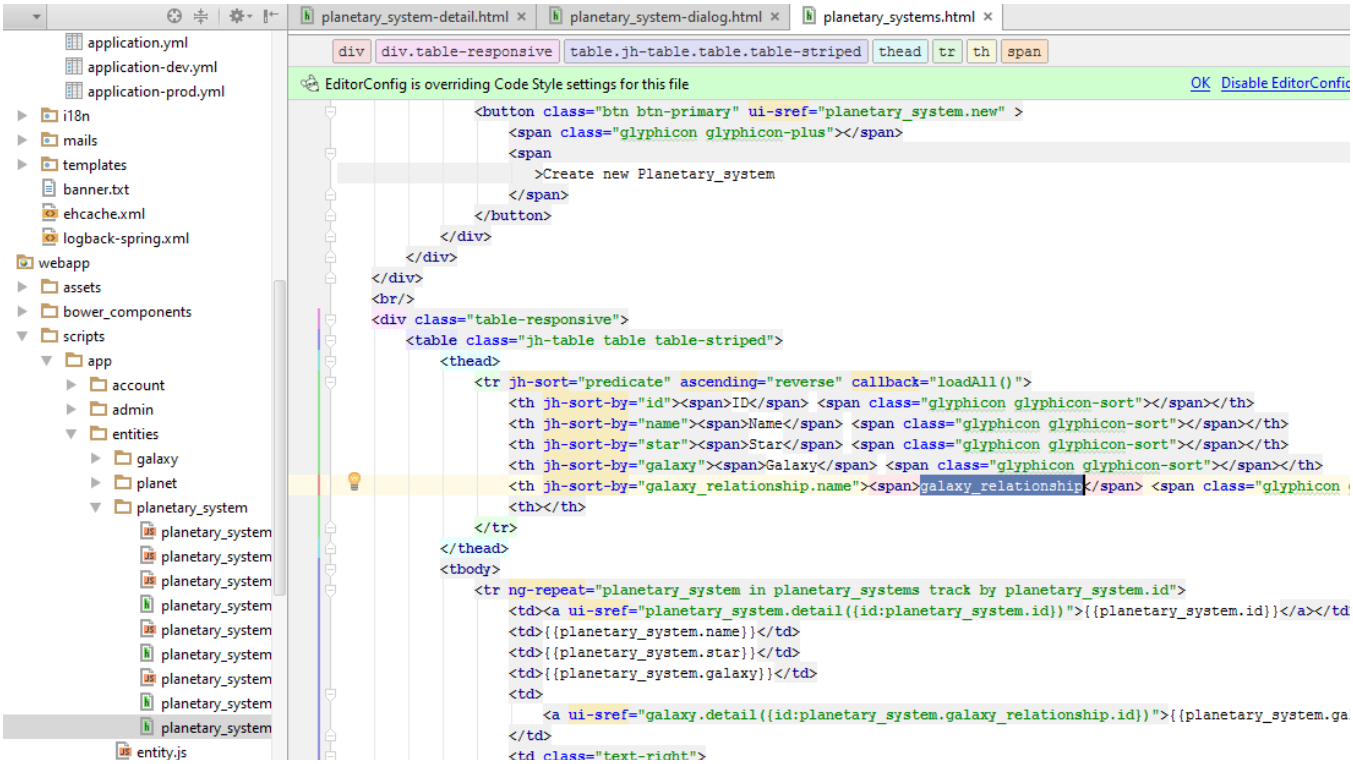

Image 6.6.: Changing a label

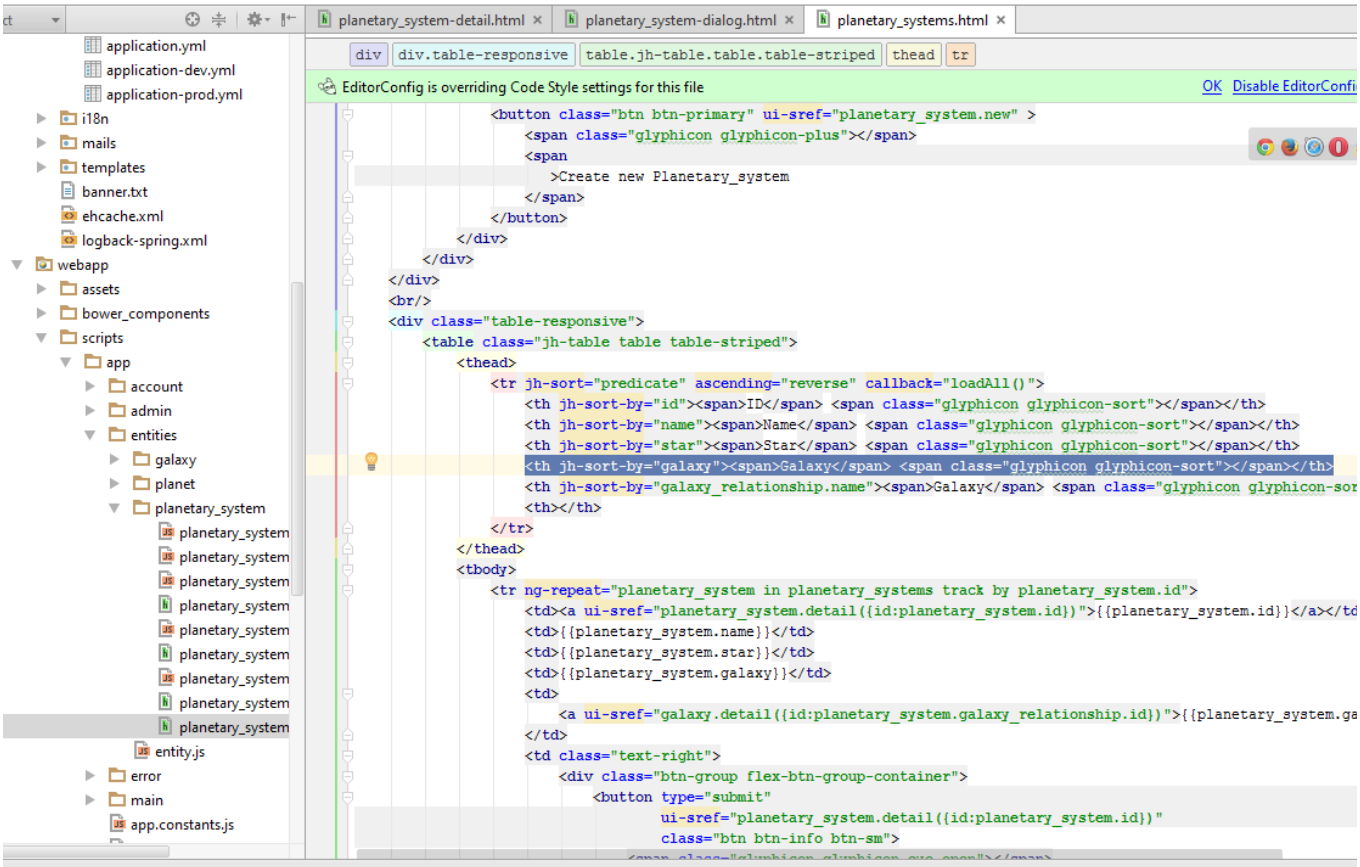

Image 6.7.: Deleting unnecessary code

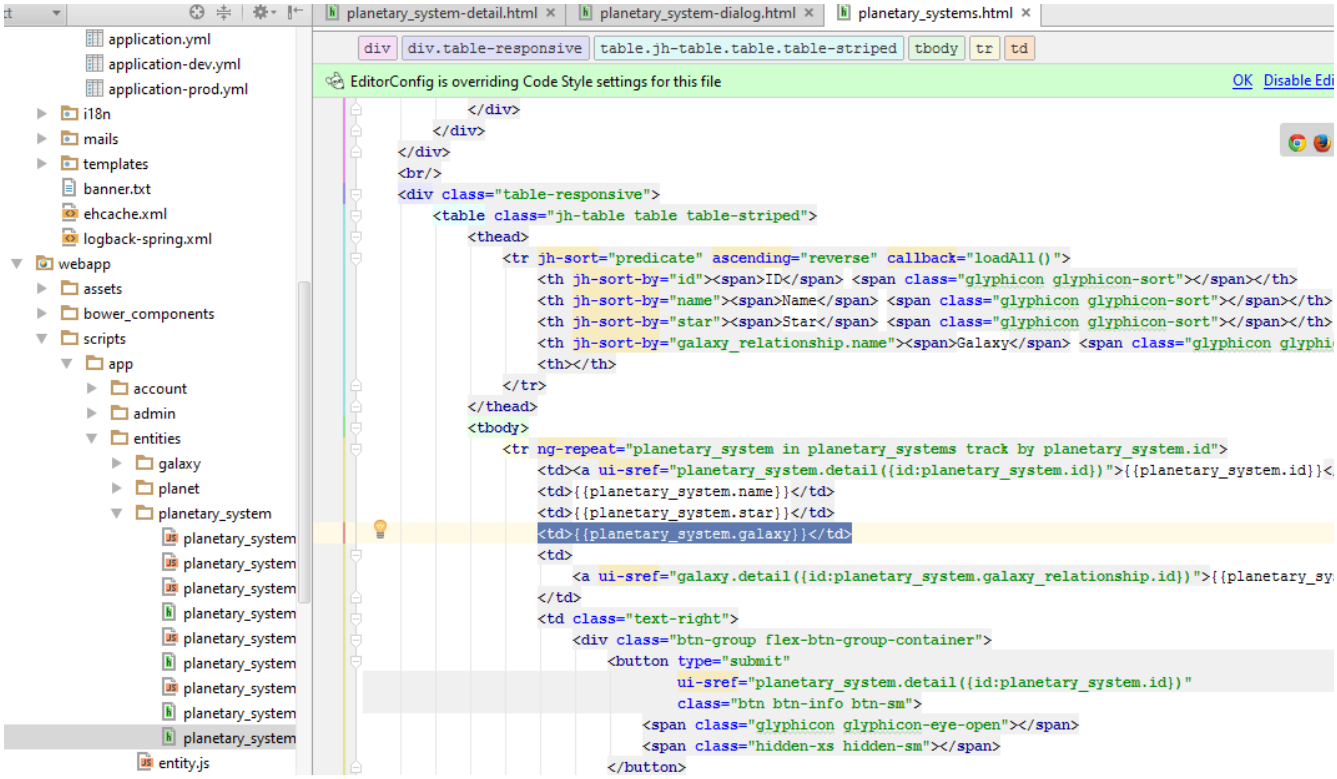

Image 6.8.: Deleting unnecessary code

Once you delete or change all the code that was related to the field you created by error, you're done!

As it has been said on the warning in the part 2.2, in the next version of JHipster (3.0) you won't need to do that because you'll be able to update the entities from the generator.

7. ANNEX III: Versions used

Maybe you don't know this, but most of the software that it's used on this tutorial is updated every few days, and it's really hard to keep it up to date. Also, this software can change a lot, and it's interesting to make a list of all the versions that I have used to do this, hopefully this will help you be aware if you see anything weird.

Here's the list:

- Java: 1.8.0_71
- Maven: 3.3.9
- NPM: 3.3.12
- Node: 5.2.0
- Bower: 1.7.7
- Yeoman: 1.6.0
- Grunt-cli: 0.1.13
- IntelliJ IDEA: 15
- pgAdmin III
- Postgre SQL: 9.5.1
- JHipster generator: 2.27.0
- Heroku
    - -apps: 1.2.4
    - -cli-addons: 0.2.0
    - -fork: 4.1.1
    - -git: 2.4.5
    - -local: 4.1.7
    - -pipelines: 1.0.1
    - -run: 2.9.2
    - -spaces: 2.0.13
    - -status: 2.1.0