Spring Lab

Spring Lab for beginner

In this Lab you'll learn how to build a Spring Application that uses JPA, Spring MVC, REST Web Service with an AngularJS front end.

Introduction

Warning: you'll need to understand every step and every file provided in this Lab. If you do not understand, you will not be able complete the Project.

Tools you'll need

In order to complete this lab you'll need a you'll need the following tools:

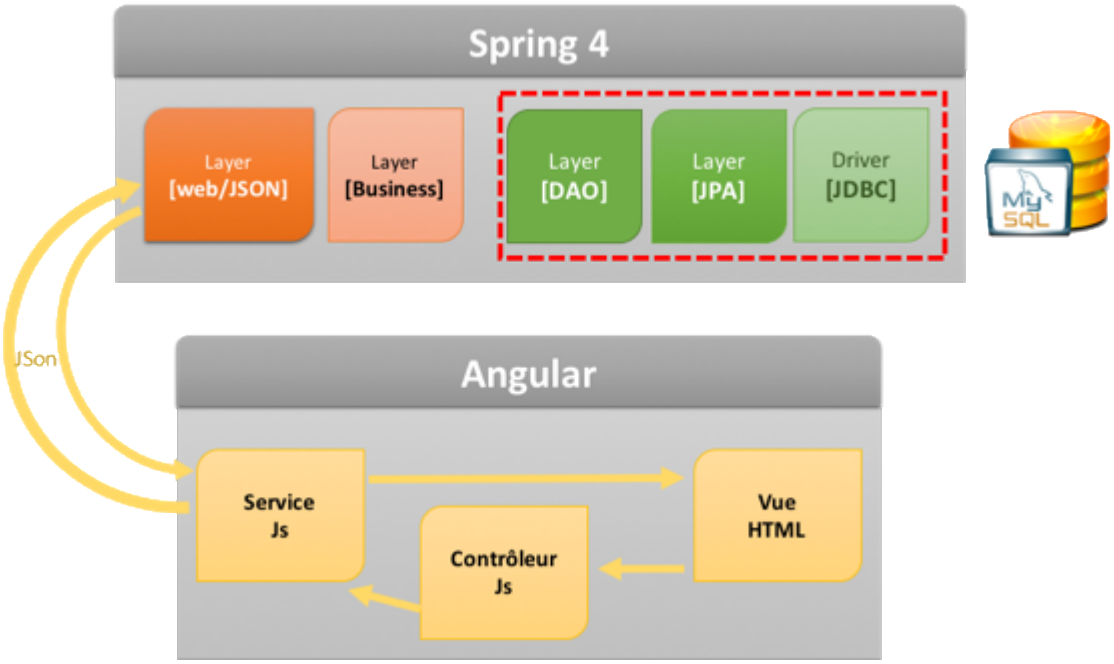
- **Spring Tool Suite 3.72**: https://spring.io/tools/sts/all
- **MySQL Database 5.X**: M
  - MAC OS & Windows MAMP : https://www.mamp.info/en/downloads/
  - For linux install mysql & phpmyadmin
- **Database structure and content**: http://downloads.mysql.com/docs/sakila-db.zip
- **Chrome Plugin**: https://chrome.google.com/webstore/detail/hgmloofddffdnphfgcellkdfbfbjeloo
- **Node.JS:** https://nodejs.org/en/download/

And last but not least: you need an Internet Connection while configuring the different projects.

The Lab

In this Lab we are going to build a complete Web Application working application.

The architecture of the application will be:



The Project

What shall be done

The Project is to complete to rental store application. That application will be used by the rental shop in order. The rental shop is determined by the login.

What you should provide (by store):

1. User Login / Password (Static Security),
2. Ability to create/update a customer with its address in one operation,
3. Ability to rent a film to a customer and to give it back,
4. Ability to Add/Remove a film from the inventory,
5. Ability to create/modify/delete a film and its related information. Related information are : Film Category, Language, Actors, Text
6. Ability to create/update/delete referential tables: Actor, Country, City, Language, Category

Optional:

1. User Login / Password (Table Staff with Encryption),
2. Handle Internationalization with a list box to choose its locale,
3. Perform the cash management of the Shop,
4. For the shop managers
    a. create BI reports in order to see by store and for a given store:
        i. The Revenue per year / Month / Week,
        ii. Top 10 customers over the different laps of time,
        iii. Top 10 film over the different laps of time,
        iv. Top 10 actors over the different laps of time for generated cash,
    b. Ability to create/update/delete a staff member.

Unit Testing of the application is mandatory

How many?

The project shall be developed by groups of 3 up to 4 students.

It is forbidden to work at 1, 2, 5 or more.

How is it delivered

It it delivered by mail to odepertat@adlere.fr . Email subject should starts with "[ISEP]LAB02 " then the list of the names in capital letters separated by commas. The global project should be deliver as follow:

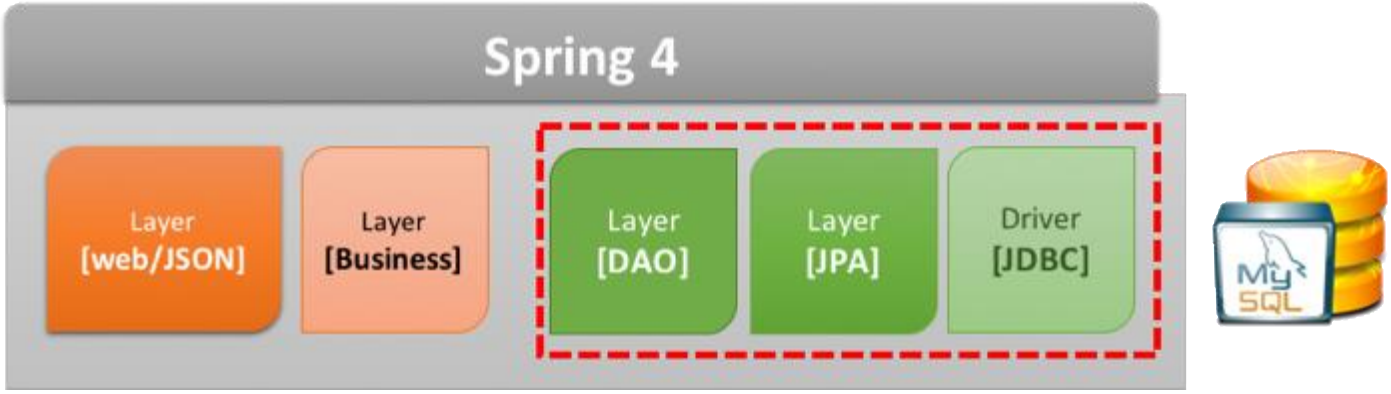
- A STS / Maven project for the DAL with Unit Testing,
- A STS / Maven project for the REST services & Business Logic with Unit Testing,
- A project for the AngularJS GUI.
  You can switch to Thymeleaf GUI or JSPs but at least the rental should be done in Angular

Word document that contains:

- Chapter 1 : Introduction, installation, what has been done, how it works … and … how to make it work
- Chapter 2, with screenshots showing:
  - At chapter 2.1 : login/password
  - At chapter 2.2:
  - …. And so on
- For optional features, do the same with chapter 3.

Unit Testing are part of the delivery.

Building the Data Access Layer



Getting Started

In order to begging, we are going to create a simple Java JPA application with a unique Entity Bean (aka. Table) named Customer.

To avoid, database connection problems, installation, we are going to use H2 Java Database.

H2 Database

H2 Database is Java SQL database. It is a real DMBS that works inside the JVM. No network configuration is required, booting fast, reliable for developers, no maintenance, no downtime… The main features of H2 are:

- Very fast, open source, JDBC API
- Embedded and server modes; in-memory databases
- Browser based Console application
- Small footprint: around 1.5 MB jar file size

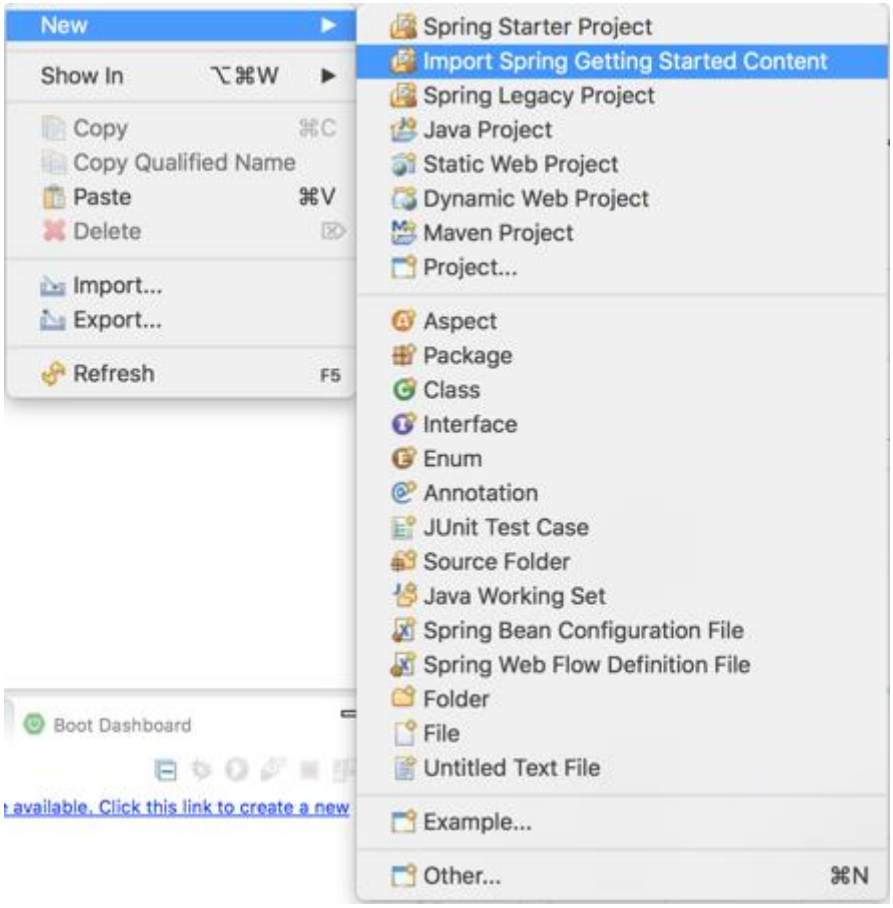
Features

	H2	Derby	HSQLDB	MySQL	PostgreSQL
Pure Java	Yes	Yes	Yes	No	No
Memory Mode	Yes	Yes	Yes	No	No
Encrypted Database	Yes	Yes	Yes	No	No
ODBC Driver	Yes	No	No	Yes	Yes
Fulltext Search	Yes	No	No	Yes	Yes
Multi Version Concurrency	Yes	No	Yes	Yes	Yes
Footprint (jar/dll size)	~1 MB	~2 MB	~1 MB	~4 MB	~6 MB

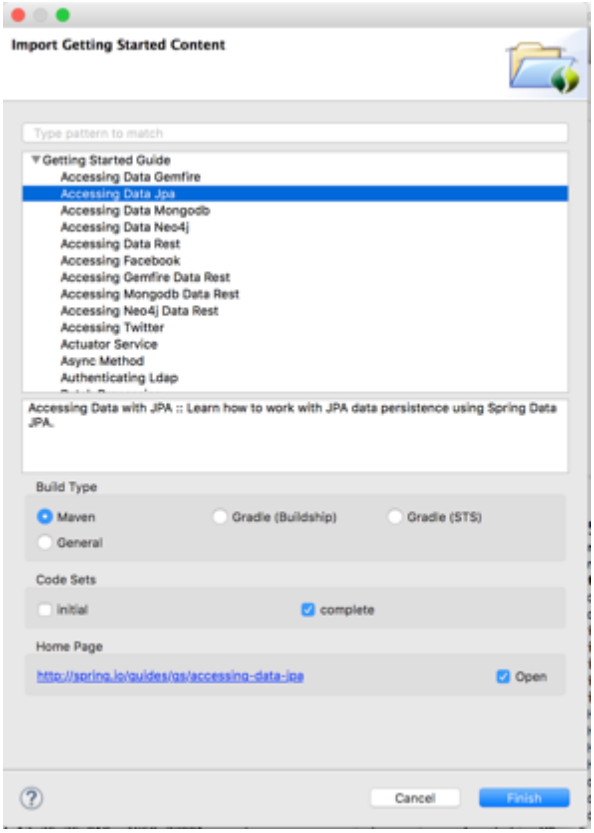
That Database is useful for Unit Testing purposes and Application Development support.

Creating The First Project

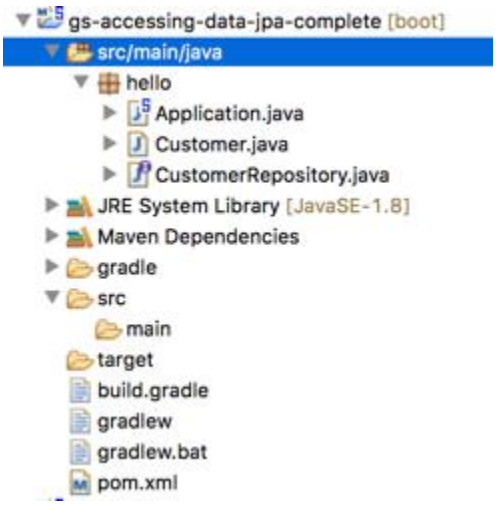
In order to create our first project, we are not going to reinvent the wheel. We are going to use the "Getting Started Contents Guides" from http://spring.io.



We Select the "Accessing Data JPA" Guide. Do not check initial if you want to have the final result, otherwise you'll have to follow the hole guide in order to get a working project (it is much more longer but better to get the right information).

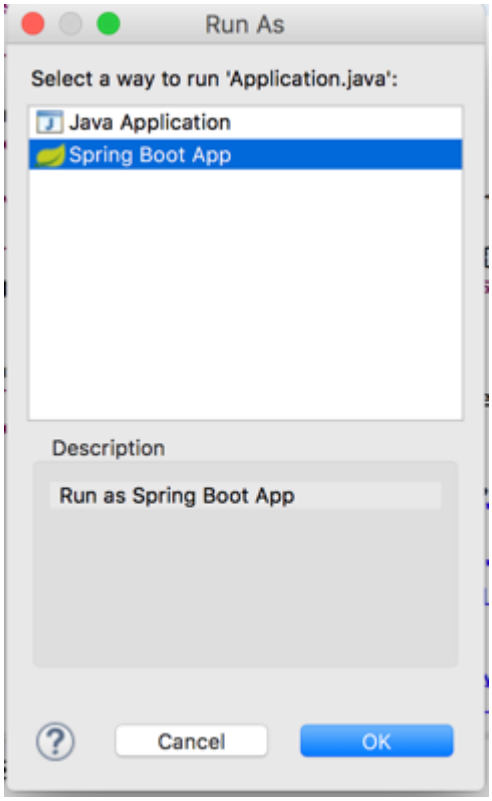


Now the created project should look like this:

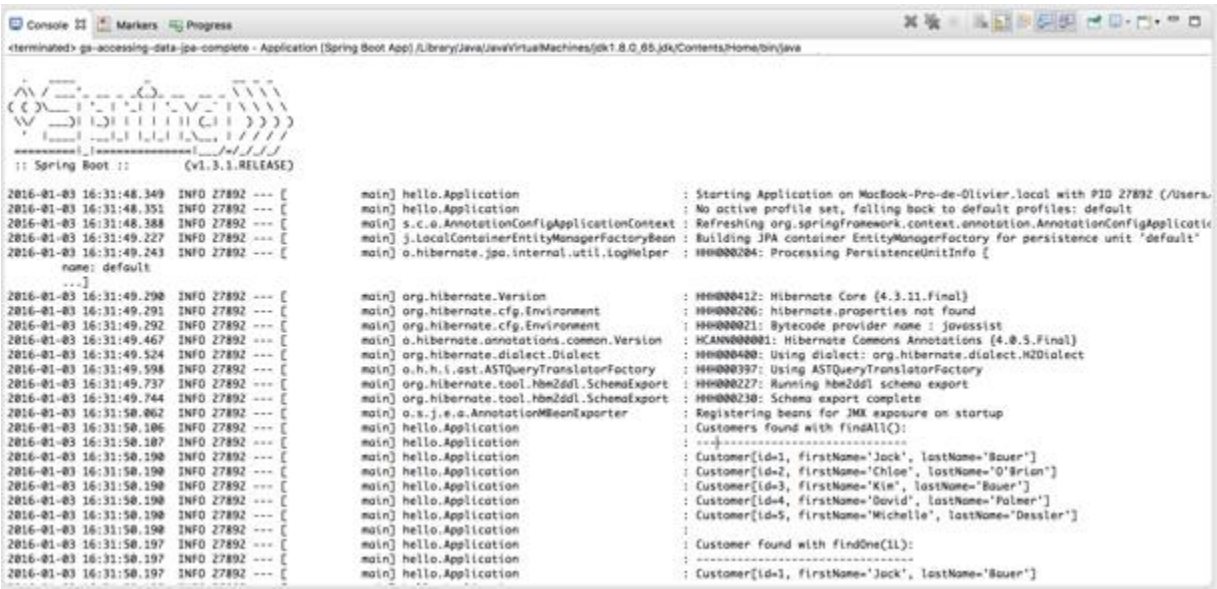


As you have been running it, you have noticed that it's a working project. But the configuration possibilities are really poor. Everything is done by default so the database used is H2 and you cannot change anything (entities are created at runtime and so on...).

You can run the created project from the command line thanks to Spring. This is very helpful in order to test debug and perform other development operations. To do so right click on the project name and select "Run As".



The result of the execution can be seen in the console. You'll find bellow a capture of the console output.



Creating a more realistic DAL

Enabling a configuration

We going to improve the process and make a bit more professional. In the previously created project, create new packages:

- `isep.web.sakila.jpa.config`: this will host Configuration of Spring,
- `isep.web.sakila.jpa.console`: this will host the main programs,
- `isep.web.sakila.jpa.entities`: this will host JPA entities,
- `isep.web.sakila.jpa.repositories`: this will host ,

Create the file `Config.java` using the code following code:

```
package isep.web.sakila.jpa.config;

import javax.persistence.EntityManagerFactory;
import javax.sql.DataSource;

import org.apache.commons.dbcp.BasicDataSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.orm.jpa.JpaTransactionManager;
import org.springframework.orm.jpa.JpaVendorAdapter;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.Database;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;

//@ComponentScan(basePackages = { "isep.web.sakila.jpa" })
//@EntityScan(basePackages = { "isep.web.sakila.jpa.entities" })
@EnableTransactionManagement
@EnableJpaRepositories(basePackages = { "isep.web.sakila.jpa.repositories" })
@Configuration
public class Config
{
	// la source de données H2
	@Bean
	public DataSource dataSource()
	{
		BasicDataSource dataSource = new BasicDataSource();
		dataSource.setDriverClassName("org.h2.Driver");
		dataSource.setUrl("jdbc:h2:./demo");
		dataSource.setUsername("sa");
		dataSource.setPassword("");
		return dataSource;
	}

	// le provider JPA
	@Bean
	public JpaVendorAdapter jpaVendorAdapter()
	{
		HibernateJpaVendorAdapter hibernateJpaVendorAdapter = new
HibernateJpaVendorAdapter();
		hibernateJpaVendorAdapter.setShowSql(false);
		hibernateJpaVendorAdapter.setGenerateDdl(true);
		hibernateJpaVendorAdapter.setDatabase(Database.H2);
		return hibernateJpaVendorAdapter;
	}

	// EntityManagerFactory
```

```
	@Bean
	public EntityManagerFactory entityManagerFactory(JpaVendorAdapter jpaVendorAdapter,
DataSource dataSource)
	{
		LocalContainerEntityManagerFactoryBean factory = new
LocalContainerEntityManagerFactoryBean();
		factory.setJpaVendorAdapter(jpaVendorAdapter);
		factory.setPackagesToScan("isep.web.sakila.jpa.entities");
		factory.setDataSource(dataSource);
		factory.afterPropertiesSet();
		return factory.getObject();
	}

	// Transaction manager
	@Bean
	public PlatformTransactionManager transactionManager(EntityManagerFactory c)
	{
		JpaTransactionManager txManager = new JpaTransactionManager();
		txManager.setEntityManagerFactory(c);
		return txManager;
	}

}
```

Creating new main applications

Create the following new application (it uses the configuration class previously created): ApplicationSpringboot.java

```
package isep.web.sakila.jpa.console;

import java.util.List;

import org.springframework.boot.SpringApplication;
import org.springframework.context.ConfigurableApplicationContext;

import isep.web.sakila.jpa.entities.Customer;
import isep.web.sakila.jpa.repositories.CustomerRepository;

public class ApplicationSpringboot
{

	public static void main(String[] args)
	{

		ConfigurableApplicationContext context =
SpringApplication.run(isep.web.sakila.jpa.config.Config.class);
		CustomerRepository repository = context.getBean(CustomerRepository.class);

		// save a couple of customers
		repository.save(new Customer("Jack", "Bauer"));
		repository.save(new Customer("Chloe", "O'Brian"));
		repository.save(new Customer("Kim", "Bauer"));
		repository.save(new Customer("David", "Palmer"));
		repository.save(new Customer("Michelle", "Dessler"));

		// fetch all customers
		Iterable<Customer> customers = repository.findAll();
		System.out.println("Customers found with findAll():");
		System.out.println("-------------------------------");
		for (Customer customer : customers)
		{
			System.out.println(customer);
		}
		System.out.println();

		// fetch an individual customer by ID
		Customer customer = repository.findOne(1L);
		System.out.println("Customer found with findOne(1L):");
```

```
            System.out.println("--------------------------------");
            System.out.println(customer);
            System.out.println();

            // fetch customers by last name
            List<Customer> bauers = repository.findByLastName("Bauer");
            System.out.println("Customer found with findByLastName('Bauer'):");
            System.out.println("--------------------------------------------");
            for (Customer bauer : bauers)
            {
                  System.out.println(bauer);
            }

            context.close();
      }

}
```

Create also the following application (notice that in this version there is no more any reference to Spring-Boot environment):

```
package isep.web.sakila.jpa.console;

import java.util.List;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import isep.web.sakila.jpa.entities.Customer;
import isep.web.sakila.jpa.repositories.CustomerRepository;

public class ApplicationSpringboot2
{
      public static void main(String[] args)
      {

            AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext(
                isep.web.sakila.jpa.config.Config.class);
            CustomerRepository repository = context.getBean(CustomerRepository.class);

            // save a couple of customers
            repository.save(new Customer("Jack", "Bauer"));
            repository.save(new Customer("Chloe", "O'Brian"));
            repository.save(new Customer("Kim", "Bauer"));
            repository.save(new Customer("David", "Palmer"));
            repository.save(new Customer("Michelle", "Dessler"));

            // fetch all customers
            Iterable<Customer> customers = repository.findAll();
            System.out.println("Customers found with findAll():");
            System.out.println("-------------------------------");
            for (Customer customer : customers)
            {
                  System.out.println(customer);
            }
            System.out.println();

            // fetch an individual customer by ID
            Customer customer = repository.findOne(1L);
            System.out.println("Customer found with findOne(1L):");
            System.out.println("--------------------------------");
            System.out.println(customer);
            System.out.println();

            // fetch customers by last name
            List<Customer> bauers = repository.findByLastName("Bauer");
            System.out.println("Customer found with findByLastName('Bauer'):");
            System.out.println("--------------------------------------------");
            for (Customer bauer : bauers)
            {
                  System.out.println(bauer);
```

```
            }

            context.close();
        }

}
```

Add to your `pom.xml` file the following dependencies:

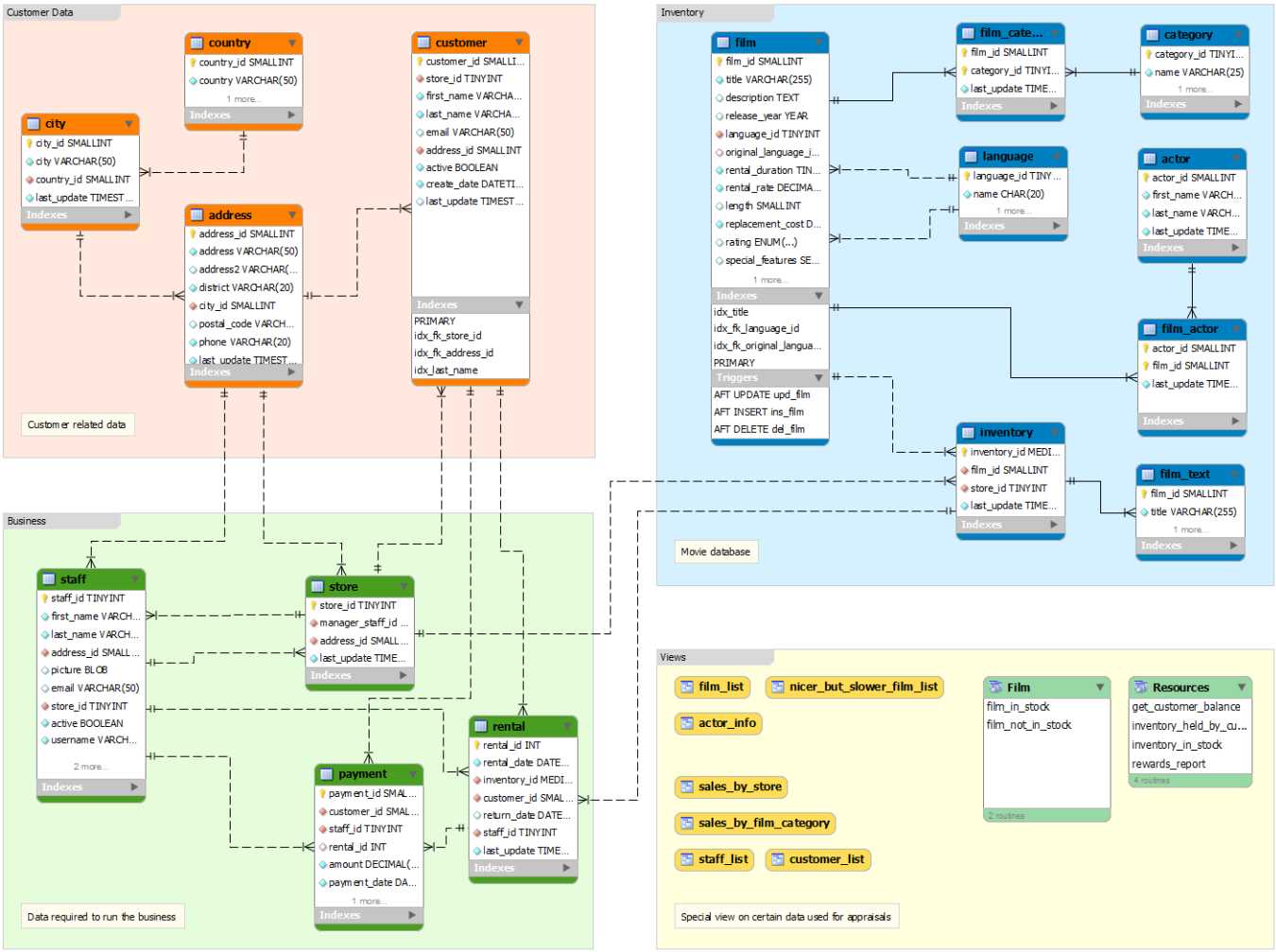
```
<dependencies>
   <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-data-jpa</artifactId>
   </dependency>
   <dependency>
     <groupId>com.h2database</groupId>
     <artifactId>h2</artifactId>
   </dependency>
   <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-core</artifactId>
   </dependency>
   <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-context</artifactId>
   </dependency>
   <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-beans</artifactId>
   </dependency>
   <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-aop</artifactId>
   </dependency>
   <dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-tx</artifactId>
   </dependency>
   <dependency>
     <groupId>org.springframework.data</groupId>
     <artifactId>spring-data-jpa</artifactId>
   </dependency>
   <dependency>
     <groupId>org.hibernate</groupId>
     <artifactId>hibernate-entitymanager</artifactId>
   </dependency>
   <dependency>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot</artifactId>
   </dependency>
   <dependency>
     <groupId>commons-dbcp</groupId>
     <artifactId>commons-dbcp</artifactId>
   </dependency>
   <dependency>
     <groupId>commons-pool</groupId>
     <artifactId>commons-pool</artifactId>
   </dependency>
 </dependencies>
```

Creating a real Data Access Layer (DAL)

Now that you have seen how the DAL looks like with the H2 Database a single table, we are going to create a real project and create the JPA mapping to our SAKILA Database.

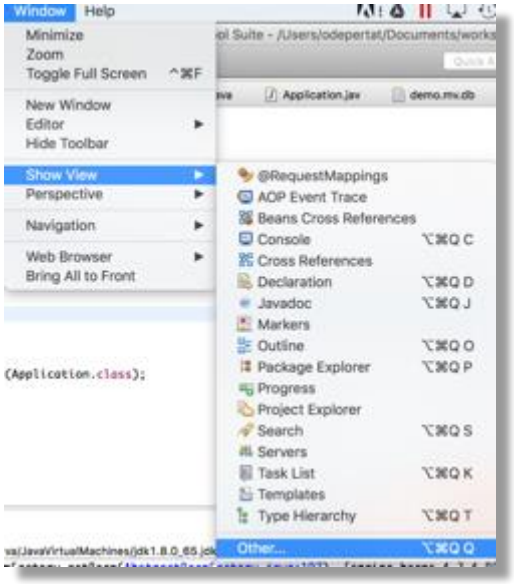
Sakila Database Structure

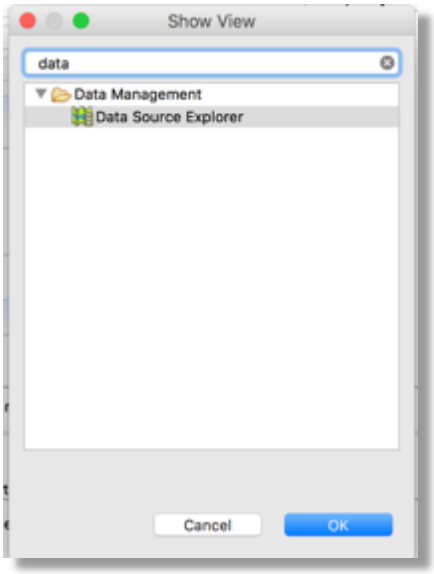
You have already worked with the Sakila Database, but we advice you to look carefully to its design. The representation bellow is quite interesting and should help you before the reading of the complete documentation at http://downloads.mysql.com/docs/sakila-en.pdf



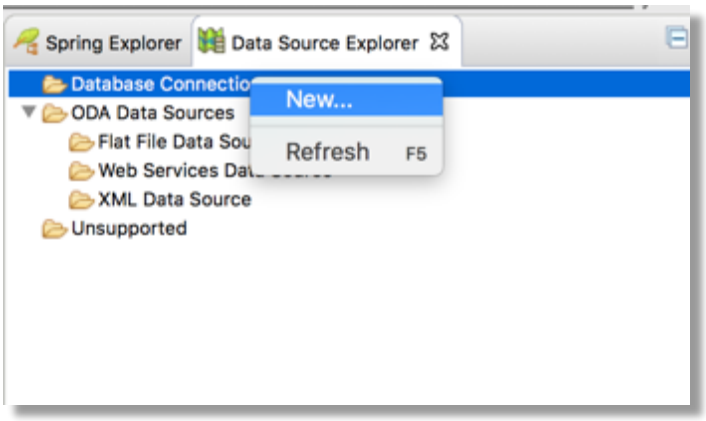
Creating a database connection to MySQL

In order to create JPA Entities we'll need a MySQL connection from STS. Unfortunately, unlike Maven Eclipse will need you to provide the MySQL Driver. You can get it from your .m2 repository or you can download it from https://dev.mysql.com/downloads/connector/j/. This will enables you to create the JPA Entities from the database. First, open the "Data Source Explorer" View.

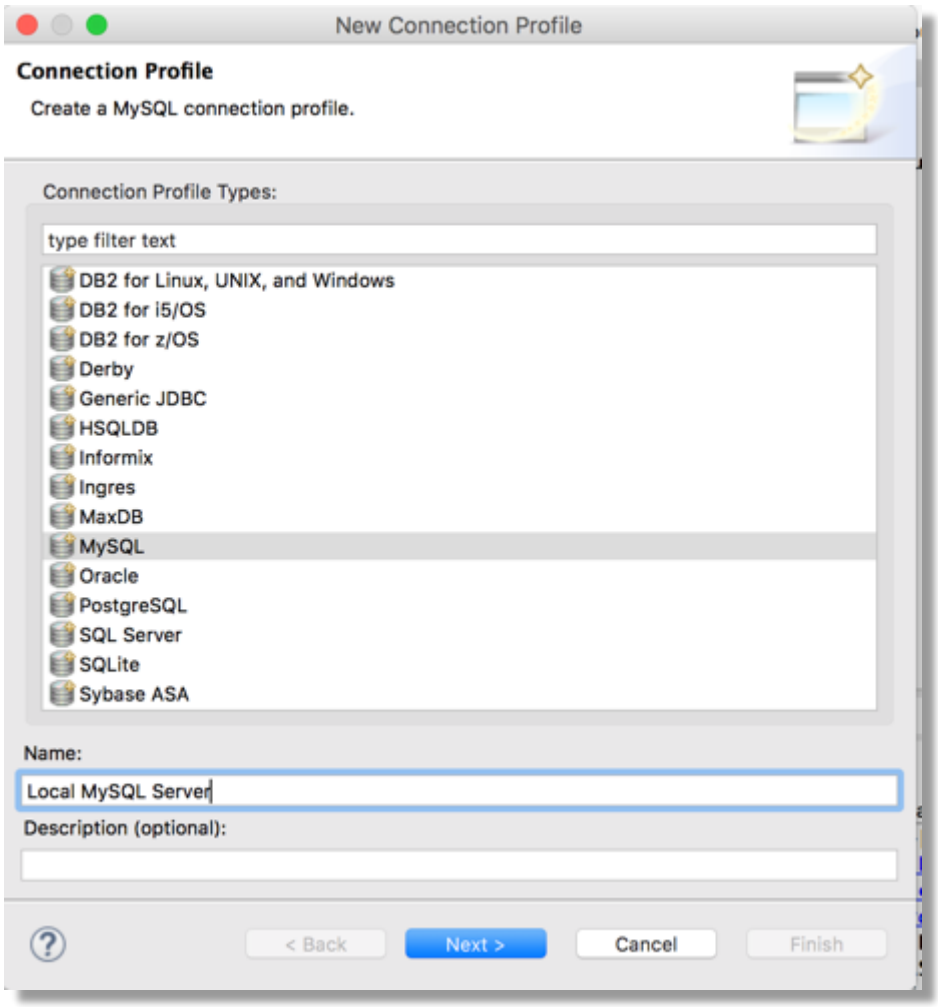




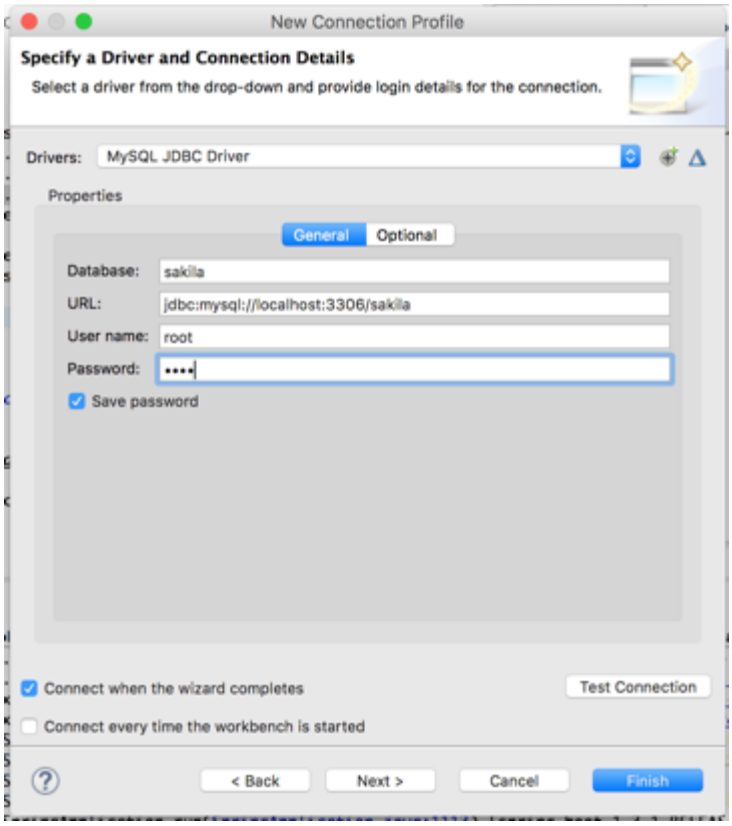
The select new from the view



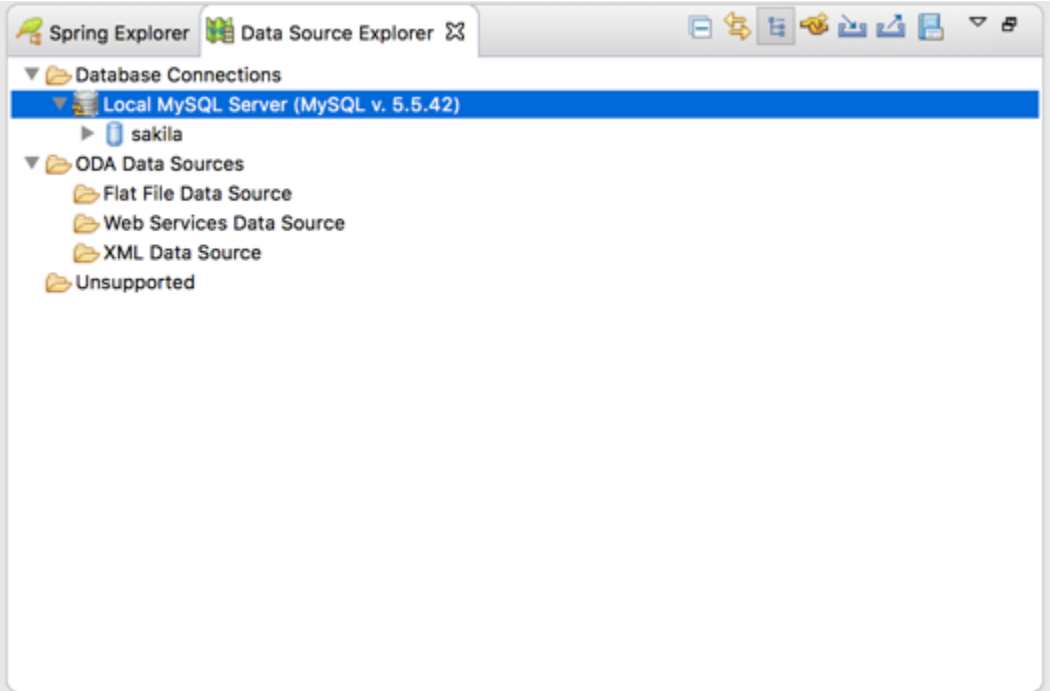
Then name your database connection and select the type.



Specify connection parameters (it depends on your installation)

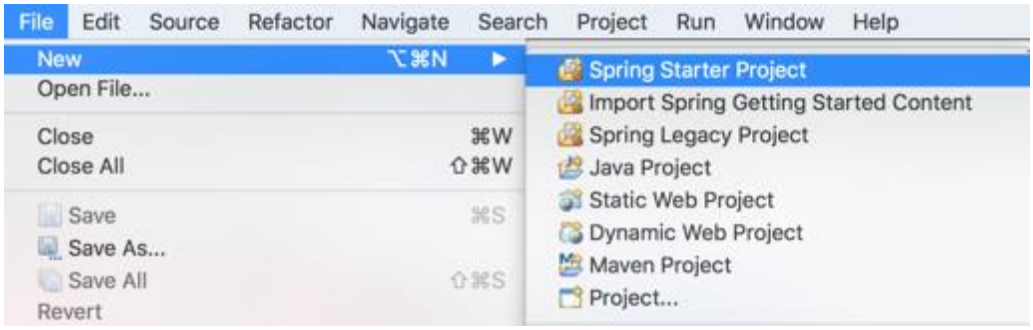


The freshly created connection appears as shown bellow:

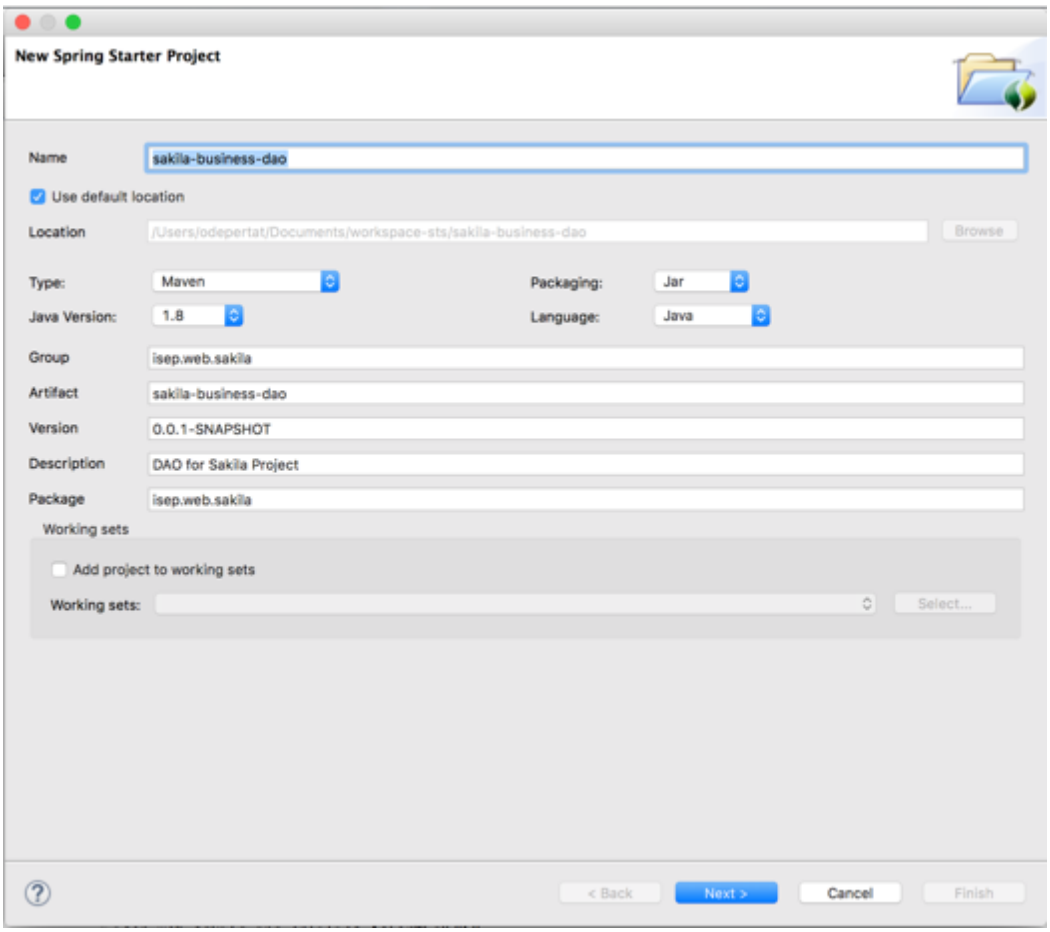


Creating the Real DAO Project

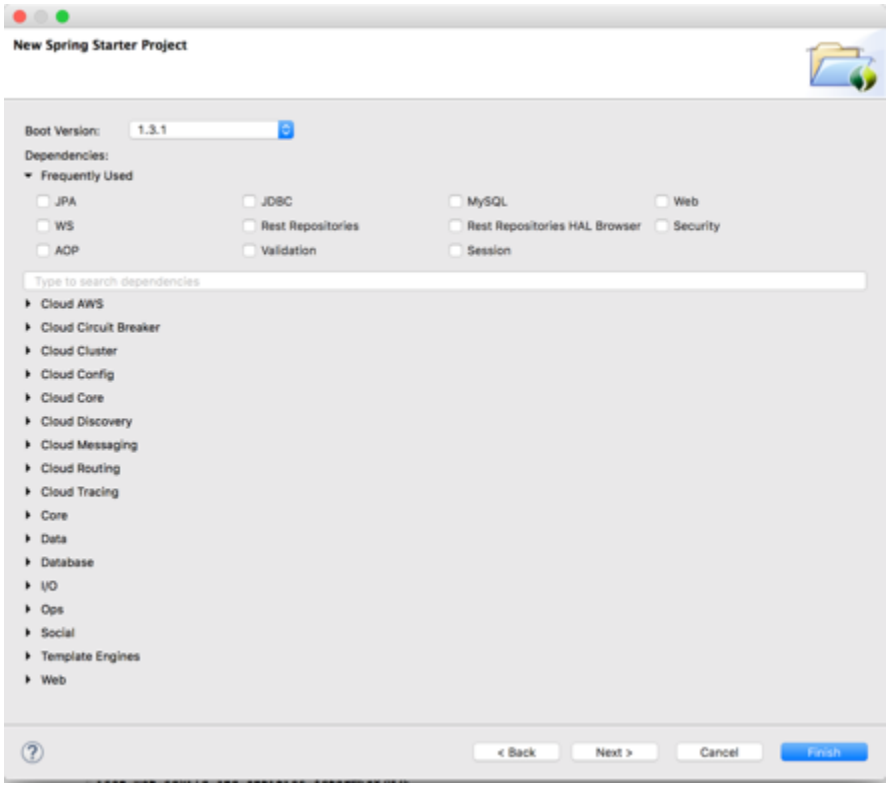
To start clean. For this we are going to create a brand new project.

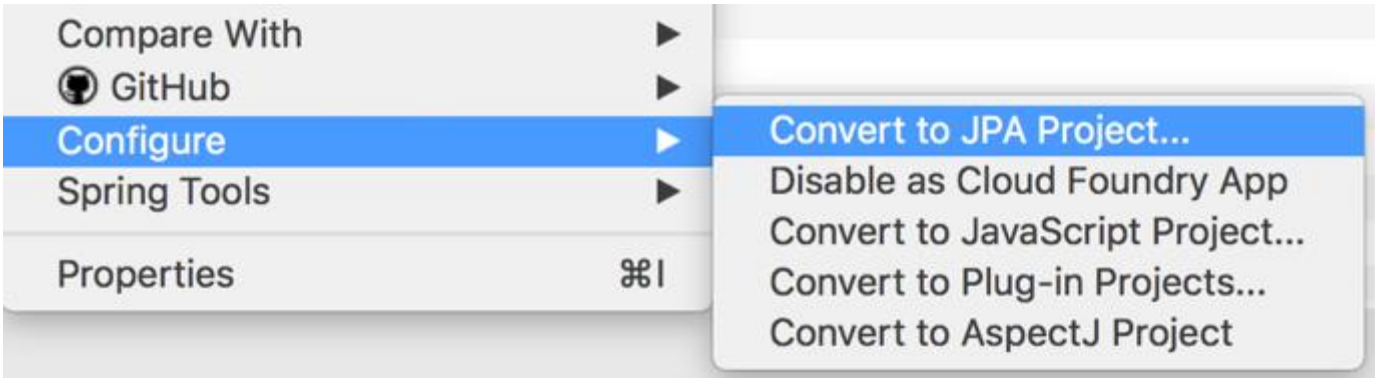


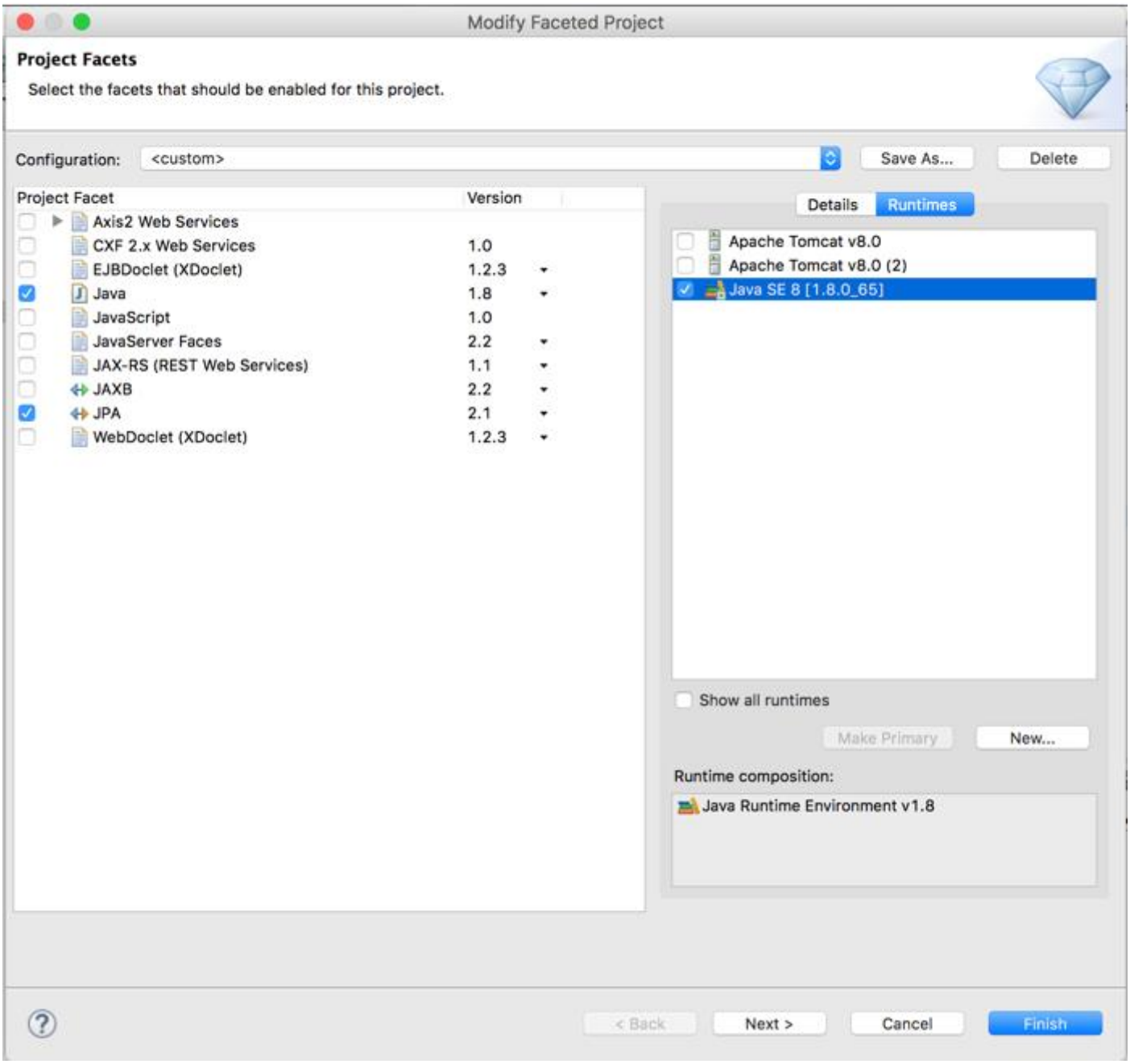
Then name the library in a more explicit way like it is detailed bellow (every parameter is important):



On the next screen leave everything empty, we'll configure the maven file by ourselves.

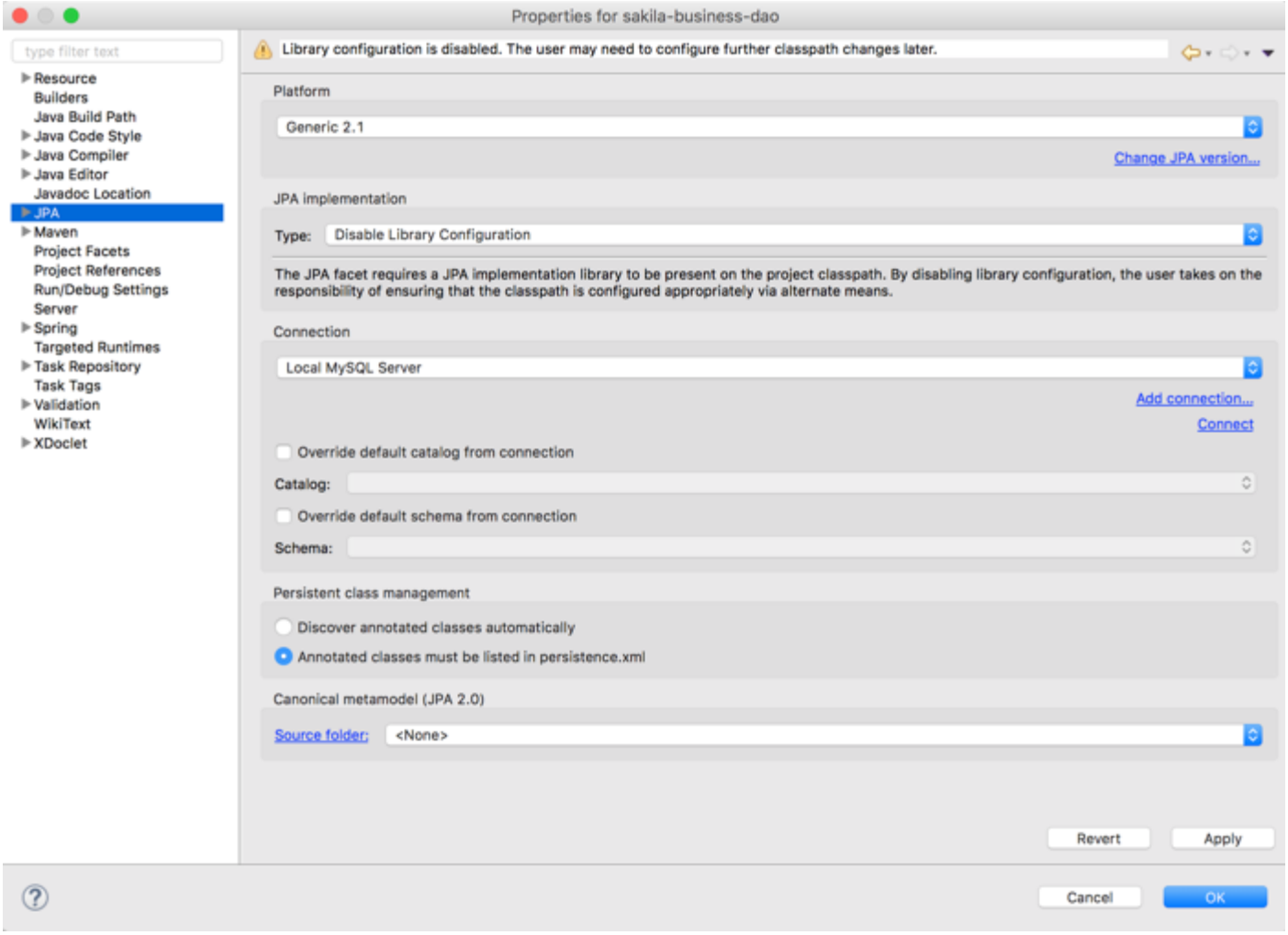



Compare With
GitHub
Configure
Spring Tools
Properties
⌘I
Convert to JPA Project...
Disable as Cloud Foundry App
Convert to JavaScript Project...
Convert to Plug-in Projects...
Convert to AspectJ Project


Modify Faceted Project
Project Facets
Select the facets that should be enabled for this project.
Configuration:
<custom>
Save As...
Delete
Project Facet
Version
Axis2 Web Services
CXF 2.x Web Services 1.0
EJBDoclet (XDoclet) 1.2.3
Java 1.8
JavaScript 1.0
JavaServer Faces 2.2
JAX-RS (REST Web Services) 1.1
JAXB 2.2
JPA 2.1
WebDoclet (XDoclet) 1.2.3
Details
Runtimes
Apache Tomcat v8.0
Apache Tomcat v8.0 (2)
Java SE 8 [1.8.0_65]
Show all runtimes
Make Primary
New...
Runtime composition:
Java Runtime Environment v1.8
< Back
Next >
Cancel
Finish

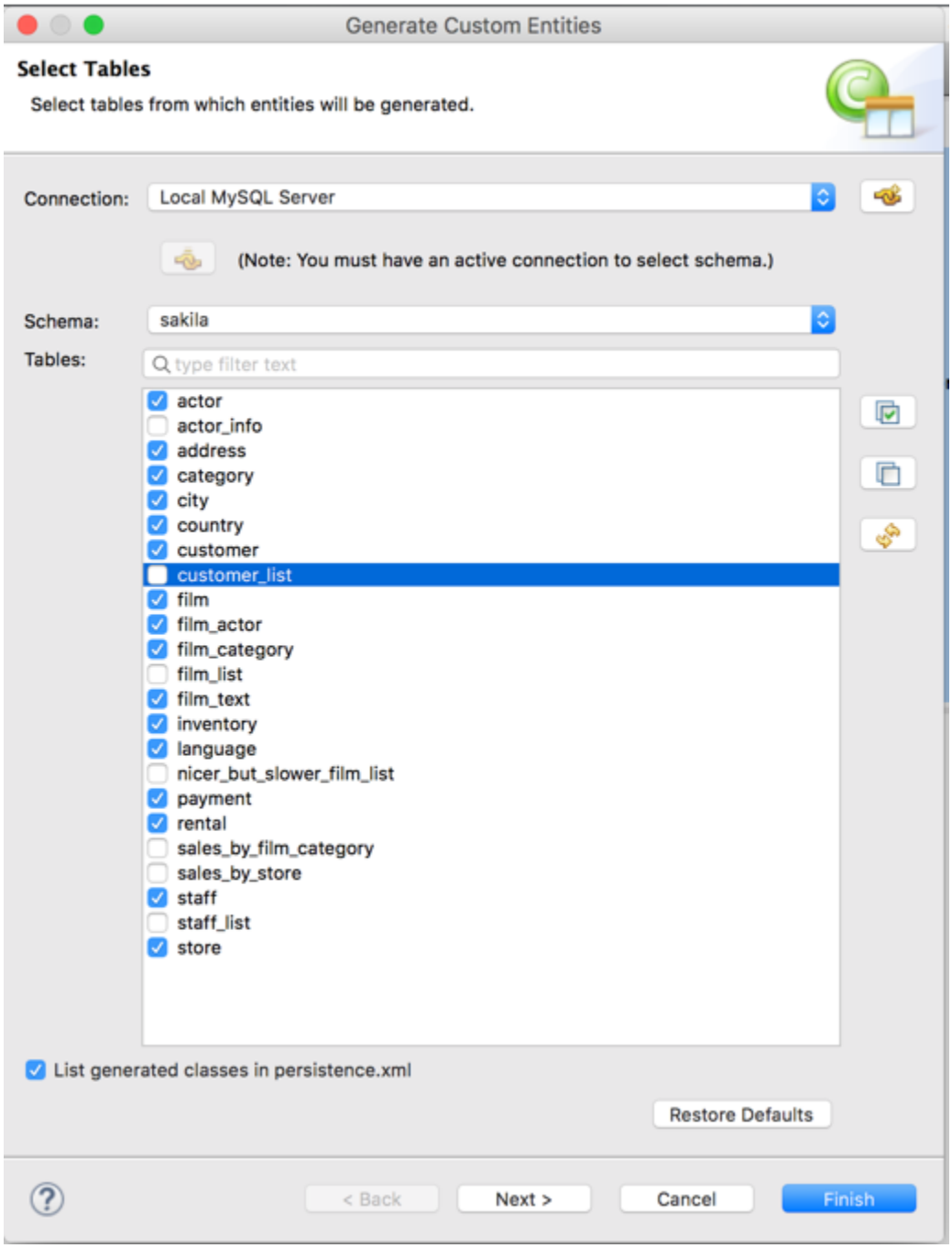
Generating JPA Entities

In the project Settings, under the JPA parameters, you can now select the connection to "Local MySQL Server". It will allow STS to connect to the database in order to perform a reverse engineering operation and generate the JPA objects.



You can now select the "Generate Entities from Tables" option from the right click on the project:

From the database connection select all the tables and unselect all the views (cf. Database schema). Warning there 7 VIEWS to unselect.



The go directly to the newt screen and click next (this screen is useful when we want to erase the Join Tables or change some cardinalities).

Next Screen, **it is very important, reproduce exactly that is specified in the screen bellow**:

Generate Custom Entities

Customize Defaults

Optionally customize aspects of entities that will be generated by default from database tables. A Java package should be specified.

Mapping defaults

Key generator: auto

Sequence name:

You can use the patterns $table and/or $pk in the sequence name. These patterns will be replaced by the table name and the primary key column name when a table mapping is generated.

Entity access: Field Property

Associations fetch: Default Eager Lazy

Collection properties type: java.util.Set java.util.List

Always generate optional JPA annotations and DDL parameters

Domain java class

Source folder: sakila-business-dao/src/main/java Browse...

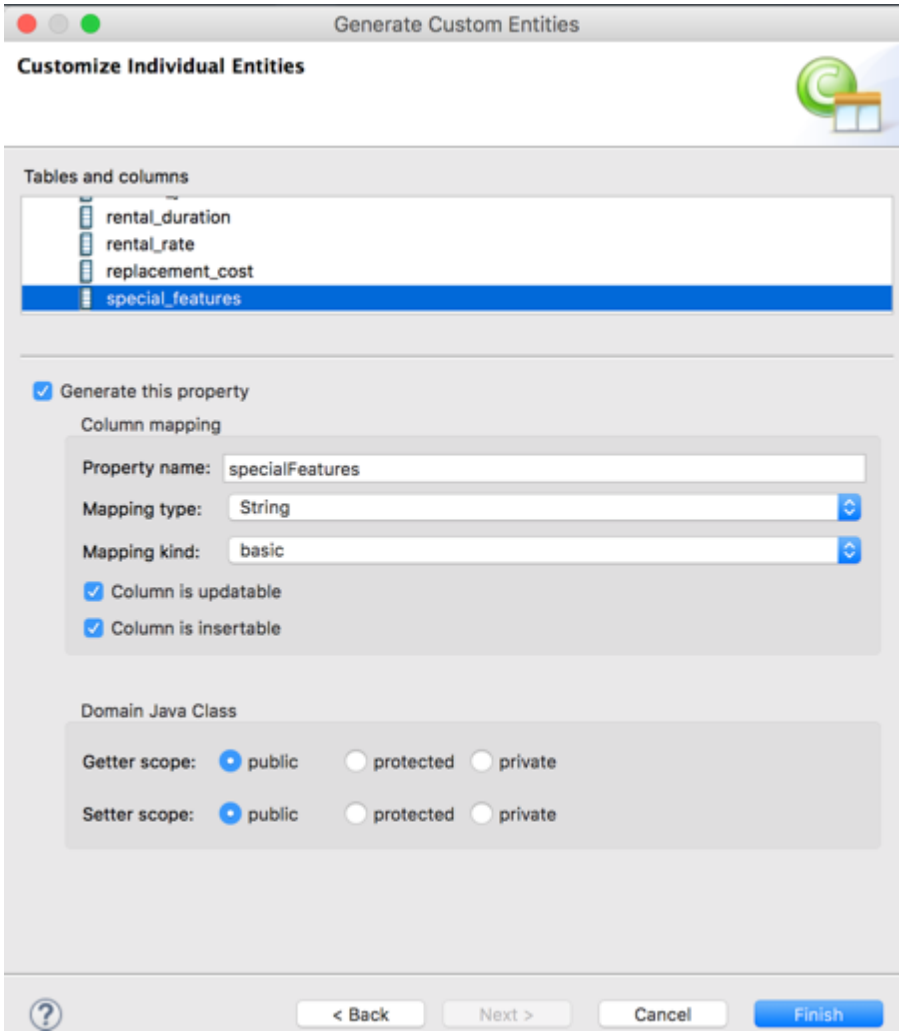
Package: isep.web.sakila.jpa.entities Browse...

Superclass: Browse...

Interfaces: java.io.Serializable Add... Remove

< Back Next > Cancel Finish

In the last screen change the Object type of the field `special_features` of Film to String. Otherwise it will not work (remember to look at the database that field it will help you a lot for film creation).



Edit the pom.xml

Edit the pom.xml and replace the dependencies with the following ones:

```
<dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
    </dependency>
    <dependency>
      <groupId>commons-dbcp</groupId>
      <artifactId>commons-dbcp</artifactId>
    </dependency>
    <dependency>
      <groupId>commons-pool</groupId>
      <artifactId>commons-pool</artifactId>
    </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
```

```
      <artifactId>jackson-databind</artifactId>
    </dependency>
    <dependency>
      <groupId>com.google.guava</groupId>
      <artifactId>guava</artifactId>
      <version>16.0.1</version>
    </dependency>
  </dependencies>
```

Create the repository

Under the package `isep.web.sakila.dao.repositories` create the following file

```
package isep.web.sakila.dao.repositories;

import org.springframework.data.repository.CrudRepository;

import isep.web.sakila.jpa.entities.Actor;

public interface ActorRepository extends CrudRepository<Actor, Integer>
{
}
```

Create the Business Layer

Create the Business Interface

```
package isep.web.sakila.dao.business;

import java.util.List;

import isep.web.sakila.jpa.entities.Actor;

public interface IBusiness
{
      public List<Actor> getAllActors();

      public Actor getByID(int actorId);

}
```

Its implementation

```
package isep.web.sakila.dao.business;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.google.common.collect.Lists;

import isep.web.sakila.dao.repositories.ActorRepository;
import isep.web.sakila.jpa.entities.Actor;

@Service("business")
public class Business implements IBusiness
{
      @Autowired
      private ActorRepository actorRepository;
```

```
	@Override
	public List<Actor> getAllActors()
	{
		return Lists.newArrayList(actorRepository.findAll());
	}

	public Actor getByID(int actorId)
	{
		return actorRepository.findOne(actorId);
	}
}
```

And its test class (you can see the dependency injection in the test case with the `@Autowired` annotation).

```
package isep.web.sakila.dao.business;

import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.SpringApplicationConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import isep.web.sakila.jpa.config.PersistenceConfig;

@SpringApplicationConfiguration(classes = PersistenceConfig.class)
@RunWith(SpringJUnit4ClassRunner.class)
public class BusinessTest
{
	@Autowired
	private IBusiness business;

	@Test
	public void testGetAllActors()
	{
		Assert.assertEquals(200, business.getAllActors().size());
	}

	@Test
	public void testGetByID()
	{
		Assert.assertEquals("GUINESS", business.getByID(1).getLastName());
	}

}
```

Create the Database connection configuration

Under the package `isep.web.sakila.jpa.config` create the following file

```
package isep.web.sakila.jpa.config;

import javax.sql.DataSource;

import org.apache.commons.dbcp.BasicDataSource;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.orm.jpa.EntityScan;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
import org.springframework.orm.jpa.JpaVendorAdapter;
import org.springframework.orm.jpa.vendor.Database;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.annotation.EnableTransactionManagement;
```

```
@EnableJpaRepositories(basePackages = { "isep.web.sakila.dao.repositories" })
@EnableAutoConfiguration
@ComponentScan(basePackages = { "isep.web.sakila" })
@EntityScan(basePackages = { "isep.web.sakila.jpa" })
@EnableTransactionManagement
public class PersistenceConfig
{

	// MySQL DataSource
	@Bean
	public DataSource dataSource()
	{
		BasicDataSource dataSource = new BasicDataSource();
		dataSource.setDriverClassName("com.mysql.jdbc.Driver");
		dataSource.setUrl("jdbc:mysql://localhost:3306/sakila");
		dataSource.setUsername("root");
		dataSource.setPassword("root");
		return dataSource;
	}

	// JPA Provider is only needed when we don't want the default one with the
	// default values provided by Spring Boot
	// Here we define it in order to activate/deactivate SQL Logs....
	@Bean
	public JpaVendorAdapter jpaVendorAdapter()
	{
		HibernateJpaVendorAdapter hibernateJpaVendorAdapter = new
HibernateJpaVendorAdapter();
		hibernateJpaVendorAdapter.setShowSql(false);
		hibernateJpaVendorAdapter.setGenerateDdl(false);
		hibernateJpaVendorAdapter.setDatabase(Database.MYSQL);
		return hibernateJpaVendorAdapter;
	}

	// EntityManageFactory and TransactionManager are defined by default by Spring
	// boot.
	// therefore, we do not declare anything here

}
```

Create the real main

This will be our main program. It is useful for development purposes. Otherwise it is useless. Think of using Unit Testing to develop the business layer...

```
package isep.web.sakila;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ConfigurableApplicationContext;

import isep.web.sakila.dao.business.IBusiness;
import isep.web.sakila.jpa.config.DomainAndPersistenceConfig;
import isep.web.sakila.jpa.entities.Actor;

@SpringBootApplication
public class SakilaBusinessDaoApplication
{

	public static void main(String[] args)
	{
		// We prepare the Spring Configuration
		SpringApplication app = new SpringApplication(DomainAndPersistenceConfig.class);
		app.setLogStartupInfo(false);

		// We launch the Application Context
		ConfigurableApplicationContext context = app.run(args);
		// Business Layer
```

```
            IBusiness business = context.getBean(IBusiness.class);

            try
            {
                    for (Actor actor : business.getAllActors())
                    {
                            System.out.println(actor);
                    }

                    Actor guiness = business.getByID(1);
                    System.out.printf("Who is ID 1? %s %s %n", guiness.getLastName(),
guiness.getFirstName());

            } catch (Exception ex)
            {
                    System.out.println("Exception : " + ex.getCause());
            }
            // Closing Spring Context
            context.close();

      }
}
```

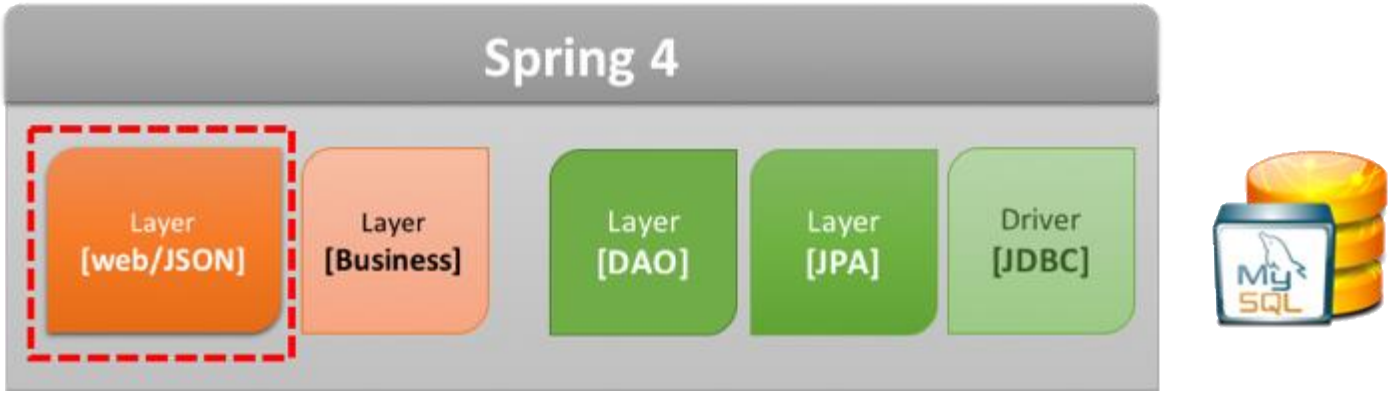
The final result should look like this:

Now we move to the service layer.

The Service Layer

In order to expose the Business Logic to your clients, you'll need a specific interface. In our project this will be completed by the Web/JSon layer.

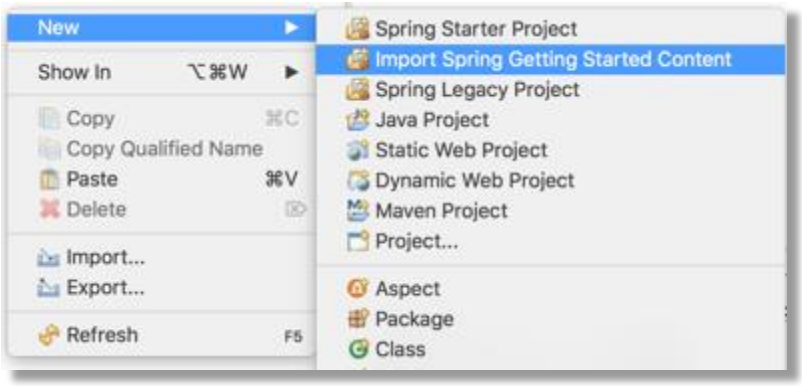
That Layer will expose Web Services at the REST format. Those Web Services will answer to the queries text formatted in JSON (JavaScript Object Notation). That kind of web application are often called Web API. We are going to implement that Web Application with Spring MVC.



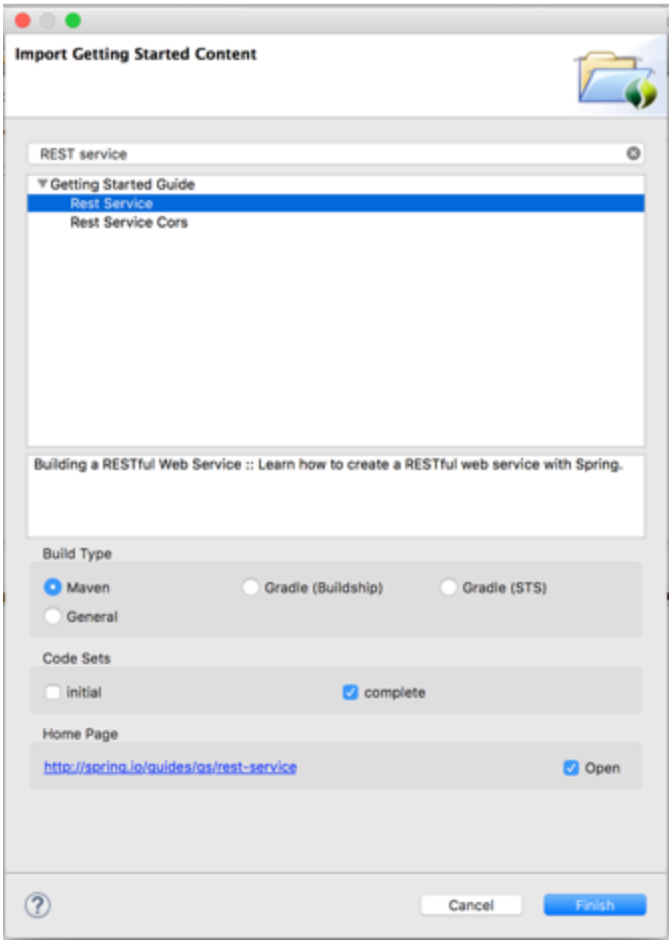
A First REST Web Service

In order to start we need an Application Server, Spring, REST API and tools to test our exposed API (because it is API, we cannot surf as a regular web site). Thanks to STS and Spring you have fully functional example. In order to run it start again with a guide from http://spring.io

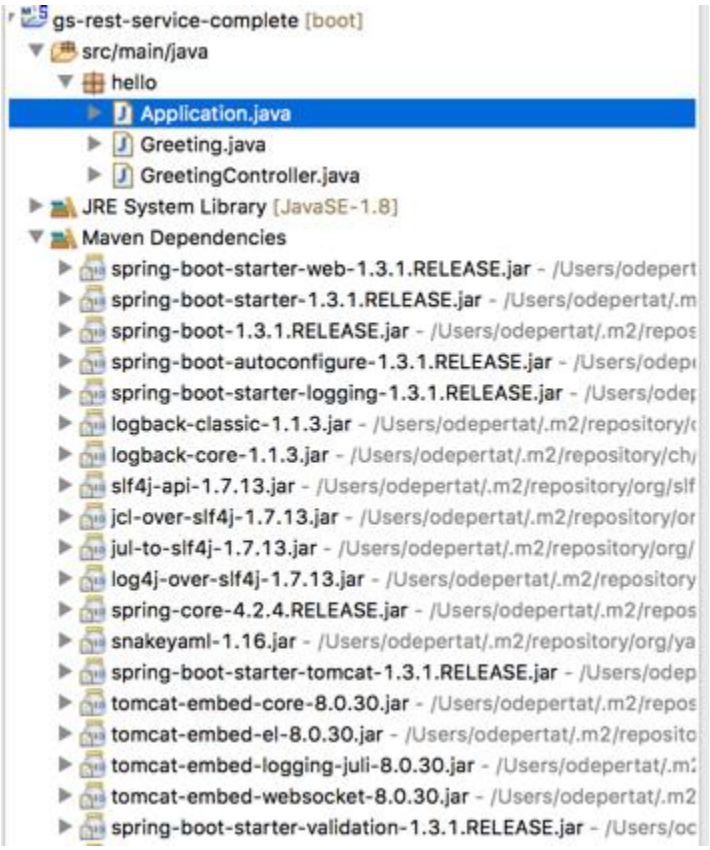
First import the project:

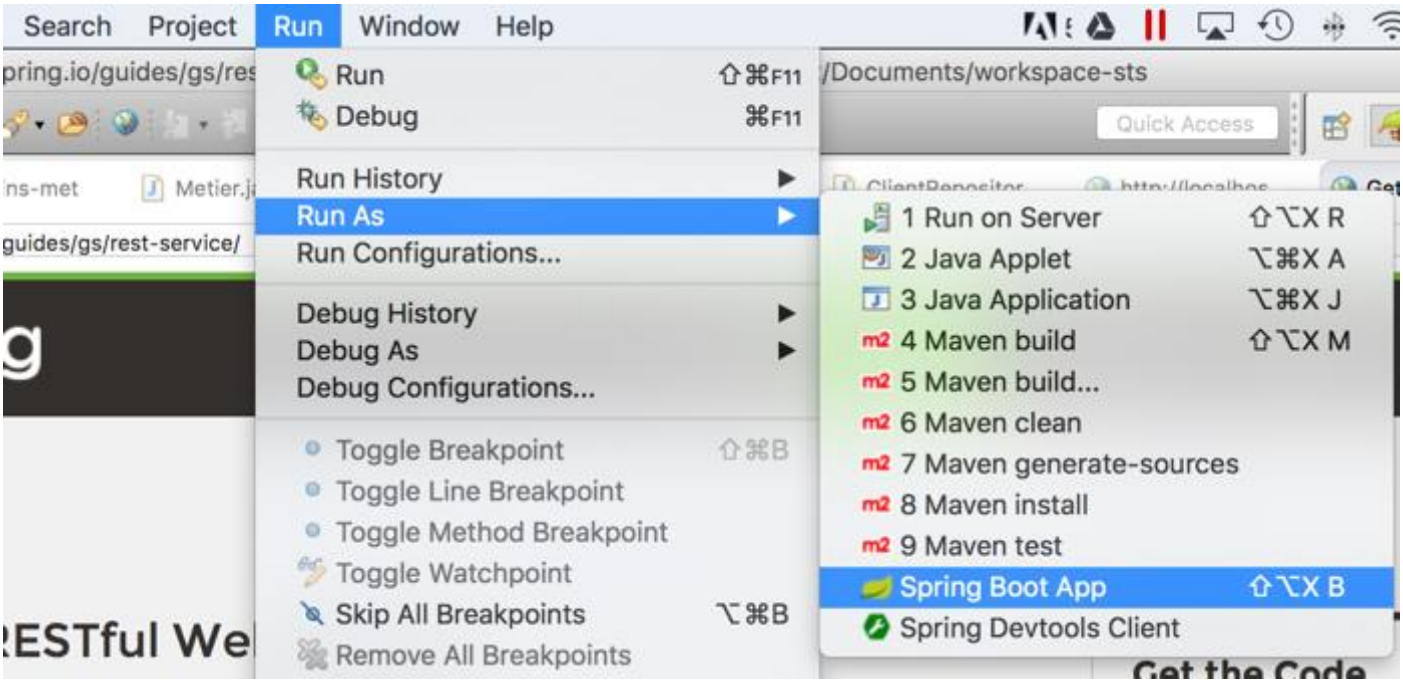


Type "REST Service" and select the given project (do not forget to uncheck initial).

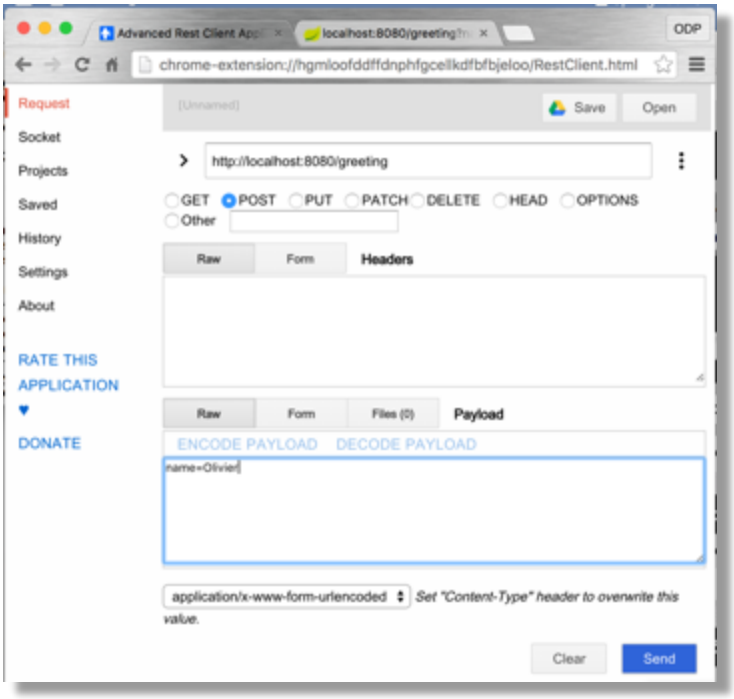


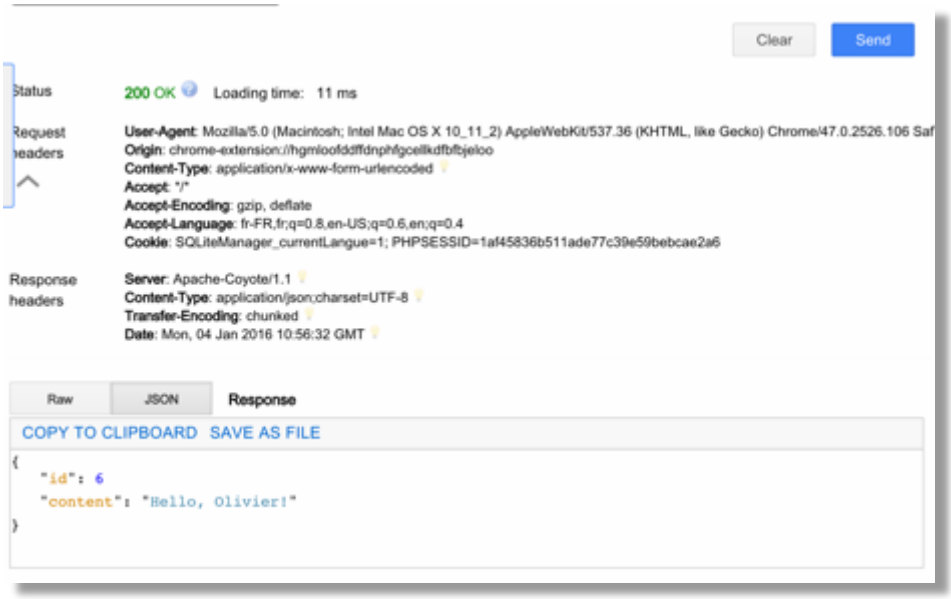
In the bran new project you just have created






Advanced Rest Client Appl
localhost:8080/greeting?name=olivier
{"id":5,"content":"Hello, olivier!"}


Advanced Rest Client Appl
localhost:8080/greeting?n
ODP
chrome-extension://hgmloofddffdnphfgcellkdfbfbjeloo/RestClient.html
Request
Socket
Projects
Saved
History
Settings
About
RATE THIS APPLICATION
DONATE
[Unnamed]
Save
Open
http://localhost:8080/greeting
GET
POST
PUT
PATCH
DELETE
HEAD
OPTIONS
Other
Raw
Form
Headers
Raw
Form
Files (0)
Payload
ENCODE PAYLOAD
DECODE PAYLOAD
name=Olivier
application/x-www-form-urlencoded
Set "Content-Type" header to overwrite this value.
Clear
Send


Clear
Send
Status
200 OK
Loading time: 11 ms
Request headers
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.106 Saf
Origin: chrome-extension://hgmloofddffdnphfgcellkdfbfbjeloo
Content-Type: application/x-www-form-urlencoded
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: fr-FR,fr;q=0.8,en-US;q=0.6,en;q=0.4
Cookie: SQLiteManager_currentLangue=1; PHPSESSID=1af45836b511ade77c39e59bebcae2a6
Response headers
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Mon, 04 Jan 2016 10:56:32 GMT
Raw
JSON
Response
COPY TO CLIPBOARD
SAVE AS FILE
{
"id": 6
"content": "Hello, Olivier!"
}

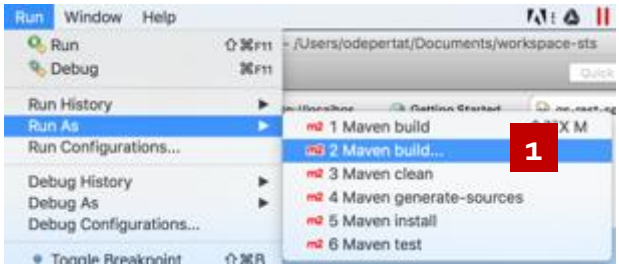
Running the project outside STS

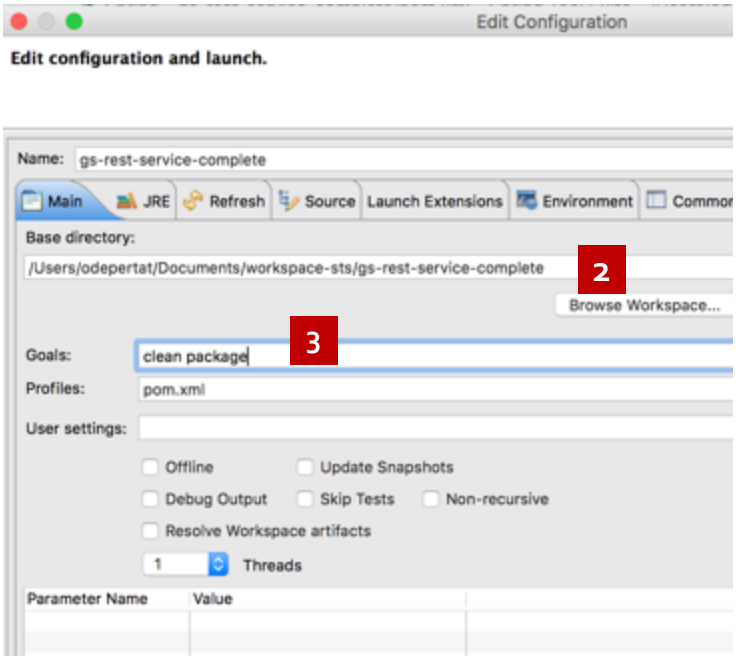
The program runs perfectly in STS but we may want to run it outside that embedded environment. To proceed we shall perform the following tasks....

Switch the `pom.xml` properties to:

```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <start-class>hello.Application</start-class>
  <java.version>1.8</java.version>
</properties>
```

Now we have to build the distributable package:





It generates a runnable that is in the target directory:

```
MacBook-Pro-de-Olivier:~/Documents/workspace-sts/gs-rest-service-complete/target$ ls -la
total 26160
drwxr-xr-x   8 odepertat  staff       272  4 jan 17:07 .
drwxr-xr-x  13 odepertat  staff       442  4 jan 17:07 ..
drwxr-xr-x   3 odepertat  staff       102  4 jan 17:07 classes
drwxr-xr-x   3 odepertat  staff       102  4 jan 17:07 generated-sources
-rw-r--r--   1 odepertat  staff  13389816  4 jan 17:07 gs-rest-service-0.1.0.jar
-rw-r--r--   1 odepertat  staff      3786  4 jan 17:07 gs-rest-service-0.1.0.jar.original
drwxr-xr-x   3 odepertat  staff       102  4 jan 17:07 maven-archiver
drwxr-xr-x   3 odepertat  staff       102  4 jan 17:07 maven-status
```

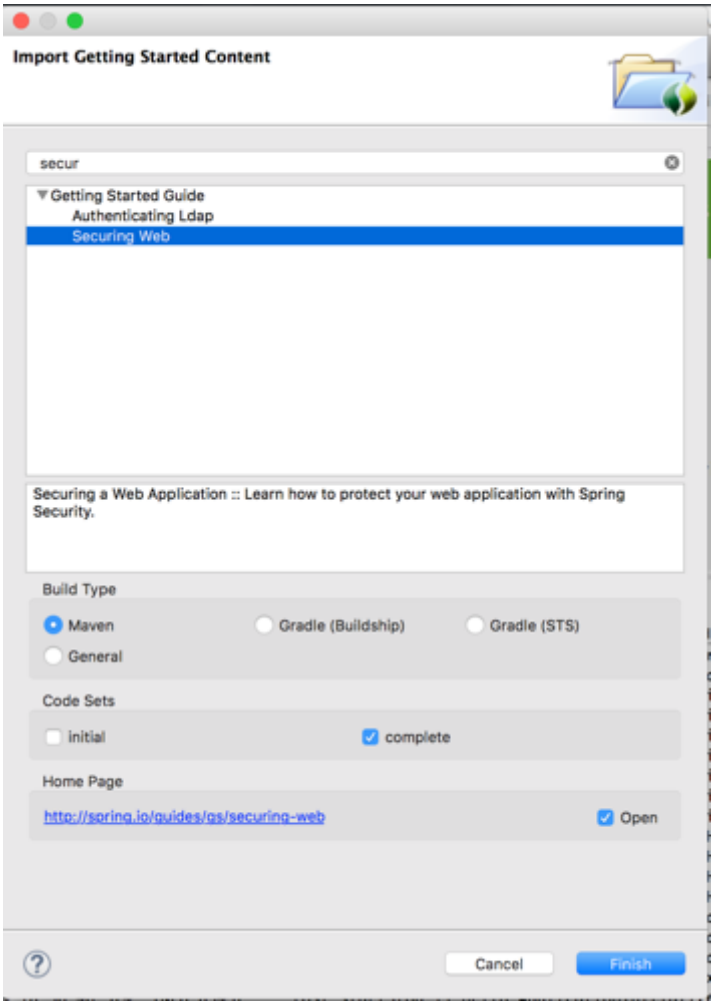
You can now run the REST application from the command line as follow:

```
MacBook-Pro-de-Olivier:~/Documents/workspace-sts/gs-rest-service-complete/target$ java -jar gs-rest-service-0.1.0.jar

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.3.1.RELEASE)

2016-01-04 17:20:32.376  INFO 81259 --- [           main] hello.Application                        : Starting Applicat
259 (/Users/odepertat/Documents/workspace-sts/gs-rest-service-complete/target/gs-rest-service-0.1.0.jar started by ode
rest-service-complete/target)
2016-01-04 17:20:32.379  INFO 81259 --- [           main] hello.Application                        : No active profile
2016-01-04 17:20:32.436  INFO 81259 --- [           main] ationConfigEmbeddedWebApplicationContext : Refreshing org.sp
beddedWebApplicationContext@792a5364: startup date [Mon Jan 04 17:20:32 CET 2016]; root of context hierarchy
2016-01-04 17:20:32.926  INFO 81259 --- [           main] o.s.b.f.s.DefaultListableBeanFactory     : Overriding bean d
rent definition: replacing [Root bean: class [null]; scope=; abstract=false; lazyInit=false; autowireMode=3; dependenc
ryBeanName=org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration$WhitelabelErrorViewConfiguration; fact
l; destroyMethodName=(inferred); defined in class path resource [org/springframework/boot/autoconfigure/web/ErrorMvcAu
]] with [Root bean: class [null]; scope=; abstract=false; lazyInit=false; autowireMode=3; dependencyCheck=0; autowireC
ngframework.boot.autoconfigure.web.WebMvcAutoConfiguration$WebMvcAutoConfigurationAdapter; factoryMethodName=beanNameV
nferred); defined in class path resource [org/springframework/boot/autoconfigure/web/WebMvcAutoConfiguration$WebMvcAut
2016-01-04 17:20:33.485  INFO 81259 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat initialize
2016-01-04 17:20:33.497  INFO 81259 --- [           main] o.apache.catalina.core.StandardService   : Starting service
2016-01-04 17:20:33.498  INFO 81259 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet
2016-01-04 17:20:33.581  INFO 81259 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spri
2016-01-04 17:20:33.581  INFO 81259 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root WebApplicati
2016-01-04 17:20:33.822  INFO 81259 --- [ost-startStop-1] o.s.b.c.e.ServletRegistrationBean        : Mapping servlet:
2016-01-04 17:20:33.827  INFO 81259 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean  : Mapping filter: '
2016-01-04 17:20:33.827  INFO 81259 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean  : Mapping filter: '
2016-01-04 17:20:33.827  INFO 81259 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean  : Mapping filter: '
2016-01-04 17:20:33.827  INFO 81259 --- [ost-startStop-1] o.s.b.c.embedded.FilterRegistrationBean  : Mapping filter: '
2016-01-04 17:20:34.100  INFO 81259 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @Cont
d.AnnotationConfigEmbeddedWebApplicationContext@792a5364: startup date [Mon Jan 04 17:20:32 CET 2016]; root of context
2016-01-04 17:20:34.181  INFO 81259 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/greeti
```

It will works the same way as before very handy for a build !

Enabling Security

Security is essential and should exists at the very beginning of every project. In order to understand how security works within Spring Framework, we are going to use again a Spring Guide. This time the guide used is “Securing Web”.

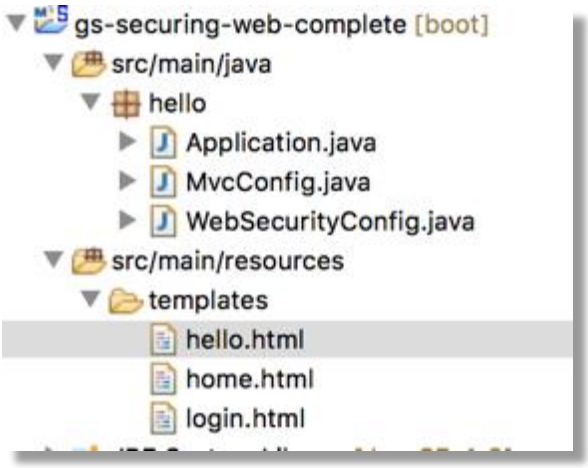


Once all the libraries are downloaded you can go and check the new project called gs-securing-web-complete.

Thymeleaf Views

This time the project is a bit different because we have a GUI and Views. Thymeleaf is rendering engine that replaces JSPs that were since the beginning the default rendering engine for Spring. The rendered pages are built on the server side and sent back to the client.

It uses a template system that relies on HTML pages with tag extensions.



These views are stored in `template` folder under `src/main/resources`.

The generated views are : `hello.html`

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
    <head>
        <title>Hello World!</title>
    </head>
    <body>
        <h1 th:inline="text">Hello [[${#httpServletRequest.remoteUser}]]!</h1>
        <form th:action="@{/logout}" method="post">
            <input type="submit" value="Sign Out"/>
        </form>
    </body>
</html>
```

View `home.html` :

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
    <head>
        <title>Spring Security Example</title>
    </head>
    <body>
        <h1>Welcome!</h1>

        <p>Click <a th:href="@{/hello}">here</a> to see a greeting.</p>
    </body>
</html>
```

View login.html

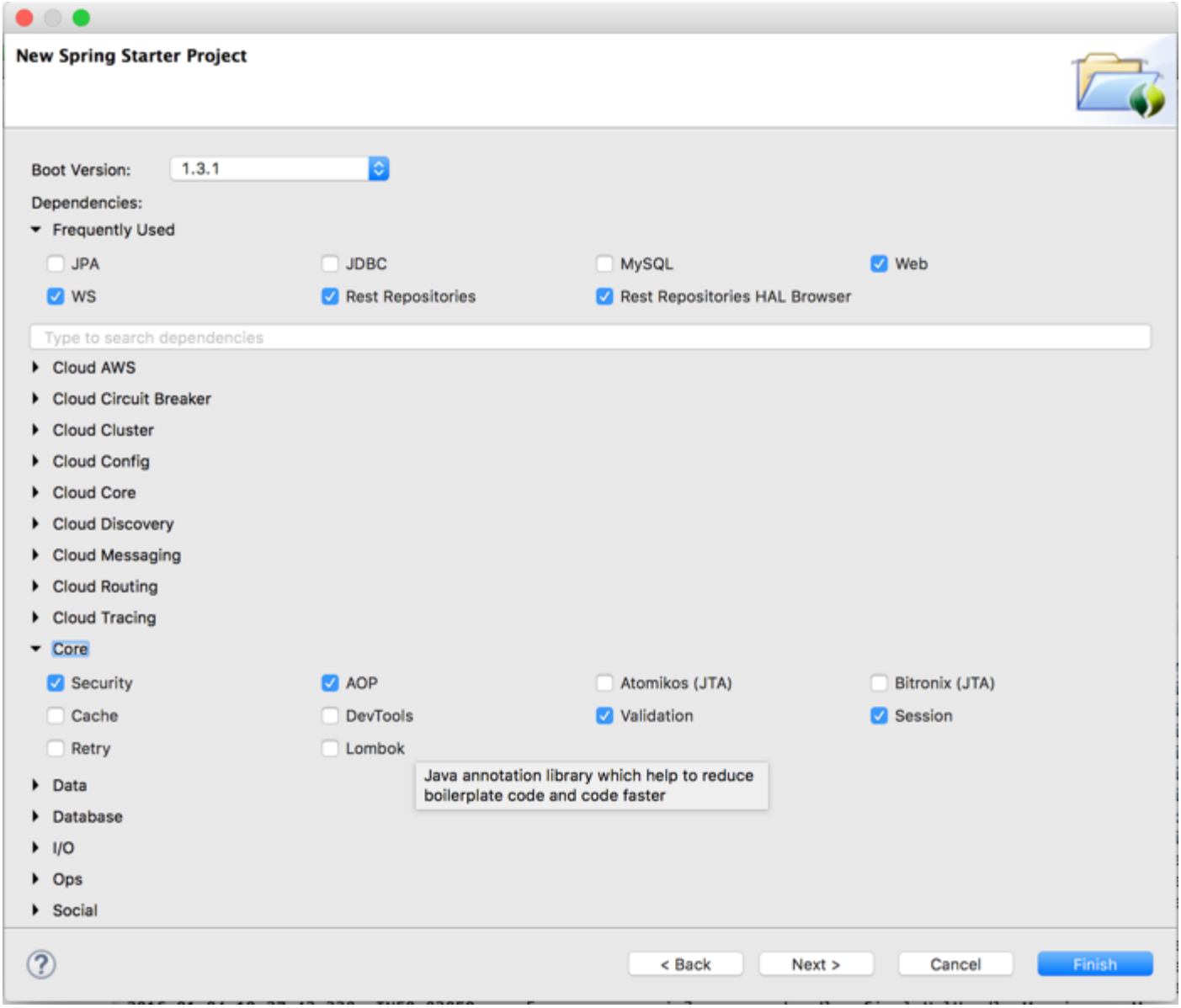
```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
    <head>
        <title>Spring Security Example </title>
    </head>
    <body>
        <div th:if="${param.error}">
            Invalid username and password.
        </div>
        <div th:if="${param.logout}">
            You have been logged out.
        </div>
        <form th:action="@{/login}" method="post">
            <div><label> User Name : <input type="text" name="username"/> </label></div>
            <div><label> Password: <input type="password" name="password"/> </label></div>
            <div><input type="submit" value="Sign In"/></div>
        </form>
    </body>
</html>
```

The MVC

The MvcConfig.java is simply binding the URLs and the ThymeLeaf views

```
package hello;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
```

```
@Configuration
public class MvcConfig extends WebMvcConfigurerAdapter
{

      @Override
      public void addViewControllers(ViewControllerRegistry registry)
      {
            registry.addViewController("/home").setViewName("home");
            registry.addViewController("/").setViewName("home");
            registry.addViewController("/hello").setViewName("hello");
            registry.addViewController("/login").setViewName("login");
      }

}
```



Configure the pom.xml

Add to dependencies the dependency of the DAO layer:

```
    <dependency>
      <groupId>isep.web.sakila</groupId>
      <artifactId>dao</artifactId>
      <version>0.0.1-SNAPSHOT</version>
    </dependency>
```

Building The Service Layer

Start with an empty project. To do so, this select "New Spring Starter Project" and select no checkbox and name the project `sakila-business-webapi`.

The Configuration

Pom.xml

Replace the pom.xml by the one below:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>isep.web.sakila</groupId>
  <artifactId>wepapi</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>sakila-business-webapi</name>
  <description>Web API for Sakila Project</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.1.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-aop</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-ws</artifactId>
    </dependency>
```

```
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
    </dependency>
    <dependency>
      <groupId>commons-dbcp</groupId>
      <artifactId>commons-dbcp</artifactId>
    </dependency>
    <dependency>
      <groupId>commons-pool</groupId>
      <artifactId>commons-pool</artifactId>
    </dependency>
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
    </dependency>
    <dependency>
      <groupId>com.google.guava</groupId>
      <artifactId>guava</artifactId>
      <version>16.0.1</version>
    </dependency>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
    </dependency>
    <dependency>
      <groupId>isep.web.sakila</groupId>
      <artifactId>sakila-business-dao</artifactId>
      <version>0.0.1-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</artifactId>
      <version>1.2</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>


</project>
```

Boot class

The create your boot class SakilaBusinessWebapiApplication. You can see how to take advantage of Logging.Logging level is configured trough the `application.properties` file.

```
package isep.web.sakila.webapi;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication
public class SakilaBusinessWebapiApplication
{
	private static final Log log = LogFactory.getLog(SakilaBusinessWebapiApplication.class);

	public static void main(String[] args)
	{
		log.debug(String.format("****************************************"));
		log.debug(String.format("  Starting Embeded Tomcat"));
		log.debug(String.format("****************************************"));
		SpringApplication.run(SakilaBusinessWebapiApplication.class, args);
		log.debug(String.format("****************************************"));
	}
}
```

Logging configuration

The file `application.properties` under the directory `src/main/resources` should be as follow.

```
# LOGGING
logging.level.org.springframework.web=DEBUG
logging.level.org.hibernate=ERROR
```

Configuration Class

This empty class is very important because it configures the Spring framework. Here you can see :

- @Import: that makes the link with the DAO layer
- @ComponentScan: that indicates what is the root package of your beans.

```
package isep.web.sakila.webapi.config;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

import isep.web.sakila.jpa.config.PersistenceConfig;

@Configuration
@EnableAutoConfiguration
@ComponentScan(basePackages = { "isep.web.sakila.webapi" })
@Import({ PersistenceConfig.class })
public class MvcConfig extends WebMvcConfigurerAdapter
{

}
```

The Service Layer

The model

We are creating a model that will creates the abstraction layer between the JPA component that represents the Database and the object `ActorWO` that will makes the connection between the service layer and the presentation layer. Here is our simple component

```
package isep.web.sakila.webapi.model;

import isep.web.sakila.jpa.entities.Actor;

public class ActorWO extends WebObject
{

	private static final long serialVersionUID	= -1377067679473844279L;

	protected int 	actorId;
	protected String 	lastName;
	protected String 	firstName;

	public ActorWO()
	{
		super();
	}

	public ActorWO(int actorId, String lastName, String firstName)
	{
		super();
		this.actorId = actorId;
		this.lastName = lastName;
		this.firstName = firstName;
	}

	public ActorWO(final Actor actor)
	{
		super();
		this.actorId = actor.getActorId();
		this.lastName = actor.getLastName();
		this.firstName = actor.getFirstName();
	}

	public String getFirstName()
	{
		return firstName;
	}

	public int getActorId()
	{
		return actorId;
	}

	public String getLastName()
	{
		return lastName;
	}

	public void setFirstName(String firstName)
	{
		this.firstName = firstName;
	}

	public void setActorId(int actorId)
	{
		this.actorId = actorId;
	}
```

```
	public void setLastName(String lastName)
	{
		this.lastName = lastName;
	}

	@Override
	public String toString()
	{
		return "Actor [id=" + this.actorId + ", LastNanem=" + this.lastName + ", First=" +
this.firstName + "]";
	}
}
```

The mother class of all the components.

```
package isep.web.sakila.webapi.model;

import java.io.Serializable;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class WebObject implements Serializable
{
	private static final long serialVersionUID	= 3319176093424235711L;

	@SuppressWarnings("unused")
	private static final Log	log	=
LogFactory.getLog(WebObject.class);

	@Override
	public boolean equals(Object obj)
	{
		if (this == obj)
		{
			return true;
		}
		if (obj == null)
		{
			return false;
		}
		return true;
	}
}
```

The interface of the service Layer

For each service you will provide, you will need a specific service Interface and, at least, one implementation of the underlying service.

```
package isep.web.sakila.webapi.service;

import java.util.List;

import isep.web.sakila.webapi.model.ActorWO;

public interface ActorService
{
	ActorWO findById(int id);

	void saveActor(ActorWO userWO);

	void updateActor(ActorWO userWO);

	void deleteActorById(int id);
```

```
	List<ActorWO> findAllActors();

}
```

The Service implementation

The service enables the binding between the Data and the service provided the client. It is closing to the CRUD services because the component is simple. But the more complex the service is the more comple that class will be.

Important points here are :

- The name of the service (@Service("actorService"))
- @Autowired that injects the instance of actorRepository that enables the JPA access…

```
package isep.web.sakila.webapi.service;

import java.sql.Timestamp;
import java.util.LinkedList;
import java.util.List;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import isep.web.sakila.dao.repositories.ActorRepository;
import isep.web.sakila.jpa.entities.Actor;
import isep.web.sakila.webapi.model.ActorWO;

@Service("actorService")
@Transactional
public class ActorServiceImpl implements ActorService
{
	@Autowired
	private ActorRepository		actorRepository;

	private static final Log	log	= LogFactory.getLog(ActorServiceImpl.class);

	public List<ActorWO> findAllActors()
	{
		List<ActorWO> actors = new LinkedList<ActorWO>();

		for (Actor actor : actorRepository.findAll())
		{
			actors.add(new ActorWO(actor));
			log.debug("Adding " + actor);
		}

		return actors;
	}

	public ActorWO findById(int id)
	{
		log.debug(String.format("Looking for user by Id %s", id));
		Actor actor = actorRepository.findOne(id);

		if (actor != null)
		{
			return new ActorWO(actor);
		}
		return null;
	}

	public void saveActor(ActorWO actorWO)
	{
```

```
            Actor actor = new Actor();
            actor.setLastName(actorWO.getLastName());
            actor.setFirstName(actorWO.getFirstName());
            actor.setLastUpdate(new Timestamp(System.currentTimeMillis()));
            actorRepository.save(actor);
      }

      public void updateActor(ActorWO actorWO)
      {
            Actor actor2update = actorRepository.findOne(actorWO.getActorId());
            actor2update.setLastName(actorWO.getLastName());
            actor2update.setFirstName(actorWO.getFirstName());
            actor2update.setLastUpdate(new Timestamp(System.currentTimeMillis()));
            actorRepository.save(actor2update);
      }

      @Override
      public void deleteActorById(int id)
      {
            actorRepository.delete(id);
      }

}
```

The Controller

Last but not least Java class… the Controller. It makes the link between the service (named actorService and Autowired…) and the REST requests from the GUI.

In that class you usually find the binding of data and request handling to handle the protocol disruption between HTTP and Java.

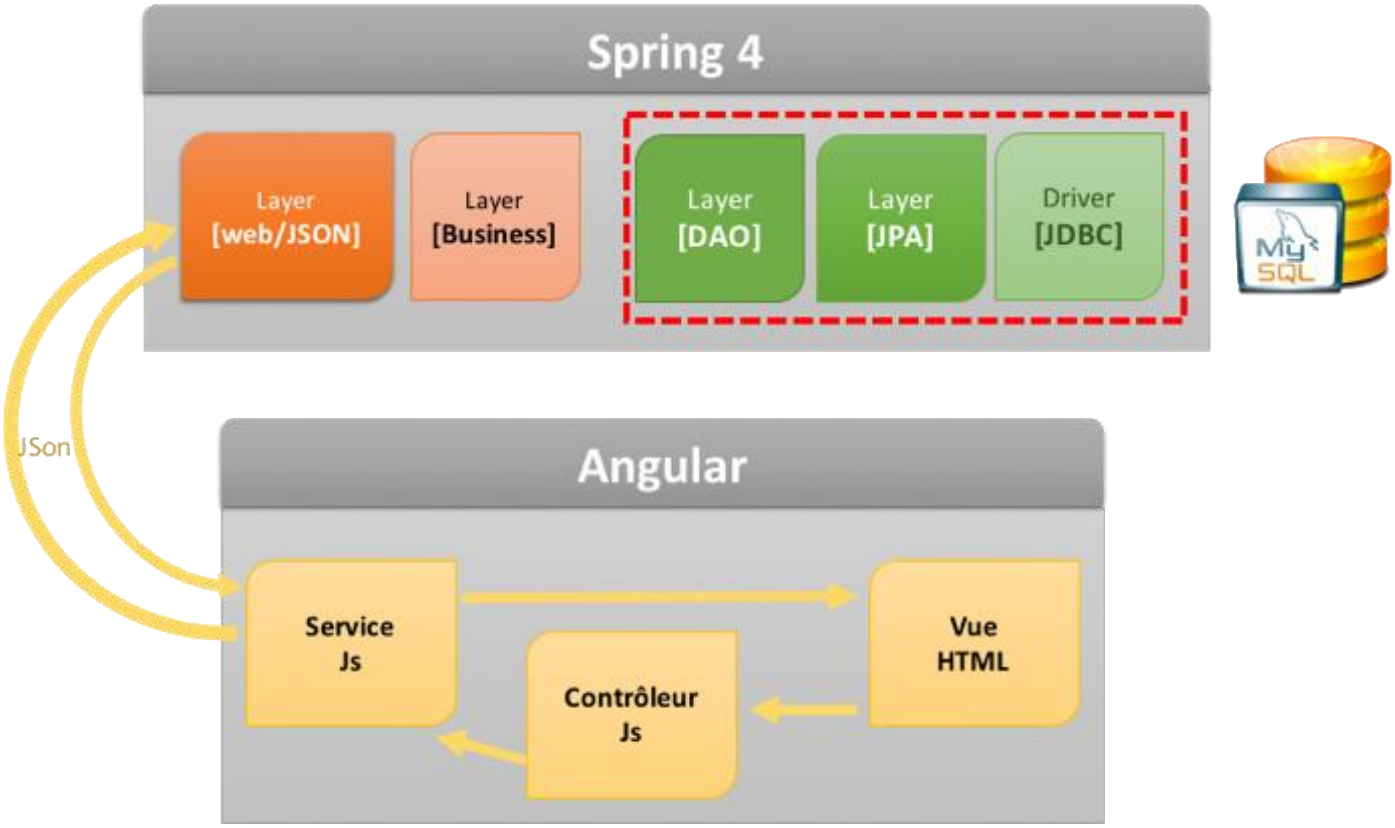
```
package isep.web.sakila.webapi.controller;

import java.util.List;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.util.UriComponentsBuilder;

import isep.web.sakila.webapi.model.ActorWO;
import isep.web.sakila.webapi.service.ActorService;

@RestController
public class ActorRestController
{

      @Autowired
      ActorService                                          actorService;

      private static final Log   log    = LogFactory.getLog(ActorRestController.class);

      @RequestMapping(value = "/actor/", method = RequestMethod.GET)
      public ResponseEntity<List<ActorWO>> listAllActors()
```

```
	{
		List<ActorWO> actors = actorService.findAllActors();
		if (actors.isEmpty())
		{
			return new ResponseEntity<List<ActorWO>>(HttpStatus.NO_CONTENT);
		}
		return new ResponseEntity<List<ActorWO>>(actors, HttpStatus.OK);
	}

	@RequestMapping(value = "/actor/{id}", method = RequestMethod.GET, produces =
MediaType.APPLICATION_JSON_VALUE)
	public ResponseEntity<ActorWO> getActor(@PathVariable("id") int id)
	{
		System.out.println("Fetching Actor with id " + id);
		ActorWO staffWO = actorService.findById(id);
		if (staffWO == null)
		{
			System.out.println("Actor with id " + id + " not found");
			return new ResponseEntity<ActorWO>(HttpStatus.NOT_FOUND);
		}
		return new ResponseEntity<ActorWO>(staffWO, HttpStatus.OK);
	}

	// -------------------Create a Actor----------------------------------

	@RequestMapping(value = "/actor/", method = RequestMethod.POST)
	public ResponseEntity<Void> createActor(@RequestBody ActorWO actorWO,
UriComponentsBuilder ucBuilder)
	{
		System.out.println("Creating Actor " + actorWO.getLastName());

		actorService.saveActor(actorWO);

		HttpHeaders headers = new HttpHeaders();

	headers.setLocation(ucBuilder.path("/actor/{id}").buildAndExpand(actorWO.getActorId()).to
Uri());
		return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
	}

	@RequestMapping(value = "/actorUpdate/", method = RequestMethod.POST)
	public ResponseEntity<ActorWO> updateActor(@RequestBody ActorWO actorWO,
UriComponentsBuilder ucBuilder)
	{
		log.error(String.format("Updating Actor %s ", actorWO.getActorId()));
		ActorWO currentActor = actorService.findById(actorWO.getActorId());

		if (currentActor == null)
		{
			System.out.println("Actor with id " + actorWO.getActorId() + " not found");
			return new ResponseEntity<ActorWO>(HttpStatus.NOT_FOUND);
		}

		currentActor.setLastName(actorWO.getLastName());
		currentActor.setFirstName(actorWO.getFirstName());
		actorService.updateActor(currentActor);

		return new ResponseEntity<ActorWO>(currentActor, HttpStatus.OK);
	}

	@RequestMapping(value = "/actorDelete/{id}", method = RequestMethod.GET)
	public ResponseEntity<ActorWO> deleteActor(@PathVariable("id") int id)
	{

		System.out.println("Fetching & Deleting Actor with id " + id);

		ActorWO staffWO = actorService.findById(id);
		if (staffWO == null)
		{
			System.out.println("Unable to delete. Actor with id " + id + " not found");
			return new ResponseEntity<ActorWO>(HttpStatus.NOT_FOUND);
		}
```

```
            actorService.deleteActorById(id);
            return new ResponseEntity<ActorWO>(HttpStatus.NO_CONTENT);
        }
}
```

AngularJS front end

Our front end is written in Angular. It implements a MVC model as shownin the schema below:

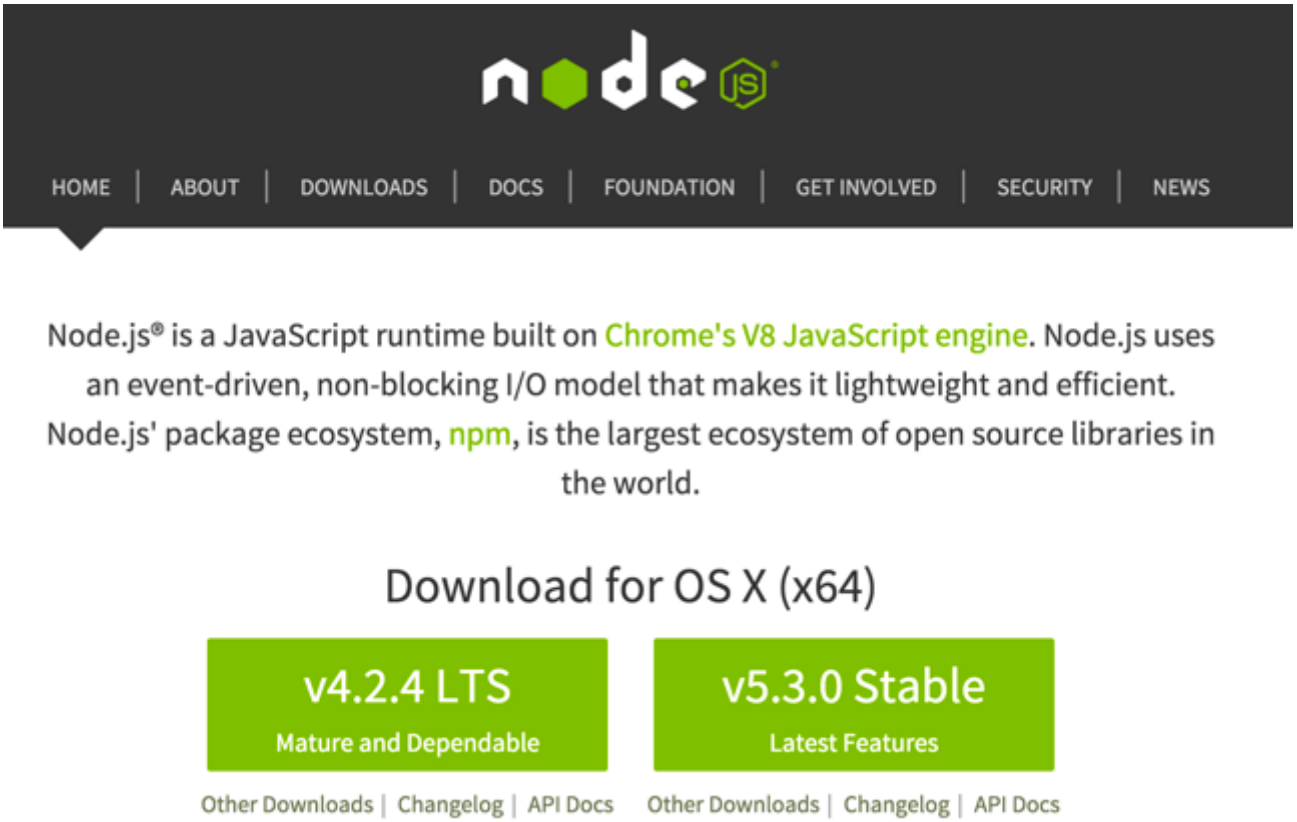


AngularJS intallation

Take a breath... You can use directly the web link of the AngularJS sources or create a real development environment from the command line. We have done for the example the first case, but you will quickly need for development purpose a real development environment and an abstract server that helps you to go faster.

To build you environment you will need Node.JS, bower and gulp.

Bower Installation



this will give us the `npm` command line utility ☺

Bower Installation

Bower is a quid of maven for AngularJS. It is handy to get libraries et build your project. You can do it with npm, but it is much more simple to perform these tasks with bower.

Bower installation for everyone is done

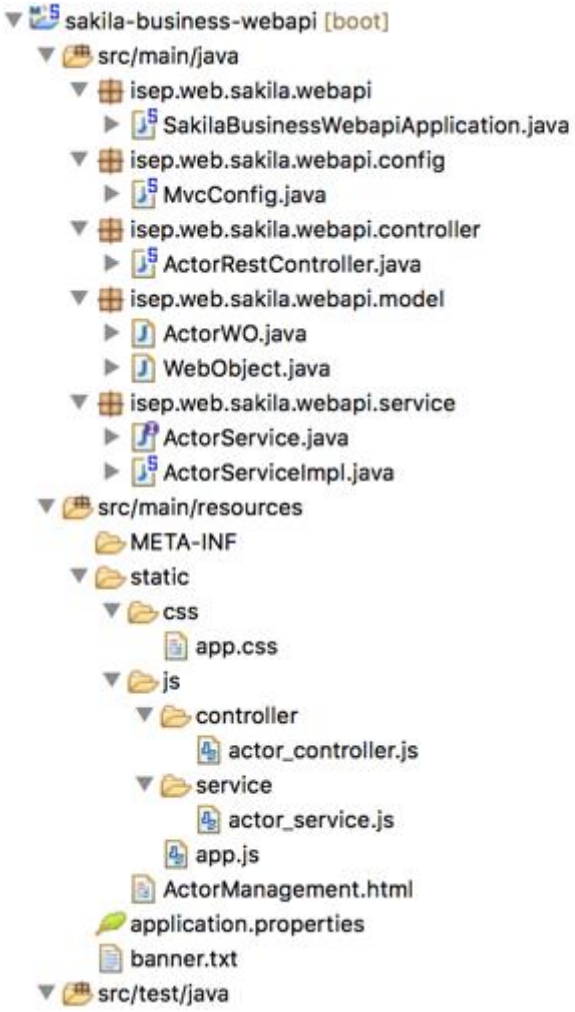
```
MacBook-Pro-de-Olivier:~$ sudo npm install -g bower
Password:
/usr/local/bin/bower -> /usr/local/lib/node_modules/bower/bin/bower
/usr/local/lib
└─┬ bower@1.7.2
  └── semver-utils@1.1.1
```

It is the very easy to install jquery in one command...

```
MacBook-Pro-de-Olivier:~$ bower install jquery
? May bower anonymously report usage statistics to improve the tool over time? Yes
bower not-cached    git://github.com/jquery/jquery.git#*
bower resolve       git://github.com/jquery/jquery.git#*
bower download      https://github.com/jquery/jquery/archive/2.1.4.tar.gz
bower extract       jquery#* archive.tar.gz
bower resolved      git://github.com/jquery/jquery.git#2.1.4
bower install       jquery#2.1.4
jquery#2.1.4 bower_components/jquery
```

Going Back to STS project

Our final project looks like this:



File `static/css/app.css`

First we create the CSS file named `app.css` it located in the `static/css` folder.

```
body, #mainWrapper {
      height: 70%;
      background-color:rgb(245, 245, 245);
}

body, .form-control{
      font-size:12px!important;
}

.floatRight{
      float:right;
      margin-right: 18px;
}

.has-error{
      color:red;
}

.formcontainer{
      background-color: #DAE8E8;
      padding: 20px;
```

```
}

.tablecontainer{
      padding-left: 20px;
}

.generic-container {
  width:80%;
  margin-left: 20px;
  margin-top: 20px;
  margin-bottom: 20px;
  padding: 20px;
  background-color: #EAE7E7;
  border: 1px solid #ddd;
  border-radius: 4px;
  box-shadow: 0 0 30px black;
}

.custom-width {
    width: 80px !important;
}
```

File js/app.js

This is the global definition of our AngularJS application.

```
'use strict';

var App = angular.module('myApp',[]);
```

File js/controller/actor_controller.js

This is the AngularJS controller it is called by the HTML GUI and it calls the Angular service layer in order to perform the REST WebServices calls.

```
'use strict';

App.controller('ActorController', ['$scope', 'ActorService', function($scope, ActorService) {
          var self = this;
          self.actor={actorId:null,lastName:'',firstName:''};
          self.actors=[];

          self.fetchAllActors = function(){
              ActorService.fetchAllActors()
                  .then(
                                                function(d) {
                                                        self.actors = d;
                                                },
                                        function(errResponse){
                                                console.error('Error while fetching Currencies');
                                        }
                                );
          };

          self.createActor = function(actor){
              ActorService.createActor(actor)
                          .then(
                      self.fetchAllActors,
                                        function(errResponse){
                                                console.error('Error while creating Actor.');
                                        }
                  );
          };

         self.updateActor = function(actor){
              ActorService.updateActor(actor)
```

```
            .then(
                    self.fetchAllActors,
                                        function(errResponse){
                                                console.error('Error while updating Actor.');
                                        }
                );
        };

       self.deleteActor = function(actorId){
            ActorService.deleteActor(actorId)
                        .then(
                                        self.fetchAllActors,
                                        function(errResponse){
                                                console.error('Error while deleting Actor.');
                                        }
                );
        };

        self.fetchAllActors();

        self.submit = function() {
            if(self.actor.actorId==null){
                console.log('Saving New Actor', self.actor);
                self.createActor(self.actor);
            }else{
                console.log('Actor updating with id ', self.actor.actorId);
                console.log('Actor: ', self.actor);
                self.updateActor(self.actor);
            }
            self.reset();
        };

        self.edit = function(actorId){
            console.log('id to be edited', actorId);
            for(var i = 0; i < self.actors.length; i++){
                if(self.actors[i].actorId == actorId) {
                   self.actor = angular.copy(self.actors[i]);
                   break;
                }
            }
        };

        self.remove = function(actorId){
            console.log('id to be deleted', actorId);
            for(var i = 0; i < self.actors.length; i++){
                if(self.actors[i].actorId == actorId) {
                   self.reset();
                   break;
                }
            }
            self.deleteActor(actorId);
        };


        self.reset = function(){
            self.actor={actorId:null,lastName:'',firstName:''};
            $scope.myForm.$setPristine(); //reset Form
        };

    }]);
```

File js/ service /actor_service.js

This file is the service component. It calls the REST Services.

```
'use strict';
```

```
App.factory('ActorService', ['$http', '$q', function($http, $q){

      return {

                    fetchAllActors: function() {
                                  return $http.get('http://localhost:8080/actor/')
                                                .then(
                                                              function(response){
                                                                     return response.data;
                                                              },
                                                              function(errResponse){
                                                                     console.error('Error while
fetching actors');
                                                                     return
$q.reject(errResponse);
                                                              }
                                                );
                    },

                 createActor: function(actor){
                                  return $http.post('http://localhost:8080/actor/', actor)
                                                .then(
                                                              function(response){
                                                                     return response.data;
                                                              },
                                                              function(errResponse){
                                                                     console.error('Error while
creating actor');
                                                                     return
$q.reject(errResponse);
                                                              }
                                                );
                 },

                 updateActor: function(actor, actorId){
                 console.log("XXX", actor);
                                  return $http.post('http://localhost:8080/actorUpdate/', actor)
                                                .then(
                                                              function(response){
                                                                     return response.data;
                                                              },
                                                              function(errResponse){
                                                                     console.error('Error while
updating actor');
                                                                     return
$q.reject(errResponse);
                                                              }
                                                );
                    },

                    deleteActor: function(actorId){
                                  return $http.get('http://localhost:8080/actorDelete/'+actorId)
                                                .then(
                                                              function(response){
                                                                     return response.data;
                                                              },
                                                              function(errResponse){
                                                                     console.error('Error while
deleting actor');
                                                                     return
$q.reject(errResponse);
                                                              }
                                                );
                    }

      };

}]);
```

File ActorManagement.html

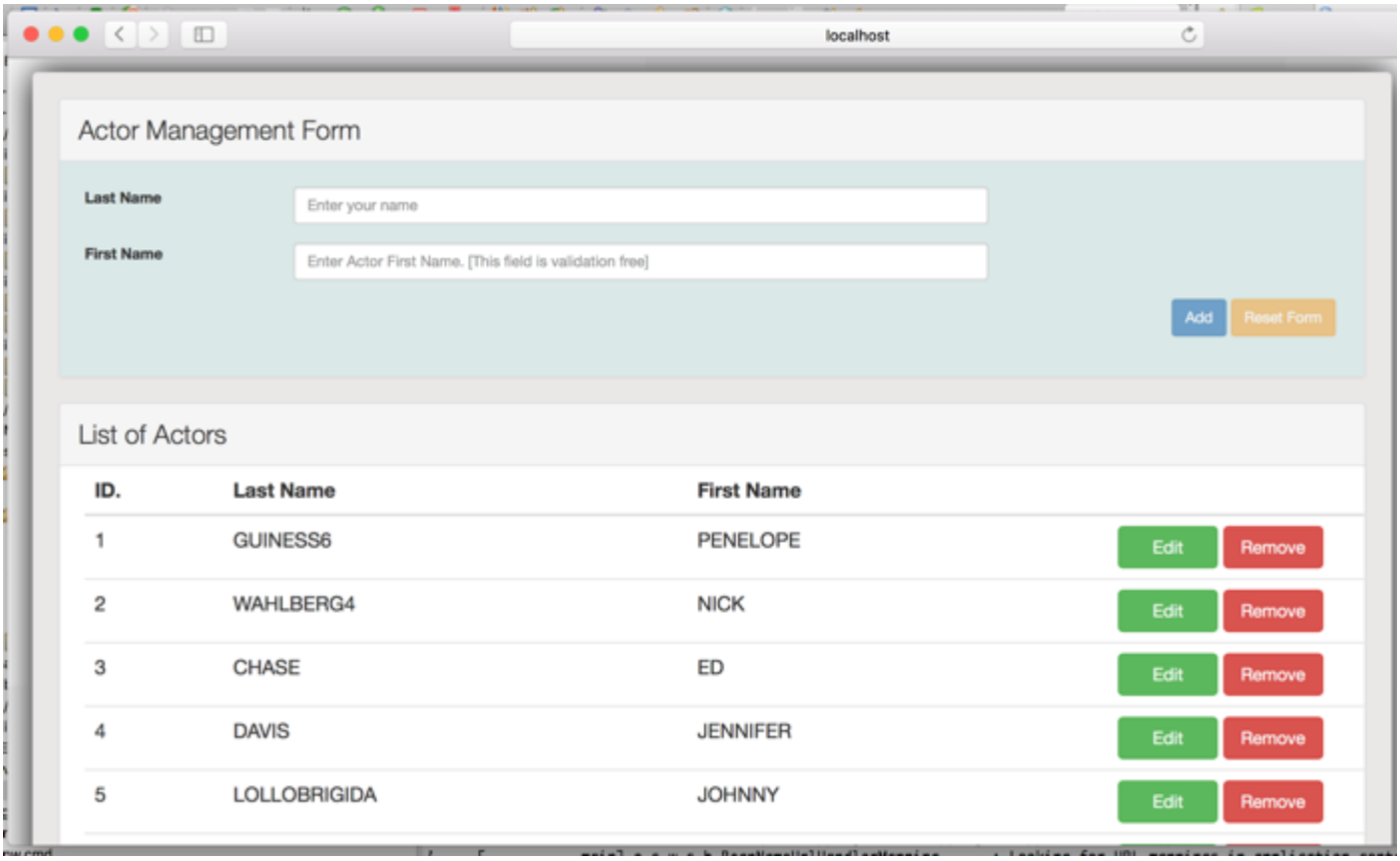
This is the view.

```
<html>
  <head>
    <title>Actor Management</title>
    <style>
      .username.ng-valid {
          background-color: lightgreen;
      }
      .username.ng-dirty.ng-invalid-required {
          background-color: red;
      }
      .username.ng-dirty.ng-invalid-minlength {
          background-color: yellow;
      }

      .email.ng-valid {
          background-color: lightgreen;
      }
      .email.ng-dirty.ng-invalid-required {
          background-color: red;
      }
      .email.ng-dirty.ng-invalid-email {
          background-color: yellow;
      }

    </style>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
    <link href="css/app.css" rel="stylesheet"></link>
  </head>
  <body ng-app="myApp" class="ng-cloak">
      <div class="generic-container" ng-controller="ActorController as ctrl">
          <div class="panel panel-default">
              <div class="panel-heading"><span class="lead">Actor Management Form
</span></div>
              <div class="formcontainer">
                  <form ng-submit="ctrl.submit()" name="myForm" class="form-horizontal">
                      <input type="hidden" ng-model="ctrl.actor.actorId" />
                      <div class="row">
                          <div class="form-group col-md-12">
                              <label class="col-md-2 control-lable" for="file">Last
Name</label>
                              <div class="col-md-7">
                                  <input type="text" ng-model="ctrl.actor.lastName"
name="lastName" class="form-control input-sm" placeholder="Enter your name" required ng-
minlength="3"/>
                                  <div class="has-error" ng-show="myForm.$dirty">
                                      <span ng-show="myForm.lastName.$error.required">This is
a required field</span>
                                      <span ng-show="myForm.lastName.$error.minlength">Minimum
length required is 3</span>
                                      <span ng-show="myForm.lastName.$invalid">This field is
invalid </span>
                                  </div>
                              </div>
                          </div>


                      <div class="row">
                          <div class="form-group col-md-12">
                              <label class="col-md-2 control-lable" for="file">First
Name</label>
                              <div class="col-md-7">
                                  <input type="text" ng-model="ctrl.actor.firstName"
name="firstName"  class="form-control input-sm" placeholder="Enter Actor First Name. [This
field is validation free]"/>
                                  <div class="has-error" ng-show="myForm.$dirty">
```

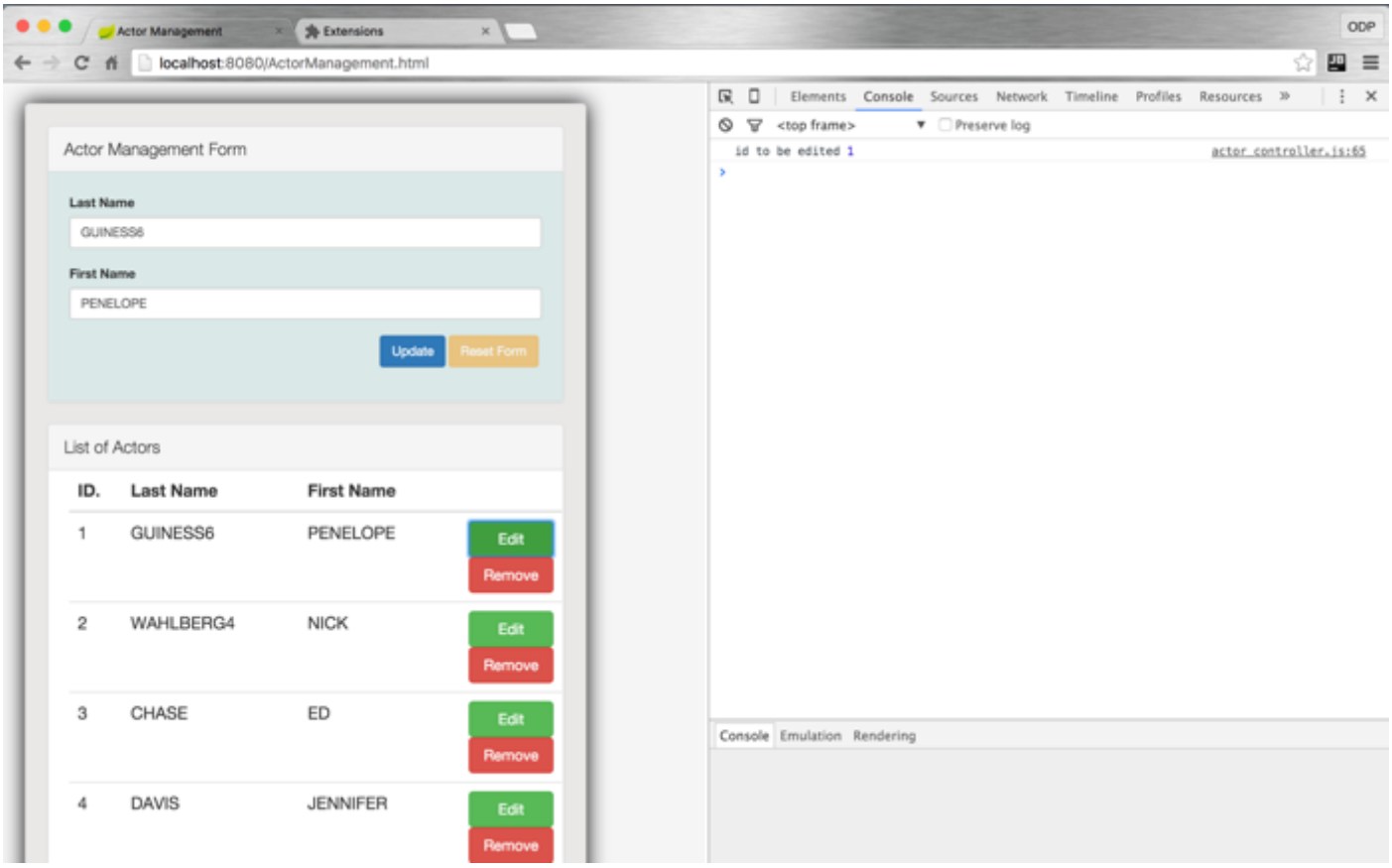
```
                                        <span ng-show="myForm.firstName.$error.required">This is
a required field</span>
                                    </div>
                                </div>
                            </div>
                        </div>

                        <div class="row">
                            <div class="form-actions floatRight">
                                <input type="submit"  value="{{!ctrl.actor.actorId ? 'Add' :
'Update'}}" class="btn btn-primary btn-sm" ng-disabled="myForm.$invalid">
                                <button type="button" ng-click="ctrl.reset()" class="btn btn-
warning btn-sm" ng-disabled="myForm.$pristine">Reset Form</button>
                            </div>
                        </div>
                    </form>
                </div>
            </div>
            <div class="panel panel-default">
                  <!-- Default panel contents -->
                <div class="panel-heading"><span class="lead">List of Actors </span></div>
                <div class="tablecontainer">
                    <table class="table table-hover">
                        <thead>
                            <tr>
                                <th>ID.</th>
                                <th>Last Name</th>
                                <th>First Name</th>
                                <th width="20%"></th>
                            </tr>
                        </thead>
                        <tbody>
                            <tr ng-repeat="u in ctrl.actors">
                                <td><span ng-bind="u.actorId"></span></td>
                                <td><span ng-bind="u.lastName"></span></td>
                                <td><span ng-bind="u.firstName"></span></td>
                                <td>
                                <button type="button" ng-click="ctrl.edit(u.actorId)" class="btn
btn-success custom-width">Edit</button>  <button type="button" ng-
click="ctrl.remove(u.actorId)" class="btn btn-danger custom-width">Remove</button>
                                </td>
                            </tr>
                        </tbody>
                    </table>
                </div>
            </div>
        </div>

        <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"></script>
        <script src="js/app.js"></script>
        <script src="js/service/actor_service.js""></script>
        <script src="js/controller/actor_controller.js"></script>
    </body>
</html>
```

Screen shots…

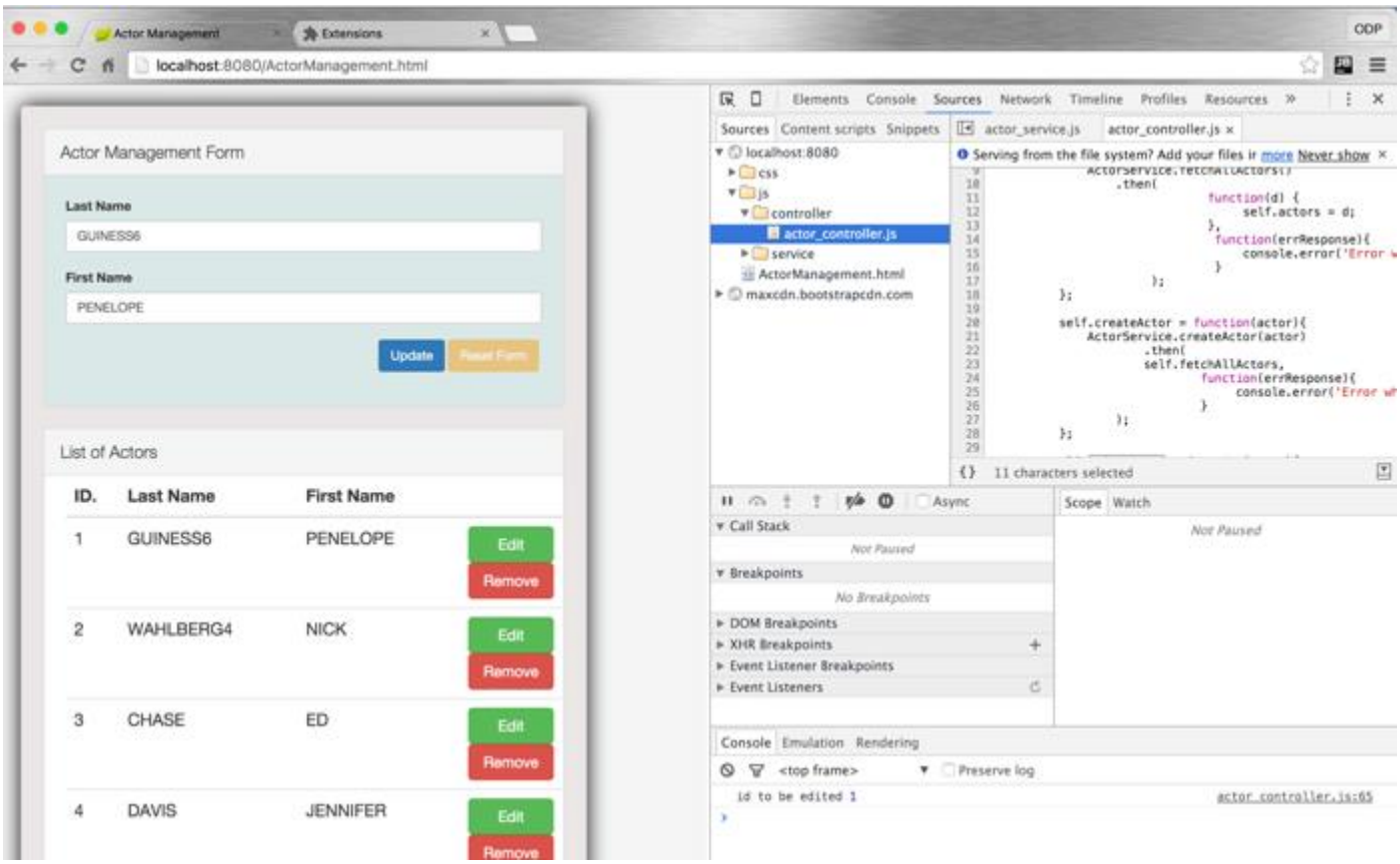
An screenshot of the final result:



An screenshot of debugging tool under chrome with the console:



You can notice that you can dynamically edit the code…



Under chrome the wonderfull plugin "Avanced Rest Client App"….