**Intro**

Via the URL <https://spring.io/guides/> you can find numerous getting started guides.

For reference documentation you can navigate to <http://docs.spring.io/spring-boot/docs/1.1.x/reference/htmlsingle/>

**Lab 1**

**Create a new REST service with Spring Boot**

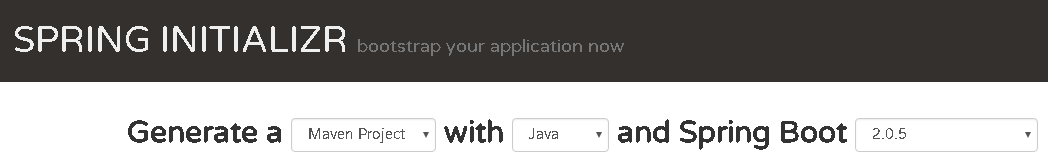
In this lab you are going to create a first REST service with Springboot. This REST service will (for now) retrieve a list of drinks. After creation you are going to do some tests in which you call the service.

**Create your first project**

For the creation of a new Springboot service you can make use of Spring Initializr, which generates an ‘out of the box’ project structure for you. For learning purposes you will make use of the web based Spring Initializr first. Besides, Spring Initializr is also available in popular IDE’s such as Eclipse and IntelliJ

* Go to the Spring Initializr web page at <http://start.spring.io/>.

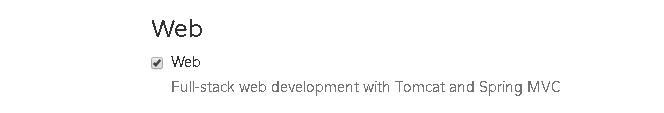
From here you will generate a new Java project, which uses the Springboot framework and Maven as build tooling. Another available build technology is Gradle, which won’t be used forthis lab.

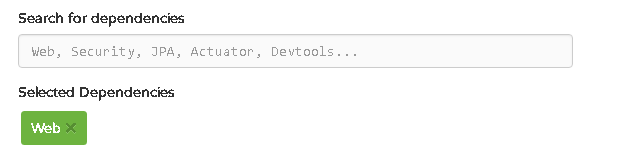


* On the web page choose ‘Switch to the full version



* Name the Artifact ‘shoppingservice’
* Fill in ‘demo project for Spring Boot’ for the description
* Change package name to: nl.sjop.service
* Set the packaging option to jar
* Choose java version 8 (or higher, depending on your installed JDK version)
* Choose WEB as a dependency





* Generate the project

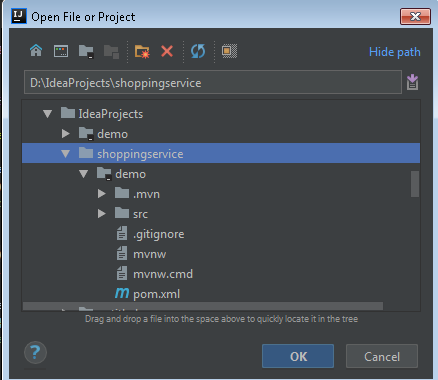


If things go well, you now got a zip file wich contains the project structure

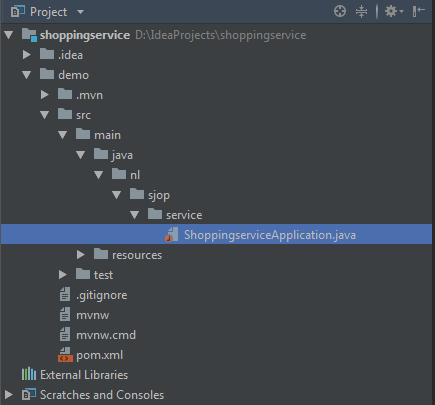
* Unzip the generated file into a directory of your choice
* Open the project in your favorite IDE

**Opening the project in IntelliJ**

* From the top main menu choose : file -> open project
* Select the unzipped project created by the Spring Initializer (in the shown case below, this is D:\IdeaProjects\shoppingservice)

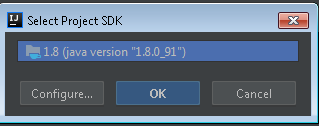


* Navigate to the Java class named ShoppingserviceApplication and double click the file to see its content

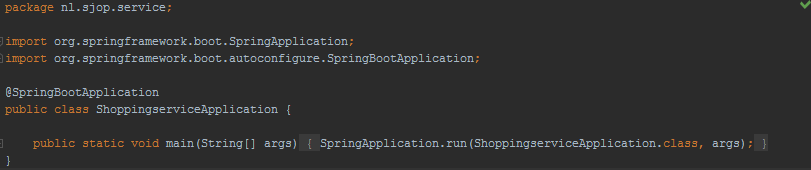


* If the below message is shown in the editor screen, select the ‘setup SDK’ link and choose your SDK (at least Java 8).

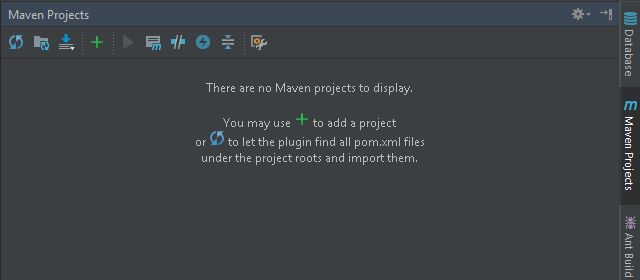
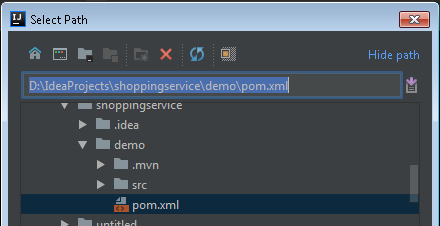
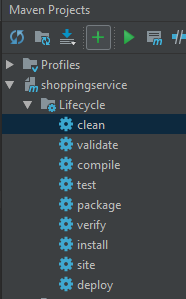
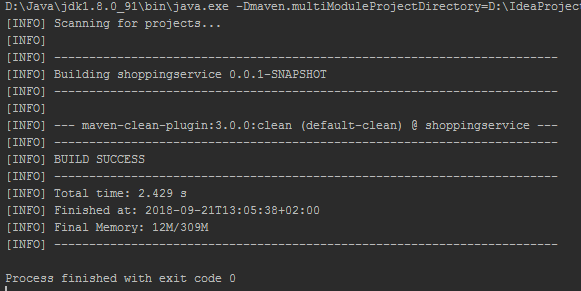




In the editor you should now see the following Java class code



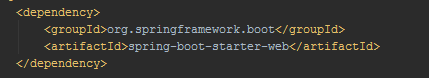
Setting Maven fort his project

* From the tab options on the right side of the screen, choose maven 
* Press the + button
* Navigate to and select the pom.xml file for this project
* IntelliJ recognizes this is a Maven project. It should now show some lifecycles. Double click the option ‘clean’ from the lifecycle options
* Check the message if the build of the projects is succesfull
* 

**Dependencies**

Spring Initializr generated a file called ‘pom.xml’, based on the technologies chosen for the project. Because you selected web as a technology for the project, the spring-boot-starter-web was added as a dependency

Pom.xml is used by the build tool maven and contains info on the libraries used in the project. At build time these dependent libraries are downloaded from a repository, configured in the settings for maven and placed on the projects classpath.



The spring-boot-starter-web provides access to technologies frequently used for building web applications, such as Spring for the Model ViewController and,Tomcat as the default embedded container.

The starter is kind of an umbrella specification, under which a bunch of dependent libraries are bundled and for which the collaboration is tested/guaranteed

To find out which (transitive) dependencies are included in your project, you can run the following maven command from the command line:

**mvn dependency:tree**

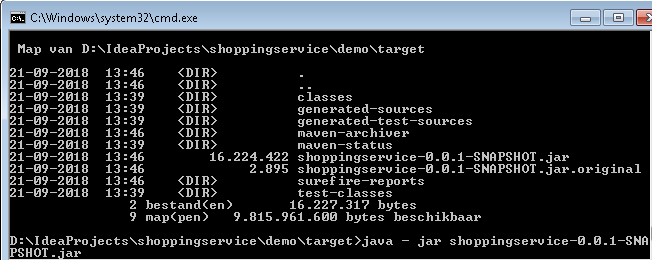
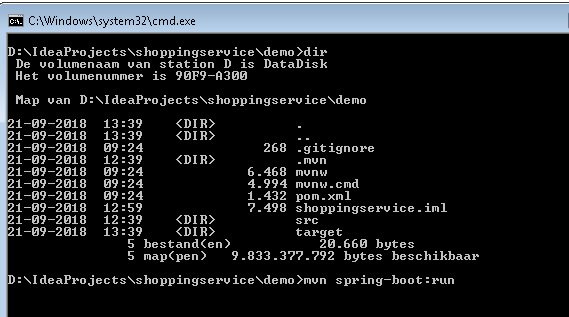
As you can see there are a number of additional dependencies, including the Tomcat web server and Spring Boot itself.

**Running the application**

Spring Initizalizr has generated an application class named ShoppingserviceApplication.java. If you have not opened this file in your editor already, than open it now. As you can see it contains a method called main, which is the entry point of the program.

The main method receives argument values of the type String from the command line. A String is an (immutable) sequence of characters in Java

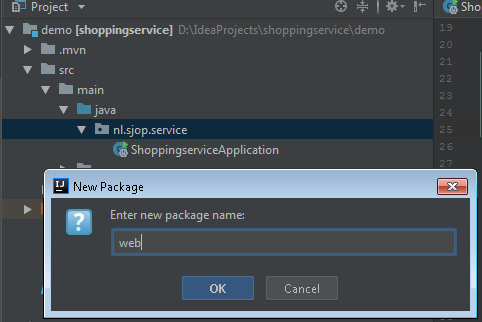
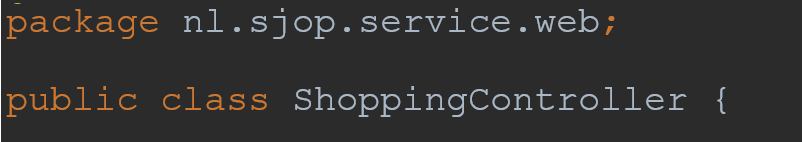
There are some ways to execute your application:

1. With the ShoppingserviceApplication in your IDE editor, you can run (or debug) it from the menu options.
2. Package the application via the mvn package lifecycle. After packaging has finished, a jar file has been generated in a new folder called target. Run the application from this jar file via the command**: java –jar [projectname]-0.0.1-SNAPSHOT.jar** 
3. From the command line: navigate to the folder wich contains the pom.xml file and type **mvn spring-boot run** 

* Try it out.
* What output do you see from the application and why?

**Add a JSON Controller**

To give your application some functionality which can be accessed fom a web client (eg. Browser) you are going to add a rest controller to your project.

* Add a new folder to the package ‘nl.sjop.service’ and name it ‘web’
* Add a new Java class to the web folder and name it ‘ShoppingController.java’

To ‘mark’ this class as a REST controller you need to annotate it.

* Add the following text line above the class definition:

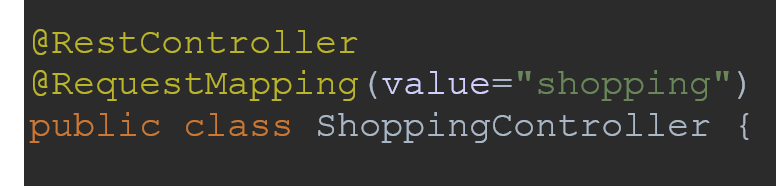
**@RestController**

If all goes well, an extra line with the import of the RestController is automatically added

To be able to reach this controller fromthe web via an URL, another annotation needs to be added.

* Add the following annotation to the shoppingController class:

**@RequestMapping(value=“shopping”)**



Now the shopping controller is reacheable via the URL <http://[servername]:[port]/shopping>, but still has no functionality to it. Therefor you are going to add some code for retrieving one or more drinks from a predefined list.

* First add a list with drinks to your shopping controller class:

private static final List<String> drinks =

Arrays.asList("Glenfiddich 18 years old 70 cl",

"Old Pulteney 17 years Single Malt 70 cl",

"Aberlour 12 Years Double Cask Matured 70 cl",

"Laphroaig 10 Years 70CL");

* Then add a method for retrieving the above list of drinks. The method will be invoked by an http GET method and the URL <http://[servername]:[port]/shopping/drinks>. For this add the following:

@RequestMapping(value="drinks")

public static List<String> retrieveDrinks() {

return drinks;

}

To retrieve a specific drink from the list, based on the identifier shown in the list, add another method named ‘retrieveDrink’. This method wil be invoked by theURL [http://[servername]:[port]/shopping/drink/{id}](http://[servername]:[port]/shopping/drink/%7bid%7d). Note that the id of the drink is passed in via the URL.

* add a second method called ‘retrieveDrink’ and annotate the method with a request mapping ‘drink/{id}’. Id is a so-called path variable and needs to be mapped onto the corresponding method parameter, which is productId in this case. For the mapping you use the annotation @PathVariable.

**@RequestMapping(value="drink/{id}", method=RequestMethod.GET)**

public static String retrieveDrink**(@PathVariable("id") String productId)** {

final String drinkNotFound = "drink not found";

try {

int keuze = Integer.parseInt(productId)-1;

if (keuze<drinks.size()) {

return drinks.get(keuze);

} else {

return drinkNotFound;

}

} catch (NumberFormatException nfe) {

return drinkNotFound;

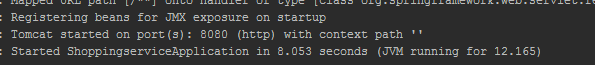
}

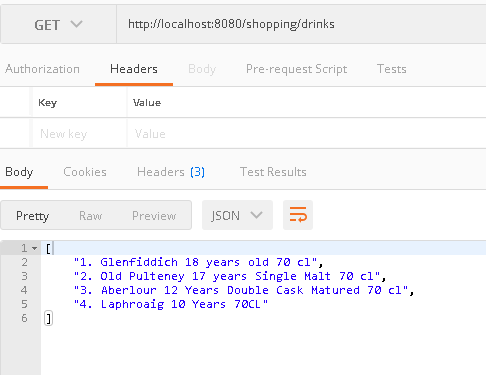
}

**Test the Shopping Service**

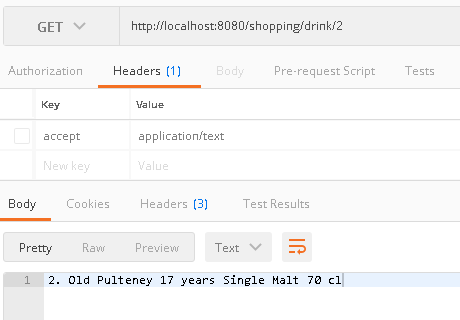
For testing purposes, you can use Postman.

* Start the shopping service application as explained earlier (3 ways to run the app).

When started in IntelliJ, you can see the portnumber through which the application can be reached. 

* Open postman
* Choose GET
* Type the URL: http:[server]:[port]/shopping/drinks with the placeholders [server] and [port] replaced with your actual values: likely localhost and 8080.
* Press send.
* If all goes well, than the list is returned
* Now let’s query one of the products listed:

http:[server]:[port]/shopping/drink/2



Is this correct?

The response from the method retrieveDrink is returned as plain text, but json is wanted.

To achieve this, add a new class for the String response and wrap the answer, eg.

package nl.sjop.service.web;

public class StringResponse {

public String getResponse() {

return response;

}

public void setResponse(String response) {

this.response = response;

}

private String response;

}

Enhance the annatation with the method retrieveDrink and implement the StringWrapper

@RequestMapping(value="drink/{id}", method=RequestMethod.GET, **produces = "application/json")**

public static StringResponse retrieveDrink(@PathVariable("id") String productId) {

final StringResponse drinkNotFound = new StringResponse();

drinkNotFound.setResponse("drink not found");

try {

int keuze = Integer.parseInt(productId)-1;

if (keuze<drinks.size()) {

final StringResponse stringResponse = new StringResponse();

stringResponse.setResponse(drinks.get(keuze));

return stringResponse;

} else {

return drinkNotFound;

}

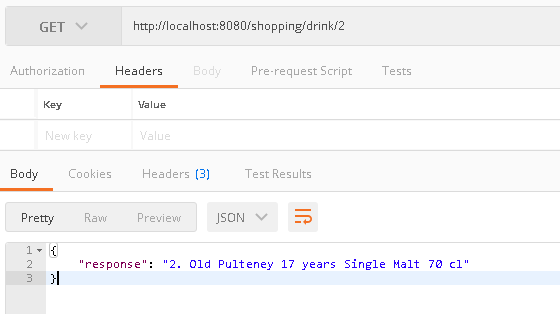
} catch (NumberFormatException nfe) {

return drinkNotFound;

}

}

Now the response is valid JSON

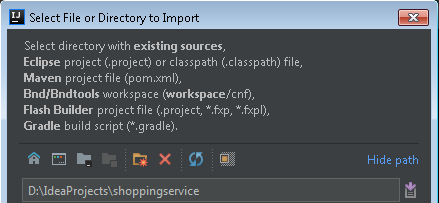


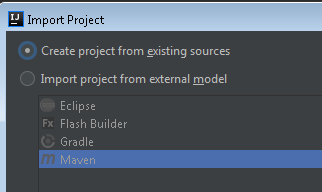
Bijlage

Opening a project in IntelliJ from existing sources

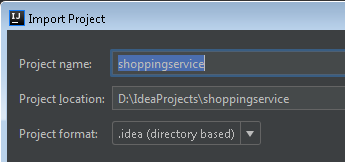
Prerequisite: under the project folder ther exists an .idea folder

* file -> new -> import from existing souces

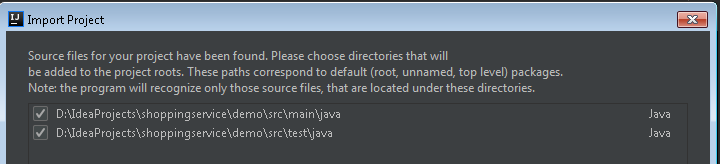




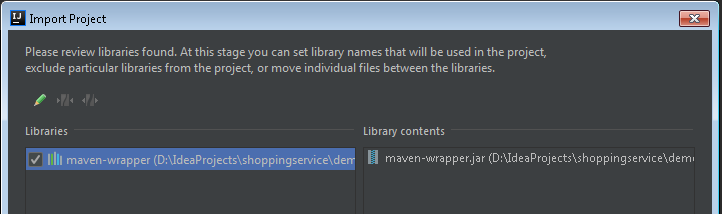
* Choose next



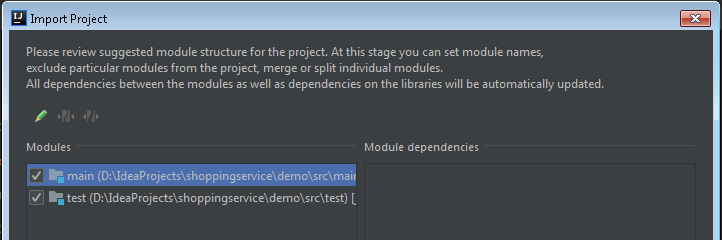
* Choose next



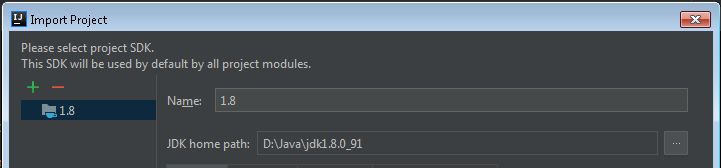
* Choose next



* Choose next



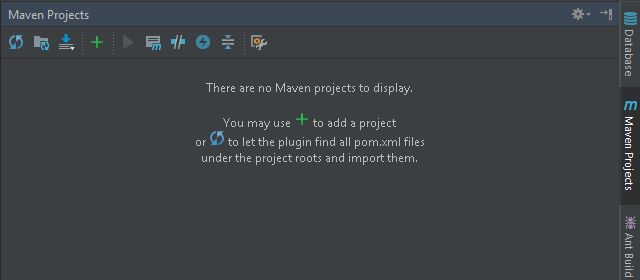
* Choose next and finish





The project should now be opened in IntelliJ.

* From the tab options on the right side of the screen, choose maven.
* Press the refresh button
* IntelliJ recognizes this is a Maven project and should now show some lifecycles



package nl.sjop.service.web;  
  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.Arrays;  
import java.util.List;  
  
@RestController  
@RequestMapping(value="shopping")  
public class ShoppingController {  
  
 private static final List<String> *drinks* =  
 Arrays.*asList*("1. Glenfiddich 18 years old 70 cl",  
 "2. Old Pulteney 17 years Single Malt 70 cl",  
 "3. Aberlour 12 Years Double Cask Matured 70 cl",  
 "4. Laphroaig 10 Years 70CL");  
  
 @RequestMapping(value="drinks", method=RequestMethod.*GET*)  
 public static List<String> retrieveDrinks() {  
 return *drinks*;  
 }  
  
 @RequestMapping(value="drink/{id}", method=RequestMethod.*GET*)  
 public static String retrieveDrink(@PathVariable("id") String productId) {  
 final String drinkNotFound = "drink not found";  
 try {  
 int keuze = Integer.*parseInt*(productId)-1;  
 if (keuze<*drinks*.size()) {  
 return *drinks*.get(keuze);  
 } else {  
 return drinkNotFound;  
 }  
 } catch (NumberFormatException nfe) {  
 return drinkNotFound;  
 }  
 }  
}

package nl.sjop.service.web;  
  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.Arrays;  
import java.util.List;  
  
@RestController  
@RequestMapping(value="shopping")  
public class ShoppingController {  
  
 private static final List<String> *drinks* =  
 Arrays.*asList*("1. Glenfiddich 18 years old 70 cl",  
 "2. Old Pulteney 17 years Single Malt 70 cl",  
 "3. Aberlour 12 Years Double Cask Matured 70 cl",  
 "4. Laphroaig 10 Years 70CL");  
  
 @RequestMapping(value="drinks", method=RequestMethod.*GET*)  
 public static List<String> retrieveDrinks() {  
 return *drinks*;  
 }  
  
 @RequestMapping(value="drink/{id}", method=RequestMethod.*GET*, produces = "application/json")  
 public static StringResponse retrieveDrink(@PathVariable("id") String productId) {  
 final StringResponse drinkNotFound = new StringResponse();  
 drinkNotFound.setResponse("drink not found");  
 try {  
 int keuze = Integer.*parseInt*(productId)-1;  
 if (keuze<*drinks*.size()) {  
 final StringResponse stringResponse = new StringResponse();  
 stringResponse.setResponse(*drinks*.get(keuze));  
 return stringResponse;  
 } else {  
 return drinkNotFound;  
 }  
 } catch (NumberFormatException nfe) {  
 return drinkNotFound;  
 }  
 }  
}