Spring Boot Crash Course Exercise

Spring Boot

These exercises will show how easy it is to create a Java Web App with Spring Boot. The functionality will be explained in more detail in other modules.

Exercise 1 - Create a Spring Boot Project Structure

The project structure of a Spring Boot project can be generated in different ways. If you have IntelliJ IDEA Ultimate edition, you can generate it with IDEA. Otherwise you can generate it with a web service at start.spring.se. The end result will be the same as IDEA uses the same web service. Select one of the alternatives:

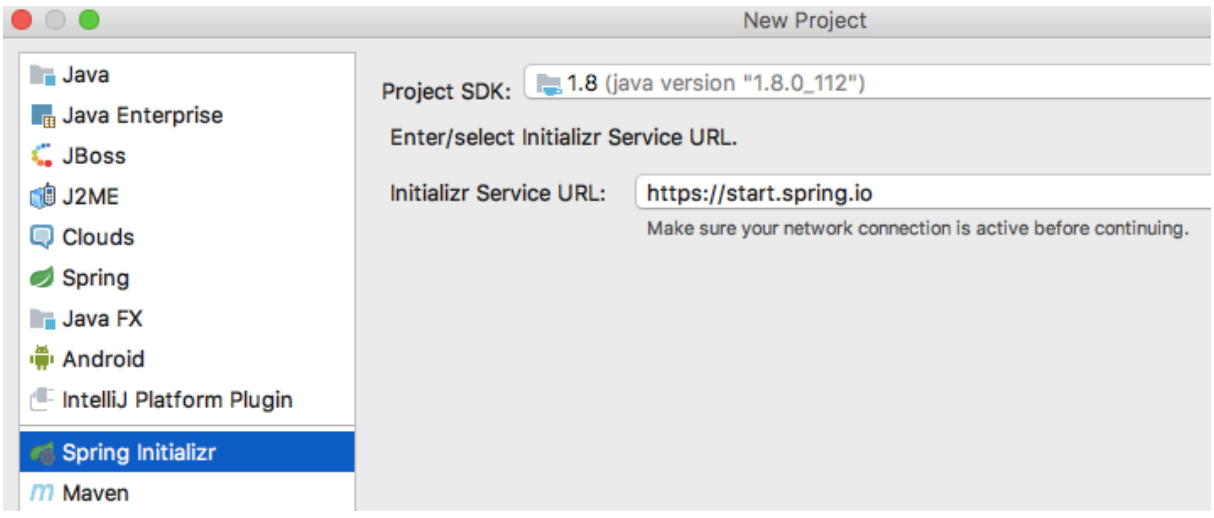
Alternative 1 – Generate with IntelliJ IDEA Ultimate Edition

Create a new project in IntelliJ IDEA.

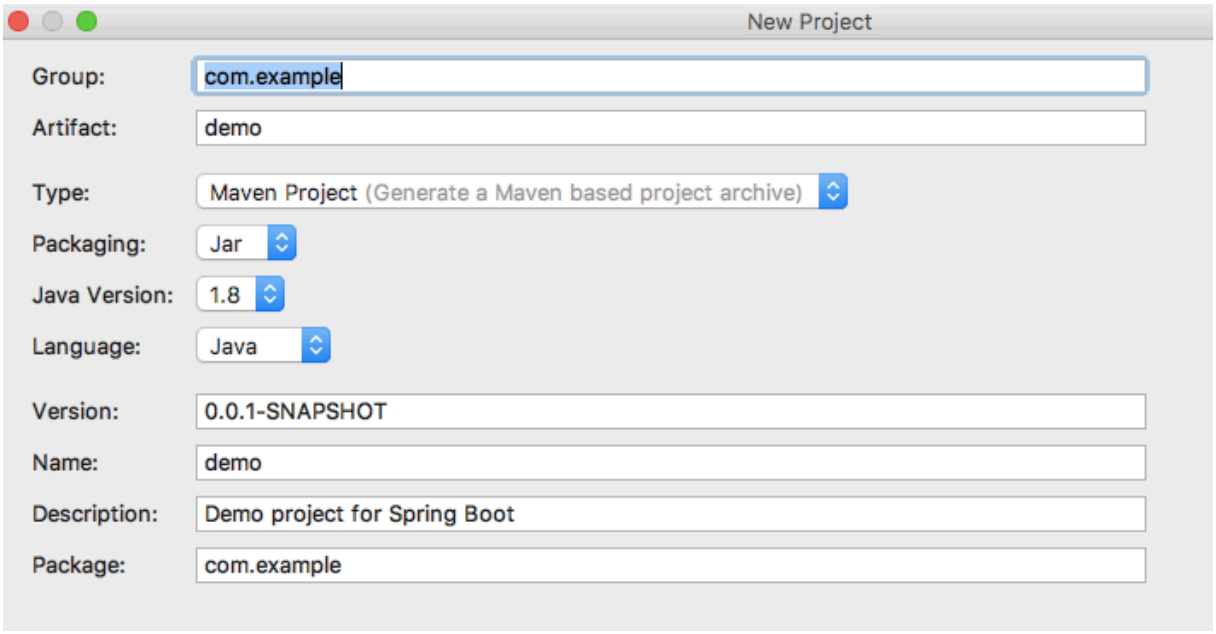
If you are already in a project, click **File – New – Project**…

If you have the IDEA Welcome screen open, click **Create New Project**.

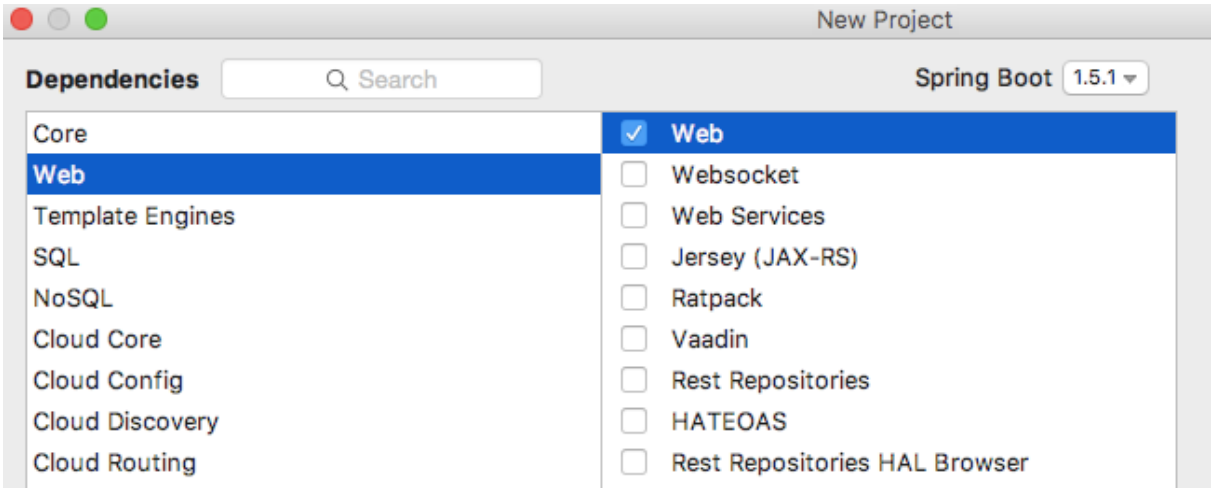
Select **Spring Initializr**



Click **Next** and you will see something like this:



Click **Next** again. Here you can select dependencies. Select **Web** and check the **Web** checkbox.



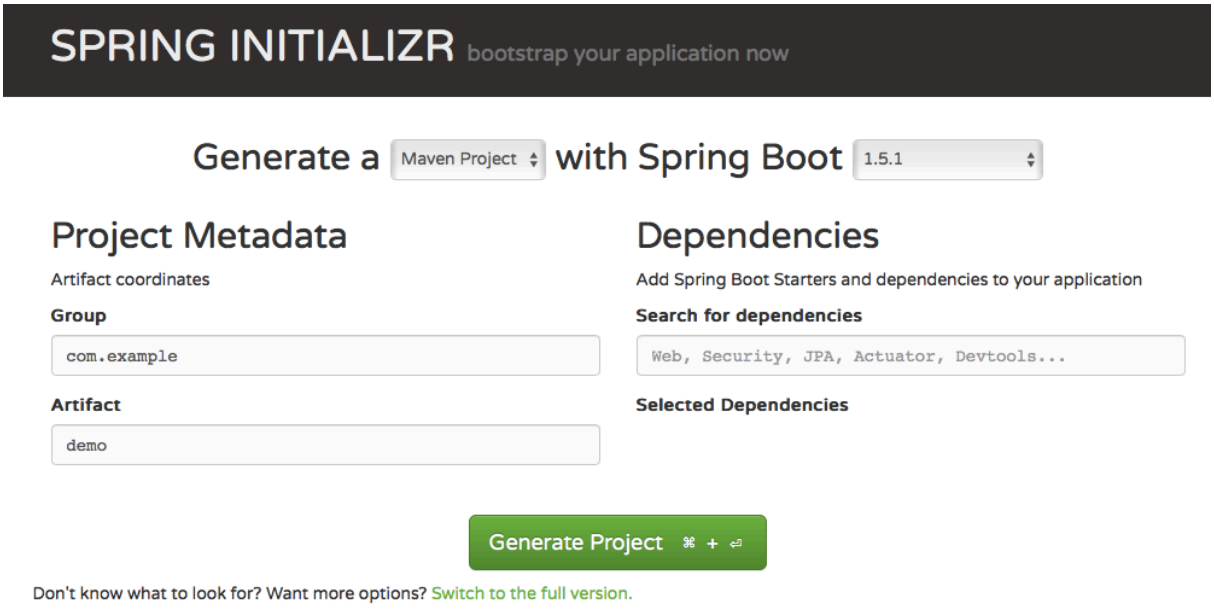
Also, select **Template Engines** and check the **Thymeleaf** checkbox.

Click Next. Here you can change the project name if you want to.

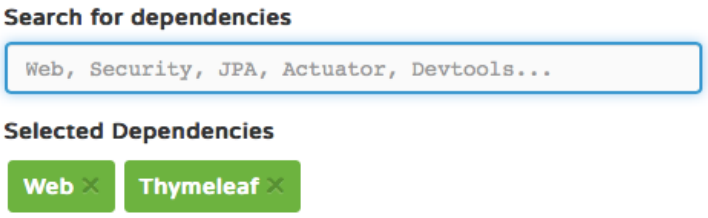
Click **Finish** and you have created the project structure and it will open in IntelliJ IDEA!

Alternative 2 – Generate with start.spring.io

Go to **start.spring.io** with a web browser. You will see something like this:



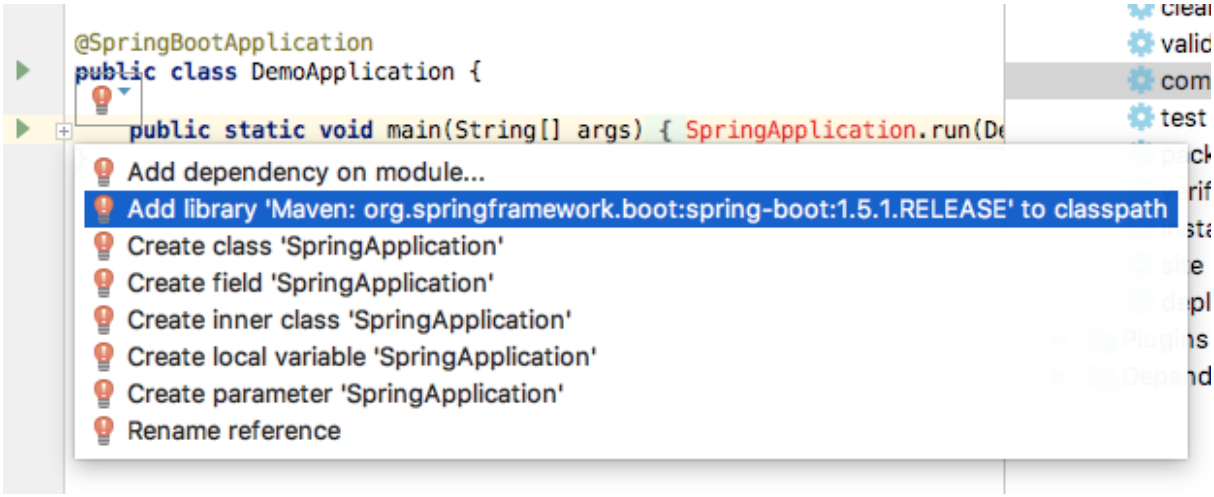
Enter the word **web** in the Search for dependencies text field and press enter. Then enter the word **thymeleaf** in the same text field and press enter again. You should see that you have selected the dependencies for **Web** and **Thymeleaf**, it should look something like this:



Then click the **Generate Project** button and you will download the project structure in a zip file.

Extract the zip file and open the project in IntelliJ IDEA by selecting **File – New – Project** from Existing Sources and select the folder with the extracted project structure. Click **Next** as many times as needed, and then click **Finish** to open the project IntelliJ IDEA!

Look at the class **com.example.DemoApplication**. If the annotation **@SpringBootApplication** or the class name **SpringApplication** is marked as errors, then click the little light bulb and select **Add library 'Maven: org.springframework.boot:spring-boot:…** as shown here:

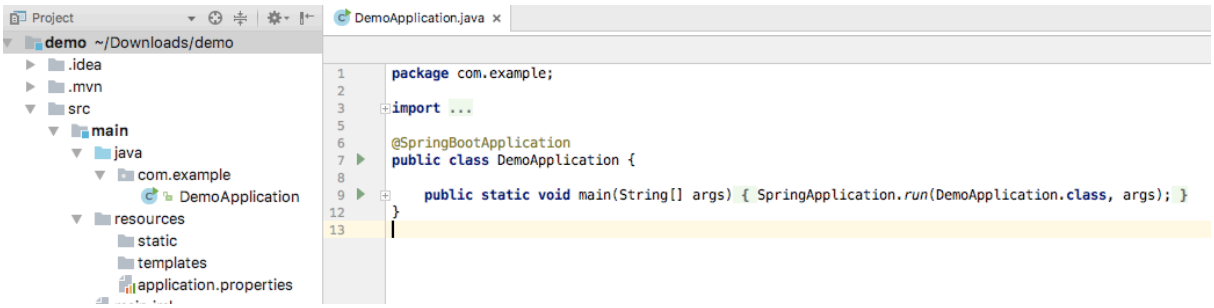


IntelliJ didn’t recognize these classes because this dependency wasn’t added to the classpath.

Exercise 2 - Create a simple Java Web App with the Spring Boot Project Structure

Now that you have a Spring Boot project structure, you can create a simple Java Web app with only 5 lines of code!

Look at the **DemoApplication** class, it should look something like this:



The **DemoApplication** class already contains a public static void main method that will run the Spring Boot functionality for you.

By adding a few lines of code, we will make this class a so-called **Controller** with methods that will automatically be called when web requests with a certain URL is sent to the application.

First add a new annotation called **@RestController** above the **@SpringBootApplication** annotation. This will make Spring Boot look in this class for methods that are marked with certain annotations that means that the method should be called when web requests with certain URLs are sent to the application.

Then add a method with a name of your choice (the name doesn't matter), it should be public and return a String and it should not take any arguments. It should only contain one line of code that returns a string, for example "Hello World Academy!".

Put one annotation on this method with this signature:

@GetMapping(**"/"**)

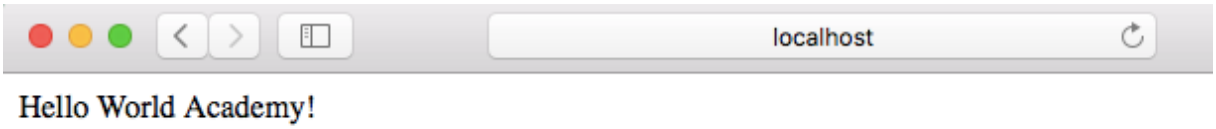
The method should now look something like this:

```
@GetMapping("/")
public String hello() {
    return "Hello World Academy!";
}
```

Now run the application and Spring Boot will start up and provide an embedded Tomcat web container so you don't have to deploy the web app to a web container. You don't even need an installed web container anymore since everything the application needs is handled by Spring Boot.

The application will start up with the host name localhost and the port 8080.

Go to a web browser and enter the URL **http://localhost:8080** and you should see your Spring Boot Web App respond with something like this:



Congratulations! You have now created a Java web application with Spring Boot and five lines of code!

The hello method is a normal method that returns a string. The "magic" all happens because of the annotations. The **@RestController** annotation on the class tells Spring Boot that this class is a **RestController** and that it should look for methods annotated with some special annotations, like for example the **@GetMapping** annotation that tells Spring Boot to call this method when it receives a request for "/", which is the root of the web app and what is called with the URL to only the host and port number: **http://localhost:8080**

Exercise 3 – Return JSON data from a Spring Boot Web App

The job for a web app is often to return data to a client application like a mobile app or a JavaScript client or some other client. With Spring Boot it is very easy to return data in the JSON format.

Create a new class in your project with some variables and the corresponding getters and setters, like for example the Customer class from exercise 14:

```
package com.example;

class Customer {
    int id;
    String name;
    String address;
    int zipcode;
    String city;
    String email;

    public Customer(int id, String name, String address, int zipcode, String city, String
email) {
        this.id = id;
        this.name = name;
        this.address = address;
        this.zipcode = zipcode;
        this.city = city;
        this.email = email;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public int getZipcode() {
        return zipcode;
    }

    public void setZipcode(int zipcode) {
        this.zipcode = zipcode;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }
}
```

Then create a new method in the **DemoApplication** class with a new name (the name doesn't matter) that is public and returns a Customer object.

Create and return a Customer object inside the method.

Put this annotation on the method:

@GetMapping(**"/customer"**)

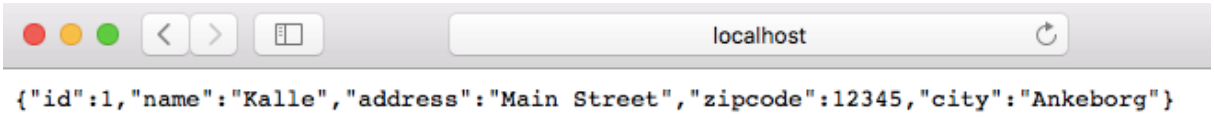
The method should look something like this:

```
@GetMapping("/customer")
public Customer customer() {
   Customer customer = new Customer(1, "Kalle", "Main Street",
         12345, "Ankeborg", "kalle@anka.com");
   return customer;
}
```

Run the application and enter this URL in a web browser:

http://localhost:8080/customer

You should see a result like this:



Congratulations! You have created a Spring Boot Web App that returns JSON data created from the values in the variables of Java objects.

The GetMapping to "/customer" tells Spring Boot to call the customer method when a request enters the web app with the URL of the host, port number, and path /customer, like this URL: http://localhost:8080/customer

Exercise 4 – Handle request parameters

In this exercise, we will send a parameter as a request parameter in the URL to the web app and let the web app read this parameter and print it back to the user.

Add a method to the **DemoApplication** that is similar to the hello method from exercise 2, but with the name "user" and a value in GetMapping that is "/user". It could look something like this:

```
@GetMapping("/user")
public String user() {
   return "Hello World Academy!";
}
```

This method will be called with this URL: **http://localhost:8080/user**

We want the method to get hold of a request parameter with the name "name". We can just tell Spring Boot that we want an argument of this kind by specifying a String name as an input argument in the method, and annotate it with the annotation **@RequestParam**.

The signature of the method will then look something like this:

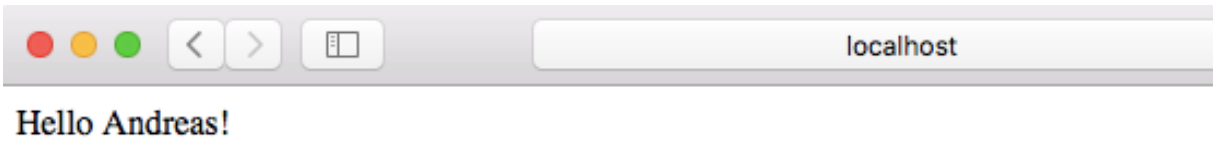
```
public String user(@RequestParam String name)
```

You can now use the name variable in the method and add the name to the string that the method returns. Change the string so that it will print Hello and then the value of the name variable.

To add the name parameter to the URL, you add a question mark at the end of the URL and then the name of the parameter, an equals character, and then the value of the parameter, like this:

http://localhost:8080/user?name=Andreas

Restart the web app and try this URL. Did it respond with a Hello to the name in the URL parameter? The URL above should result in something like this:



If not, check that your method looks something like this:

```
@GetMapping("/user")
public String user(@RequestParam String name) {
   return "Hello " + name + "!";
}
```

Congratulations, you have now created a method that handles request parameters.

Stretch Tasks (if you have time)

Stetch Task 1

Try to send other request parameters to a method. For example, use a request parameter named uppercase, and return a string in upper case if the uppercase parameter has the value y and in lower case if it has a value of n or any other value.

Stetch Task 2

Try to send many parameters. In the URL, you just add an ampersand & and then another name-value pair as a parameter, for example:

?name=Andreas&uppercase=y

The method can handle both parameters by just adding all of them as input arguments, separated with a comma, just as usual (but don't forget to add the **@RequestParam** annotation before each variable).

Stetch Task 3

Try to send an integer parameter. The argument to the method does not need to be a string (even though all request parameters are sent as strings in the URL). Spring can convert the parameter to another type by just specifying the type of the input argument to the method.

Try having an input argument to a method that looks like this:

```
@GetMapping("/stretch3")
public String stretch3(@RequestParam Integer number)
```

Now you can send a request parameter with the name number and a numeric value, and Spring Boot will put the value in the Integer number variable. You can now use the Integer value in the method, for example multiply the number by 2 and return the result to the user.

The URL for this method should look something like this:

http://localhost:8080/stretch3?number=3

If you have even more time, check out the Spring Boot Guides here:

https://spring.io/guides

Look at Getting Started with Spring Boot with Josh Long:

https://www.youtube.com/watch?v=sbPSjI4tt10