Spring MVC Exercise

Spring MVC

Spring MVC builds on the Model View Controller pattern.

Controller

This is the Java class we create and annotate with **@RestController** or **@Controller**. Here we create methods that will respond to http requests. We specify the URL of the request in an annotation on the method, **@GetMapping** for get request, **@PostMapping** for post requests and so on. In the method, we create the code that should run for every http request.

Model

The model is where we put all data that we want to show the user in the html that we will create. We create a Model as a new Model object or a new **ModelAndView** object. We add objects (could be any Java object) to the model by calling the method **addObject** on the model.

View

This is a template of a html page where the data from the model will be added so that the user can see the actual data from the Java code in a html page generated from the template and the data from the model.

Read more about MVC in general here:

https://sv.wikipedia.org/wiki/Model-View-Controller

Read about serving web content with Spring MVC here:

https://spring.io/guides/gs/serving-web-content/

We will use Thymeleaf as a templating technology. Thymeleaf has an excellent integration with Spring MVC. Read about the Thymeleaf Spring MVC support here:

http://www.thymeleaf.org/doc/articles/springmvcaccessdata.html

Full documentation here:

http://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html

Exercise 1 - Create a Spring MVC application

Create a new Spring Boot project, like in the Spring Boot Crash Course Exercise. Select both dependencies **Web – Web** and **Templating Engines – Thymeleaf**.

The dependency to Web will give you all the functionality for MVC and the Thymeleaf dependency will give you the functionality for Thymeleaf templates that we are going to use to create the views.

In the Spring Boot Crash Course we used the **DemoApplication** class to act also as a **RestController**. Now we are going to create a new Controller class to handle the controller functionality.

Create a new class called **DemoController** in the **com.example** package where the **DemoApplication** is.

Annotate the class with an **@Controller** annotation (just above the line public class Controller…).

If it IntelliJ doesn't suggest the import, add this import line to you imports:

```
import org.springframework.stereotype.Controller;
```

The **@Controller** annotation specifies that this is a MVC controller (as opposed to a RestController), and that the controller methods will return a view name that will render the view to the client (web browser of the user).

Create this method inside the **DemoController** class:

```
@GetMapping("hello")
public ModelAndView hello() {
    return new ModelAndView("hello");
}
```

This Controller method will now respond to the URL http://localhost:8080/hello and will try to use a view template with the name of hello. Let's create the view template.

Create a new HTML file with the name hello.html in the **src.main.resources.templates** folder.

Inside the **<body>** tags, add a heading with a hello message:

```
<h1>Hello MVC World!</h1>
```

If you create it with the help of IntelliJ IDEA you will have to change one thing to make the html file xml compatible. You must close all tags, because Thymeleaf parses the templates as xml files. When IDEA creates HTML files, it creates the **<meta>** tag without a closing tag.

Change:

```
<meta charset="UTF-8">
```

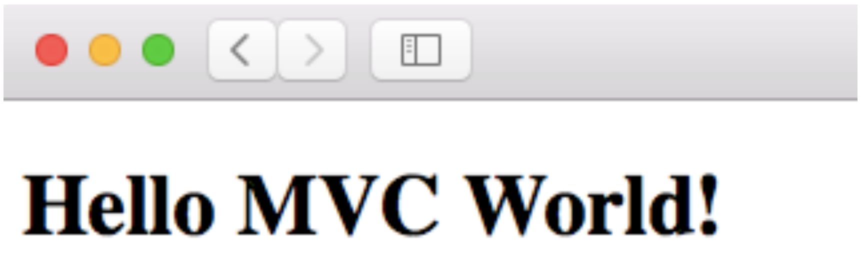
to:

```
<meta charset="UTF-8"/>
```

Try to run the application and enter this URL in a web browser:

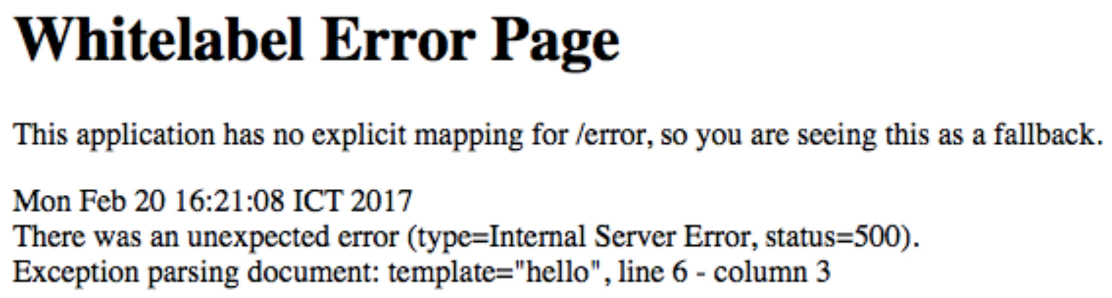
http://localhost:8080/hello

You should see something like this:



Hello MVC World!

If you got this error message:



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

```
Mon Feb 20 16:21:08 ICT 2017
There was an unexpected error (type=Internal Server Error, status=500).
Exception parsing document: template="hello", line 6 - column 3
```

Then there is something wrong with the HTML file, it could be that you have not closed all tags. Look at the HTML file again and try to correct all tags.

Congratulations, you have created a MVC application with Spring MVC!

But wait, you didn't use the model in this example, let's do that next!

Exercise 2 - Create a new Controller method that uses the model

Create a new method in **DemoController** that looks just like the previous one but change all 3 occurrences of hello to hello2 (change the **GetMapping**, name of method, and the view name).

Now you will add an object to the model that the view can display to the user.

Just call the **addObject** method on the **ModelAndView** and specify the name of the object and the value, like this:

```
return new ModelAndView("hello2").addObject("name", "Andreas");
```

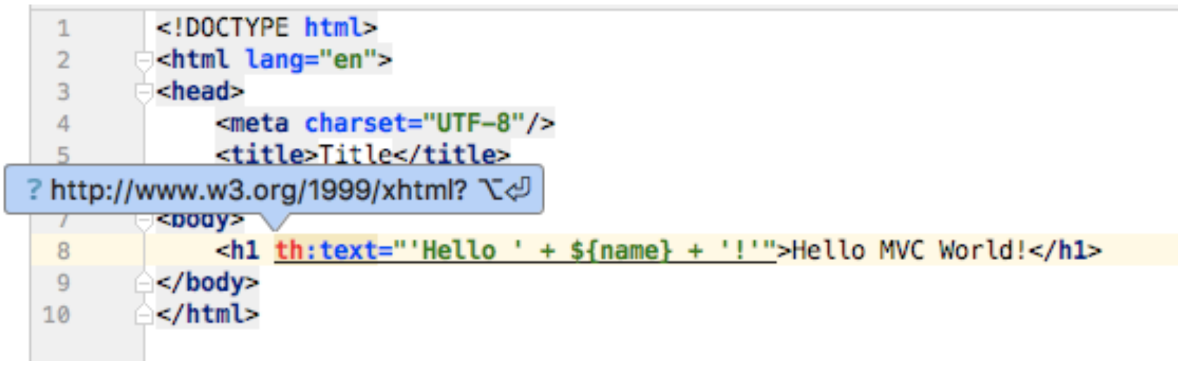
This time the name and value are both strings, the object could be any other object, but then it is important that the object has standard Java getters so that Thymeleaf can read the values from the object.

Copy the view to a new file with the name hello2.

Change the heading tag to this:

```
<h1 th:text="'Hello ' + ${name} + '!'">Hello MVC World!</h1>
```

The **th** part will probably become marked red like this:



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8"/>
    <title>Title</title>
<body>
    <h1 th:text="'Hello ' + ${name} + '!'">Hello MVC World!</h1>
</body>
</html>
```

? http://www.w3.org/1999/xhtml? ⌥⏎

Accept the suggestion from IntelliJ IDEA to add the **th namespace** to the html tag.

You can do this manually by changing the **<html>** tag like this:

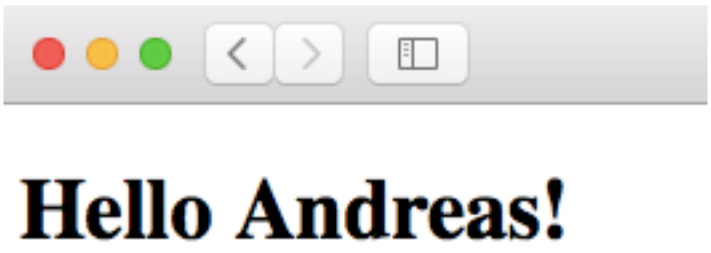
```
<html lang="en" xmlns:th="http://www.w3.org/1999/xhtml">
```

Now the red warning should go away. This is just a warning, the program will work anyway.

Ok, let’s try out our new version of MVC where we use the model. Run the application and enter the URL:

http://localhost:8080/hello2

You should see something like this:


Hello Andreas!

But wait, what happened with the Hello MVC World! Text inside the heading?

It got replaced by the **th:text** attribute. The text inside the **<h1>** tag will only show if you look directly at the template HTML file, like if you double click on it. This is an advantage with the Thymeleaf templating technology, that the templates are real html files that can be viewed in a web browser by double clicking on them. But when you run the template in a web application the value inside the **<h1>** tag will be replaced by what is in the **th:text** attribute.

Exercise 3 - Handle request parameters in the request

You have already done an exercise with request parameters in Exercise 4 in Spring Boot Crash Course. Copy the hello2 method to a new hello3 method and change the **GetMapping** to hello3. We will use the same hello2.html file, so you should still create the **ModelAndView** with the view name hello2.

Now try to change the method hello3 so that it can handle a request parameter with the name “name” like in the Spring Boot Crash Course exercise. Add the name variable to the model so that it shows up in the Thymeleaf template.

Try out the exercise by calling the URL **http://localhost:8080/hello3?name=your_name**

Can you get the method and template to say Hello to whatever name you send in the request parameter?

Exercise 4 - Handle path variables in the request

Request parameters isn't the only way to send data to the Controller method. It can also be done as a part of the URL, like this:

http://localhost:8080/hello4/your_name

For the method to get hold of the data we must use the path variable annotation.

Copy the hello3 method to a hello4 method and change **GetMapping** to hello4. We will use the same view (hello2.html) again.

Change the **GetMapping** to this:

```
@GetMapping("hello4/{name}")
```

The **{name}** is there after the slash to indicate that this method is mapped to any URL that begins with **hello4/** and then anything that comes after the slash is handled as a path variable with the name **name**.

You must also change the **@RequestParam** annotation on the String name variable that is an input argument to the method, so that it looks like this:

```
public ModelAndView hello4(@PathVariable String name)
```

Now, anything you enter in the URL after the **hello4/** will be the value in the **@PathVariable** name.

The rest of the method still works as before, and the hello2 view should now show the value from the path variable.

Try the exercise by calling the URL:

http://localhost:8080/hello4/enter_you_name_here_or_whatever

Exercise 5 – Handle objects in the model

This far we have only added a string to the model. What about other objects, and how can we get the Thymeleaf template to display data from these objects?

Create a new class in the **com.example** package with the name **User**. Add one private String variable in the User class with the name **username**.

Copy the hello4 method to a new method with the name hello5, and change the **GetMapping** to map to hello5. Also, change the view name to hello5 because we must change the view to be able to show information from the User class.

Copy the hello2.html file to a new file named hello5.html.

In the hello5 method, create a new **User** object from the **User** class. Set the **username** of the User object to the variable that comes in to the method as the path variable. Set the user object in the model instead of the name parameter, and give it the name **user**.

Now, in the view we must change what we want to display to the user. There is no more name parameter of type String in the model, but an object of type **User**. Specify what you want to print to the view by dot notation, like this:

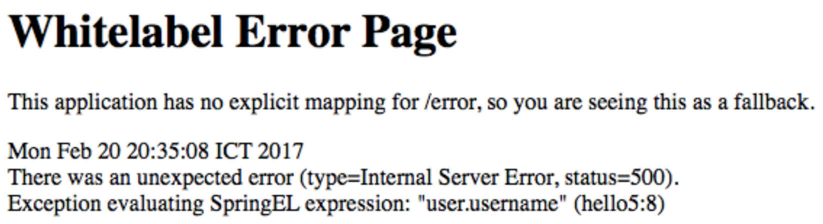
```
${user.username}
```

Try if it works by calling the URL

http://localhost:8080/hello5/enter_you_name_here_or_whatever

Assuming that you have a **private username** and **public** getters and setters, try changing the getter in the **User** class from **getUsername** to only **getName**. It should still return the **username** variable. What happens if you try this?

Maybe something like this:



```
Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Mon Feb 20 20:35:08 ICT 2017
There was an unexpected error (type=Internal Server Error, status=500).
Exception evaluating SpringEL expression: "user.username" (hello5:8)
```

Even if the variable in the **User** object still is **username**, the **user.username** reference in the template won't work because what Thymeleaf is really doing is using the appropriate getter method. So, if you have this reference in a Thymeleaf template:

```
${user.username}
```

What Thymeleaf does is calling the user objects getter method:

user.getUsername()

Thymeleaf takes what is after the dot, uppercase the first letter, and ads get in front of it, and calls a method with this name. It’s important to use standard getters and setters in Java objects. Many frameworks depend on this standard.

Stretch Tasks (if you have time)

Try out the exercise we looked at before:

https://spring.io/guides/gs/serving-web-content/