Spring MVC Workshop
Lab 1

1 Lab setup

Make sure you have the following items installed

- Latest version of JDK 8 / 11 (note: labs are tested with JDK 8 but should work on JDK 11 with no or minimal changes)
- Eclipse Enterprise Edition for Java (or a suitable alternative IDE for Enterprise Java)
- Latest version of Maven
- Tomcat integrated with Eclipse
- A suitable text editor (Notepad ++)
- A utility to extract zip files

In each of the main lab folders, there are two subfolders: `changes` and `final`. The `changes` subfolder just holds the source code files for the lab, while the `final` subfolder holds the complete Eclipse project starting from its project root folder. We will use the code from the `changes` subfolder to build up our applications from scratch and you can always fall back on the complete Eclipse project if you encounter any errors while building up the application.

2 Creating a Spring MVC application

The source code for this lab is found in `SpringMVC-Basic-Project/changes` folder.

We can create a new Maven web app project from scratch following the steps that we did for the earlier lab. Alternatively, we can create a new copy based on the previous lab project and modify it appropriately so that it can run properly.

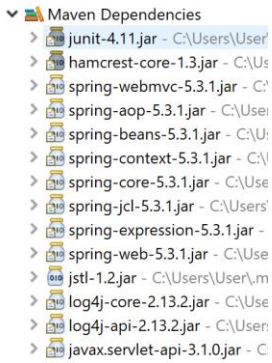
In the Project Explorer, right click on `JSPMavenProject`, select Copy and then right click in any empty space in the Explorer and select Paste.

For the new project name, type: `BasicSpringMVC`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select Maven -> Update Project, and click OK. Notice that it includes dependencies for Spring MVC, JSTL, Log4J2 as well as Servlet 3.1 API.

```
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>${spring.framework.version}</version>
		</dependency>
```

If you go to the Libraries entry, you should see the various JARs for Spring `core`, `context`, `web` and `webmvc` – they are all transitive dependencies that are added in via the `spring-webmvc` dependency.



Right click on the project entry, select Properties - > Resource. Click on arrow next to the Location field. Go into `BasicSpringMVC/.settings` and modify the settings in this file: `org.eclipse.wst.common.component` to reflect the new project name.

```
<?xml version="1.0" encoding="UTF-8"?><project-modules id="moduleCoreId" project-
version="1.5.0">
    <wb-module deploy-name="BasicSpringMVC">
        <wb-resource       deploy-path="/"       source-path="/target/m2e-wtp/web-
resources"/>
        <wb-resource         deploy-path="/"         source-path="/src/main/webapp"
tag="defaultRootSource"/>
        <wb-resource             deploy-path="/WEB-INF/classes"             source-
path="/src/main/java"/>
        <wb-resource             deploy-path="/WEB-INF/classes"             source-
path="/src/main/resources"/>
        <property                                                name="java-output-path"
value="/BasicSpringMVC/target/classes"/>
        <property name="context-root" value="BasicSpringMVC"/>
    </wb-module>
</project-modules>
```

Save the file and refresh the project.

Delete the package `com.workshop.servlets`
Delete all the JSP files in `webapp`
Delete `web.xml`

We are now ready to populate the Maven web app project with the relevant resources to create a proper Spring-MVC project

2.1 Using a XML configuration

Create the following files in WEB-INF
`spring-mvc-configuration.xml`
`web.xml`

Create a folder `views` in WEB-INF. Place these files there:
`main-menu.jsp`
`awesome.jsp`

Place `log4j2.properties` in `src/main/resources`

In `src/main/java`, create a package: `com.workshop.mvc`
Create a class here: `FirstController.java`

Delete `index.jsp` in Deployed Resources -> Webapp, if you haven't already done so. The reason for this is that the Tomcat server will automatically look for this file first when we access the root deployment URL: http://localhost:8080/BasicSpringMVC/, which will override the access to the mapping in `FirstController.java`

Right click on the Project, select Run As -> Run on Server

Access these URLs:

http://localhost:8080/BasicSpringMVC/
http://localhost:8080/BasicSpringMVC/first

and verify that the appropriate JSP file is returned in accordance with the URL path mapping. You can also check for the output of DEBUG messages in the Console view that gives some insight into the working of the Spring MVC architecture.

Try typing a random URL that does not map to any @Controller @RequestMapping method, and check the DEBUG message output in the Console view as well.

2.2 Using a Java-based configuration

In the package: `com.workshop.mvc`, create the following classes:

`MyWebAppInitializer.java`
`WebConfig.java`

Delete these XML files in WEB-INF:

```
spring-mvc-configuration.xml
web.xml
```

Redeploy the app and verify that you can still access these URLs and they return the correct JSP files:

http://localhost:8080/BasicSpringMVC/
http://localhost:8080/BasicSpringMVC/first

For the next lab, we will work using an XML configuration. Delete the two new Java classes you created and restore the 2 XML files into WEB-INF again.

3 Using @RequestMapping, @GetMapping, @PostMapping

We continue with the same project from the previous lab.

Create these files in `WEB-INF/views`:

```
cool.jsp
fancy.jsp
first-form.jsp
```

In the package `com.workshop.mvc`, create these files:

```
SecondController.java
ThirdController.java
```

Access these URLs below and verify that they map correctly to their respective @RequestMapping methods by checking the log output in the Console view:

http://localhost:8080/BasicSpringMVC/stuff/interesting
http://localhost:8080/BasicSpringMVC/stuff/fancy/pants
http://localhost:8080/BasicSpringMVC/stuff/cool
http://localhost:8080/BasicSpringMVC/stuff/reallycool

Note: If there are too many DEBUG messages appearing in the Console view, remember that you can filter these out by changing the log level to info in `log4j2.properties` in `src/main/resources`

```
rootLogger.level = info
```

Navigate to:

http://localhost:8080/BasicSpringMVC/firstform

and complete the form and submit it. Since the default HTTP method for a form submission is GET, verify that the corresponding @RequestMapping method is called.

Make this change:

`first-form-v2.jsp`

which now explicitly makes the submit method POST, and verify that a different @RequestMapping is called in this instance.

Make another change:

`ThirdController-v2.java`

Navigate again to:

http://localhost:8080/BasicSpringMVC/firstform

and complete the form and submit it. Switch between the two implementations (`first-form.jsp` – using GET) and `first-form-v2.jsp` – using POST) and verify that the Console output indicates the correct matching method is called again.

Make another change:

`spring-mvc-configuration-v2.xml`

Navigate to these URLs:

http://localhost:8080/BasicSpringMVC/happy
http://localhost:8080/BasicSpringMVC/funny

and confirm the correct views are returned. We use the <mvc:view-controller> element when we need to map paths directly to views without the need for any additional interceding business logic that would typically go in the @RequestMapping methods.

4 Working with Model

The source code for this lab is found in `SpringMVC-Forms/changes` folder.

We can create a new Maven web app project from scratch following the steps that we did for the earlier lab. Alternatively, we can create a new copy based on the previous lab project and modify it appropriately so that it can run properly.

In the Project Explorer, right click on `BasicSpringMVC`, select Copy and then right click in any empty space in the Explorer and select Paste.

For the new project name, type: `FormsSpringMVC`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select Maven -> Update Project, and click OK. Notice that it includes dependencies for Spring MVC, JSTL, Log4J2 as well as Servlet 3.1 API.

Right click on the project entry, select Properties - > Resource. Click on arrow next to the Location field. Go into `FormsSpringMVC/.settings` and modify the settings in this file: `org.eclipse.wst.common.component` to reflect the new project name.

```
    <wb-module deploy-name="FormsSpringMVC">
        <wb-resource       deploy-path="/"       source-path="/target/m2e-wtp/web-
resources"/>
        <wb-resource         deploy-path="/"         source-path="/src/main/webapp"
tag="defaultRootSource"/>
        <wb-resource             deploy-path="/WEB-INF/classes"             source-
path="/src/main/java"/>
        <wb-resource             deploy-path="/WEB-INF/classes"             source-
path="/src/main/resources"/>
        <property                                              name="java-output-path"
value="/FormsSpringMVC/target/classes"/>
        <property name="context-root" value="FormsSpringMVC"/>
    </wb-module>
```

Save the file and refresh the project.

Delete all the classes in `com.workshop.mvc`
Delete all the JSP files in `WEB-INF/views`

In WEB-INF, create the following files:
`web.xml`
`spring-mvc-configuration.xml`

We are now ready to populate the Maven web app project with the relevant resources to create a proper Spring-MVC project

4.1 Adding attributes to and displaying attributes from Model

In WEB-INF/views, create these files:
`main-menu.jsp`
`employee-details.jsp`
`employee-form.jsp`

Place `log4j2.properties` in `src/main/resources`

Create these classes in the package: `com.workshop.mvc`
`HomeController.java`
`EmployeeController.java`

Delete `index.jsp` in Deployed Resources -> Webapp if it still exists. The reason for this is that the Tomcat server will automatically look for this file first when we access the root deployment URL: http://localhost:8080/FormsSpringMVC/, which will override the access to the mapping in `HomeController.java`

Deploy the app:

http://localhost:8080/FormsSpringMVC/

Click on the employee details form link and complete the form that is returned. Check that the form data is correctly returned. Notice that we use JSP EL in `employee-details.jsp` to obtain the parameter values.

Add the following classes to `com.workshop.mvc`
`JobDetails.java`

Make the following changes:
`EmployeeController-v2.java`
`employee-details-v2.jsp`
`employee-form-v2.jsp`

Deploy the app:

http://localhost:8080/FormsSpringMVC/

Click on the employee details form link and complete the form that is returned. Check that the form data and model attributes is correctly returned. Notice that we use JSTL Core tags in `employee-details.jsp` to perform an if-else evaluation (we could have also done this using a scriptlet Java code block as well).

4.2 Using @RequestParam to bind form parameters

Make the following changes:

`EmployeeController-v3.java`

Deploy the app:

http://localhost:8080/FormsSpringMVC/

Click on the employee details form link and complete the form that is returned. Check that the form data and model attributes is correctly returned. Notice that we us @RequestParam annotation in the method signature of processFormv2 in order to bind the request parameters to the method parameters.