Spring MVC Workshop
Lab 1

1 Lab setup

Make sure you have the following items installed

- Latest version of JDK 11 (note: labs are tested with JDK 11 but should work with higher versions with no or minimal changes)
- Eclipse Enterprise Edition for Java (or a suitable alternative IDE for Enterprise Java)
- Tomcat 9.0 installed as a Windows service
- Latest version of Maven
- A suitable text editor (Notepad ++)
- A utility to extract zip files (7-zip)

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.

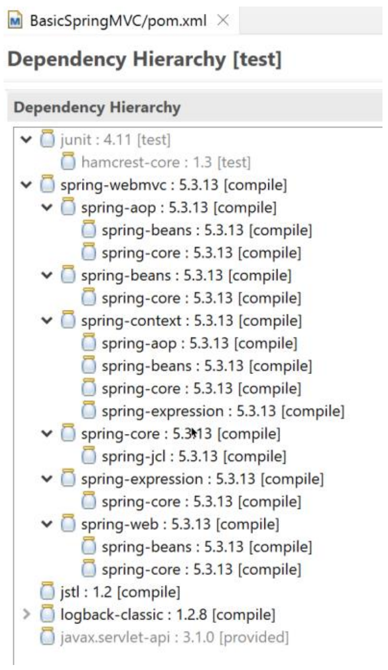
For the new project name, type: `BasicSpringMVC`

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, Logback 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 Dependency Hierarchy view of the pom.xml, you will see that the various dependencies related to Spring Core: such as `beans`, `context`, `expression`, `etc`–are all transitive dependencies that are added in via the `spring-webmvc` dependency.


BasicSpringMVC/pom.xml

Dependency Hierarchy [test]

Dependency Hierarchy

```
junit : 4.11 [test]
    hamcrest-core : 1.3 [test]
spring-webmvc : 5.3.13 [compile]
    spring-aop : 5.3.13 [compile]
        spring-beans : 5.3.13 [compile]
        spring-core : 5.3.13 [compile]
    spring-beans : 5.3.13 [compile]
        spring-core : 5.3.13 [compile]
    spring-context : 5.3.13 [compile]
        spring-aop : 5.3.13 [compile]
        spring-beans : 5.3.13 [compile]
        spring-core : 5.3.13 [compile]
        spring-expression : 5.3.13 [compile]
    spring-core : 5.3.13 [compile]
        spring-jcl : 5.3.13 [compile]
    spring-expression : 5.3.13 [compile]
        spring-core : 5.3.13 [compile]
    spring-web : 5.3.13 [compile]
        spring-beans : 5.3.13 [compile]
        spring-core : 5.3.13 [compile]
jstl : 1.2 [compile]
logback-classic : 1.2.8 [compile]
javax.servlet-api : 3.1.0 [provided]
```

Go to Properties -> Web Project Settings and ensure that the Context root: is set to `BasicSpringMVC`.
If it is not set it to this value, click Apply -> Ok and then click Apply and Close.

Right click on the project entry, select Show in - > System Explorer. Navigate 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

Copy the following files from `changes` into `src/main/webapp/WEB-INF`

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

Create a folder `views` in WEB-INF.

Copy the following files from `changes` into `src/main/webapp/WEB-INF/views`

```
main-menu.jsp
awesome.jsp
```

In `src/main/java`, create a package: `com.workshop.mvc`

Copy the following files from `changes` into `com.workshop.mvc` in `src/main/java`:

```
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, as well as INFO messages from the @RequestMapping methods in FirstController.

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

Copy the following files from `changes` into `com.workshop.mvc` in `src/main/java`:

```
MyWebAppInitializer.java
WebConfig.java
```

Delete these XML files in `src/main/webapp/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

3 Using @RequestMapping, @GetMapping, @PostMapping

We continue with the same project from the previous lab.

Copy the following files from `changes` into `WEB-INF/views`:

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

Copy the following files from `changes` into `com.workshop.mvc` in `src/main/java`:

```
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

To control the level of detail of the logging messages that appear in the Console view that are related to the Spring MVC architecture internals, we can introduce a configuration file for the Logback framework.

Copy the following files from `changes` into `src/main/resources`:

`logback.xml`

Here we set the root logging level to `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 changes to following files in `WEB-INF/views` from `changes`:

`first-form-v2.jsp`

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

Make changes to the following files in `com.workshop.mvc` in `src/main/java` from `changes`:

`ThirdController-v2.java`

This uses the shorter annotation @GetMapping to replace @RequestMapping(method = RequestMethod.GET) and @PostMapping to replace @RequestMapping(method = RequestMethod.POST)

Navigate again to:

http://localhost:8080/BasicSpringMVC/firstform

and complete the form and submit it. Check how the request looks like in the Network tab of Chrome DevTools when the submit method of POST is used.

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 changes to the following files in `com.workshop.mvc` in `src/main/java` from `changes`:

`WebConfig-v2.java`

Notice that we have now added a direct mapping between two URL paths and two view names directly in the `addViewControllers` method without the need to provide a separate @GetMapping handler method in any @Controller class to do this mapping. This is simple way to perform mapping if there is no need to execute any additional business logic in the handler method

Navigate to these URLs:

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

and confirm the correct views are returned.

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, Logback as well as Servlet 3.1 API.

Go to Properties -> Web Project Settings and ensure that the Context root: is set to `FormsSpringMVC`.
If it is not set it to this value, click Apply -> Ok and then click Apply and Close.

Right click on the project entry, select Show in - > System Explorer. Navigate 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`

Copy the following files from `changes` into `com.workshop.mvc` in `src/main/java`:

```
MyWebAppInitializer.java
WebConfig.java
```

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

Copy the following files from `changes` into `src/main/webapp/WEB-INF/views`

```
main-menu.jsp
employee-details.jsp
employee-form.jsp
```

Copy the following files from `changes` into `src/main/resources`

```
logback.xml
```

Copy the following files from `changes` into `com.workshop.mvc` in `src/main/java`:

```
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 in the usual manner.

Navigate to this link:

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.

Copy the following files from `changes` into `com.workshop.mvc` in `src/main/java`:

```
JobDetails.java
```

Make changes to the following files in `com.workshop.mvc` in `src/main/java` from `changes`:

`EmployeeController-v2.java`

Make changes to following files in `WEB-INF/views` from `changes`:

```
employee-details-v2.jsp
employee-form-v2.jsp
```

Deploy the app in the usual manner and navigate to:

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 changes to the following files in `com.workshop.mvc` in `src/main/java` from `changes`:

`EmployeeController-v3.java`

Deploy the app in the usual manner and navigate to:

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 @RequestParam annotation in the method signature of processFormv2 in order to bind the request parameters to the method parameters.

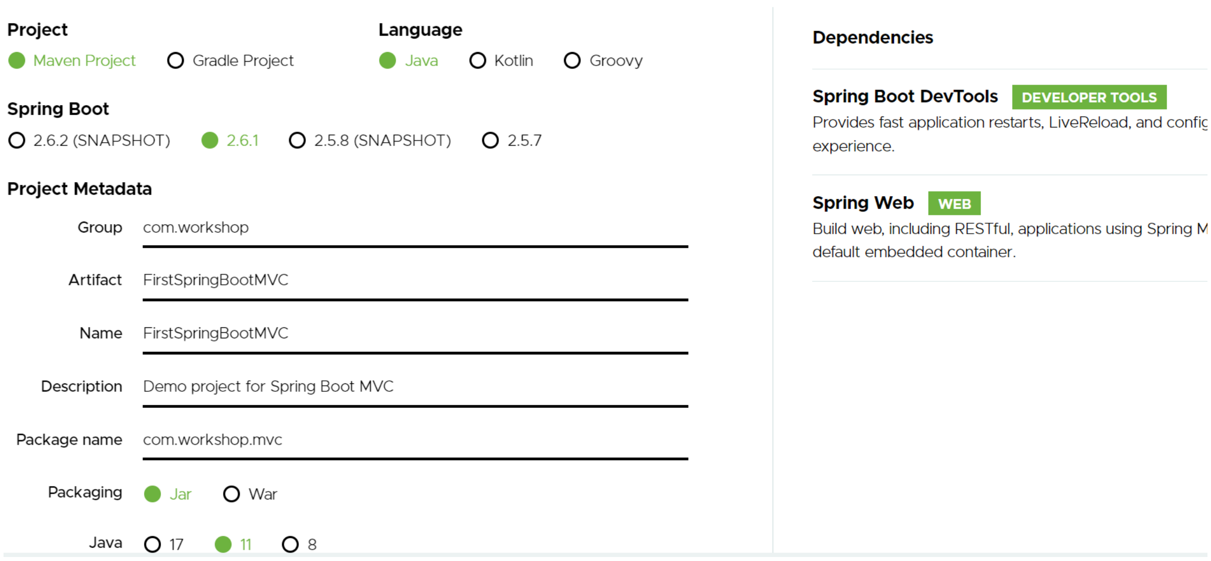
5 Creating a Spring Boot MVC application

Navigate to the Spring Initialzr at:

https://start.spring.io/

Key in the values for the fields as shown below as well as the following 2 dependencies:

- Spring Web - this is a Spring Boot Starter template which includes `spring-webmvc` as well as a few other related dependencies such as `spring-boot-starter-json, spring-boot-starter-tomcat`, etc
- Spring Boot DevTools - this provides live reloads, which facilitates application development and debugging



Project
● Maven Project ○ Gradle Project

Language
● Java ○ Kotlin ○ Groovy

Spring Boot
○ 2.6.2 (SNAPSHOT) ● 2.6.1 ○ 2.5.8 (SNAPSHOT) ○ 2.5.7

Project Metadata

Group	com.workshop
Artifact	FirstSpringBootMVC
Name	FirstSpringBootMVC
Description	Demo project for Spring Boot MVC
Package name	com.workshop.mvc
Packaging	● Jar ○ War
Java	○ 17 ● 11 ○ 8

Dependencies

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and config
experience.

Spring Web WEB
Build web, including RESTful, applications using Spring M
default embedded container.

Click Generate. Download the Zip file and unzip it in a suitable folder anywhere (you can also place it directly in your Eclipse workspace directory).

In Eclipse, switch to Java EE perspective.

Go to File -> Import -> Maven -> Existing Maven Project. Select the FirstSpringBootMVC folder that you just unzipped and click Finish.

Open the pom.xml. Notice that `spring-boot-devtools` has `runtime` scope, which means application code will not be able to access the libraries from this dependency as it is meant to only be used at runtime when performing a hot reload of the web app.

Add the following dependencies to the <dependencies> list in the POM:

```
    <!-- This allows the compilation and rendering of JSP pages -->
    <dependency>
      <groupId>org.apache.tomcat.embed</groupId>
      <artifactId>tomcat-embed-jasper</artifactId>
      <scope>provided</scope>
    </dependency>

    <!-- This provides the JSTL tags supported required in JSP pages -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
    </dependency>
```

Notice that we do not specify <version> for both of these dependencies because they are already specified in the <dependencyManagement> of `spring-boot-dependencies`, the base POM for all autogenerated Spring Boot project POMs.
https://repo1.maven.org/maven2/org/springframework/boot/spring-boot-dependencies/2.6.1/spring-boot-dependencies-2.6.1.pom

Create a `src/main/webapp/WEB-INF/views` folder in the project folder structure.

Make sure that src/main/webapp is on the project build path. Right click -> Build Path -> Use as Source Folder.

Configure the Spring MVC view resolver by adding in these properties in application.properties:

```
#Configuring Spring MVC
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
```

If there are any static HTML files (such as the normal start-up index.html), they should be placed in the static folder in src/main/resources. This is also where other static resources such as JS, CSS and images should be placed.

There is no need for any further configuration once this is done as the @EnableAutoConfiguration that is an implicit part of the @SpringBootApplication will already perform all the configuration automatically to set up a Spring MVC app correctly.
You should avoid any other kind of @Configuration class that also includes @EnableWebMvc as that will override the existing default settings provided by Spring Boot under the hood.

The source code for this lab is found in First-Spring-Boot-MVC/changes folder.

Copy the following files from changes into com.workshop.mvc in src/main/java:

```
EmployeeController.java
HomeController.java
JobDetails.java
```

Copy the following files from changes into src/main/webapp/WEB-INF/views

```
main-menu.jsp
employee-details.jsp
employee-form.jsp
```

This is the identical implementation of the previous Spring MVC app built using Maven.

To run this project in the embedded Tomcat server, we will run it via a Maven build.

First, make sure that the integrated Eclipse Tomcat instance has been stopped.

Right click on the project entry and select Run As -> 4 Maven Build -> and use spring-boot:run as the goal in the Run Configuration.
This uses the run goal of the spring-boot-maven-plugin

This starts up the embedded Tomcat server and automatically deploys the web app within the server.

Navigate to:

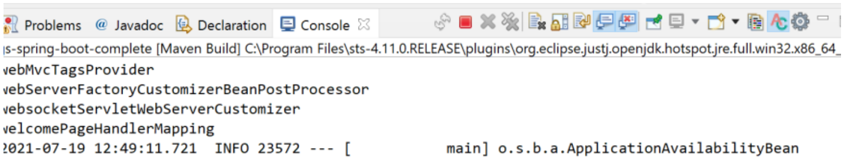
http://localhost:8080/

Notice now that the web app is accessible directly at the root path of the web server, without the need to add a qualifying context root (unlike for an app deployed into the integrated Eclipse Tomcat server instance or standalone Tomcat server).

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.

Make some minor modifications (for e.g. put an extra space in the String) in any of the @Controller classes in `com.workshop.mvc` and save. Notice that the server automatically reloads a couple of seconds later (hot reload from the `spring-boot-devtools` functionality).

If you are running from inside Eclipse, you will need to click on the Red button in the upper right hand corner of the Console view to terminate the embedded Tomcat server.



```
s-spring-boot-complete [Maven Build] C:\Program Files\sts-4.11.0.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_
vebMvcTagsProvider
vebServerFactoryCustomizerBeanPostProcessor
vebsocketServletWebServerCustomizer
velcomePageHandlerMapping
2021-07-19 12:49:11.721  INFO 23572 --- [           main] o.s.b.a.ApplicationAvailabilityBean
```

We will now add in some additional Spring Boot specific properties to `application.properties` to demonstrate how we can configure the app deployment:

```
#Configuring Spring MVC
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp

#Provides a context-root to deploy the app on
server.servlet.context-path=/simpleapp

#Change from the default port of 8080
server.port=8081
```

Stop the server and restart it - right click on the project entry and select Run As -> 3 Maven Build

Navigate to:

http://localhost:8081/simpleapp/

NOTE: If you accidentally restart the app without first stopping a previously running instance, the embedded Tomcat server instance may fail to start up due to a port conflict with this previously running instance. In that case, you will need to locate the previously running server instance and terminate it first by searching for it at the port number you configured for it to deploy on:

```
netstat -aon | findstr :port-number

TCP    0.0.0.0:port-number    0.0.0.0:0          LISTENING      PID
TCP    [::]:port-number       [::]:0             LISTENING      PID

tasklist | findstr PID
```

```
someprogram.exe    PID Console                  11    219,532 K
```

```
taskkill /PID PID /F
```

```
SUCCESS: The process with PID PID has been terminated.
```

5.1 Creating a simple data storage / retrieval app

We will refactor this app to introduce a few more packages and classes to structure it as a simple data storage / retrieval app.

Delete all the *.java classes from `com.workshop.mvc` with the exception of `FirstSpringBootMvcApplication`

Create the following package: `com.workshop.mvc.model`
Copy the following files from `changes` into this package:

```
EmployeeDetails.java
```

Create the following package: `com.workshop.mvc.service`
Copy the following files from `changes` into this package:

```
EmployeeService.java
```

Create the following package: `com.workshop.mvc.repository`
Copy the following files from `changes` into this package:

```
EmployeeRepository.java
InMemoryEmployeeRepository.java
```

Create the following package: `com.workshop.mvc.controller`
Copy the following files from `changes` into this package:

```
HomeController.java

EmployeeController-v2.java
```

Delete the following files in `WEB-INF/views`:

```
employee-details.jsp
employee-form.jsp
```

Make changes to following files in `WEB-INF/views` from `changes`:

```
main-menu-v2.jsp
```

Copy the following files from `changes` into `WEB-INF/views`

```
display-all-employees.jsp
```

```
display-single-employee.jsp
get-employee-form.jsp
new-employee-form.jsp
```

Notice that we now apply the @Repository and @Service annotations to the InMemoryEmployeeRepository and EmployeeService classes respectively, which means that the container will automatically load and register them as beans for purposes of DI into classes with @Autowired dependencies that need to use them.

The @Service class is typically used as an intermediate layer between the @Controller classes and the @Repository implementations. It is not specifically needed here because its simply exposes methods that function as wrappers around its single EmployeeRepository dependency. For more complex applications however, the @Service class could incorporate business logic within it that augments the basic CRUD functionality offered by the EmployeeRepository dependency with additional processing.

By providing a EmployeeRepository interface that exposes common CRUD-related methods, we provide ourselves the flexibility of introducing multiple implementations at a later period of time (for e.g. SQLEmployeeRepository, NoSQLEmployeeRepository, etc, etc) and then using either the @Primary or @Qualifier annotations to select one of the possible candidate beans as discussed earlier in a previous lab.

Stop the server and restart it - right click on the project entry and select Run As -> 3 Maven Build

Navigate to:

http://localhost:8081/simpleapp/

Verify the basic functionality of the app by navigating to the various pages and executing the requisite functionality.

5.2 Creating a Spring Boot app via STS

The Spring Tool Suite (STS) is a plugin for the Eclipse Enterprise IDE which provides additional features that supports development of Spring projects. This can be installed as a plugin:

https://www.codejava.net/ides/eclipse/install-spring-tool-suite-for-existing-eclipse-ide

or downloaded as a full Eclipse version by itself at:

https://spring.io/tools

Start up STS and select an appropriate folder for your workspace.

If you are starting with a new workspace, the latest version of STS comes with its own installed JRE for Java 16. It would be better to change the installed JRE and compiler compliance to the current version of JDK installed on your machine.

Go to Window -> Preferences -> Java -> Installed JREs.

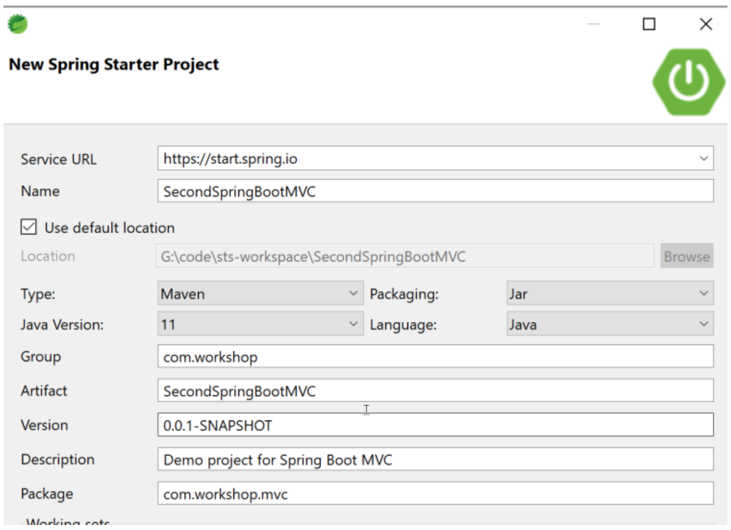
Click on Add. In the JRE Type dialog box, select Standard VM. Click Next.

In the JRE Home entry, click the Directory button and navigate and select the installation directory of the JDK, for e.g: C:\Program Files\Java\jdk-11.0.13
Click Finish, then select the JDK entry and click Apply and Close.

Go to Window -> Preferences -> Java -> Compiler.
In the Compiler Compliance Settings, set it to the version of the JDK installed on your machine (for e.g. 11). Click Apply and Close.

Switch to the Java perspective.

Go to File -> New -> Spring Starter Project. The dialog box that appears provides the same options that are presented at Spring Initialzr (https://start.spring.io/). Complete it with the following details:

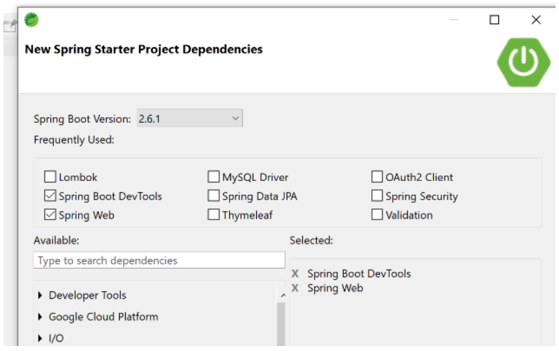


New Spring Starter Project

Service URL	https://start.spring.io
Name	SecondSpringBootMVC
☑ Use default location	
Location	G:\code\sts-workspace\SecondSpringBootMVC
Type:	Maven
Packaging:	Jar
Java Version:	11
Language:	Java
Group	com.workshop
Artifact	SecondSpringBootMVC
Version	0.0.1-SNAPSHOT
Description	Demo project for Spring Boot MVC
Package	com.workshop.mvc

Click Next and select the following dependencies.

```
Web -> Spring Web
Developer Tools -> Spring Boot DevTools
```



New Spring Starter Project Dependencies

Spring Boot Version: 2.6.1
Frequently Used:

☐ Lombok	☐ MySQL Driver	☐ OAuth2 Client
☑ Spring Boot DevTools	☐ Spring Data JPA	☐ Spring Security
☑ Spring Web	☐ Thymeleaf	☐ Validation

Available:
Type to search dependencies
- Developer Tools
- Google Cloud Platform
- I/O

Selected:
X Spring Boot DevTools
X Spring Web

Click Finish.

This generates a Maven project with a Maven wrapper that is identical to the one that we created via the Spring Initialzr in the earlier lab.

Add the following dependencies to the <dependencies> list in the POM:

```
    <!-- This allows the compilation and rendering of JSP pages -->
    <dependency>
      <groupId>org.apache.tomcat.embed</groupId>
      <artifactId>tomcat-embed-jasper</artifactId>
      <scope>provided</scope>
    </dependency>

    <!-- This provides the JSTL tags supported required in JSP pages -->
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
    </dependency>
```

Save the POM and do a Maven -> Update Project

Create a `src/main/webapp/WEB-INF/views` folder in the project folder structure.
Make sure that `src/main/webapp` is on the project build path. Right click -> Build Path -> Use as Source Folder.

We populate this new project with the same package and classes as the previous project.

Create the following package: `com.workshop.mvc.model`
Copy the following files from `changes` into this package:

```
EmployeeDetails.java
```

Create the following package: `com.workshop.mvc.service`
Copy the following files from `changes` into this package:

```
EmployeeService.java
```

Create the following package: `com.workshop.mvc.repository`
Copy the following files from `changes` into this package:

```
EmployeeRepository.java
InMemoryEmployeeRepository.java
```

Create the following package: `com.workshop.mvc.controller`
Copy the following files from `changes` into this package:

```
HomeController.java
EmployeeController-v2.java (rename to EmployeeController)
```

Copy the following files from `changes` into `WEB-INF/views`

```
display-all-employees.jsp
display-single-employee.jsp
get-employee-form.jsp
main-menu-v2.jsp (rename to main-menu.jsp)
new-employee-form.jsp
```

Replace `application.properties` in `src/main/resources` with the version from `changes:`

Right click on the project entry, select Run As -> Spring Boot App.

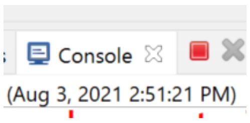
The same console output that you saw in the previous lab is produced, but this time with color formatting to facilitate viewing:

Once the application has started up successfully, navigate to:

http://localhost:8081/simpleapp/

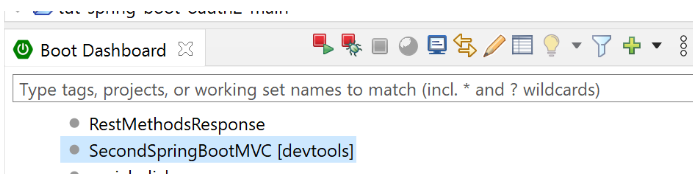
and check for the same results as in the previous lab.

To stop the embedded Tomcat server, click on the red button next to the Console view.

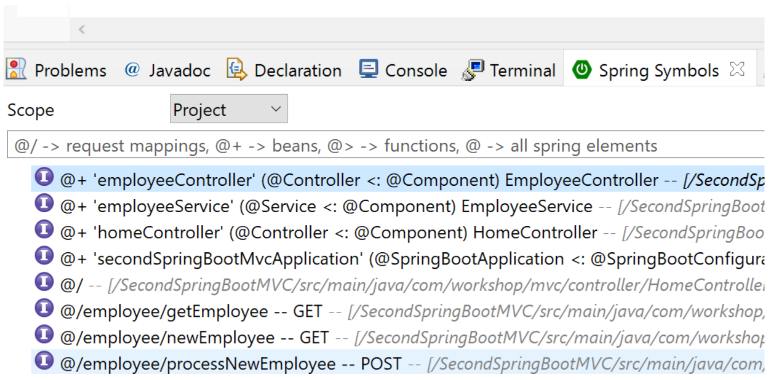


The Boot Dashboard on the left helps facilitate working with the Spring Boot apps. If you can't see this view, go to Window -> Show View -> Other -> look in the Other folder -> Boot Dashboard.

From the drop down list for local, you can see the list of Spring Boot apps available for interacting with. Select an entry and a list of action icons (such as running, stopping or debugging the app)


Boot Dashboard
Type tags, projects, or working set names to match (incl. * and ? wildcards)
RestMethodsResponse
SecondSpringBootMVC [devtools]

STS provides a Spring Symbols view which helps to identify Spring related annotations in your various classes. If you can't see this view, go to Window -> Show View -> Other -> look in the Other folder -> Spring Symbols. Select the SecondSpringBootMVC project entry and select Project in the Spring Symbols view. You should be able to see all the Spring annotated annotations in the Project:



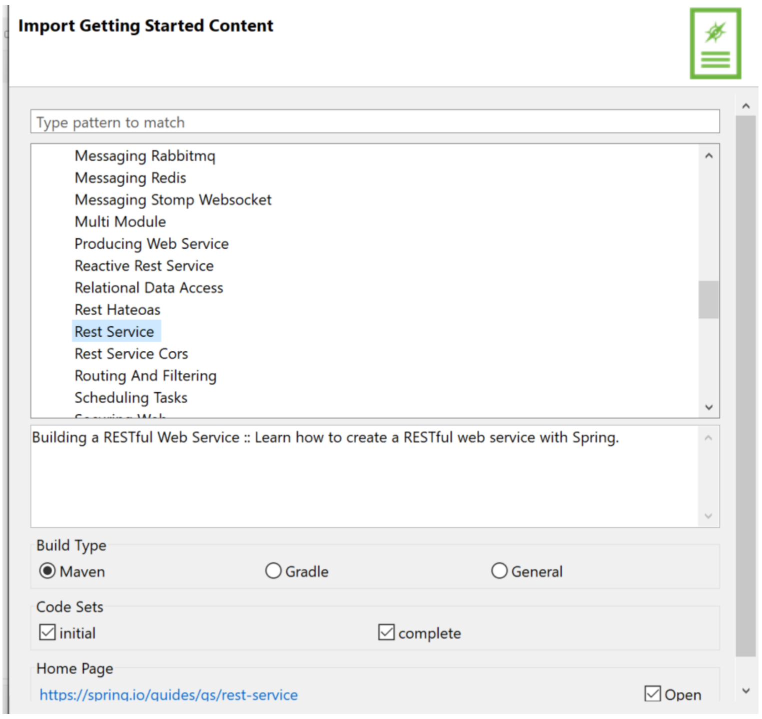
```
@/ -> request mappings, @+ -> beans, @> -> functions, @ -> all spring elements
@+ 'employeeController' (@Controller <: @Component) EmployeeController -- [/SecondSp
@+ 'employeeService' (@Service <: @Component) EmployeeService -- [/SecondSpringBoot
@+ 'homeController' (@Controller <: @Component) HomeController -- [/SecondSpringBoo
@+ 'secondSpringBootMvcApplication' (@SpringBootApplication <: @SpringBootConfigura
@/ -- [/SecondSpringBootMVC/src/main/java/com/workshop/mvc/controller/HomeControlle
@/employee/getEmployee -- GET -- [/SecondSpringBootMVC/src/main/java/com/workshop,
@/employee/newEmployee -- GET -- [/SecondSpringBootMVC/src/main/java/com/workshop
@/employee/processNewEmployee -- POST -- [/SecondSpringBootMVC/src/main/java/com,
```

Similarly, you can go to the individual source code files (`EmployeeController,etc`) and select File in the Spring Symbols view to see the annotations at the individual file level.

STS also provides the ability to directly import any of the projects from the Getting Started guides from the main Spring home page:
https://spring.io/guides#getting-started-guides

For e.g. this tutorial details how we can build a simple REST service in the Spring framework step-by-step:
https://spring.io/guides/gs/rest-service/

To create a new Spring Boot project that illustrates this tutorial, go to File -> New -> Import Getting Started Content:
In the Dialog Box, select Getting Starting Guide -> Rest Service and check the other options as follows:



Import Getting Started Content

Type pattern to match

Messaging Rabbitmq
Messaging Redis
Messaging Stomp Websocket
Multi Module
Producing Web Service
Reactive Rest Service
Relational Data Access
Rest Hateoas
Rest Service
Rest Service Cors
Routing And Filtering
Scheduling Tasks

Building a RESTful Web Service :: Learn how to create a RESTful web service with Spring.

Build Type
[✓ Maven] Gradle General

Code Sets
[✓ initial] [✓ complete]

Home Page
https://spring.io/guides/gs/rest-service [✓ Open]

This creates two projects in the Package Explorer:

- gs-rest-service-initial (the initial code base from which you can follow the tutorial)
- gs-rest-service-complete (the final code base at the end of the tutorial)

as well as serving up the webpage documenting that particular tutorial.

Select `gs-rest-service-complete` in the Boot Dashboard and run it.

Once the application has started up successfully, open a browser tab at:

http://localhost:8080/greeting

You can use STS in this way to facilitate walking through the ever growing number of tutorials available on the Spring ecosystem.