**Hands On Lab 2**

**Spring MVC with JSP**

In this lab, you will learn how to create a Spring Boot project for developing a Java web application with Spring MVC and JSP, running in an embedded Tomcat server. The project in this tutorial is developed with JDK 11 and Spring Tool Suite IDE with Maven.

If you ever developed Spring MVC applications before Spring Boot’s times, you know that it requires writing pretty much boiler-plate code just for the configuration which you use more or less the same for almost all projects. This repeated and boring task consumes programmer’s time which should be spent on developing the business logic for the application.

And now with Spring Boot, things get a lot easier. All the configurations are automatically done for you (auto configuration) with sensible defaults. So you can quickly jump to coding the business logic without worrying about the configuration details.

Now, let follow this lab, step by step, to understand how to use the sensible defaults when developing a Spring MVC web application with JSP/ThymeLeaf.

**Table of Contents:**

**1. Create Spring Boot Project in Eclipse**

**2. Default welcome page**

**3. Create Model, Business and Controller Classes**

**4. How to use JSP with Spring Boot**

**5. How to use CSS, Javascript and Images**

**6. Configure Logging**

7. Configure Server Port Number and Context Path

8. How to package the application to JAR/WAR

## **1. Create Spring Boot Project in Eclipse**

You need to generate a Spring Boot project using Spring Initializr tool in STS with the web dependency.

Open the pom.xml file to see Maven dependencies. You see it’s very simple, there are only two dependencies declared:

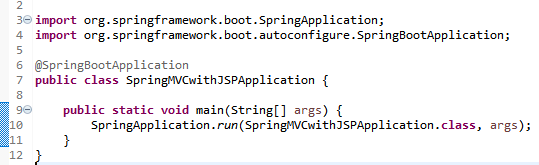
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <**dependencies**>      <**dependency**>          <**groupId**>org.springframework.boot</**groupId**>          <**artifactId**>spring-boot-starter-web</**artifactId**>      </**dependency**>        <**dependency**>          <**groupId**>org.springframework.boot</**groupId**>          <**artifactId**>spring-boot-starter-test</**artifactId**>          <**scope**>test</**scope**>      </**dependency**>  </**dependencies**> |

The dependency ****spring-boot-starter-web**** enables web development with Spring MVC and embedded Tomcat server. It requires all common dependencies (Spring framework, logging, validation, JSON, …) with proper versions so you don’t have to struggle with a lot of dependencies and their versions like normal Spring MVC application development.

You can remove the dependency spring-boot-starter-test if you don’t use tests. When doing so, also delete the test class SpringMVCwithJSPApplication.java (no hyphens) under src/test/java directory.

Expand the ****Maven Dependencies**** folder to see various JAR files required by the project, and the good thing is you don’t have to do anything to configure this. Spring Boot includes all these defaults.

SpringMVCwithJSPApplication****.java**** (no hyphens) :this is the entry point class to run the application. It has the following code:



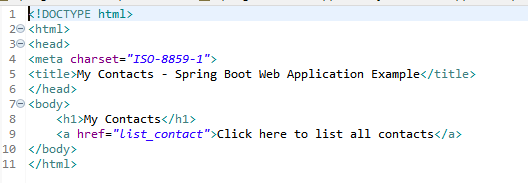
The ****@SpringBootApplication****annotation does the magic work to start the embedded Tomcat server, configure Spring Dispatcher Servlet, etc.

And there are two default directories and one properties file in the src/main/resources folder.

* ****static:****put your static files here, e.g. HTML files.
* ****templates:****put your template files here, e.g. ThymeLeaf files.
* ****application.properties****: specify additional configurations here, e.g. logging, Spring MVC view resolver, server port number, etc.

## **2. Code default welcome page**

If you have the static welcome page (index.html) for the application, put in under the src/main/resources/static directory. For example, create the index.html file with the following content:



Use the Boot Dashboard in STS to run the application. You can see the application is started with Spring Boot logo and logging messages from Tomcat and Spring framework.

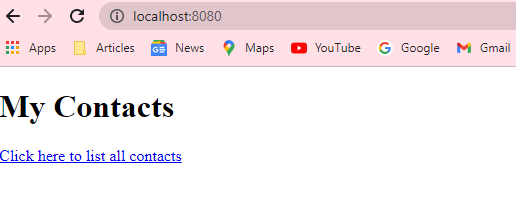
Notice this line in the *Console* view at the end of the logs:



By default, the server is listening on port number 8080 with empty context path, so type the following URL in your web browser:

|  |  |
| --- | --- |
| **1** | **http://localhost:8080/** |

Then you can see the welcome page gets displayed:



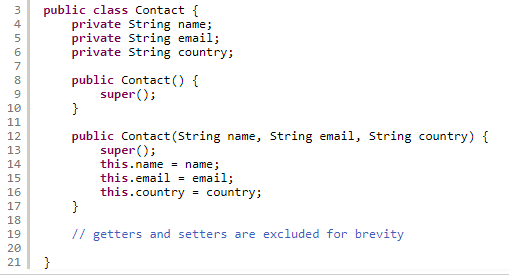
You can override the default welcome page by creating a handler method in a [Spring controller](https://www.codejava.net/frameworks/spring/14-tips-for-writing-spring-mvc-controller" \t "https://www.codejava.net/frameworks/spring-boot/_blank) class like this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @RequestMapping("/")  **public** String welcome() {    **return** "index";  } |

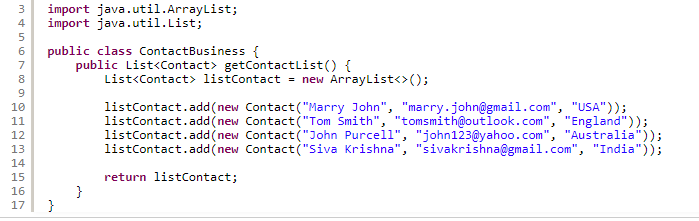
As you can see, this method handles requests with the URL pattern “/” which is the context root (home page) of the application, and it returns a view named “index” which is resolved to actual view page by Spring view resolver.

## **3. Code Model, Business and Controller Classes**

Suppose that when the user clicks the hyperlink Click here to list all contacts, the web application displays a list of contacts. Create the model class Contact as follows:



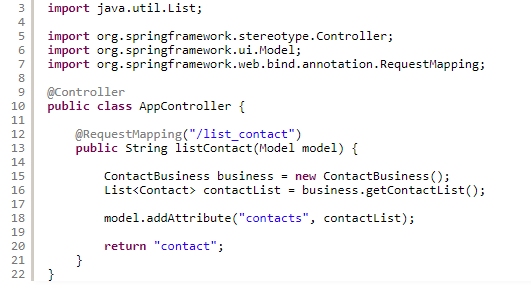
Next, create a business class that implements a method to return a hard-coded List of Contact objects like this:



Next, create a Spring controller class to handle requests coming from the hyperlink in the welcome page above:



Name this class as AppController with the following code:

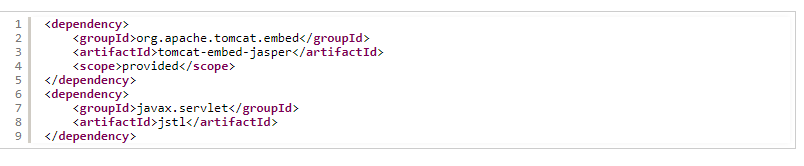


As you can see, this is standard Spring web controller class annotated with the @Controller annotation. The @RequestMapping("/list\_contact") annotation specifies that the listContact() method is responsible to handle requests coming from the /list\_contact URL.

This method uses the ContactBusinessclass to get a List of Contact objects and puts this collection to a Model object, which will be then used by the view. Finally it returns the view name “contact”, which can be resolved to a JSP page or a ThymeLeaf template, depending on our configuration.

## **4. How to use JSP with Spring Boot**

We will create JSP page with JSTL to display the contact list, so add the following two dependencies in the pom.xml file:

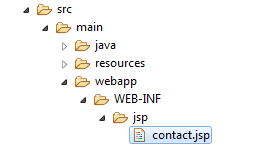


The dependency tomcat-embed-jasper is JSP Engine for Tomcat. It is required to enable JSP with Spring Boot.

By default, JSP pages are looked up inside /webapp directory. So under the src/main directory, create a new directory named webapp. If no view resolvers are configured, a handler method should return the actual view file name, for example:



To follow common convention and to protect JSP pages, create directories WEB-INF/jsp inside webapp. And create contact.jsp file under src/main/webapp/WEB-INF/jsp as shown in the following screenshot:



To configure Spring MVC view resolver, open the application.properties file and put the following properties:

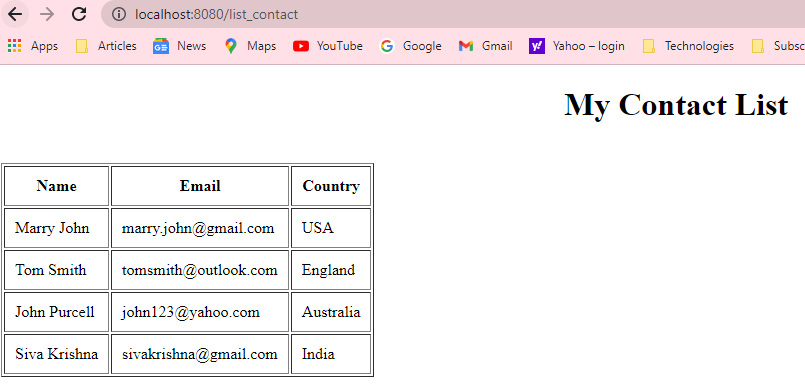


Now, let’s update code for the contact.jsp file as follows:



As you can see, this JSP page uses JSTL’s forEach tag to iterate over Contact objects in the collection named contacts in the model, and generate HTML code to display the contact list.

Stop and re-run the SpringMVCwithJSPApplication program. Click the hyperlink in the welcome page and the contact list appears, as shown in the following screenshot:



Voila, it works as expected!

## **5. How to use CSS, Javascript and Images**

You can use these types of web files as normal, just make sure to put them under src/main/resources/templates if you use ThymeLeaf and under src/main/webapp/WEB-INF in case you use JSP.

## **6. Configure Logging**

Spring Boot uses various logging libraries: Commons Logging for its core and Java Util Logging, Log4J2 and Logback for the dependencies which use different logging libraries. The default log format is pretty fine and the default log level is INFO.

If you want to change the log level for all libraries, specify the following entry in the application.properties file:

|  |  |
| --- | --- |
| 1 | logging.level.root=<LEVEL> |

The log level can be one of FATAL, ERROR, WARN, INFO, DEBUG, TRACE or OFF. For example:

|  |  |
| --- | --- |
| 1 | logging.level.root=WARN |

This changes the log level to WARN (show only warning and error messages) for all loggers.

For fine tuning, you can change log level for a specific Java package by specifying the following entry:

|  |  |
| --- | --- |
| 1 | logging.level.<package\_name>=<LEVEL> |

For example:

|  |  |
| --- | --- |
| 1 | logging.level.com.mycompany.springbootwebapp=ERROR |
|  |  |

This changes log level for all classes in the com.mycompany.springbootwebapp package to ERROR.

To enable debug mode for the application, specify ****debug=true**** in your application.properties file or specify the ****--debug**** flag as an argument when running the program, for example:

|  |  |
| --- | --- |
| 1 | java -jar yourappname.jar --debug |

Be patient when running the application in debug mode because it prints a huge amount of information, which slows down the application significantly.

## **7. Configure Server Port Number and Context Path**

The default port number of embedded Tomcat server is 8080. You can override this default by specifying the ****server.port**** property in the application.properties file like this:



And the default context path is empty (‘’) so you can access the web application via http://localhost. To override this default, specifying the ****server.servlet.context-path**** in the application.properties file, for example:



With two changes above, you now see the contact list via this URL:

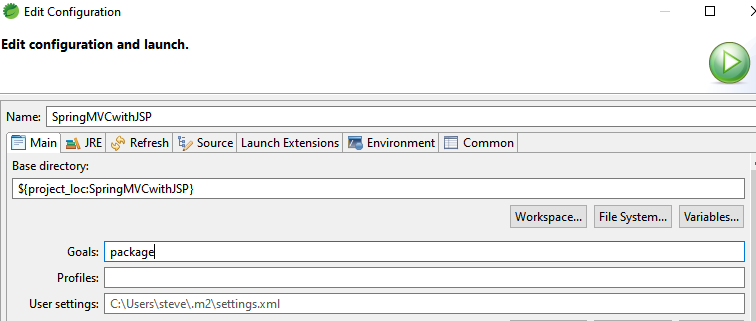


## **8. How to package Spring Boot web application to JAR/WAR**

A major advantage of Spring Boot is that it packages your web application and embedded Tomcat server into a single executable JAR file, which is very convenient. However, JSP doesn’t work with JAR packaging . So if you use JSP, you must change the packaging type to WAR in your pom.xml as follows:



To package your Spring Boot web application to a JAR or WAR file in Eclipse, right-click on the project name, click ****Run As > Maven build…**** In the *Edit Configuration* dialog, type ****package**** as the goal and click ****Run****:



If using command line, change the current directory is your project directory, and simply type:

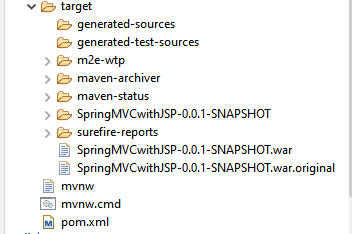
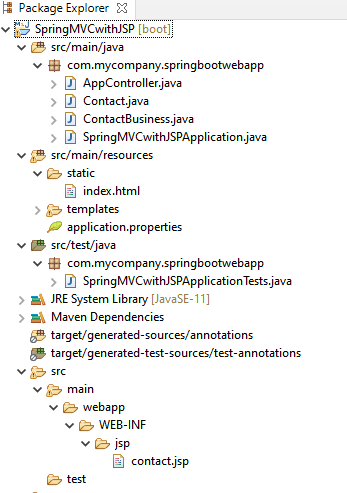


Then check the target folder in your project directory, a JAR/WAR file is generated depending on the packaging type you use. And run the application:



****Note:**** Use the same command to run a WAR file because it is executable.

And for your reference, we have the final project structure which looks like this:



Good Job! That’s how to develop a Spring MVC web application with Spring Boot in STS IDE. Almost the configuration is done by Spring Boot so you can quickly start coding your business logic, following the sensible defaults. And keep in mind that you have to use war packaging with JSP.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*