Spring MVC Tutorial – Contact List Application

Step 01: Hello, MVC

SOFTWARE GUILD

The Learning House

427 S 4th Street #300

Louisville KY 40202

Step 01: Hello, MVC

Overview

In Step 1, we will create and look at a new Spring MVC web application using the custom SWCGuild Spring MVC Maven archetype. This tutorial assumes that you have Java, NetBeans, Maven, and Tomcat properly installed and configured. If this is not the case, please see the Software Craftsmanship Guild **Maven/Tomcat/Spring Environment Setup** document.

This tutorial also assumes that you are familiar with the configuration, structure, and operation of Servlet/JSP based web applications.

Install SWCGuild Spring MVC Maven Archetype Locally

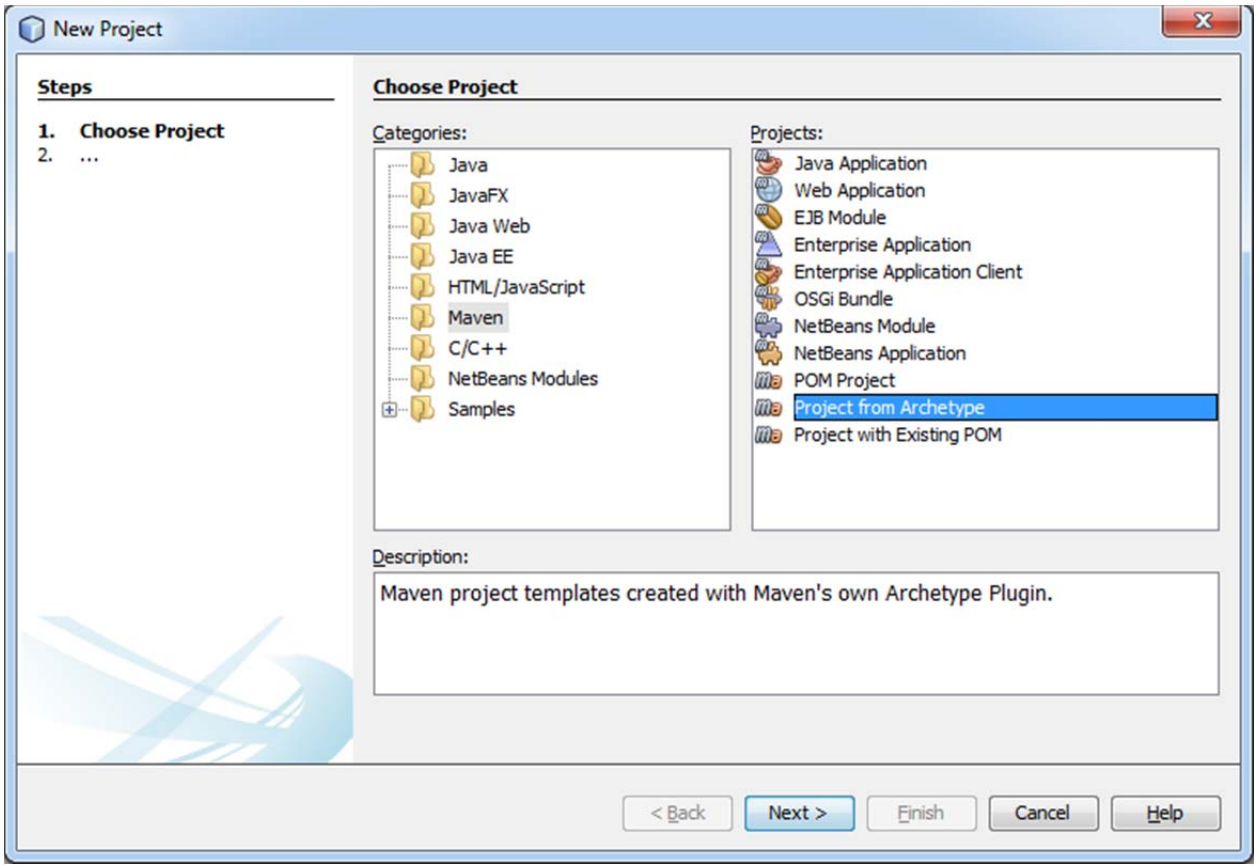
Before we can create our new Spring MVC application, we need to install the SWCGuild Spring MVC Maven Archetype:

1. Clone the swcguild/Tools repository
2. Go to the **Java/maven/Spring MVC Archetype Project** directory and follow the directions in the README.txt file

Create New Project

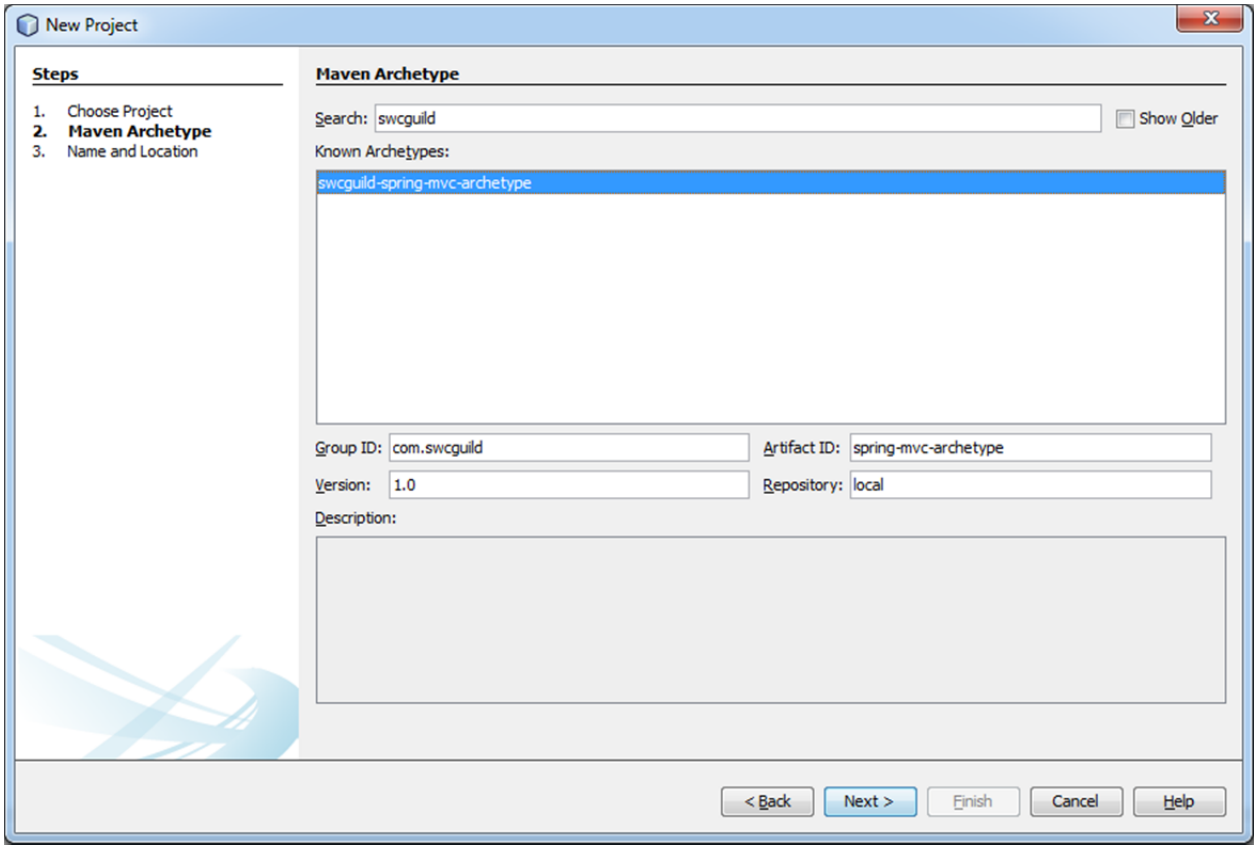
The next step is to create our new Spring MVC project:

1. Open NetBeans
2. Select **File → New Project**
3. In the **Categories** pane, choose **Maven**
4. In the **Projects** pane, choose **Project from Archetype**



5. Click **Next**

6. Type in **swcguild** in the Search textbox
7. Select **swcguild-spring-mvc-archetype** under Known Archetypes

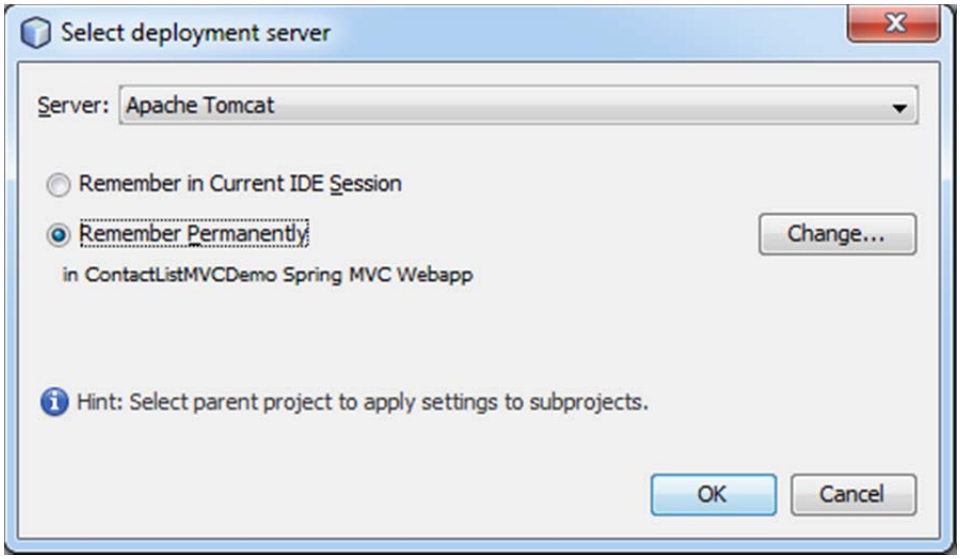


8. Click **Next**
9. Type **ContactListMVC** for the Project Name
10. Type in **com.swcguild** for your Group Id
11. Click **Finish**

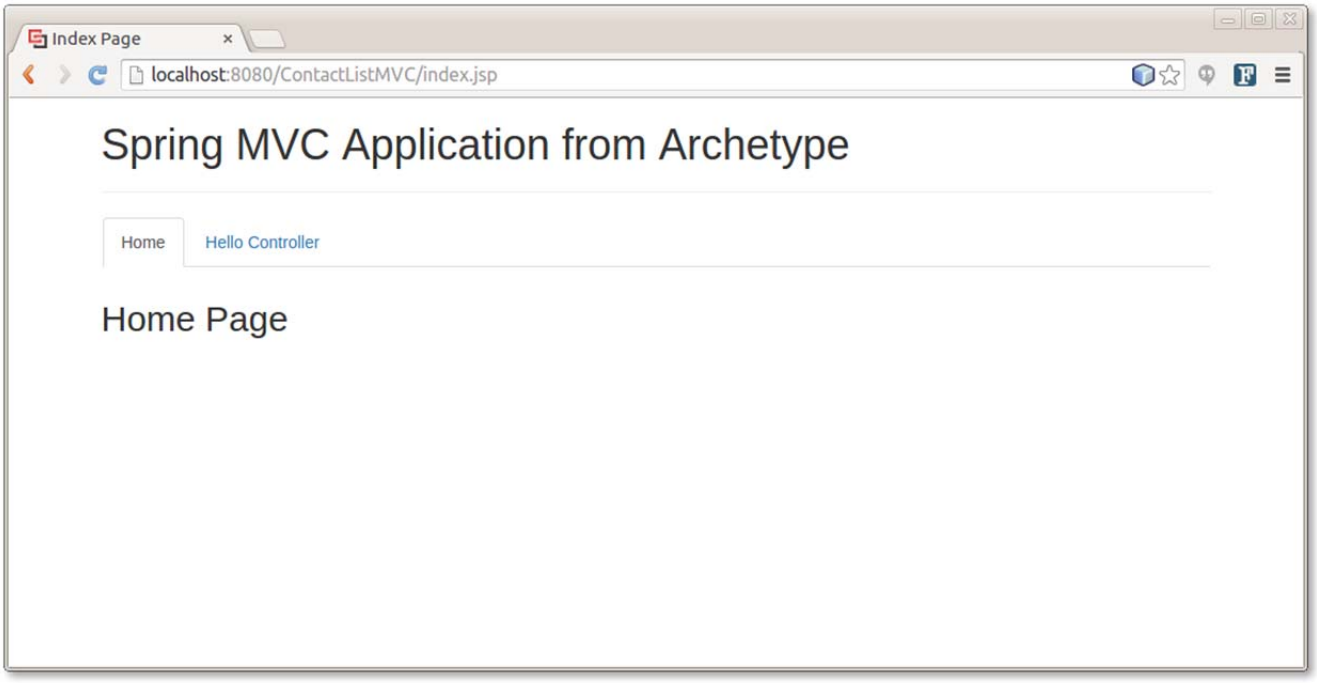
Test New Project

Now we need to test our new project:

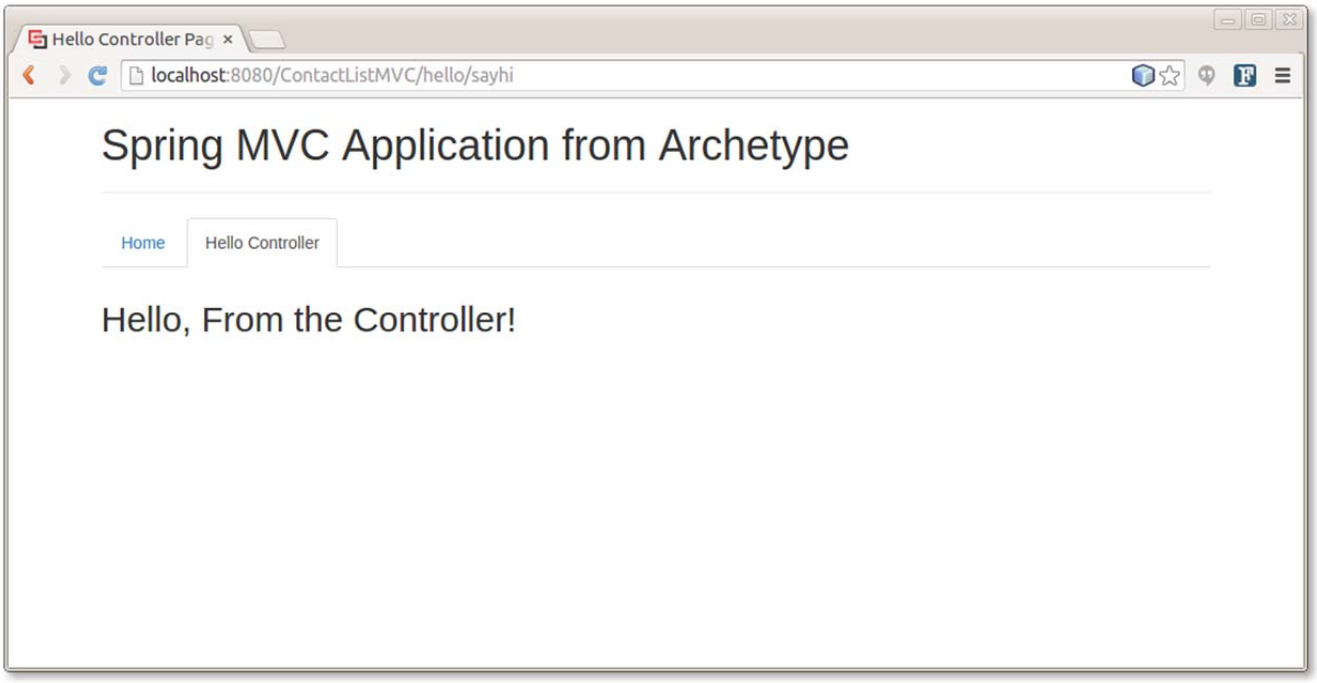
1. Highlight the new project (**ContactListMVC Spring MVC Webapp**) in the Projects pane
2. Select **Run** → **Run Project** (or click the green Run Project arrow, or hit F6)
3. You will be prompted to select a deployment server: select your Tomcat server, click the **Remember Permanently** radio button, and then click **OK**



4. Your project should build, deploy to Tomcat, and then display in your browser like this:



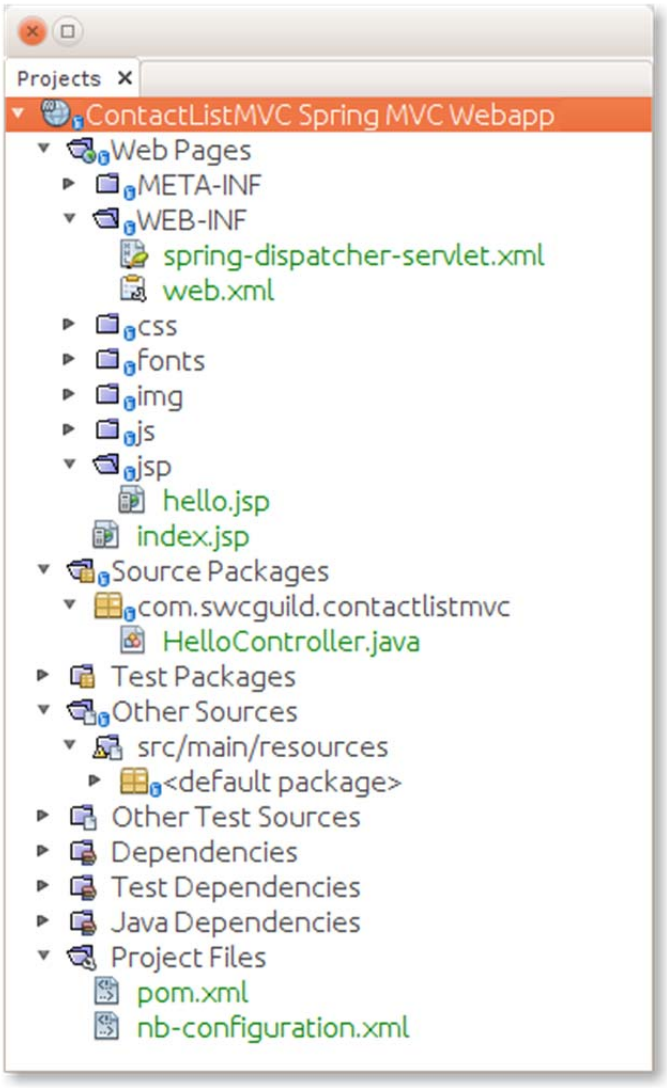
5. When you click on the **Hello Controller** link, you should see this:



Explore the New Project

The SWC Guild Maven Spring MVC Archetype creates a complete Spring MVC application. Although this application is simple, it contains all the basic building blocks needed for a more substantial application.

The figure below highlights the important files of our initial application:



Notation Conventions:

The following sections display the contents of the important files in our applications. Each file is listed with an overview and a numbered list containing explanations of specific parts of the file. Each number in the list corresponds to a number comment in the code which follows the description (all such comments will appear in red font).

spring-dispatcher-servlet.xml

The spring-dispatcher-servlet.xml contains the configuration for Spring MVC. This file has four important pieces:

1. All the XML namespaces for Spring configuration
2. Entry that allows us to specify Spring MVC components via annotations
3. Entry that specifies where Spring should start looking for annotated Spring MVC components
4. The View Resolver configuration:
    a. In our case, we are using the InternalResourceViewResolver
    b. We configure it to prefix all view names returned from controllers with /jsp/ (this tells Spring MVC to look for view files in the jsp folder)
    c. We configure it to consider all files ending in .jsp

```
<!-- #1: XML namespace declarations start -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context-3.0.xsd
  http://www.springframework.org/schema/mvc
  http://www.springframework.org/schema/mvc/spring-mvc.xsd">
<!-- #1: XML namespace declarations end -->

  <!-- #2 Specify Spring MVC configuration is annotation driven start  -->
  <mvc:annotation-driven />
  <!-- #2 Specify Spring MVC configuration is annotation driven end  -->

<!-- #3 Specify where Spring should look for MVC components start  -->
<context:component-scan base-package="com.swcguild.contactlistmvc" />
<!-- #3 Specify where Spring should look for MVC components end  -->

<!-- #4 Specify our View Resolver start  -->
<bean
    <!-- #4-a Use InternalResourceViewResolver  -->
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!-- #4-b Prefix all view names with /jsp/  -->
        <property name="prefix">
            <value>/jsp/</value>
        </property>
         <!-- #4-c Append .jsp to all view names  -->
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>

 </beans>
<!-- #4 Specify our View Resolver end  -->
```

web.xml

The web.xml is a standard configuration file for servlet-based applications. In web.xml, we configure four things:

1. Our welcome file list
2. Servlet configuration:
    a. Register the Spring DispatcherServlet
    b. Create a servlet-mapping entry for it (our servlet will be called for any web request starting with /ContactListMVC/). The Spring DispatcherServlet is the only servlet we register in our application because Spring MVC uses the Front Controller design pattern (see here in Section 13.2). This means that all requests to our application will first be handled by the DispatcherServlet, which will route the requests to the proper controller.
    c. Tell Tomcat to use the **default** servlet when serving up static files such as JavaScript, CSS, and images
3. Location of Spring context file(s): we use the <context-param> entry to tell Spring where to find its configuration files
4. Context Loader Listener: tells the servlet container (Tomcat) to load the Spring context when the servlet context is loaded; this means that Spring is automatically loaded when your web application starts up

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
id="WebApp_ID" version="2.5">
  <display-name>Archetype Created Spring MVC Web Application</display-name>
  <!-- #1 Specify welcome files start  -->
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
  <!-- #1 Specify welcome files end  -->

  <!-- #2 Servlet mapping start  -->
  <!-- #2-a Register Spring DispatcherServlet  -->
  <servlet>
        <servlet-name>spring-dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  </servlet>
  <!-- #2-b Create mapping for DispatcherServlet  -->
  <servlet-mapping>
        <servlet-name>spring-dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
  </servlet-mapping>
  <!-- #2-c Create mapping for DispatcherServlet  -->
  <servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>*.js</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>*.css</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>*.eot</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>*.svg</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>*.ttf</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>*.woff</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>*.jpg</url-pattern>
  </servlet-mapping>
```

```
  <servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>*.png</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
      <servlet-name>default</servlet-name>
      <url-pattern>*.gif</url-pattern>
  </servlet-mapping>
  <!-- #2 Servlet mapping end -->

  <!-- #3 Spring context file entries start  -->
  <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/spring-dispatcher-servlet.xml
            classpath:spring-persistence.xml
        </param-value>
    </context-param>
    <!-- #3 Spring context file entries end  -->

    <!-- #4 Tell Tomcat to load Spring context when web app is loaded start  -->
    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>
    <!-- #4 Tell Tomcat to load Spring context when web app is loaded end  -->
</web-app>
```

HelloController.java

This is the one and only controller component in our simple application. It just sends a message ("Hello from the controller") to the view JSP (hello.jsp):

1. @Controller annotation: tells Spring that this class is a Controller component. Items #2 and #3 in the spring-dispatcher-servlet.xml section above show how we configure Spring to start looking for MVC components in the com.swcguild.contactlistmvc package.
2. @RequestMapping annotation: tells the Spring MVC framework that methods in this class should be mapped to the web request starting with /ContactListMVC/hello/. Items #2a and #2b in the web.xml section above show how we configure Spring to pay attention to all web requests starting with /ContactListMVC/.
3. Another @RequestMapping annotation: tells the Spring MVC framework that this particular method (sayHi) should be called when the web request /ContactListMVC/hello/sayhi is received. Note that this @RequestMapping annotation has an attribute called 'method' — this means that this method will only get called for HTTP GET requests to the /ContactListMVC/hello/sayhi URL.
4. Put an object (in our case, a String) into the model map. The model then gets passed to the view component (hello.jsp) where it can be accessed and/or displayed.
5. Return the name of the view we want. Items #4a, #4b, and #4c in the spring-dispatcher-servlet.xml section above show how we configure Spring to find views.

```java
package com.swcguild.contactlistmvc;

import java.util.Map;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

// #1 - Controller annotation
@Controller
// #2 - Request Mapping annotation
@RequestMapping({"/hello"})
public class HelloController {
    public HelloController() {
    }
    // #3 - Request Mapping annotation
    @RequestMapping(value="/sayhi", method=RequestMethod.GET)
    public String sayHi(Map<String, Object> model) {
        // #4 - Objects put in the model are passed to the view
        model.put("message", "Hello from the controller" );
        // #5 - Return the name of the view we want (i.e. hello.jsp)
        return "hello";
    }
}
```

hello.jsp

This jsp is the one and only view component for our simple application. Hello.jsp simply takes the message ("Hello from the controller" which is found in the ${message} variable) from the controller (HelloController.java) and displays it to the screen.

1. Directives for this page: the first three import the taglibs for JSTL and Spring so they are available for use in the page. These taglibs are not used in this version, but the imports are included so hello.jsp can be used as a template for other JSP files. The fourth entry is a page directive that specifies the content type (HTML) and the encoding (UTF-8).
2. Access and display the variable 'message' using expression language (EL) notation. This value was set by the controller (see Item #4 in the HelloController.java section above to see how this value was set).

```
<!-- #1 Directives  -->
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="s" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>Hello Controller Page</title>
        <!-- Bootstrap core CSS -->
        <link href="${pageContext.request.contextPath}/css/bootstrap.min.css" \
              rel="stylesheet">
        <!-- Custom styles for this template -->
        <link href="${pageContext.request.contextPath}/css/starter-template.css"
              rel="stylesheet">
        <!-- SWC Icon -->
        <link rel="shortcut icon" href="${pageContext.request.contextPath}/img/icon.png">
    </head>
    <body>
        <div class="container">
            <h1>Spring MVC Application from Archetype</h1>
            <hr/>
            <div class="navbar">
                <ul class="nav nav-tabs">
                <li role="presentation">
                    <a href="${pageContext.request.contextPath}/index.jsp">
                        Home
                    </a>
                </li>
                <li role="presentation" class="active">
                    <a href="${pageContext.request.contextPath}/hello/sayhi">
                        Hello Controller
                    </a>
                </li>
                </ul>
            </div>
            <h2>Controller</h2>
            <!-- #2 Access the variable called 'message' and display it  -->
            <h3>${message}</h3>
        </div>
        <!-- Placed at the end of the document so the pages load faster -->
        <script src="${pageContext.request.contextPath}/js/jquery-1.11.0.min.js"></script>
        <script src="${pageContext.request.contextPath}/js/bootstrap.min.js"></script>
    </body>
</html>
```

spring-persistence.xml

This file holds the configuration information for the database and related components. Although it is empty right now, it will contain all the MySQL and DAO configuration information as the application is developed. All XML namespace entries required for AOP, transactions, MVC, and DI are included.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
              xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/mvc
  http://www.springframework.org/schema/mvc/spring-mvc.xsd
  http://www.springframework.org/schema/tx
  http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context.xsd
  http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-
3.2.xsd">

    <!-- Bean definitions go here -->

</beans>
```

pom.xml

The Maven POM file for the project contains the project description, dependencies, and compiler settings needed for the project. Most of the dependencies are not needed by the initial “Hello, World” version of the application but will be needed as database, transaction, AOP, and web service support is added.

1. Maven project information for our web application
2. Properties indicating which version of JUnit and Spring to use
3. External library dependencies for this project; most of these are not needed for the initial version but will be used as the project is developed
4. This setting allows Maven to replace tokens in files with real values from profiles, if needed
5. Specifies the version of the Java compiler to use for this project

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <!-- #1 Maven project information -->
  <groupId>com.swcguild</groupId>
  <artifactId>ContactListMVCDemo</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>ContactListMVCDemo Spring MVC Webapp</name>
  <url>http://maven.apache.org</url>

        <!-- #2 Version information for Spring and JUnit -->
        <properties>
              <spring.version>3.2.4.RELEASE</spring.version>
              <junit.version>4.11</junit.version>
                  <netbeans.hint.deploy.server>Tomcat</netbeans.hint.deploy.server>
        </properties>
      <!-- #3 Libraries on which we depend -->
      <dependencies>
              <dependency>
                      <groupId>org.springframework</groupId>
                      <artifactId>spring-core</artifactId>
                      <version>${spring.version}</version>
              </dependency>
              <dependency>
                      <groupId>org.springframework</groupId>
                      <artifactId>spring-web</artifactId>
                      <version>${spring.version}</version>
              </dependency>
              <dependency>
                      <groupId>org.springframework</groupId>
                      <artifactId>spring-webmvc</artifactId>
                      <version>${spring.version}</version>
              </dependency>
              <dependency>
                      <groupId>org.springframework</groupId>
                      <artifactId>spring-jdbc</artifactId>
                      <version>${spring.version}</version>
              </dependency>
              <dependency>
                      <groupId>org.springframework</groupId>
                      <artifactId>spring-aop</artifactId>
                      <version>${spring.version}</version>
              </dependency>
              <dependency>
                      <groupId>org.springframework</groupId>
                      <artifactId>spring-test</artifactId>
                      <version>${spring.version}</version>
              </dependency>
              <dependency>
                      <groupId>aspectj</groupId>
                      <artifactId>aspectjtools</artifactId>
                      <version>1.5.4</version>
              </dependency>
              <dependency>
                      <groupId>commons-dbcp</groupId>
                      <artifactId>commons-dbcp</artifactId>
                      <version>1.4</version>
              </dependency>
```

```
        <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <version>5.1.5</version>
        </dependency>

        <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>javax.servlet-api</artifactId>
                <version>3.1.0</version>
        </dependency>
        <dependency>
                <groupId>javax.inject</groupId>
                <artifactId>javax.inject</artifactId>
                <version>1</version>
        </dependency>
        <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>jstl</artifactId>
                <version>1.2</version>
        </dependency>
        <dependency>
                <groupId>javax.validation</groupId>
                <artifactId>validation-api</artifactId>
                <version>1.1.0.Final</version>
        </dependency>

        <dependency>
                <groupId>org.codehaus.jackson</groupId>
                <artifactId>jackson-mapper-asl</artifactId>
                <version>1.9.13</version>
                </dependency>
        <dependency>
                <groupId>org.codehaus.jackson</groupId>
                <artifactId>jackson-core-asl</artifactId>
                <version>1.9.13</version>
        </dependency>
        <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>${junit.version}</version>
                <scope>test</scope>
        </dependency>
        <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-validator</artifactId>
                <version>4.3.1.Final</version>
        </dependency>
        <dependency>
                <groupId>javax.servlet.jsp</groupId>
                <artifactId>jsp-api</artifactId>
                <version>2.1</version>
                <scope>provided</scope>
        </dependency>

</dependencies>
```

```
        <build>
                <finalName>spring-mvc-webapp</finalName>
                <!-- #4 This allows the build process to replace tokens in files -->
                <!--    the resource directories real values                     -->
                <resources>
                        <resource>
                                <directory>src/main/resources</directory>
                                <filtering>true</filtering>
                        </resource>
                </resources>
               <!-- #5 Specifies Java compiler version -->
                <plugins>
                        <plugin>
                               <groupId>org.apache.maven.plugins</groupId>
                               <artifactId>maven-compiler-plugin</artifactId>
                               <version>2.3.1</version>
                               <configuration>
                                       <source>1.8</source>
                                       <target>1.8</target>
                               </configuration>
                       </plugin>
                </plugins>
          </build>
</project>
```

Wrap-up

That concludes this portion of the tutorial. In this step, we did the following:

1. Built and installed a custom Maven archetype for Spring MVC projects
2. Created a new Spring MVC project using the custom archetype
3. Toured all of the major components of our new project and learned how our project is configured

In the next step, we'll create the initial version of our controller and view components for the application.