Spring MVC Tutorial – Contact List Application

Step 02: Basic Layout and Navigation

SOFTWARE GUILD

The Learning House

427 S 4th Street #300

Louisville KY 40202

Step 02: Basic Layout and Navigation

Overview

In this step, we will set up the shell of our application by creating views and controllers for the main sections (Home, Search, and Stats) of our application. In this version, our views will contain no live data and the controllers will do nothing but return the correct view name, but we will have the basic plumbing of the application in place when we're done.

Controllers

The first components that we'll create for our application are the controllers. We will have one controller for each section of the application. The initial version of each of our controllers will be very simple — the only functionality each will have is to map a URL to the appropriate view component.

Home Controller

Create a new Java class called HomeController and put it in the com.swcguild.contactlistmvc.controller package. Modify the class so it looks like this:

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HomeController {

    @RequestMapping(value={"/", "/home"}, method=RequestMethod.GET)
    public String displayHomePage() {
        return "home";
    }
}
```

Note that both "/" and "/home" will map to the "home" view component, which will be home.jsp in our case (although we have not yet created it).

Search Controller

Create a new Java class called SearchController and put it in the com.swcguild.contactlistmvc.controller package. Modify the class so it looks like this:

```
package com.swcguild.contactlistmvc.controller;

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

@Controller
public class SearchController {

    @RequestMapping(value="/search", method=RequestMethod.GET)
    public String displaySearchPage() {
        return "search";
    }
}
```

Note that “/search” maps to the “search” view component, which will be search.jsp in our case (although we have not yet created it).

Stats Controller

Create a new Java class called StatsController and put it in the com.swcguild.contactlistmvc.controller package. Modify the class so it looks like this:

```
package com.swcguild.contactlistmvc.controller;

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

@Controller
public class StatsController {

    @RequestMapping(value="/stats", method=RequestMethod.GET)
    public String displayStatsPage() {
        return "stats";
    }
}
```

Note that “/stats” maps to the “stats” view component, which will be stats.jsp in our case (although we have not yet created it).

Hello Controller

Finally, we need to remove the Hello Controller from the application. This controller was great for the initial "Hello, World!" version of our application but now we no longer need it. Right-click on HelloController.java and select the **Delete** menu item.

View Components

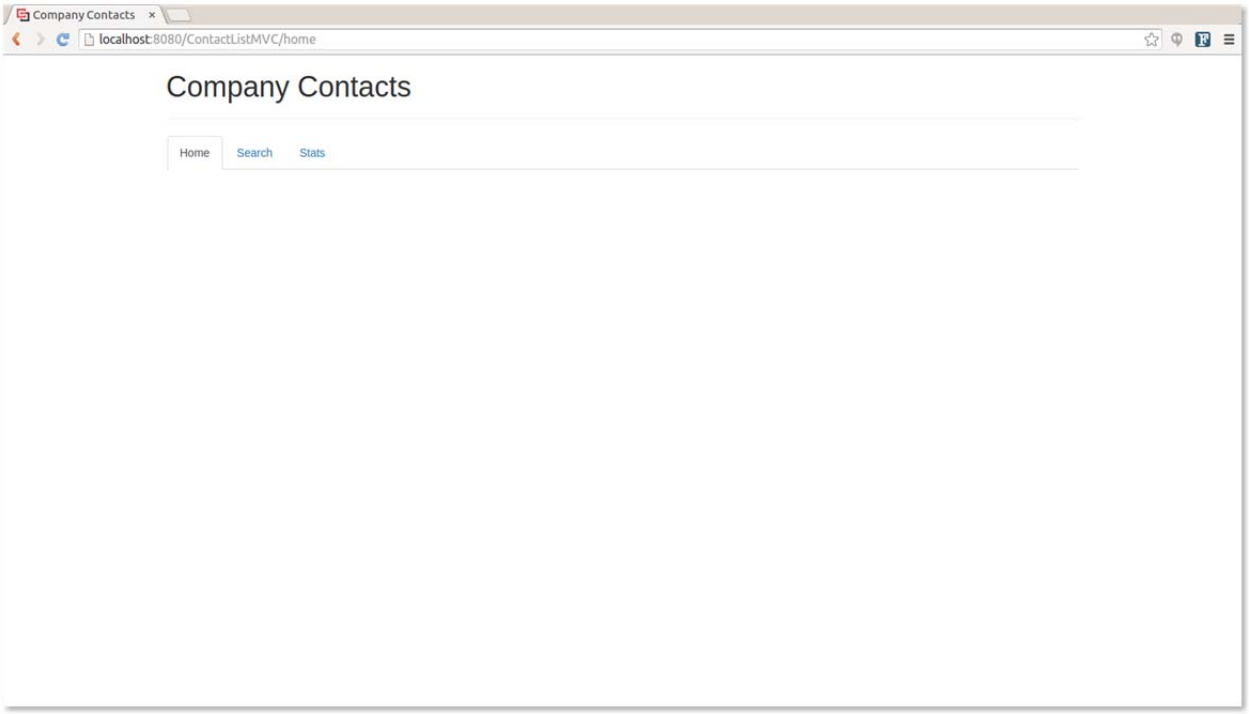
Now that we have created the controllers, we must create the matching views. In the previous step, our controller code specified the logical view names which map to the following JSPs based on our Spring MVC configuration:

1. home.jsp
2. search.jsp
3. stats.jsp

For now, we'll create static versions of these view components that only contain titles and the main navigation. Later in the tutorial, we'll modify these JSPs so they display live data.

Home Screen View

The initial version of our Home screen will look like this:



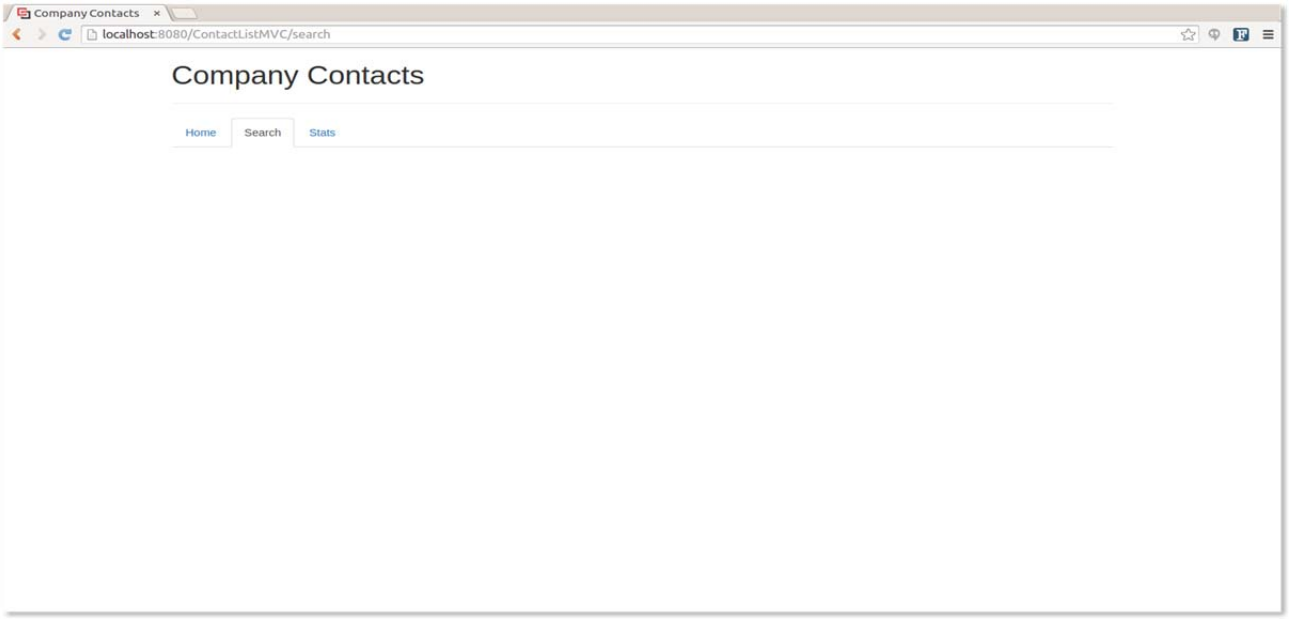
Create a new JSP file called **home.jsp** in the **jsp** folder. Modify the file so it looks like the following:

```
<%@ 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>Company Contacts</title>
        <!-- Bootstrap core CSS -->
        <link href="${pageContext.request.contextPath}/css/bootstrap.min.css"
              rel="stylesheet">

        <!-- SWC Icon -->
        <link rel="shortcut icon" href="${pageContext.request.contextPath}/img/icon.png">

    </head>
    <body>
        <div class="container">
            <h1>Company Contacts</h1>
            <hr/>
            <div class="navbar">
                <ul class="nav nav-tabs">
                    <li role="presentation" class="active">
                          <a href="${pageContext.request.contextPath}/home">Home</a>
                    </li>
                    <li role="presentation">
                         <a href="${pageContext.request.contextPath}/search">Search</a>
                    </li>
                    <li role="presentation">
                         <a href="${pageContext.request.contextPath}/stats">Stats</a>
                    </li
                 </ul>
            </div>
        </div>

        <!-- Placed at the end of the document so the pages load faster -->
        <script src="${pageContext.request.contextPath}/js/jquery-1.11.1.min.js"></script>
        <script src="${pageContext.request.contextPath}/js/bootstrap.min.js"></script>

    </body>
</html>
```

Note that our tabbed navigation consists of an unordered list of links with the “nav” and “nav-tabs” classes applied. Also note that we use the “active” class on the Home link to highlight it as the currently selected tab item (see screenshot above).

Search Screen View

The initial version of the Search screen will look like this:


Company Contacts
localhost:8080/ContactListMVC/search
Company Contacts
Home
Search
Stats

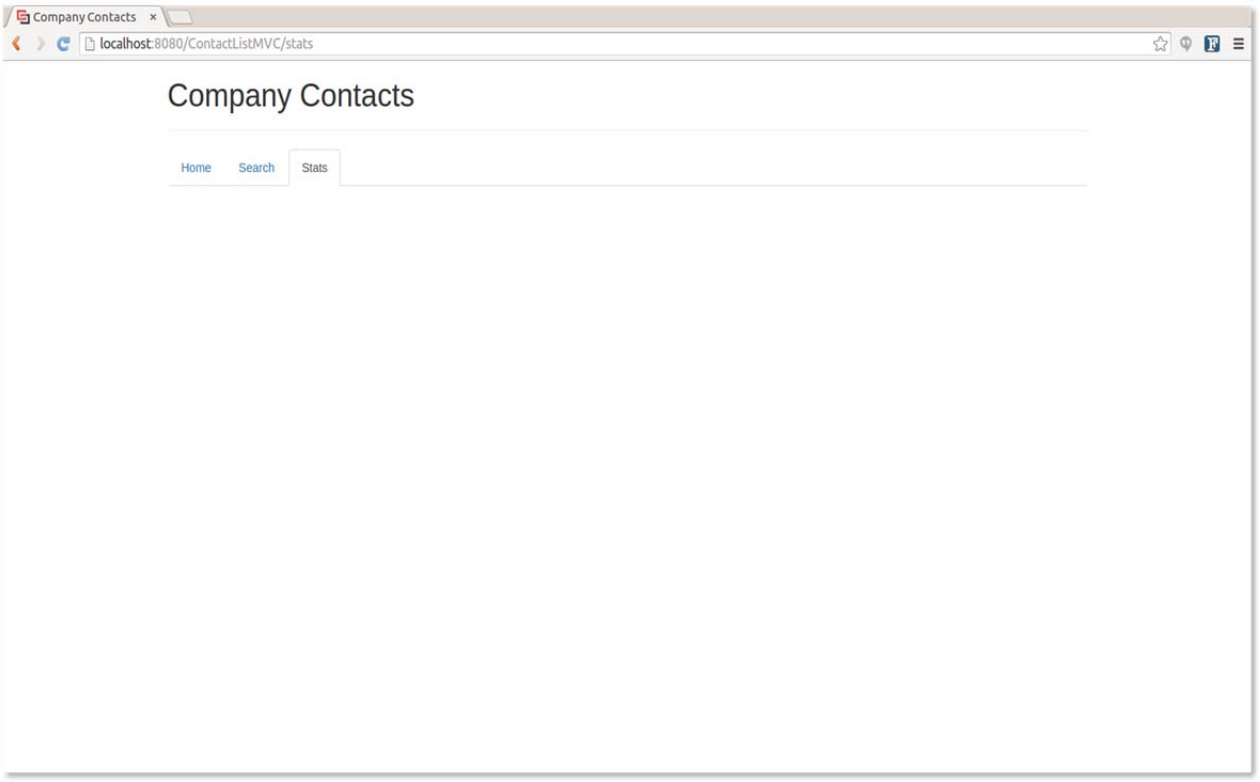
Create a new JSP file called **search.jsp** in the **jsp** folder. Modify it so it looks like the following:

```
<%@ 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>Company Contacts</title>
        <!-- Bootstrap core CSS -->
        <link href="${pageContext.request.contextPath}/css/bootstrap.min.css"
              rel="stylesheet">

        <!-- SWC Icon -->
        <link rel="shortcut icon" href="${pageContext.request.contextPath}/img/icon.png">

    </head>
    <body>
        <div class="container">
            <h1>Company Contacts</h1>
            <hr/>
            <div class="navbar">
                <ul class="nav nav-tabs">
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/home">Home</a></li>
                    <li role="presentation" class="active">
                        <a href="${pageContext.request.contextPath}/search">Search</a></li>
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/stats">Stats</a></li>
                </ul>
            </div>
        </div>

        <!-- Placed at the end of the document so the pages load faster -->
        <script src="${pageContext.request.contextPath}/js/jquery-1.11.1.min.js"></script>
        <script src="${pageContext.request.contextPath}/js/bootstrap.min.js"></script>

    </body>
</html>
```

Note that this file is identical to home.jsp except that the active class is applied to the Search link instead of the Home link.

Stats Screen View

The initial version of the Stats screen will look like this:



Create a new JSP file called **stats.jsp** in the **jsp** folder. Modify it so it looks like the following:

```
<%@ 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>Company Contacts</title>
        <!-- Bootstrap core CSS -->
        <link href="${pageContext.request.contextPath}/css/bootstrap.min.css" rel="stylesheet">

        <!-- SWC Icon -->
        <link rel="shortcut icon" href="${pageContext.request.contextPath}/img/icon.png">

    </head>
    <body>
        <div class="container">
            <h1>Company Contacts</h1>
            <hr/>
            <div class="navbar">
                <ul class="nav nav-tabs">
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/home">Home</a></li>
                    <li role="presentation">
                        <a href="${pageContext.request.contextPath}/search">Search</a></li>
                    <li role="presentation" class="active">
                        <a href="${pageContext.request.contextPath}/stats">Stats</a></li>
                </ul>
            </div>
        </div>

        <!-- Placed at the end of the document so the pages load faster -->
        <script src="${pageContext.request.contextPath}/js/jquery-1.11.1.min.js"></script>
        <script src="${pageContext.request.contextPath}/js/bootstrap.min.js"></script>

    </body>
</html>
```

Note that this file is identical to home.jsp except that the active class is applied to the Stats link instead of the Home link.

Index.jsp

Finally, we will remove **index.jsp** and **hello.jsp** from the application. Like the Hello Controller we removed in the previous section, these files are no longer needed.

Wrap-up

In this step of the tutorial, we did the following:

1. Created our controller components
2. Created our view components
3. Mapped URL requests to views
4. Removed unneeded components created by the Maven archetype.

The basic shape of our application is now in place. In the next several steps, we'll build out the client-side view functionality.