Spring Labs – Part 2

Lab 4 – Update PhoneBookApplication

Now that we have talked about using annotations for configuration, let us try some of them out.

1. Make a copy of the **PhoneBookApplication** and call it **PhoneBookApplication2**
2. Modify **PhoneServiceBean** to require the repository to be injected
3. Modify the project to use component scanning
4. Add a method that is annotated with **@PostConstruct** and have it log that the bean has been constructed. Note, you will have to explicitly import `javax.annotation` because Eclipse will not find it
5. Run the application to see that the logs are written out.

WebMVC PhoneBook

This part of the labs moves to using a server. We will be using Tomcat 8.5, which we will explain how to load in Lab 5.1 below.

With most of the exercises, we will be adding dependencies into our pom.xml file. The information needed is given in the exercise. After saving the file, make sure that you update the Maven build. This can be done by right clicking on the project > Maven > Update Project.

From time to time, it may be necessary to remove the application from the server. Usually it is because of a namespace conflict, but often enough it is because a change occurred in an XML file and is not being picked up when the application is re-run on the server. The easiest solution is to go to the servers view (found by going from the Window menu at the top > Show Views > Servers), click on the arrow next to your server instance so that you can see what is running on that server, then right click and select remove. It would also be useful at that point to go back to the server's right click menu and select Clean Tomcat Work Directory.

Setup Tomcat

For many of the following labs, we will be using web applications. Because of this, it is advisable that we load a web server. For this class, we are going to use Tomcat 8.5.

1. In the **Tools** directory, there is a **tomcat 8.5** directory. Inside of there is the .zip file for Tomcat. Unzip that .zip file somewhere that you have permissions to read and write files.
2. In Eclipse, we want to look at the Server view. It is down in the same area as your console. If you do not have it, then go **Window > Show View > Server**
    - Once you have the Server View, then click on the "**add new server**" link, or right click in the view and select **New > Server**
    - In the **Define a New Server** window, select **Apache > Tomcat 8.5** and click **Next**

- In the next window, click on the **Browse** button, and find the root to your Tomcat installation. Select **Finish**.

3. You should now have a Tomcat 8.5 server in your Server window.

Lab 5 – WebMVC PhoneBook

Now that we have a working **PhoneBook** application, let's turn it into a web application. To do this, we will first start with importing a Maven project called **RESTfulPhoneBook**. This is the same **PhoneBookApplication** that we have been working on, it just has been reconfigured for a web application.

1. Update the `pom.xml` file to add the following dependencies
   a. groupId: org.springframework
      artifactId: spring-webmvc
      version: 4.3.10.RELEASE
   b. groupId: org.springframework
      artifactId: spring-web
      version: 4.3.10.RELEASE
   c. groupId: javax.servlet
      artifactId: jstl
      version: 1.2
2. Define a new class called `MVCConfig` that contains the necessary MVC configuration information
3. Define a new class called `WebAppInitializer` and configure the `DispatcherServlet`
4. Modify `PhoneServiceBean`. This class is our `Controller`
   a. Each method should be given an appropriate `RequestMapping`
   b. Modify the `getAllContacts()` method so that it can be used to get a JSP page that would then read the model (the contacts themselves) **Note:** You may want to modify the method declaration for this method. If so, don't forget to modify the declaration in `PhoneService`. The JSP page that will allow you to show the all the contacts has already been developed for you. It is called **contactList.jsp** and can be found under **WEB-INF/views**.

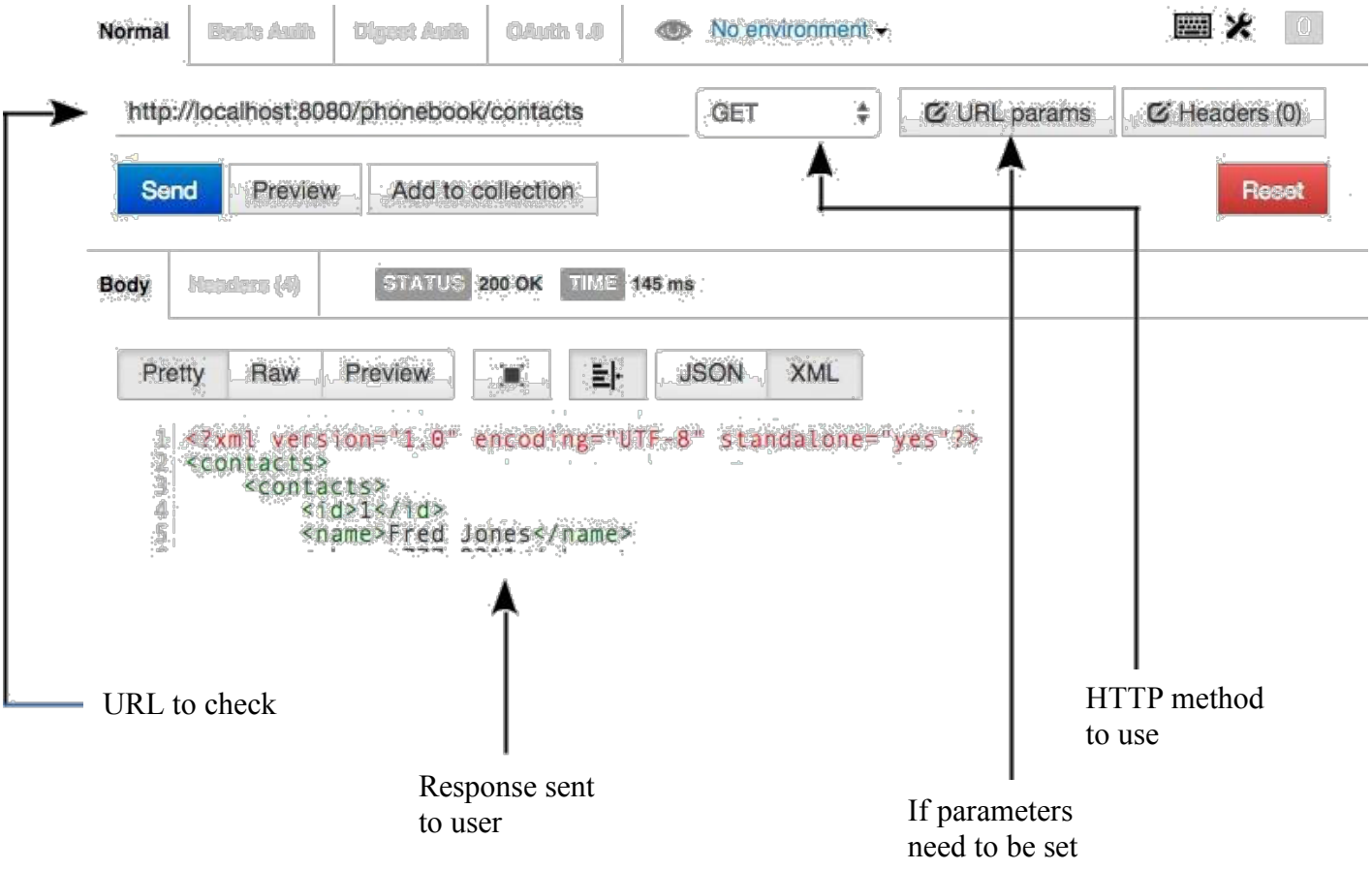
Lab 6 – RESTful PhoneBook

Now that we have a working **PhoneBook** application using MVC, let's turn it into an actual RESTful application. To do this, we will start with copying our **RESTfulPhoneBook** to make a new project called **RESTfulPhoneBook2**.

1. Update the `pom.xml` file to add the following dependency
   a. groupId: javax.xml.bind
      artifactId: jaxb-api
      version: 2.2.11
   b. groupId: com.fasterxml.jackson.core
      artifactId: jackson-databind
      version: 2.8.5

c. groupId: com.fasterxml.jackson.dataformat
   artifactId: jackson-dataformat-xml
   version: 2.8.5

2. Define a new class called `RestConfig` that contains the necessary configuration information
3. Look at your `WebAppInitializer`. Are any changes needed? If so, make them.
4. Modify `PhoneServiceBean`
   a. This class is our `RestController`
   b. Change the `addContact` to take an instance of a contact instead of individual attributes
   c. Each method should be given an appropriate `RequestMapping` and HTTP method
   d. Each method should be given an appropriate `ResponseStatus`
5. Modify the `Contact` class as necessary
6. Deploy the application to your server.
7. At this point, there are two possible ways to test it. One is to see what happens in the browser window. For that, though, only GET and POST methods would work. The second way is to use an application designed for testing RESTful web services
   a. For this class, I am using an application called Postman. If you already have something to test RESTful applications, you may use it. If not, then in the Tools directory is a directory called postman. From there, pick the version that you want (Mac or Windows) and install it. It will look something like what you see below.



If the HTTP method is POST, PATCH, PUT, or DELETE, then a section for adding form-data (ie: the body of the HTTP message) will appear immediately below the URL line. Any data that should be sent this way is entered in as Key/Value pairs. There is a separate entry for each. You will likely want to send the information as x-www-form-urlencoded.