Lab 24: Spring MVC RESTful Service

In this lab, you will create a simple RESTful service using Spring MVC.

Objectives:

- To demonstrate RESTfulness with Spring MVC.

Part 1: Creating the Service

Steps:

_1. Create a new Dynamic Web project named **Lab24Web** associated with *Lab24EAR*. Be sure to generate a Web deployment descriptor.

_2. As you did in earlier labs, copy from the *Lab14Part2Web* project:

- The *com.oaktreeair.ffprogram* package.
- The *spring.xml* and *springmvc-servlet.xml* files.

_3. Continue setting up the new project:

a. Import the Spring JARs into the WEB-INF/lib folder.

b. Configure web.xml to have the ContextLoaderListeners and configure the Spring MVC servlet, but this time, use a *url-pattern* of /.

c. Add the following to springmvc-servlet.xml:

```
<mvc:annotation-driven />

<!--  enables serving static resources like
  HTML files from Web app root directory -->
<mvc:default-servlet-handler />
```

d. Delete all of the *bean* definitions from springmvc-servlet.xml.

_4. Next, let's write a Web application context listener that will create an in-memory "database" of Segment objects and store them at application scope:

a. Right-click on the project and create a new class in the *com.oaktreeair.listeners* package named **MyContextListener** that implements *ServletContextListener* from the *javax.servlet* package.

b. Annotate the class with **@WebListener** - this registers the listener.

c. In the *contextInitialized* method, write the following to create a Segment list and store it at application scope:

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
List<Segment> segments = new ArrayList<Segment>();

Segment s = new Segment();
s.setFlightNumber(5079);
s.setMiles(345);
s.setOriginatingCity("CID");
s.setSegmentDate(new Date());
s.setSegmentNumber((long)12);
segments.add(s);

s = new Segment();
s.setFlightNumber(2486);
s.setMiles(2165);
s.setOriginatingCity("SEA");
s.setSegmentDate(new Date());
s.setSegmentNumber((long)13);
segments.add(s);

s = new Segment();
s.setFlightNumber(1431);
s.setMiles(1431);
s.setOriginatingCity("ORD");
s.setSegmentDate(new Date());
s.setSegmentNumber((long)14);
segments.add(s);

sce.getServletContext().setAttribute("segments", segments);
```

_5. Annotate the Segment class with **@XmlRootElement**. This will enable Spring to transform Segment objects to/from XML.

_6. Right-click on the project and create a new class in the *com.oaktreeair.controller* package named **SegmentController**.

Complete the new class:

a. Annotate the class so that it's a Spring MVC controller with a request-mapping URL of **rest**.

b. Define a field and inject a reference to the servlet context where your in-memory "database" of Segments resides:

```
@Autowired
private ServletContext servletCtx;
```

c. Create a private, static inner class named **ResourceNotFoundException** that extends RuntimeException. Annotate the exception class with:

```
@ResponseStatus( value = HttpStatus.NOT_FOUND )
```


Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

d. Write a service method that responds to a URL ending in **/segment/{segmentNumber}** for an HTTP GET request. The method should accept the segment number as a parameter and look up the Segment in the in-memory list. You can retrieve a reference to the segment list like:

```
List<Segment> segments =
   (List<Segment>)servletCtx.getAttribute("segments");
```

If there's no Segment with the specified segment number, throw the ResourceNotFoundException.

_7. In the Servers tab, right-click on the server and add the EAR project to the server. Wait for the server to finish publishing.

_8. To test, open a Web browser and enter the following URL:

```
http://localhost:9081/Lab24Web/rest/segment/12
```

You should see an XML document that has information about Segment #12.

Try again with a URL for Segment #59 - you should get a Not Found response.

If you have cURL installed (you can download it from http://curl.haxx.se/download.html), try using it to set the Accept header to **application/json** and retrieve the JSON representation of Segment #12.

Part 2: Implementing CRUD and Writing a jQuery Ajax Client

In this part, you will finish the service so it provides the following:

- Support for retrieving all segments.
- Support for HTTP POST to create a new segment.
- Support for HTTP PUT to update an existing segment.
- Support for HTTP DELETE to remove a segment

You will then write Web pages that use jQuery Ajax to exercise the service.

Steps:

_1. Update the service by writing an additional four methods that GET all segments, CREATE a new segment, UPDATE an existing segment and DELETE a segment.

Test with cURL and/or a browser.

_2. Write five Web pages, each of which exercises one of the service methods using the jQuery Ajax support. For the methods that accept paramter(s), we recommend that you write HTML forms so the user can enter the parameter data.

Note: You can use the jQuery *ajax* method to issue all of the requests. Here's a simple example:

Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
$.ajax({
    url: "rest/segment",
    type: 'GET',
    dataType: 'json',
    contentType: 'application/json',
    success: function(ret)
    {
        console.log("Data: " + ret);
    },
    error: function(error, textStatus, errorThrown)
    {
        console.log("Error: " + error);
        console.log("Text status: " + textStatus);
        console.log("Error thrown: " + errorThrown);
    }
});
```


Copyright © Descriptor Systems 2014. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.