Lab 16: RESTful CRUD Service

In this lab, you will create a RESTful that supports the CRUD (Create, Retrieve, Update and Delete) operations using JAX-RS.

Objectives:

- To write a RESTful CRUD application.

Part 1: Creating the Service

Steps:

_1. Create a new Dynamic Web project named **lab16Web** associated with *lab16EAR*. On the last page of the wizard, be sure to put a checkmark in the box to create the Web deployment descriptor.

_2. First, create a Customer class that your service will retrieve and store:

   a. In the **com.goliath** package, create a class named **Customer**.

   b. Decorate the class definition with the **XmlRootElement** annotation and import javax.xml.bind.annotation.XmlRootElement.

   c. Add the following fields:

```
private int id;
private String name;
private String address;
private String city;
private String state;
private String zip;
```

   d. Generate get/set methods for the fields.

   e. Use *Source - Generate hashCode() and equals()* to generate those methods. Deselect all of the fields in the wizard except for the *id* field.

   f. Use *Source - Generate toString()* to generate that method that displays all of the fields.

_3. Next, let's write a servlet context listener that will create an in-memory "database" of Customer objects and store them at application scope:

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

   b. In the *contextInitialized* method, write the following to create a Customer 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<Customer> customers = new ArrayList<Customer>();

Customer c = new Customer();
c.setId(12);
c.setAddress("123 Elm");
c.setCity("Madison");
c.setName("Bill Smith");
c.setState("WI");
c.setZip("53220");
customers.add(c);

c = new Customer();
c.setId(44);
c.setAddress("456 Oak");
c.setCity("Marion");
c.setName("Sue Jones");
c.setState("IA");
c.setZip("52302");
customers.add(c);

arg0.getServletContext().setAttribute("customers", customers);
```

c. Open the web.xml deployment descriptor into the editor and press the *Source* tab at the bottom. Add the following immediately above the close tag for **/web-app**:

```
<listener>
  <listener-class>
    com.goliath.listeners.MyContextListener
  </listener-class>
</listener>
```

Part 2: Implementing the R in CRUD

In this part, you will implement retrieving Customer as XML, given a customer ID.

Steps:

_1. First, create the Java class that will act as the service's endpoint:

a. In the **com.goliath.service** package, create a class named **CustomerService**.

b. Annotate the class to assign a URI to the service:

```
@Path("/customer/{id}")
```


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

Note that we use the "{id}" syntax to specify a placeholder in the URI for the customer's ID. In other words, URIs from the client will look like:

```
http://localhost:8080/lab20Web/rest/customer/xxxxx
```

Where 'xxxxx' is the customer ID.

Import the *javax.ws.rs.Path* type.

c. Define a field and annotate it so your service can access the ServletContext object that represents application scope:

```
@Context
ServletContext servletContext;
```

Import the *javax.servlet.ServletContext* and *javax.ws.rs.core.Context* types.

d. Start writing a method that will provide the "R" in CRUD, i.e. it will retrieve a customer in XML form, given a customer ID:

```
@GET
@Produces("text/xml")
public Response find(@PathParam("id") int id)
{
    Response ret = null;

    return ret;
}
```

Import *javax.ws.rs.core.Response* and *javax.ws.rs.Produces*.

Note how we annoate this method: @GET and @Produces configure the HTTP verb and content type and @PathParam maps the "{id}" URI template to the parameter on the method.

e. In the method, after the *String ret = null;* line but before the *return*, retrieve a reference to the customer list that your listener stored at application scope:

```
List<Customer> customers = (List<Customer>) servletContext
        .getAttribute("customers");
```

f. Then iterate the list, looking for a customer with the input ID:

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

```
Customer theCust = null;
for(Customer c : customers)
{
    if (c.getId() == id)
    {
        theCust = c;
        break;
    }
}
```

g. Next, write an *if-else* statement that checks if a there was a customer with the specified ID:

```
if (theCust != null)
{

}
else
{

}
```

h. If theCust isn't null, use JAXB automatic object mapping to create an XML response from the customer object:

```
ret = Response.ok(theCust).build();
```

i. In the *else* block (for theCust == null), send back HTTP status code "404 Not Found":

```
ret = Response.status(
        Response.Status.NOT_FOUND).build();
```

_2. Next, in the *com.goliath.service* package, create a class named **CustomerApplication** that extends *Application* from the *javax.ws.rs.core* package.

Complete the class:

a. Annotate the class to provide a prefix for the RESTful URL:

```
@ApplicationPath("/rest/")
```

b. Use the *Source - Override/Implement Methods* menu to override the *getClasses* method.

You can delete the generated call to the superclass.

c. Add code to the method so that it creates a Set containing all of your RESTful sources (only one in this case):

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

```
        Set<Class<?>> s = new HashSet<Class<?>>();
        s.add(CustomerService.class);
        return s;
```

_3. Deploy the application's EAR by adding it to the JBoss server.

_4. Test the RESTful service by opening a Web browser and entering this URL:

```
http://localhost:8080/lab16Web/rest/customer/44
```

You should see a simple Web page showing the XML for customer #44.

_5. Modify the URL so that the customer ID is 566 - you should get a 404 Not Found.

_6. Using the sample in the course notes as a guide, implement as many of the other CRUD methods as you have time for. To test PUT and DELETE, you will need to write a standalone client program, since browsers will not generate those HTTP methods.

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

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