5. Assignment 2 - XML, REST JAX-RS

- **Customer object with attributes firstname, lastname, email**
- **XML Data sample**

```
<customer>

    <firstname>Bill</firstname>

     <lastname>Burke</lastname>

   <email>bill.burke@gmail.com</email>

</customer>
```

- **Create com.rest.domain.Customer in eclipse project**

```
package com.rest.domain;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlElement;

@XmlRootElement(name="customer")
public class Customer   {
// Maps a object property to a XML element derived from property name.

@XmlElement public int id;
@XmlElement public String firstname;
@XmlElement public String lastname;
@XmlElement public String email;

}
```

- **Create Resourse com.rest.resource.CustomerResource**

```
package com.rest.resource;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.atomic.AtomicInteger;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.WebApplicationException;
import javax.ws.rs.core.Response;



@Path("customers")
public class CustomerResource {
// ConcurrentHashMap - A hash table supporting full concurrency of retrievals and adjustable expected concurrency for
```

```
updates.
static private Map<Integer, Customer> customerDB = new ConcurrentHashMap<Integer, Customer>();
// An AtomicInteger is used in applications such as atomically incremented counters
private AtomicInteger idCounter = new AtomicInteger();


@POST
@Consumes("application/xml")
public Customer createCustomer(Customer customer) {
  customer.id = idCounter.incrementAndGet();
  customerDB.put(customer.id, customer);
  return customer;
}

@GET
@Path("{id}")
@Produces("application/xml")
public Customer getCustomer(@PathParam("id") int id) {

 Customer customer = customerDB.get(id);

 return customer;

}


@PUT
@Path("{id}")
@Consumes("application/xml")
public void updateCustomer(@PathParam("id") int id, Customer update)  {

   Customer current = customerDB.get(id);
   current.firstname = update.firstname;
   current.lastname = update.lastname;
   current.email = update.email;
   customerDB.put(current.id, current);
}

@DELETE
@Path("{id}")
public void deleteCustomer(@PathParam("id") int id) {
   Customer current = customerDB.remove(id);
   if (current == null) throw new WebApplicationException(Response.Status.NOT_FOUND);

  }
}
```

- **Show CRUD using CURL with XML data**

```
curl -H "Content-Type: application/xml" -X POST -d
 "<customer><firstname>Bill</firstname><lastname>Burke</lastname><email>bill.burke@gmail.com</email></customer>" http://localhost:8080/lab2/rest/customers

curl -H "Content-Type: application/xml" -X GET http://localhost:8080/lab2/rest/customers/1

curl -H "Content-Type: application/xml" -X PUT -d
"<customer><firstname>Ed</firstname><lastname>Burke</lastname><email>ed.burke@gmail.com</email></customer>" http://localhost:8080/lab2/rest/customers/1

curl -H "Content-Type: application/xml" -X DELETE  http://localhost:8080/lab2/rest/customers/1
```

Assignment 2 - Optional

- **Prereqs**
  - Lab1 is setup correctly
  - CURL is installed
- **Create domain object - Podcast**
  - A podcast will have the following attributes or properties :
    - `feed` – url feed of the podcast :String

        - `title` – title of the podcast : String
        - `description` – a short description of the podcast : String
    - **Create XML Data Sample**
    - **Create CURL commands for POST, GET, PUT and DELETE with XML data**
    - **Create PodcastResource for POST, GET, PUT and DELETE**
    - **Run it in Eclipse using JETTY or Tomcat**
    - **Execute CURL commands and verify results in POST, GET, PUT, GET, DELETE and GET order.**
