LS/EECS Building Ecommerce Applications

Lab 7: SIS-v3

Objectives

-understand model view control programming model
-learn how to program JDBC
-understand interoperability, XML, JSON
-learn how to program REST web services

Before You Start

- Go over slides and APIs introduced in the Lecture
- Export your SIS-v2 project (make sure you include the source files in the export) to an archive file SIS-v2.zip on the desktop.
- Import the war file just created to a new Maven project, named *SIS-v3*.
- Read and understand the lecture on REST services
- Import, run and understand the samples that come with REST lecture
- Add REST dependencies in the pom.xml, as in the sample code that comes with the lectures (check lecture presentations)
- Add the front controller to the web.xml file

Requirement. Enable your application to be accessed with REST web services APIs

Use Case: Third parties from University would like to use your SIS application through REST APIs. You have to add to your application a CRUD interface that allow the third partied the query your application. Specifically, consider Enrollment and Students as web resources. You have to have the following CRUD REST interfaces:

-CREATE: insert a new student row in the student table by providing all values of the row
-READ: read a set of rows from student table whose last name contain a given string
-DELETE: delete the row with a specified SID from student table

How to test it: You can use "curl" to invoke the APIs. Here are 3 curl invocations and the return strings:

CREATE: creates a new student. -X POST invokes a POST method of the HTTP. The method returns the number of inserted rows. Note that url is between `` characters.

```
~ curl –X POST 'http://localhost:8080/SIS-
v3/rest/student/create?sid=003&givenName=John&surName=Wayne&creditTake
n=3&creditGraduate=1'

~ insertedRows:1
```

READ: reads the students whose last name contains a string. Use GET method and return the student record(s) as an XML file

```
~ curl –X GET ‘http://localhost:8080/SIS-
v3/rest/student/read?studentName=Way’

~ <sisReport namePrefix="Way" creditTaken="0">
    <studentList>
        <sid>003</sid>
        <name>John, Wayne</name>
        <credit_taken>3</credit_taken>
        <credit_graduate>1</credit_graduate>
        <credit_taking>0</credit_taking>
 </studentList>
```

DELETE: delete the row with a given student id. Is implemented as DELETE

```
~ curl –X DELETE ‘http://localhost:8080/SIS-
v3/rest/student/delete?sid=003’

~ deletedRows:1
```

How to do it:

1. The model, model.Sis.java.

It is time to make the model a singleton. A singleton is a pattern that makes sure that the class has one instance only. The reason you want that is because you want one model in your application (as defined in MVC pattern). This is how you do it:

```
public class SisModel {
     //define the static and private field, give it a name
     private static SisModel instance;
```

```
     //these are the pointers to the DAO objects…DAO objects
will support all operations on database
     private StudentDAO studentData;
     private EnrollmentDAO enrollmentData;


//getInstance will return that ONE instance of the pattern
//with the the DAO objects initialized..
public static SisModel getInstance()throws
ClassNotFoundException{
          if (instance==null) {
               instance =new SisModel();
               instance.studentData= new StudentDAO();
               instance.enrollmentData = new EnrollmentDAO();

          }
          return instance;
}
//note that the constructor is private, cannot be called from
other classes
private SisModel() {
}
```

…add other methods here..

1. The Controller

Modify the init method of `your existing controller(servlet)` so it gets the model through SisModel.getInstance(), something like this:

```
          super.init();
          try {
               SisModel sis = SisModel.getInstance();
               this.getServletContext().setAttribute("SIS", sis);
          } catch (ClassNotFoundException e) {
               throw new ServletException("Class Not Found" + e);
          }
```

Test your application so it has the functionality of Lab 5 and 6.

2. The rest package

Add a new package, rest, and in that package add a Java class, Student, that represents the web resource, student.

Add the following methods to the class:

```
@Path("student")  //this is the path of the resource

public class Student {

@GET
@Path("/read/")
@Produces("text/plain")
public String getStudent(@QueryParam("studentName") String name)
throws Exception {

//add the body of the method here… you can try in simple steps,
//for example, just return the query parameter so you can see
//it works..
//you should call a method from the model,
//SisModel.getInstance().getAsXML(name) for example or you might
//already have a method from previous labs

}

//do not copy and paste, type it so you will better remember the
//patterns

@POST
@Path("/create/")
@Consumes("text/plain")
@Produces("text/plain")
public String createStudent(@QueryParam("sid")String sid,
@QueryParam("givenName")String givenname,
@QueryParam("surName")String surname,
@QueryParam("creditTaken")String credittaken,
@QueryParam("creditGraduate")String creditgraduate )
{
//add the body of the method here… you can try in simple steps,
//for example, just return the query parameters so you can see
//it works..
//later, you should call a method from the model,
//SIS.getInstance().create(….)

}

@DELETE
```

```
@Path("/delete/")
@Consumes("text/plain")
@Produces("text/plain")
public String delete(@QueryParam("sid")String sid) {
//add the body of the method here..
//you should call a method from the model,
//SIS.getInstance().delete(….)

}
```

3. The Model: SIS.java

```
Add methods that implement the create, and delete methods above.
  -sanitize the query string
  -convert String to int types, where needed
  -validate the inputs are correct
  -call DAO methods( see below)
```

4. DAO: StudentDAO

Add methods to implement, select, insert and delete. Use PreparedStatements.

Here is how insert can look like:

```
public int insert(String sid, String givenname, String surname,
int credittake, int creditgraduate)throws SQLException,
NamingException {

     //note that the query parameters are set as ?

     String preparedStatement ="insert into students
     values(?,?,?,?,?)";

     Connection con = this.ds.getConnection();
     //PreparedStatements prevent SQL injection
     PreparedStatement stmt =
     con.prepareStatement(preparedStatement);
     //here we set individual parameters through method calls
     //first parameter is the place holder position in the ?
     //pattern above
        stmt.setString(1, sid);
        stmt.setString(2, givenname);
        stmt.setString(3, surname);
        stmt.setInt(4, credittake);
        stmt.setInt(5, creditgraduate);
```

```
        return stmt.executeUpdate();
    }
```

And here is delete..

```
     public int delete(String sid)throws SQLException,
NamingException{

     String preparedStatement ="delete from students where
sid=?";
        Connection con = this.ds.getConnection();
        PreparedStatement stmt =
con.prepareStatement(preparedStatement);
        stmt.setString(1, sid);
        return stmt.executeUpdate();
    }
```

5. Questions

a. How would you create a student by providing an XML file as input? (XML as defined in the XML schema, Lab 7)

b. How would you create a student by providing a JSON string as input (same inputs as in XML, but using JSON format)

c. How would you update a student row with new credit_to_graduate data

d. Can you think of similar operations for Enrolment resource?