LS/EECS Building Ecommerce Applications

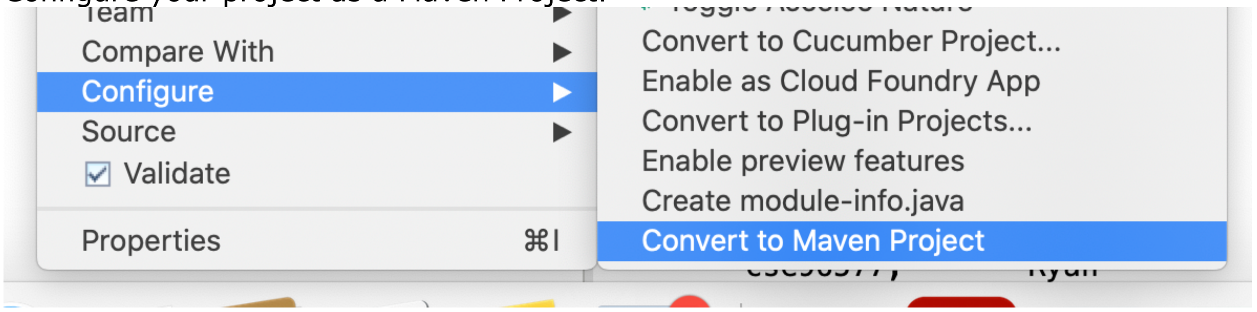
Lab 6: SIS-v2

Objectives

-understand model view control programming model
-understand JDBC programming model
-understand interoperability, XML, JSON

Before You Start

- Run and understand all samples that come with the Lecture
- Go over slides and APIs introduced in the Lecture
- Export your SIS-v1 project (make sure you include the source files in the export) to a war file SIS-v1.war on the desktop.
- Import the war file just created to a new project named *SIS-v2*.
- Configure your project as a Maven Project.



-
- Add Json and jaxb dependency in the maven build file, pom.xml.

```
   <dependencies>
  <dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.1</version>
</dependency>
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
    <version>2.3.1</version>
    <scope>runtime</scope>
</dependency>
<dependency>
   <groupId>org.glassfish</groupId>
   <artifactId>javax.json</artifactId>
   <version>1.0.4</version>
</dependency>
</dependencies>
```

Requirement. Enable your application to generate and XML, JSON reports for third parties

Use Case: Third parties would like to use the reports from your application. They would like Report*XML and ReportJSON* buttons that enable them to: export the generated report to an xml or JSON document. The xml document has to follow a specified xsd (XML Schema Definition). As a design constraint, you will use JAXB, Java API for XML Binding and json-api.

How to do it:

1. The Form.

Add the *ReportXML* and ReportJSON to your HTML and enable the functionality

The source file of the UI, note how the buttons are linked to the form through the form="form1" Also, note that ReportXML and ReportJSON do not call Ajax, they simply submit the form.

```
<fieldset>
    <legend>SIS-v2</legend>
    <form action="Sis" method="POST" class="osapForm" id="form1">
        <label for="namePrefix"> Name Prefix: </label> <input type="text"
            id="namePrefix" name="namePrefix" value="" /> <label
            for="credit_taken"> Minimum Credit Taken: </label> <input
            type="text" id="credit_taken" name="credit_taken" value="" />
    </form>
    <button type="submit" value="Ajax" id="html" name="report"
        onclick="reportAjax('/SIS-v2/Sis?report=Ajax')">ReportAjax</button>
    <button type="submit" value="XML" form="form1" id="XML"
        name="report">ReportXML</button>
    <button type="submit" value="JSON" form="form1" id="JSON"
        name="report">ReportJSON</button>

</fieldset>
<p id="result"></p>
```

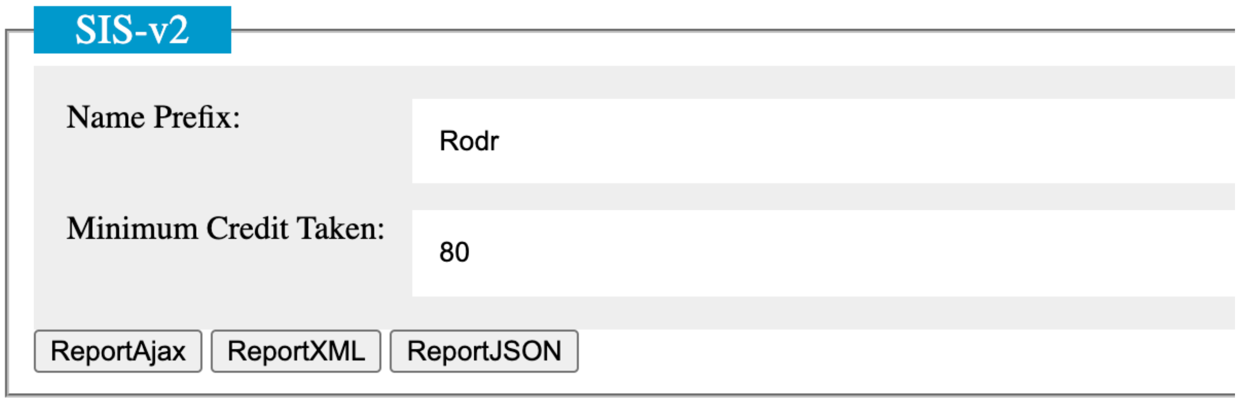
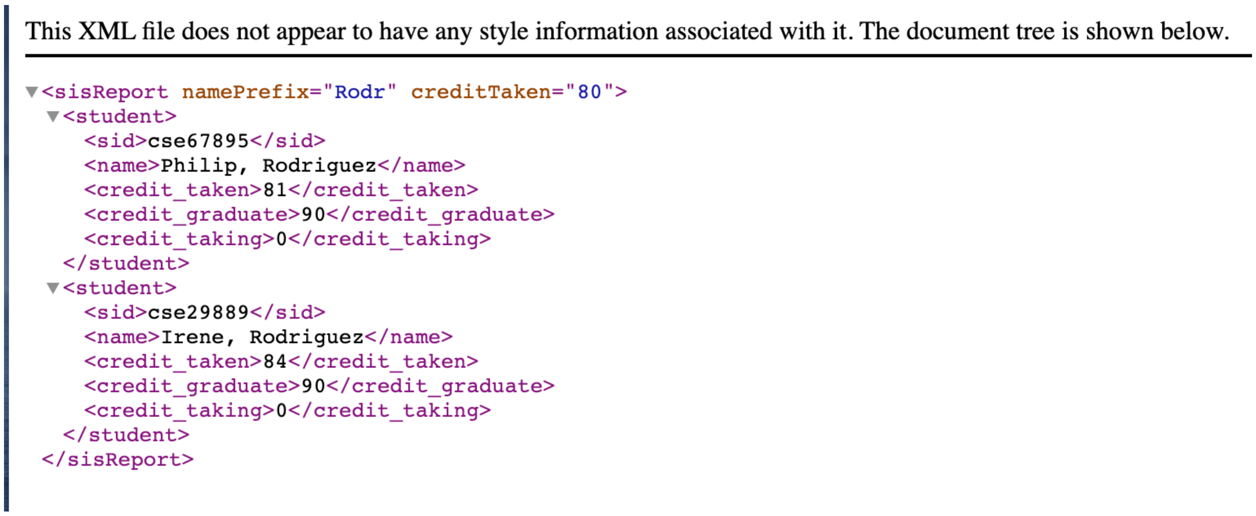


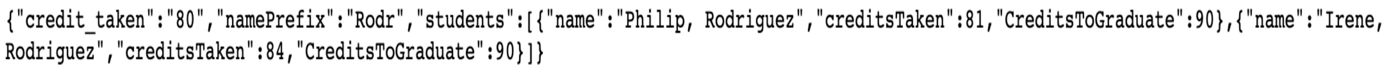
Fig 1. The UI. Note that the buttons are not included in the form frame



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<sisReport namePrefix="Rodr" creditTaken="80">
  ▼<student>
     <sid>cse67895</sid>
     <name>Philip, Rodriguez</name>
     <credit_taken>81</credit_taken>
     <credit_graduate>90</credit_graduate>
     <credit_taking>0</credit_taking>
   </student>
  ▼<student>
     <sid>cse29889</sid>
     <name>Irene, Rodriguez</name>
     <credit_taken>84</credit_taken>
     <credit_graduate>90</credit_graduate>
     <credit_taking>0</credit_taking>
   </student>
 </sisReport>
```

Fig. 2. The result of ReportXML



```
{"credit_taken":"80","namePrefix":"Rodr","students":[{"name":"Philip, Rodriguez","creditsTaken":81,"CreditsToGraduate":90},{"name":"Irene,
Rodriguez","creditsTaken":84,"CreditsToGraduate":90}]}
```

Fig 3. The JSON result

1. The Controller

Modify `Sis.java` so it detects when the *ReportXML* button is clicked and return in a *new page the xml file*. If clicked, the servlet should extract the form parameters and retrieve the model instance (as in the *ReportAjax* button case) but it should now invoke the model's `export` method:

```
public String export(String namePrefix, String minGPA) throws Exception
```

The parameters are the same as the model's `retrieve` method. This method returns a String that contains the XML elements as described below.

2. The Model: ListWrapper

Create a bean, ListWrapper that encapsulates the report, see below. In Java, you can map classes and attributes to XML elements and attributes using annotations. Important annotations: @XmlRootElement, @XmlElement, @XmlAttribute

```
@XmlRootElement(name="sisReport")
public class ListWrapper
{
	@XmlAttribute
	private String namePrefix;
	@XmlAttribute(name="creditTaken")
	private int credit_taken;
	@XmlElement(name="studentList")
	private List<StudentBean> list;
```

Add a default constructor and one that initializes the attributes. You can use Eclipse "source-Generate Constructors" for that..

3. The Model: SIS

Implement the `export()` method. It should start by invoking creating a `ListWrapper` object passing the needed parameters. Let the object name be `lw`. The method continues like this

```
  JAXBContext jc = JAXBContext.newInstance(lw.getClass());
  Marshaller marshaller = jc.createMarshaller();
  marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
  marshaller.setProperty(Marshaller.JAXB_FRAGMENT, Boolean.TRUE);


  StringWriter sw = new StringWriter();


  sw.write("\n");
  marshaller.marshal(lw, new StreamResult(sw));

  System.out.println(sw.toString()); // for debugging

  //return XML
  return sw.toString();
```

4. The Beans: StudentBean

You *could* add annotation to the bean: `@XmlElement` before each of the four attributes; or do it just before the class using `@XmlType(propOrder={"sid"…}.` This is not needed unless the requirement called for different names for the generated XML tags (i.e. different from the attribute names).

5. JSON report

The same behavior as "ReportXML" but this time it returns JSON in a new page

Appendix

Json Archive

https://search.maven.org/artifact/org.glassfish/json/2.0.0/pom

jaxb Archive

https://search.maven.org/search?q=g:org.glassfish.jaxb