

Mark Bellingham

14032098

Enterprise Programming 6G6Z1003

Assignment Web Services 1CWK50

Access the data using simple http web service calls	2
Options to return the data in text, JSON or XML	2
Google App Engine to implement the application on a remote cloud based server	4
A WSDL description of the interface to the web service	7
Access to the data using REST type interaction.....	9
An Ajax based web front end to retrieve the data and display in a suitable format using library based routines for an enhanced user interface	12
Code Design using MVC, DAO and modular routines.	16

Access the data using simple http web service calls

Options to return the data in text, JSON or XML

The web service has options to output the data in three formats - JSON, XML or plain text. Below is a section of the code in the servlet that processes these options. The program will operate by simply submitting a GET request so if the user desires, they need not use any user interface at all. If the user does not supply an 'action' parameter, they will be redirected to the interface page. If they do not supply a format parameter, the system chooses JSON by default. Once the action parameter is processed, the program uses a switch statement to process the format type. This is to avoid using a long if/else ladder and improve code readability.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // Get the parameters from the request and create the Strings
    String output = "";
    String action = "";
    String format = "";
    // If we somehow arrived here without specifying an action, return to the CRUD page
    if(request.getParameter("action") == null || request.getParameter("action") == "") {
        response.sendRedirect("coursesCRUD.jsp");
    } else {
        action = request.getParameter("action");
    }
    if(request.getParameter("format") == null || request.getParameter("format") == "") {
        format = "json";
    } else {
        format = request.getParameter("format");
    }
    String search = request.getParameter("searchText");
    String id = request.getParameter("id");
    String name = request.getParameter("name");
    String credits = request.getParameter("credits");
    String duration = request.getParameter("duration");
    String tutor = request.getParameter("tutor");

    System.out.println("search = " + search);

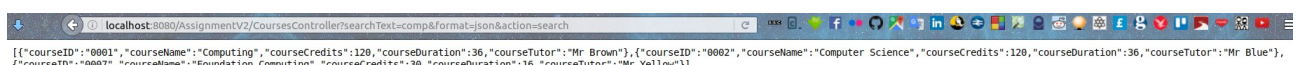
    // If the user performs a Search action, get a list of courses from the search query
    // and format the result according to the request.
    if("search".equals(action)) {
        courses = dao.searchCourse(search);

        switch(format) {
            case "json":
            {
                // Uses the GSON library to convert from java object to JSON notation
                output = new Gson().toJson(courses);
                System.out.println("json output: " + output);
                response.setContentType("application/json");
                response.setCharacterEncoding("UTF-8");
                response.getWriter().write(output);
                break;
            }
            case "xml":
            {
                // Uses the XStream library to convert from java object to XML notation
                XStream xstream = new XStream();
                xstream.alias("course", Course.class);
                xstream.addImplicitCollection(CoursesList.class, "list");

                response.setContentType("application/xhtml+xml");
                output = xstream.toXML(courses);
                System.out.println("xml output: " + output);
                response.getWriter().write(output);
                break;
            }
            case "text":
            {
                // Uses the XStream library to convert from java object to XML notation
                XStream xstream = new XStream();
                xstream.alias("course", Course.class);
                xstream.addImplicitCollection(CoursesList.class, "list");

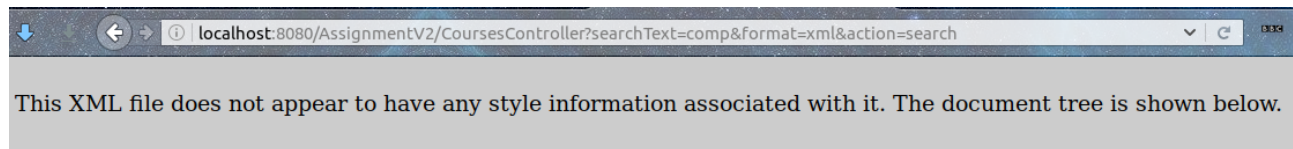
                response.setContentType("text/plain");
                output = xstream.toXML(courses);
                System.out.println("text output: " + output);
                response.getWriter().write(output);
                break;
            }
        }
    }
}
```

The next screenshot shows the output when using just the address bar with the parameters: action=search, format=JSON, searchText=comp.



```
[{"courseID": "0001", "courseName": "Computing", "courseCredits": 120, "courseDuration": 36, "courseTutor": "Mr Brown"}, {"courseID": "0002", "courseName": "Computer Science", "courseCredits": 120, "courseDuration": 36, "courseTutor": "Mr Blue"}, {"courseID": "0007", "courseName": "Foundation Computing", "courseCredits": 30, "courseDuration": 16, "courseTutor": "Mr Yellow"}]
```

This screenshot shows XML output

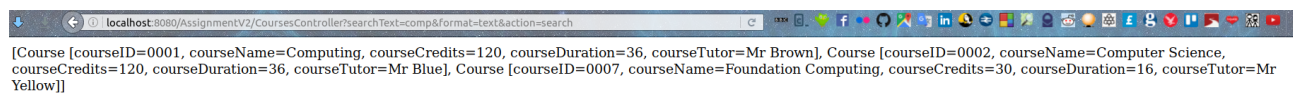


```

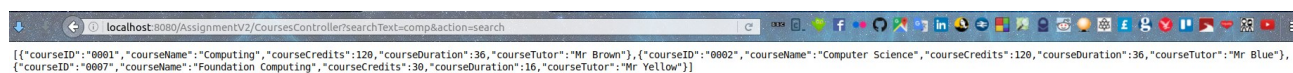
- <list>
  - <course>
    <courseID>0001</courseID>
    <courseName>Computing</courseName>
    <courseCredits>120</courseCredits>
    <courseDuration>36</courseDuration>
    <courseTutor>Mr Brown</courseTutor>
  </course>
  - <course>
    <courseID>0002</courseID>
    <courseName>Computer Science</courseName>
    <courseCredits>120</courseCredits>
    <courseDuration>36</courseDuration>
    <courseTutor>Mr Blue</courseTutor>
  </course>
  - <course>
    <courseID>0007</courseID>
    <courseName>Foundation Computing</courseName>
    <courseCredits>30</courseCredits>
    <courseDuration>16</courseDuration>
    <courseTutor>Mr Yellow</courseTutor>
  </course>
</list>

```

Here is the same search query but using 'text' as the output format.



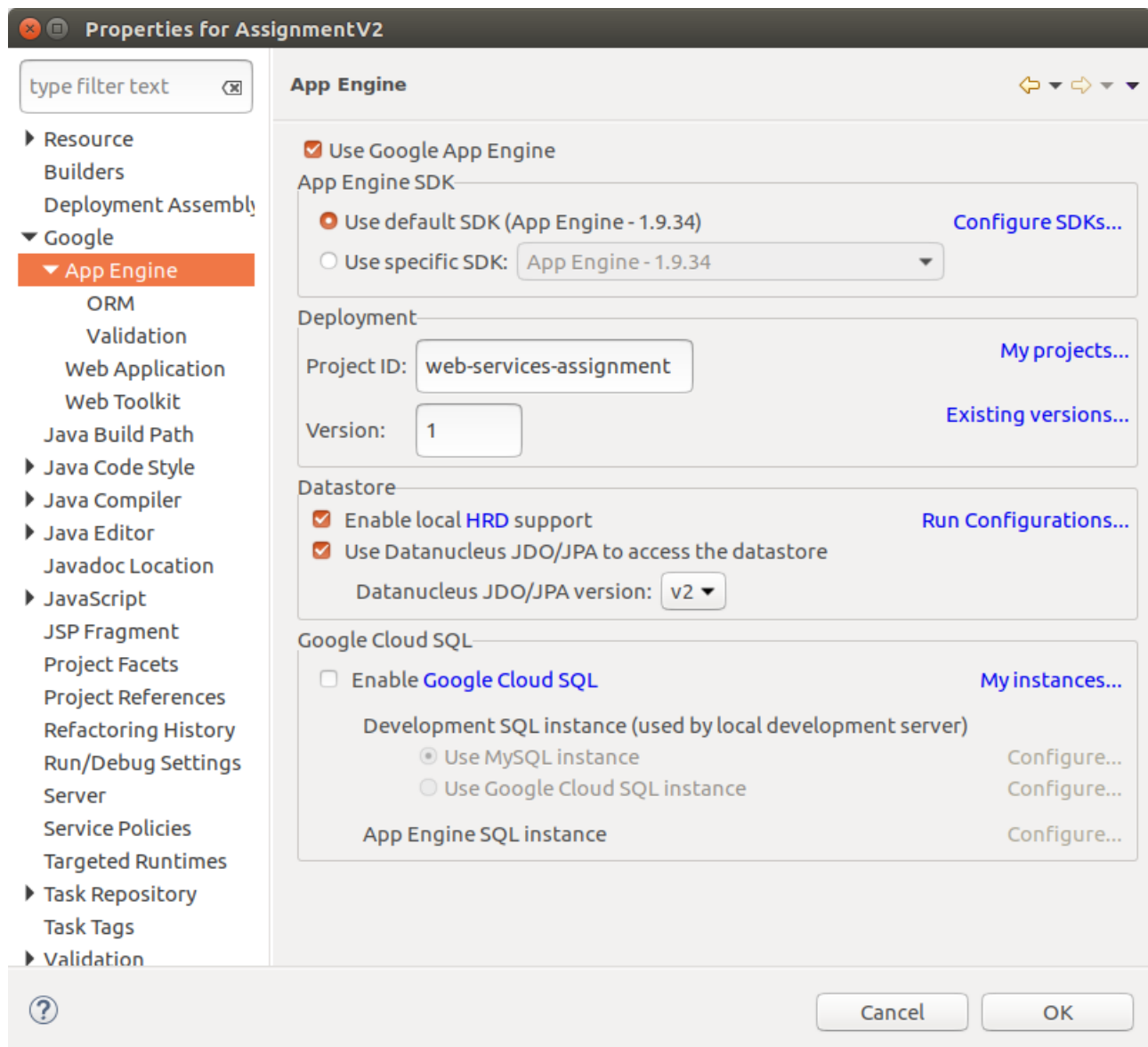
Finally, this screenshot demonstrates performing a search when the format parameter is not set.



Google App Engine to implement the application on a remote cloud based server

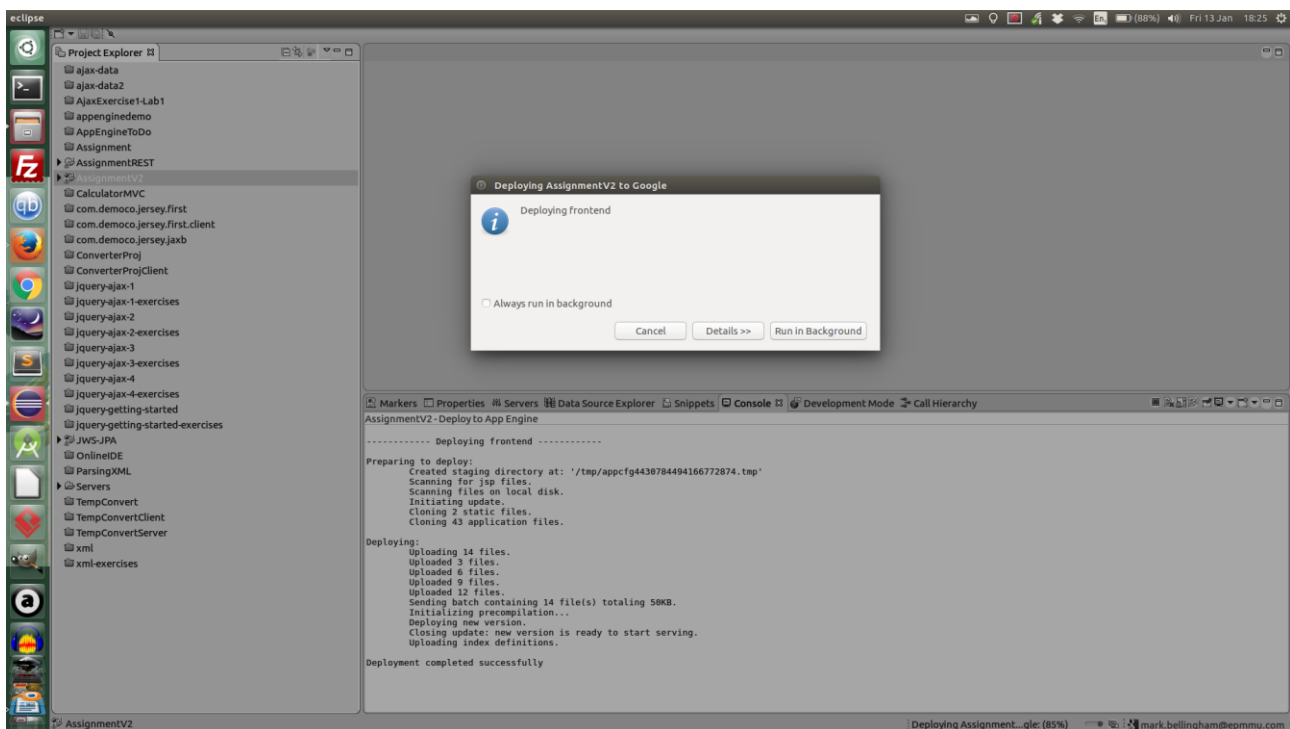
This feature was not completed for two main reasons. Firstly, I struggled to understand the methods to create the database using a Data Store. For this reason I chose to continue developing the application using MySQL, because having a working system and continuing with the other elements of the assignment is better than spending a long time on one single feature. Since I also found the other elements to be difficult, perhaps this was the right decision even though it will cost me a lot of marks.

The second major problem I had with this section was that on trying to upload the code I had so far onto Google, Eclipse would upload successfully but then crash on deployment. I'm not sure if this is related to my decision to use a MySQL database. I also tried to upload it using a university computer with the same results, maybe there is a setting somewhere that I missed. I have included below some screenshots of my attempts.



This is the settings console showing App Engine enabled and Project ID set.

This screenshot shows Eclipse after it has crashed. You can see the messages in the console stating that deployment completed successfully.



Here is the web browser message that is shown when you try to navigate to the site.



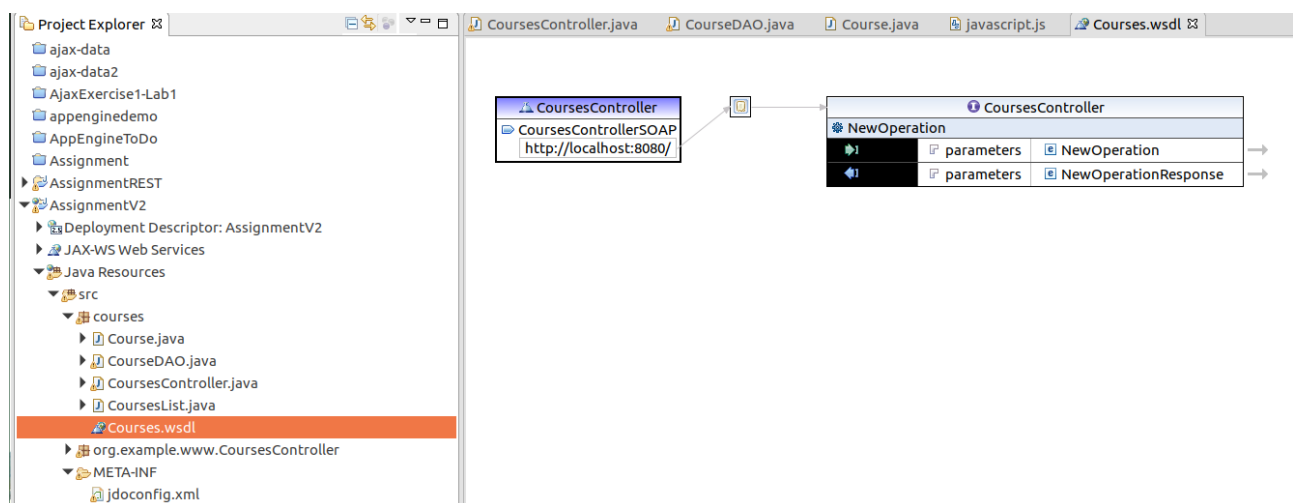
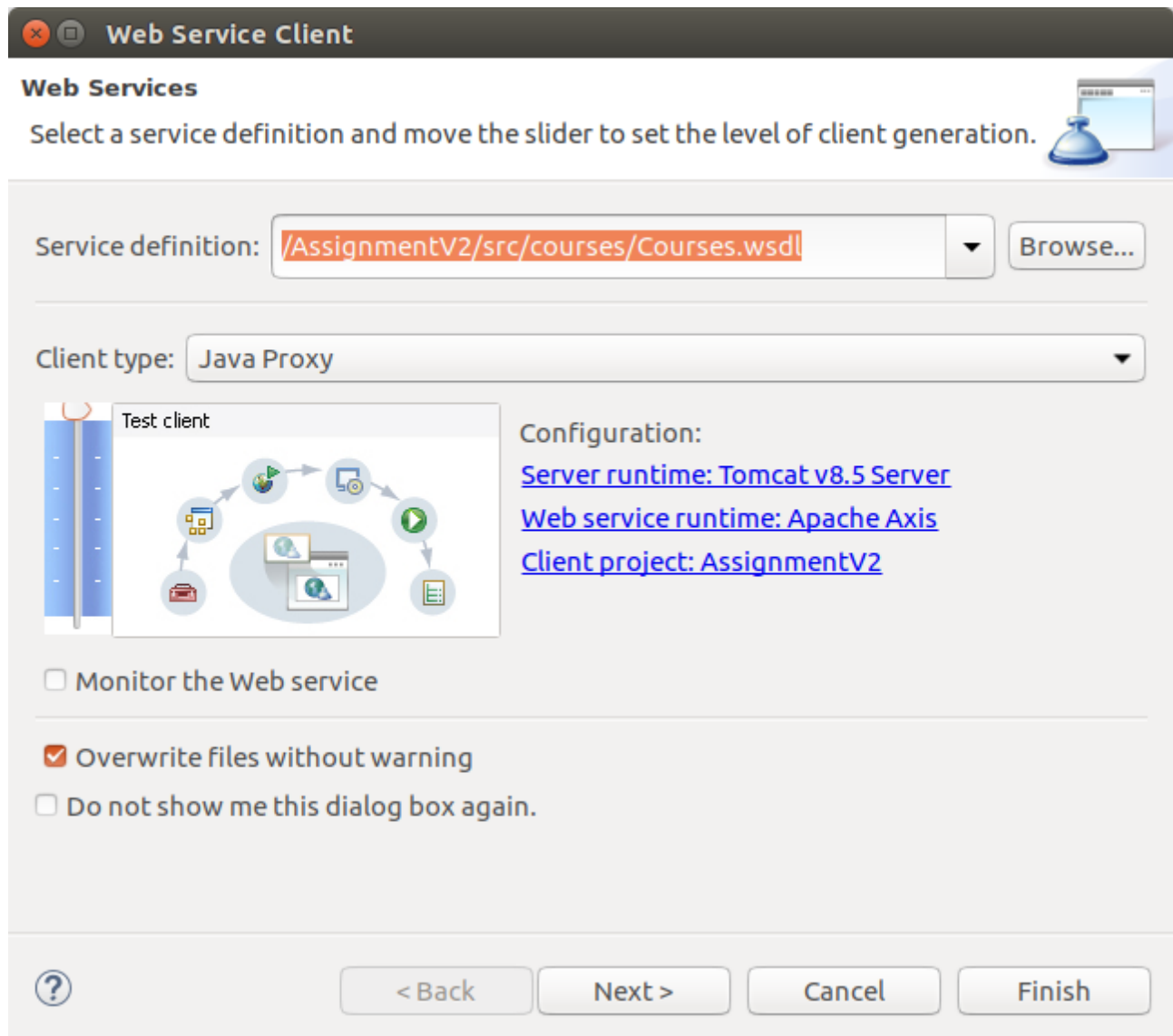
The screenshot shows the Google Cloud Platform Activity console. The browser address bar displays <https://console.cloud.google.com/home/activity?project=web-services-assignment>. The page header indicates the user has £234.33 in credit and 52 days left in their free trial. The left sidebar shows the 'Activity' tab selected. The main content area displays a list of activities grouped by date:

Date	Time	Status	Description
Today	6:25 PM	Completed update of a n...	mark.bellingham@epmmu.com completed update of a new default version: 1
	6:25 PM	Deployed a new version	mark.bellingham@epmmu.com deployed a new version: 1
1/11/17	5:32 PM	Completed update of a n...	mark.bellingham@epmmu.com completed update of a new default version: 1
	5:32 PM	Deployed a new version	mark.bellingham@epmmu.com deployed a new version: 1
1/7/17	12:43 AM	Update instance	mark.bellingham@epmmu.com updated web-services-assignment
1/6/17	11:09 PM	Completed update of a n...	mark.bellingham@epmmu.com completed update of a new default version: 1
	11:09 PM	Deployed a new version	mark.bellingham@epmmu.com deployed a new version: 1
	10:12 PM	Completed update of a n...	mark.bellingham@epmmu.com completed update of a new default version: 1
	10:12 PM	Deployed a new version	mark.bellingham@epmmu.com deployed a new version: 1
	3:09 PM	Completed update of a n...	mark.bellingham@epmmu.com completed update of a new default version: 1
	3:09 PM	Deployed a new version	mark.bellingham@epmmu.com deployed a new version: 1
	2:30 PM	Completed update of a n...	mark.bellingham@epmmu.com completed update of a new default version: 1
	2:30 PM	Deployed a new version	mark.bellingham@epmmu.com deployed a new version: 1
1/5/17	7:00 PM	Completed update of a n...	mark.bellingham@epmmu.com completed update of a new default version: 1
	7:00 PM	Deployed a new version	mark.bellingham@epmmu.com deployed a new version: 1
	6:32 PM	Completed update of a n...	mark.bellingham@epmmu.com completed update of a new default version: 1
	6:31 PM	Deployed a new version	mark.bellingham@epmmu.com deployed a new version: 1
	5:10 PM	Completed update of a n...	mark.bellingham@epmmu.com completed update of a new default version: 1
	5:10 PM	Deployed a new version	mark.bellingham@epmmu.com deployed a new version: 1
	5:06 PM	Completed update of a n...	mark.bellingham@epmmu.com completed update of a new default version: 1
	5:06 PM	Deployed a new version	mark.bellingham@epmmu.com deployed a new version: 1

Finally a screenshot of the Google AppEngine console showing the website uploaded successfully. I also checked the Error Log on AppEngine but there is nothing there.

A WSDL description of the interface to the web service

Screenshot showing the creation of a simple WSDL client.



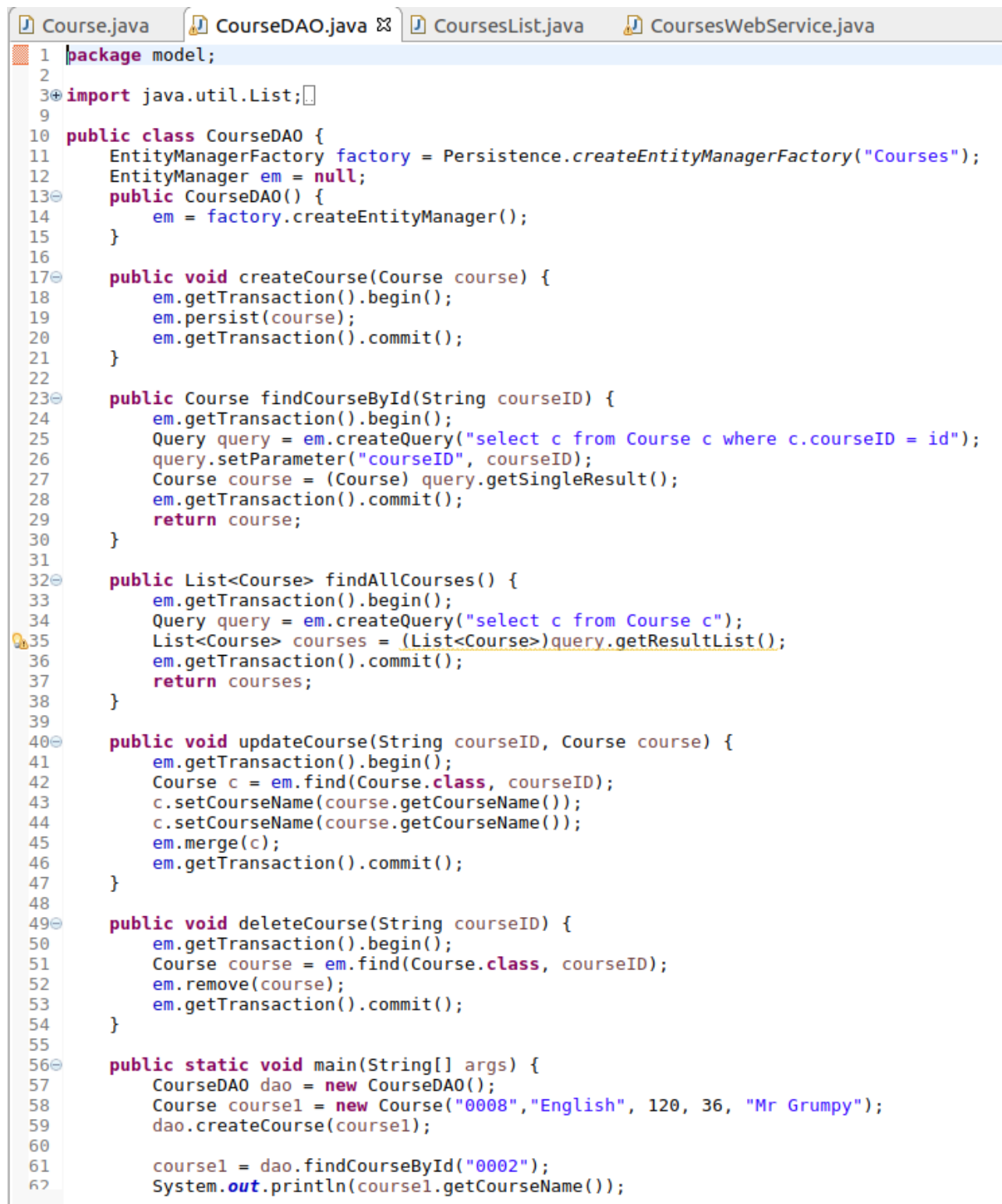
The output showing SOAP interface.

Methods	Inputs
<ul style="list-style-type: none">getEndpoint()setEndpoint(java.lang.String)getCoursesController_PortType()newOperation(java.lang.String)	<div>Invoke Clear</div>
	<div>Result</div> <div>localhost.AssignmentV2.CoursesControllerSOAPStub@2075d149</div>

Access to the data using REST type interaction

Unfortunately, this is another area that I was not able to complete successfully. I did manage to generate RESTful messages that were hard-coded in the application but I could not get the program to return data from the database. I think it may be due to a configuration problem in my persistence.xml file.

I am including screenshots showing the progress made with the code. Below is the DAO



```

1 package model;
2
3 import java.util.List;
4
5
6
7
8
9
10 public class CourseDAO {
11     EntityManagerFactory factory = Persistence.createEntityManagerFactory("Courses");
12     EntityManager em = null;
13     public CourseDAO() {
14         em = factory.createEntityManager();
15     }
16
17     public void createCourse(Course course) {
18         em.getTransaction().begin();
19         em.persist(course);
20         em.getTransaction().commit();
21     }
22
23     public Course findCourseById(String courseID) {
24         em.getTransaction().begin();
25         Query query = em.createQuery("select c from Course c where c.courseID = id");
26         query.setParameter("courseID", courseID);
27         Course course = (Course) query.getSingleResult();
28         em.getTransaction().commit();
29         return course;
30     }
31
32     public List<Course> findAllCourses() {
33         em.getTransaction().begin();
34         Query query = em.createQuery("select c from Course c");
35         List<Course> courses = (List<Course>)query.getResultList();
36         em.getTransaction().commit();
37         return courses;
38     }
39
40     public void updateCourse(String courseID, Course course) {
41         em.getTransaction().begin();
42         Course c = em.find(Course.class, courseID);
43         c.setCourseName(course.getCourseName());
44         c.setCourseName(course.getCourseName());
45         em.merge(c);
46         em.getTransaction().commit();
47     }
48
49     public void deleteCourse(String courseID) {
50         em.getTransaction().begin();
51         Course course = em.find(Course.class, courseID);
52         em.remove(course);
53         em.getTransaction().commit();
54     }
55
56     public static void main(String[] args) {
57         CourseDAO dao = new CourseDAO();
58         Course course1 = new Course("0008", "English", 120, 36, "Mr Grumpy");
59         dao.createCourse(course1);
60
61         course1 = dao.findCourseById("0002");
62         System.out.println(course1.getCourseName());

```

This shot shows the program code. It is designed to return either JSON or XML data and to either display all courses or just one when it is supplied with an ID.

```

Course.java  CourseDAO.java  CoursesList.java  CoursesWebService.java
1 package rest;
2
3 import java.util.List;
19
20 // /rest/course
21 @Path("/course")
22 public class CoursesWebService {
23
24     private CourseDAO dao = new CourseDAO();
25     private Course course = new Course();
26     private CoursesList list = new CoursesList();
27     private String output = "";
28
29     // /rest/course/json
30 @GET
31 @Produces("application/json")
32 @Path("/json")
33 public String getAllCoursesJSON() {
34     List<Course> courses = dao.findAllCourses();
35     output = new Gson().toJson(courses);
36     return output;
37 }
38
39 // /rest/course/json/{id}
40 @GET
41 @Produces("application/json")
42 @Path("/json/{id}")
43 public String getCourseForIdJSON(@PathParam("id") String courseID) {
44     course = dao.findCourseById(courseID);
45     output = new Gson().toJson(course);
46     return output;
47 }
48
49 // /rest/course/xml
50 @GET
51 @Produces("application/json")
52 @Path("/json")
53 public String getAllCoursesXML() {
54     List<Course> courses = dao.findAllCourses();
55
56     XStream xstream = new XStream();
57     xstream.alias("course", Course.class);
58     xstream.addImplicitCollection(CoursesList.class, "list");
59     output = xstream.toXML(courses);
60     return output;
61 }
62
63 // /rest/course/xml/{id}
64 @GET
65 @Produces("application/json")
66 @Path("/json/{id}")
67 public String getCourseForIdXML(@PathParam("id") String courseID) {
68     course = dao.findCourseById(courseID);
69     XStream xstream = new XStream();
70     xstream.alias("course", Course.class);
71     xstream.addImplicitCollection(CoursesList.class, "list");
72     output = xstream.toXML(course);
73     return output;

```

Finally a screenshot of the persistence.xml file



```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLoc
3 <persistence-unit name="transactions-optional">
4   <provider>org.datanucleus.api.jpa.PersistenceProviderImpl</provider>
5   <properties>
6     <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
7     <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/courses" />
8     <property name="javax.persistence.jdbc.user" value="bellingm" />
9     <property name="javax.persistence.jdbc.password" value="krIsderm2" />
10  </properties>
11 </persistence-unit>
12 </persistence>
13
```

An Ajax based web front end to retrieve the data and display in a suitable format using library based routines for an enhanced user interface

The program front-end is a web application displayed in the browser, where the user can search for courses and have the output in JSON, XML or plain text. There is a text input box where the user can insert a new course using either JSON or XML formatting. Finally there is a table which shows all the courses in the database with basic CRUD functions.

The user can enter any number of characters forming all or part of the course name. Characters can be from the beginning, middle or end of the course name. The search function uses JavaScript to submit the form and Ajax to insert the results without refreshing the page. The program shows the results differently depending on the format. JSON data is shown in a html div. Plain text is also shown in a div but the courses are displayed with bullet points for each one. XML is shown using 'pre' tags so that it is displayed correctly on the screen. The JavaScript code shows and hides the div and pre tags automatically when they are used/not used to keep the page neat.

When inserting a course using JSON or XML, the same textbox is used. The program can detect which format is inputted and divert the code as necessary. This also cuts down on clutter and keeps things neat and simple for the user.

First screenshot shows the application as it is initially presented to the user

Search Courses

Enter all or part of a course name

Insert course using JSON or XML

Single course only. For JSON end-tags should be {} - For XML use Course tag and attribute tags

Courses

Course ID	Course Name	Course Credits	Course Duration	Course Tutor	Add	Update	Clear
0001	Computing	120	36	Mr Brown	Delete	Select	
0002	Computer Science	120	36	Mr Blue	Delete	Select	
0003	Web and Multimedia	120	36	Mr Pink	Delete	Select	
0004	Games Design	120	36	Mr Black	Delete	Select	
0005	Software Engineering	120	36	Mr Red	Delete	Select	
0006	Forensics and Security	120	36	Mr Green	Delete	Select	
0007	Foundation Computing	30	16	Mr Yellow	Delete	Select	
0008	Business and Economics	120	36	Mr Happy	Delete	Select	

The three sections are clearly laid out with helpful hints for the user.

This screenshot demonstrates a search return in JSON format.

Search Courses


```
[{"courseID":"0001","courseName":"Computing","courseCredits":120,"courseDuration":36,"courseTutor":"Mr Brown"}, {"courseID":"0002","courseName":"Computer Science","courseCredits":120,"courseDuration":36,"courseTutor":"Mr Blue"}, {"courseID":"0007","courseName":"Foundation Computing","courseCredits":30,"courseDuration":16,"courseTutor":"Mr Yellow"}]
```

The same search is done with XML,

Search Courses


```
<course>
  <courseID>0001</courseID>
  <courseName>Computing</courseName>
  <courseCredits>120</courseCredits>
  <courseDuration>36</courseDuration>
  <courseTutor>Mr Brown</courseTutor>
</course>
<course>
  <courseID>0002</courseID>
  <courseName>Computer Science</courseName>
  <courseCredits>120</courseCredits>
  <courseDuration>36</courseDuration>
  <courseTutor>Mr Blue</courseTutor>
</course>
<course>
  <courseID>0007</courseID>
  <courseName>Foundation Computing</courseName>
  <courseCredits>30</courseCredits>
  <courseDuration>16</courseDuration>
  <courseTutor>Mr Yellow</courseTutor>
</course>
```

and finally the plain text version

Search Courses

 Text ▾ Submit

- courseID0001, courseNameComputing, courseCredits120, courseDuration36, courseTutorMr Brown,
- courseID0002, courseNameComputer Science, courseCredits120, courseDuration36, courseTutorMr Blue,
- courseID0007, courseNameFoundation Computing, courseCredits30, courseDuration16, courseTutorMr Yellow

This screenshot shows inserting a course using JSON format

Insert course using JSON or XML

```
{"courseID":"0010","courseName":"Business and Law","courseCredits":120,"courseDuration":36,"courseTutor":"Mr Rush"}
```

The next screenshot shows inserting a course using XML, and the successful result of the JSON insertion at the bottom

Insert course using JSON or XML

```
<courseID>0006</courseID>  
<courseName>Forensics and Security</courseName>  
<courseCredits>120</courseCredits>  
<courseDuration>36</courseDuration>  
<courseTutor>Mr Green</courseTutor>  
</course>
```

Submit

Courses

Course ID	Course Name	Course Credits	Course Duration	Course Tutor	Add	Update	Clear
0002	Computer Science	120	36	Mr Blue	Delete	Select	
0003	Web and Multimedia	120	36	Mr Pink	Delete	Select	
0004	Games Design	120	36	Mr Black	Delete	Select	
0005	Software Engineering	120	36	Mr Red	Delete	Select	
0006	Forensics and Security	120	36	Mr Green	Delete	Select	
0008	Business and Economics	120	36	Mr Happy	Delete	Select	
0009	International Business	120	36	Mr Grumpy	Delete	Select	
0010	Business and Law	120	36	Mr Rush	Delete	Select	

Finally, confirmation of the XML insert

Courses

Course ID	Course Name	Course Credits	Course Duration	Course Tutor	Add	Update	Clear
0002	Computer Science	120	36	Mr Blue	Delete	Select	
0003	Web and Multimedia	120	36	Mr Pink	Delete	Select	
0004	Games Design	120	36	Mr Black	Delete	Select	
0005	Software Engineering	120	36	Mr Red	Delete	Select	
0006	Forensics and Security	120	36	Mr Green	Delete	Select	
0008	Business and Economics	120	36	Mr Happy	Delete	Select	
0009	International Business	120	36	Mr Grumpy	Delete	Select	
0010	Business and Law	120	36	Mr Rush	Delete	Select	
0011	Business Studies	120	36	Mr Tickle	Delete	Select	

Code Design using MVC, DAO and modular routines.

The core structure of this program uses the Model View Controller approach, which is one of the most successful design patterns in programming. The View has already been demonstrated earlier in this report, with access to the core program functions using simple get requests and more complex interactive methods. Similarly the controller is set out in the first chapter, forming part of the servlet that receives requests from the view, passing the parameters to the DAO and returning the results back to the View. The Data Accessor Object is revealed below.

```

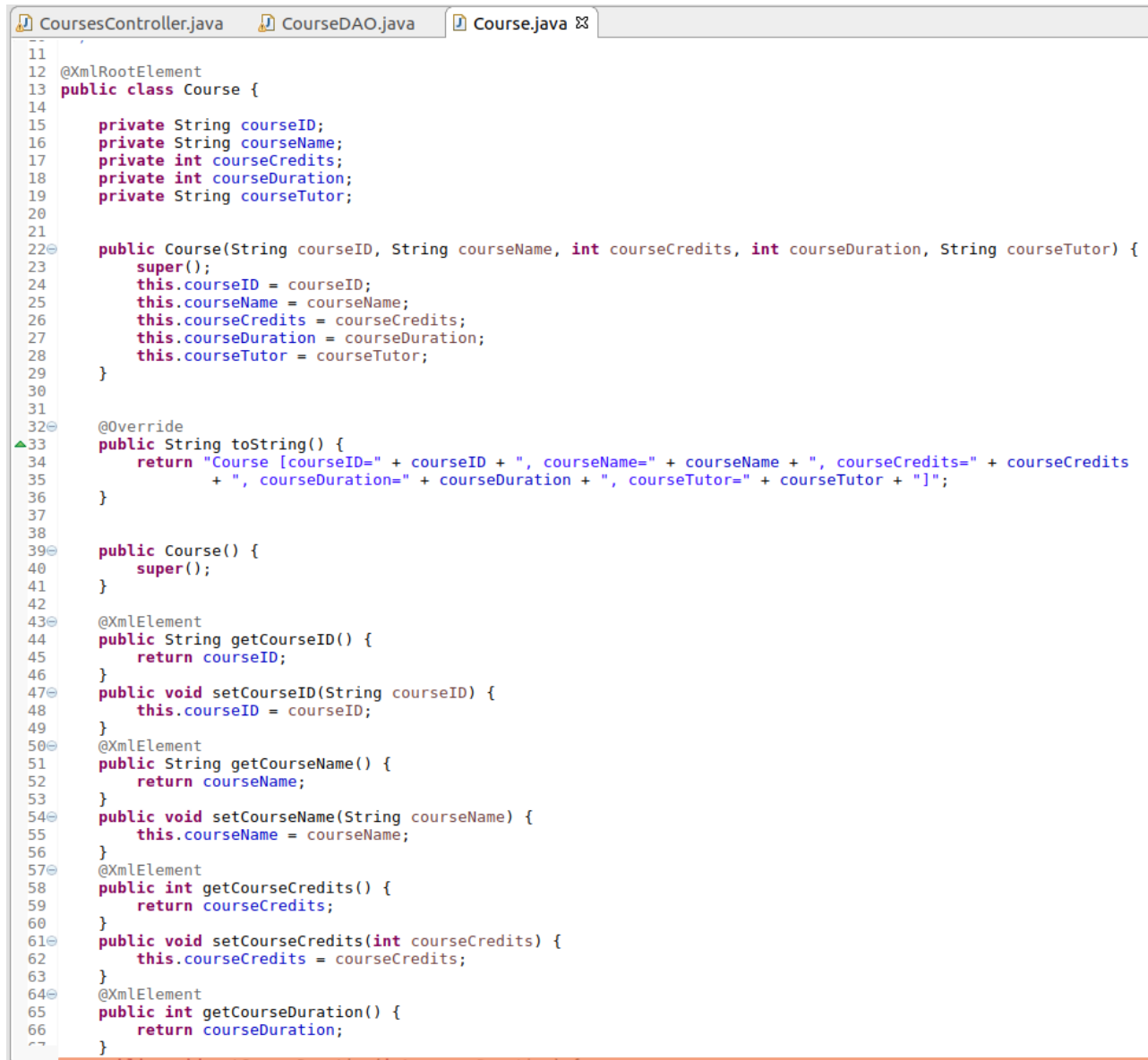
CoursesController.java  CourseDAO.java x
61      }
62
63
64      // Method for inserting a course into the database.
65      // It takes a Course object as a parameter but does not return anything.
66      public void insertCourse(Course course) {
67          String sql = "insert into courses " +
68                      "(courseID, courseName, courseCredits, courseDuration, courseTutor)" +
69                      " values (?, ?, ?, ?, ?)";
70
71          java.sql.Connection conn = openConnection();
72
73          try {
74              java.sql.PreparedStatement stmt = conn.prepareStatement(sql);
75              stmt.setString(1, course.getCourseID());
76              stmt.setString(2, course.getCourseName());
77              stmt.setInt(3, course.getCourseCredits());
78              stmt.setInt(4, course.getCourseDuration());
79              stmt.setString(5, course.getCourseTutor());
80              stmt.execute();
81          } catch (SQLException e) {
82              e.printStackTrace();
83          } finally {
84              closeConnection(conn);
85          }
86      }
87
88
89      // Method for creating a Course type object from the result set.
90      // It is not accessed directly from the controller but used as a sub-routine in other methods
91      private Course getNextCourse(java.sql.ResultSet rs) {
92          Course thisCourse = null;
93
94          try {
95              thisCourse = new Course (
96                  rs.getString("courseID"),
97                  rs.getString("courseName"),
98                  rs.getInt("courseCredits"),
99                  rs.getInt("courseDuration"),
100                 rs.getString("courseTutor")
101             );
102          } catch (SQLException e) {
103              e.printStackTrace();
104          }
105          return thisCourse;
106      }
107
108
109      // Method that returns a single course from the database when provided with an id.
110      public Course getOneCourse(String courseID) {
111          Course course = null;
112          String sql = "select * from courses where courseID = ?";
113          java.sql.Connection conn = openConnection();
114          try {
115              java.sql.PreparedStatement stmt = conn.prepareStatement(sql);
116              stmt.setString(1, courseID);
117              java.sql.ResultSet rs1 = stmt.executeQuery();
118              if (rs1.next()) {

```

The connection is separated out so that it can be reused by each function in the DAO without repeating code. There is a separate function for each of the different types of search queries required by the application, which then return a Course or list of Courses back to the controller.

Because I chose to use MySQL as the database format, which was not specified in the assignment, I had to be careful to use Prepared Statements to guard against SQL Injection attacks. Had I been able to implement the Data Store option, it would be more suitable for the Cloud Computing environment because it is more amenable to load balancing, both for the server and the network.

The Course type is instantiated as its own object, with a constructor and getters and setters so that each of its attributes can be accessed independently.



```

11
12 @XmlElement
13 public class Course {
14
15     private String courseID;
16     private String courseName;
17     private int courseCredits;
18     private int courseDuration;
19     private String courseTutor;
20
21
22     public Course(String courseID, String courseName, int courseCredits, int courseDuration, String courseTutor) {
23         super();
24         this.courseID = courseID;
25         this.courseName = courseName;
26         this.courseCredits = courseCredits;
27         this.courseDuration = courseDuration;
28         this.courseTutor = courseTutor;
29     }
30
31
32     @Override
33     public String toString() {
34         return "Course [courseID=" + courseID + ", courseName=" + courseName + ", courseCredits=" + courseCredits
35             + ", courseDuration=" + courseDuration + ", courseTutor=" + courseTutor + "];"
36     }
37
38
39     public Course() {
40         super();
41     }
42
43     @XmlElement
44     public String getCourseID() {
45         return courseID;
46     }
47     public void setCourseID(String courseID) {
48         this.courseID = courseID;
49     }
50     @XmlElement
51     public String getCourseName() {
52         return courseName;
53     }
54     public void setCourseName(String courseName) {
55         this.courseName = courseName;
56     }
57     @XmlElement
58     public int getCourseCredits() {
59         return courseCredits;
60     }
61     public void setCourseCredits(int courseCredits) {
62         this.courseCredits = courseCredits;
63     }
64     @XmlElement
65     public int getCourseDuration() {
66         return courseDuration;
67     }

```

The attributes are given XML/JSON tags so that they can be correctly identified by the functions that parse XML and JSON.

```

CoursesController.java CourseDAO.java Course.java javascript.js
21     if (window.ActiveXObject){
22         // code for Internet Explorer
23         xmlString = xmlData.xml;
24     } else {
25         // code for Firefox, Chrome and others
26         xmlString = (new XMLSerializer()).serializeToString(xmlData);
27     }
28     xmlString = xmlString.replace("<list>", "").replace("</list>", "");
29     return xmlString;
30 }
31
32 // Function that formats the plain text data coming from the server, putting it in a list
33 function formatText(input) {
34     console.log(input);
35     input = input.replace(/[\^a-zA-Z0-9, ]/g, '');
36     inputArr = input.split("Course");
37     formatted = "<ul>";
38     for (i = 0; i < inputArr.length-1; i++) {
39         formatted += ("<li>" + inputArr[i+1] + "</li>");
40     }
41     formatted += "</ul>";
42     return formatted;
43 }
44
45 // Function to get the JSON data from the server according to search query
46 function json(params) {
47     $('#search-div').show();
48     $('#search-pre').hide();
49     $.get('CoursesController?action=search&' + params, function(responseText) {
50         data: 'json',
51         $('#search-div').text(JSON.stringify(responseText));
52     });
53 }
54
55 //Function to get the XML data from the server according to search query
56 function xml(params) {
57     $('#search-div').hide();
58     $('#search-pre').show();
59     $.get('CoursesController?action=search&' + params, function(responseText) {
60         data: 'xml',
61         xmlString = xmlToString(responseText);
62         $('#search-pre').text(xmlString);
63     });
64 }
65
66 //Function to get the plain text data from the server according to search query
67 function text(params) {
68     $('#search-div').show();
69     $('#search-pre').hide();
70     $.get('CoursesController?action=search&' + params, function(responseText) {
71         data: 'text/plain',
72         responseText = formatText(responseText);
73         $('#search-div').html(responseText);
74     });
75 }
76
77 /* Function that detects when the search form is submitted and handles the
    submission so that AJAX can insert the result into the page without reloading it */

```

The JavaScript, which forms part of the View, contains many individual functions that are concerned with how the data is displayed. These functions are separated out according to task so for example, the function that implements the request to the server is separate from the one that formats the returned data.

Throughout the program the code is clean and well laid-out, with useful comments and descriptive variable names that would help any future programmer, be it myself or someone else, to adapt, update or modify the program.

Although I did not successfully implement the RESTful interface, it clearly has many useful applications such as returning data from an online newspaper or catalogue.

Similarly XML is very human-readable, if a little clunky, whereas JSON is light on resources, which is very useful in the mobile environment or in places where network traffic is under strain.