Critical Analysis and Proof of Work

Enterprise Programming Assignment

Film Website

Written By: Muhammad Salman Khan

Student ID: 16065179

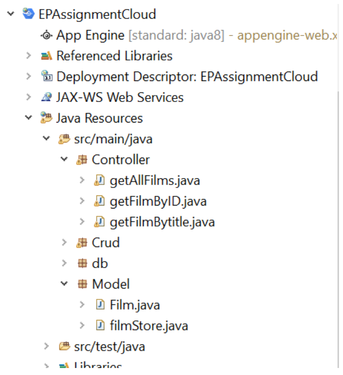
Contents

Introduction

For this assignment I was able to produce a working Film Web Service from scratch that allowed me to access data from a Google cloud database which I used. In the introduction I will also explain about what I have done in my assignment. I have attempted almost every part of the assignment. I did CRUD in jersey. I used AJAX to “getallfilms”, “filmbyId” and “filmbyTitle” as well. I used AJAX for delete, update and add. I made my own Google Cloud Database using my personal email and I populated it with data from the lab files. I was also able to successfully deploy my project into app engine which I will be explaining further down. I also used servlet for https call for getting all films, film by title and film by id and they are explained further down as well with screenshot evidence. I did add a WSDL file but could not figure out what to do so left it in between. The aim of this critical analysis is to show the proof of work done and explain all the coding techniques used and different methods used to achieve what was done. There will also be a part where I talk about coding practices and at the end a conclusion.

MVC

In this section I would be talking about MVC architecture that I used in my project. Old programming methods had input->process and then output but MVC works on Controller->Model and then view. MVC helps in separating all the aspects of project like managing the data and viewing it to the user. I used MVC for my project because its very helpful as it separates the code and it’s a lot easier to understand. Any other developer working on my code would easily understand my code and he will be able to update it if he wants to.



```
EPAssignmentCloud
  App Engine [standard: java8] - appengine-web.x
  Referenced Libraries
  Deployment Descriptor: EPAssignmentCloud
  JAX-WS Web Services
  Java Resources
    src/main/java
      Controller
        getAllFilms.java
        getFilmByID.java
        getFilmBytitle.java
      Crud
      db
      Model
        Film.java
        filmStore.java
    src/test/java
```

Figure 1: Model and controller shown in project.

Model:

Model is like the main part of the architecture as it manages all the data, the logic and the rules associated with the project. It is independent of the view and does not have any direct relation with it.

View:

The user interface depends on the view because any information that needs to be displayed to the user is handled in the view. Any type of data that needs to be displayed is done through view. Its also possible to have multiple views of the same data.

Controller:

The controller accepts the input given by the model and then pass it onto view as commands. Controllers role is to handle the user interaction and respond to it. Like for this project the controller will handle query values and pass it onto model which in turn will query into the database.

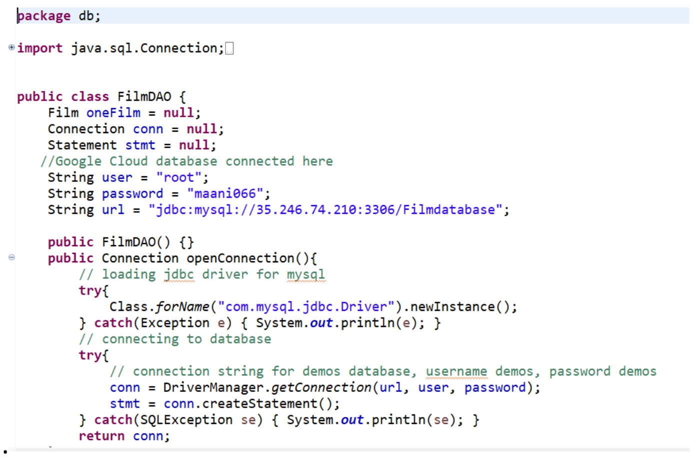
DAO pattern

Data access object is a pattern that allows the business layer to be separate from the persistence layer. The persistence layer is not able to access the database on its own and that is what the DAO is for because it lets it access only specific data that is needed to execute queries. There are many advantages of using DAO pattern and they are as below

- Common calls to retrieve objects or info.
- The general layout can be repeated for other DAOs.
- It separates the business logic from other components of your code.

There are few disadvantages as well of using DAO pattern and they are as below.

- You cannot lazy-load stuff with this easily because you have to change the whole DAO to do that.
- The code becomes tedious and it repeats over and over.

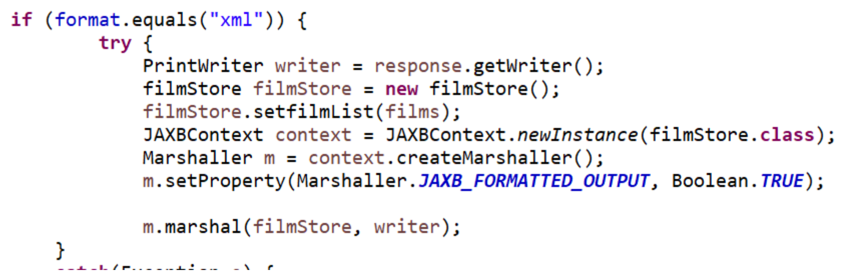


```
package db;

import java.sql.Connection;


public class FilmDAO {
    Film oneFilm = null;
    Connection conn = null;
    Statement stmt = null;
   //Google Cloud database connected here
    String user = "root";
    String password = "maani066";
    String url = "jdbc:mysql://35.246.74.210:3306/Filmdatabase";

    public FilmDAO() {}
    public Connection openConnection(){
        // loading jdbc driver for mysql
        try{
            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch(Exception e) { System.out.println(e); }
        // connecting to database
        try{
            // connection string for demos database, username demos, password demos
            conn = DriverManager.getConnection(url, user, password);
            stmt = conn.createStatement();
        } catch(SQLException se) { System.out.println(se); }
        return conn;
```

Figure 2 Shows the FilmDAO in project

JAXB

JAXB is a software framework that I used in my project so I can Marshall java objects into XML and vice versa. It was very helpful using JAXB because it allowed me to store data and retrieve data in XML format for which we only need a simple XML.jsp file and the rest was done by JAXB which formatted it to own ways and presented the data in XML format to the user.



```
 if (format.equals("xml")) {
         try {
             PrintWriter writer = response.getWriter();
             filmStore filmStore = new filmStore();
             filmStore.setfilmList(films);
             JAXBContext context = JAXBContext.newInstance(filmStore.class);
             Marshaller m = context.createMarshaller();
             m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

             m.marshal(filmStore, writer);
     }
```

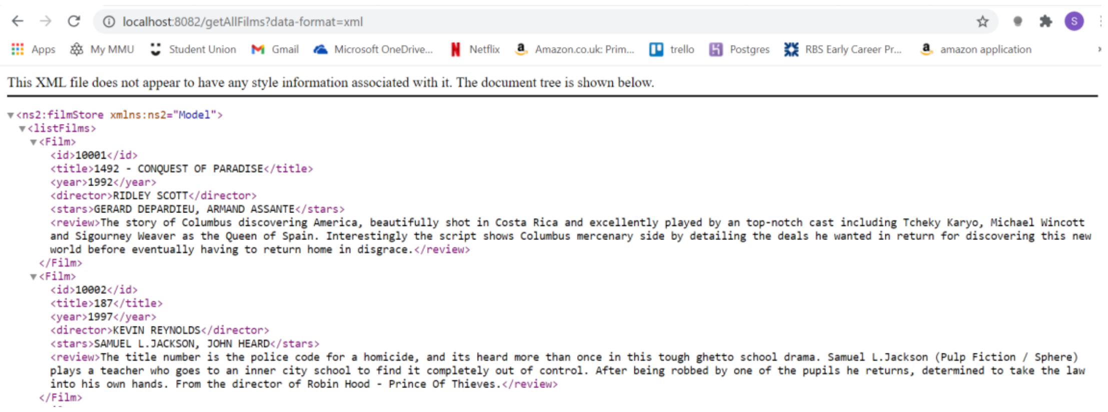
Figure 3 shows JAXB Marshaller in getAllFilms

Http calls

Getting data in xml format:

I used some http calls as I had three servlets for that. First servlet I had was of getAllFilms which gave out all the films in the database in either XML or JSON format. The image below shows that when this certain http call was made the data was returned in XML format for all the films in the database. The port in the image is 8082 but I changed it to 8080 due to some server error which caused my project to not run on 8082 but it successfully runs in 8080 now so it wont be a problem anymore and the rest of the http call is the same as given below

http://localhost:8080/getAllFilms?data-format=xml



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

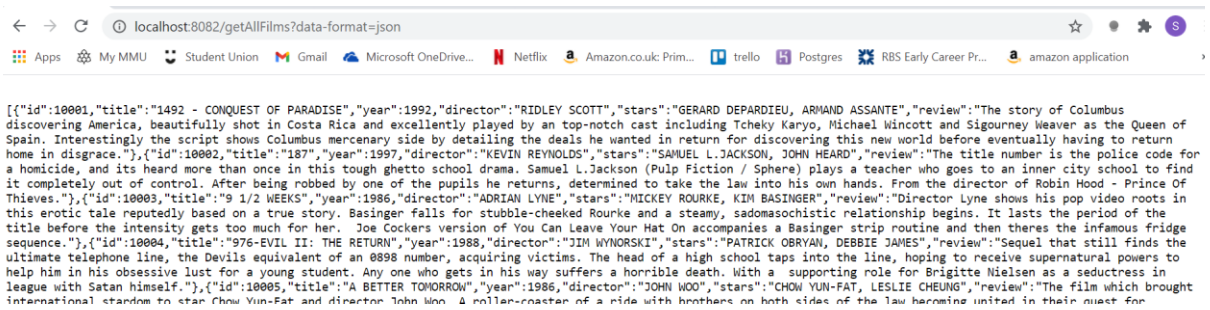
```
▼<ns2:filmStore xmlns:ns2="Model">
  ▼<listFilms>
    ▼<Film>
       <id>10001</id>
       <title>1492 - CONQUEST OF PARADISE</title>
       <year>1992</year>
       <director>RIDLEY SCOTT</director>
       <stars>GERARD DEPARDIEU, ARMAND ASSANTE</stars>
       <review>The story of Columbus discovering America, beautifully shot in Costa Rica and excellently played by an top-notch cast including Tcheky Karyo, Michael Wincott
       and Sigourney Weaver as the Queen of Spain. Interestingly the script shows Columbus mercenary side by detailing the deals he wanted in return for discovering this new
       world before eventually having to return home in disgrace.</review>
     </Film>
    ▼<Film>
       <id>10002</id>
       <title>187</title>
       <year>1997</year>
       <director>KEVIN REYNOLDS</director>
       <stars>SAMUEL L.JACKSON, JOHN HEARD</stars>
       <review>The title number is the police code for a homicide, and its heard more than once in this tough ghetto school drama. Samuel L.Jackson (Pulp Fiction / Sphere)
       plays a teacher who goes to an inner city school to find it completely out of control. After being robbed by one of the pupils he returns, determined to take the law
       into his own hands. From the director of Robin Hood - Prince Of Thieves.</review>
     </Film>
```

Figure 4 shows data outputted in XML format

Getting data in json format:

To get data in JSON format for all the films this http call was made, and it returned all the data in JSON form and for this I had a separate servlet so http call could be made. The port is changed because due to some error I had to make a new server and the new one had 8080 as a port so I just changed it to 8080 for you to use. The rest of the http call is correct as given below and you would be able to get all films from the http call given below.

http://localhost:8080/getAllFilms?data-format=json



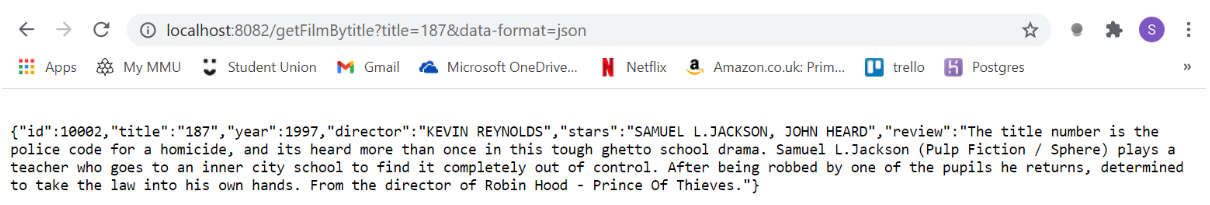
```
[{"id":10001,"title":"1492 - CONQUEST OF PARADISE","year":1992,"director":"RIDLEY SCOTT","stars":"GERARD DEPARDIEU, ARMAND ASSANTE","review":"The story of Columbus
discovering America, beautifully shot in Costa Rica and excellently played by an top-notch cast including Tcheky Karyo, Michael Wincott and Sigourney Weaver as the Queen of
Spain. Interestingly the script shows Columbus mercenary side by detailing the deals he wanted in return for discovering this new world before eventually having to return
home in disgrace."},{"id":10002,"title":"187","year":1997,"director":"KEVIN REYNOLDS","stars":"SAMUEL L.JACKSON, JOHN HEARD","review":"The title number is the police code for
a homicide, and its heard more than once in this tough ghetto school drama. Samuel L.Jackson (Pulp Fiction / Sphere) plays a teacher who goes to an inner city school to find
it completely out of control. After being robbed by one of the pupils he returns, determined to take the law into his own hands. From the director of Robin Hood - Prince Of
Thieves."},{"id":10003,"title":"9 1/2 WEEKS","year":1986,"director":"ADRIAN LYNE","stars":"MICKEY ROURKE, KIM BASINGER","review":"Director Lyne shows his pop video roots in
this erotic tale reputedly based on a true story. Basinger falls for stubble-cheeked Rourke and a steamy, sadomasochistic relationship begins. It lasts the period of the
title before the intensity gets too much for her.  Joe Cockers version of You Can Leave Your Hat On accompanies a Basinger strip routine and then theres the infamous fridge
sequence."},{"id":10004,"title":"976-EVIL II: THE RETURN","year":1988,"director":"JIM WYNORSKI","stars":"PATRICK OBRYAN, DEBBIE JAMES","review":"Sequel that still finds the
ultimate telephone line, the Devils equivalent of an 0898 number, acquiring victims. The head of a high school taps into the line, hoping to receive supernatural powers to
help him in his obsessive lust for a young student. Any one who gets in his way suffers a horrible death. With a  supporting role for Brigitte Nielsen as a seductress in
league with Satan himself."},{"id":10005,"title":"A BETTER TOMORROW","year":1986,"director":"JOHN WOO","stars":"CHOW YUN-FAT, LESLIE CHEUNG","review":"The film which brought
international stardom to star Chow Yun-Fat and director John Woo. A roller-coaster of a ride with brothers on both sides of the law becoming united in their quest for
```

Figure 5 shows data outputted in JSON format

Searching film by title:

So, for this http call I made a separate servlet class as well and called it getFilmByTitle and this had all the code to get film by the name in either JSON or XML format. The port for this was also changed because of the error I had with my server so u would be able to run this http call as it is given below using the port 8080.

http://localhost:8080/getFilmBytitle?title=187&data-format=json

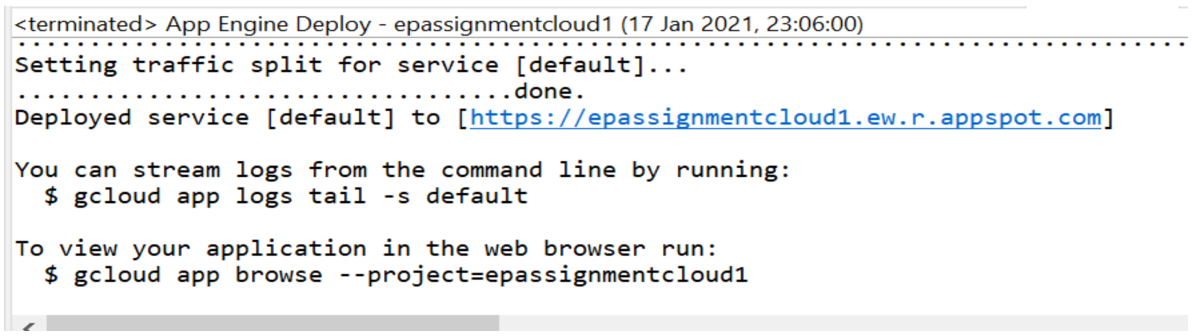


```
{"id":10002,"title":"187","year":1997,"director":"KEVIN REYNOLDS","stars":"SAMUEL L.JACKSON, JOHN HEARD","review":"The title number is the
police code for a homicide, and its heard more than once in this tough ghetto school drama. Samuel L.Jackson (Pulp Fiction / Sphere) plays a
teacher who goes to an inner city school to find it completely out of control. After being robbed by one of the pupils he returns, determined
to take the law into his own hands. From the director of Robin Hood - Prince Of Thieves."}
```

Figure 6 shows data outputted in JSON for a specific title

Google App Engine Deployment

The Project was successfully deployed onto the app engine and can be accessed by the URL given below. I used a personal email for this and in the Google Cloud Console I made a project by the name of EpAssignmentCloud1 and then implemented it accordingly and from eclipse I deployed the project using the same email address and it was successfully deployed into the Google App Engine.

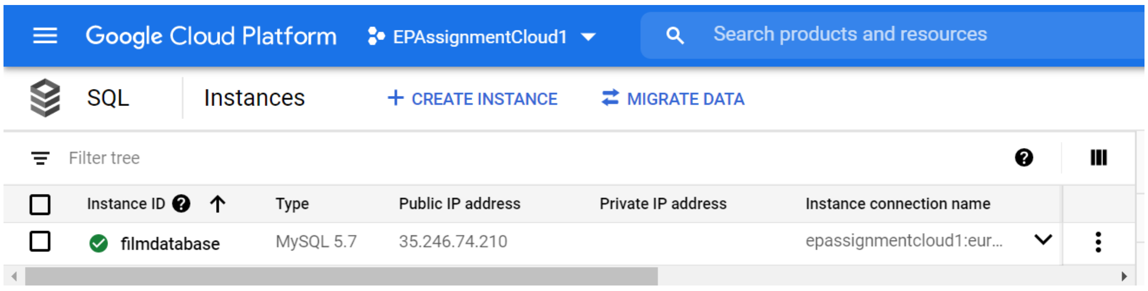


```
<terminated> App Engine Deploy - epassignmentcloud1 (17 Jan 2021, 23:06:00)
.................................................................................
Setting traffic split for service [default]...
..................................done.
Deployed service [default] to [https://epassignmentcloud1.ew.r.appspot.com]

You can stream logs from the command line by running:
  $ gcloud app logs tail -s default

To view your application in the web browser run:
  $ gcloud app browse --project=epassignmentcloud1
```

Figure 7 shows Deployment on Google App Engine

Google Cloud Database

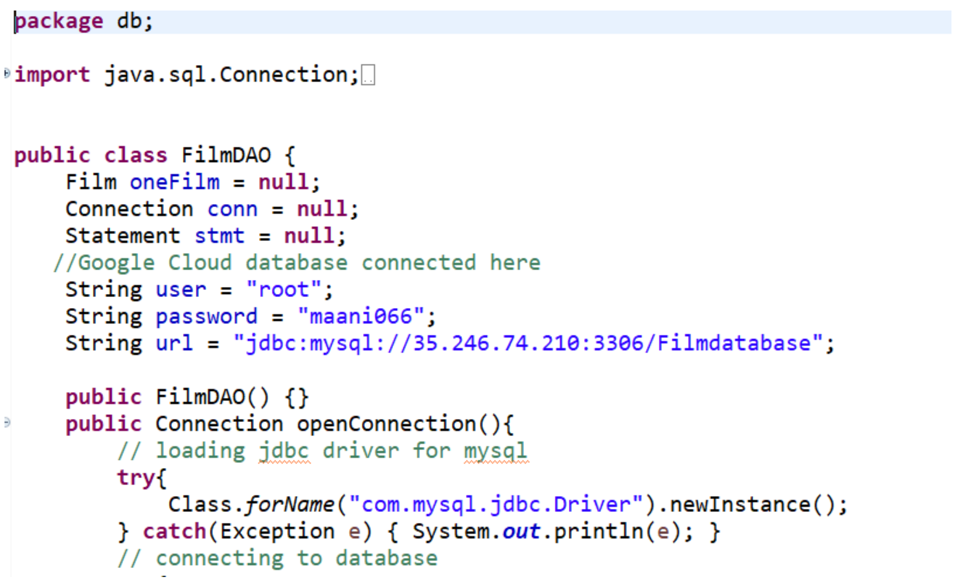
For this part I made a MYSQL Google Cloud Database using the same email as above and I was successful in populating it with the data from lab files. Then I connected my database with my project and for the IP in google cloud database I put 0.0.0.0/0 so it would be accessible by anyone from any IP. This below shows my google Cloud Console and the instance name and the IP. I used google cloud database for many reasons as this had good security than the Mudfoot one and it was my own private database that no one had access to. Its also easier to have backups on this and better network and its cloud based as well.



Instance ID	Type	Public IP address	Private IP address	Instance connection name
filmdatabase	MySQL 5.7	35.246.74.210		epassignmentcloud1:eur...

Figure 8 shows Google Cloud Database produced using personal email

This image below shows my implementation of Google Cloud Database within my project which I used to connect it to cloud database using my personal email address. The username and password to access my database is shown in the below code and the URL shows the public IP and the name of the database.

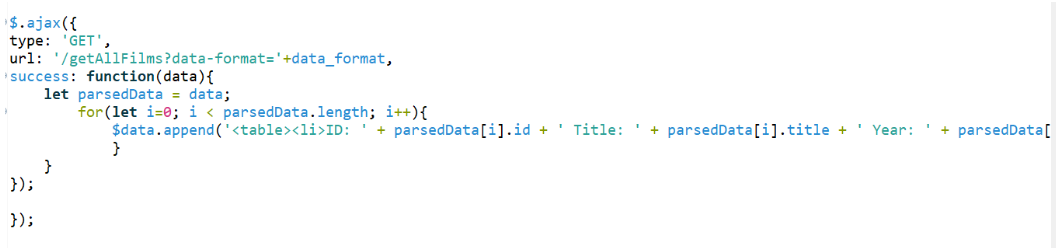


```
package db;

import java.sql.Connection;


public class FilmDAO {
    Film oneFilm = null;
    Connection conn = null;
    Statement stmt = null;
   //Google Cloud database connected here
    String user = "root";
    String password = "maani066";
    String url = "jdbc:mysql://35.246.74.210:3306/Filmdatabase";

    public FilmDAO() {}
    public Connection openConnection(){
        // loading jdbc driver for mysql
        try{
            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch(Exception e) { System.out.println(e); }
        // connecting to database
```

Figure 9 shows Google Cloud Database connected with the project

Ajax

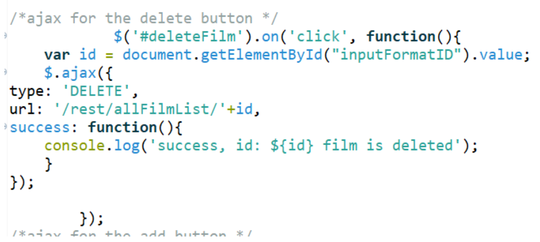
I did ajax button for getting all films which did not use any http calls but just gave the output in json format because I couldn't figure out how to do the same with xml. The data append gives all the data output in json format and it gives the data with bullet points for each separate film.



```
$.ajax({
type: 'GET',
url: '/getAllFilms?data-format='+data_format,
success: function(data){
    let parsedData = data;
        for(let i=0; i < parsedData.length; i++){
            $data.append('<table><li>ID: ' + parsedData[i].id + ' Title: ' + parsedData[i].title + ' Year: ' + parsedData[
            }
    }
});

});
```

Figure 10 shows the AJAX code for getting all Films

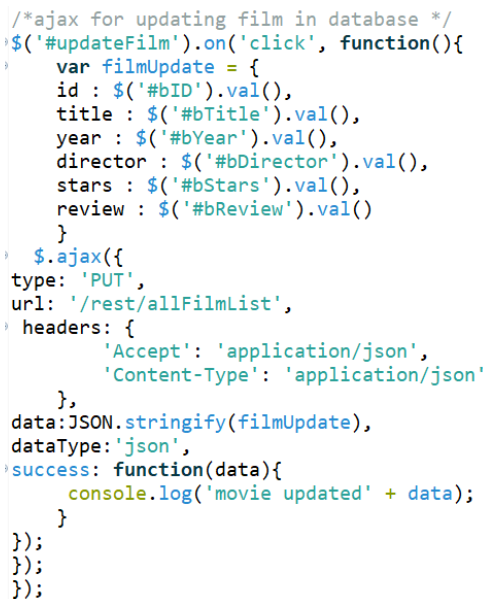
I did ajax for delete button where when u put the ID in the input field and press the button it runs this code and that id, and all the data associated with that id is deleted and in the console it shows as success the film with this id is deleted as shown in the code below.



```
/*ajax for the delete button */
            $('#deleteFilm').on('click', function(){
    var id = document.getElementById("inputFormatID").value;
    $.ajax({
type: 'DELETE',
url: '/rest/allFilmList/'+id,
success: function(){
    console.log('success, id: ${id} film is deleted');
    }
});

        });
```

Figure 11 shows AJAX button for delete

I did ajax for both add and update but due to the code too long for add I only could add update here but the add button works in my project. When u put all the details in the input field of update and click the button it runs this code, and the data is updated in the database and both xml and json formats are updated. The add works the same because when you complete all the fields and before pressing a button there is an option list where u can send data as XML or JSON and you choose any of them both and press the button and the data is sent to the database and you can also check if the data is added by searching for the same id.

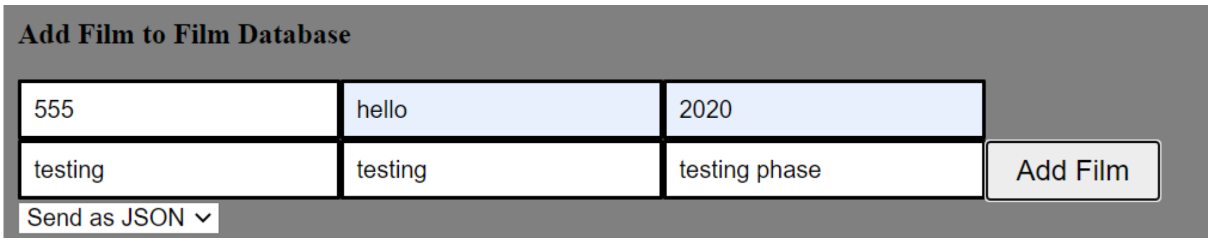


```
/*ajax for updating film in database */
$('#updateFilm').on('click', function(){
    var filmUpdate = {
    id : $('#bID').val(),
    title : $('#bTitle').val(),
    year : $('#bYear').val(),
    director : $('#bDirector').val(),
    stars : $('#bStars').val(),
    review : $('#bReview').val()
    }
  $.ajax({
type: 'PUT',
url: '/rest/allFilmList',
 headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json'
    },
data:JSON.stringify(filmUpdate),
dataType:'json',
success: function(data){
     console.log('movie updated' + data);
    }
});
});
});
```

Figure 12 shows Update Button in AJAX

Adding Film to Database

The below image shows when we try to add a new film in the database it asks for option to send as JSON or XML and once we press the Add Film button it goes onto the next part.



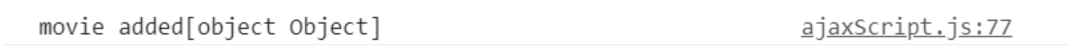
Add Film to Film Database

555	hello	2020
testing	testing	testing phase

Send as JSON

Figure 13 shows adding a new film to database

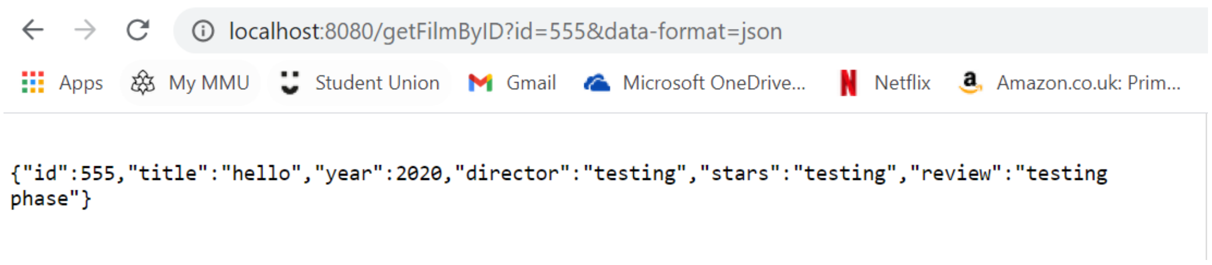
Figure 14 shows that when a film is successfully added into the database it shows the following message in the console.



```
movie added[object Object]                     ajaxScript.js:77
```

Figure 14 shows the output in console when film is added

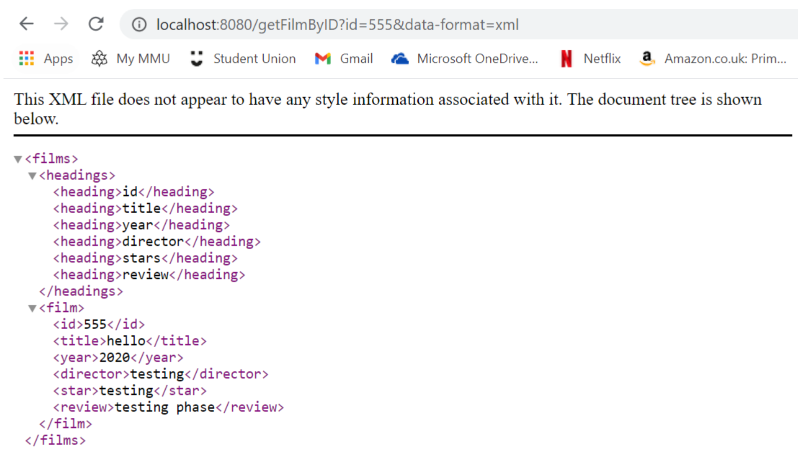
Figure 15 shows when we try to search for the same film using id it shows that the film has been added into the database successfully and it outputs as a JSON format.



```
{"id":555,"title":"hello","year":2020,"director":"testing","stars":"testing","review":"testing
phase"}
```

Figure 15 shows when the same film with the id just added is searched in JSON format

Figure 16 shows that when the same film is searched in XML format it gives the output which means the film was successfully added in XML as well.

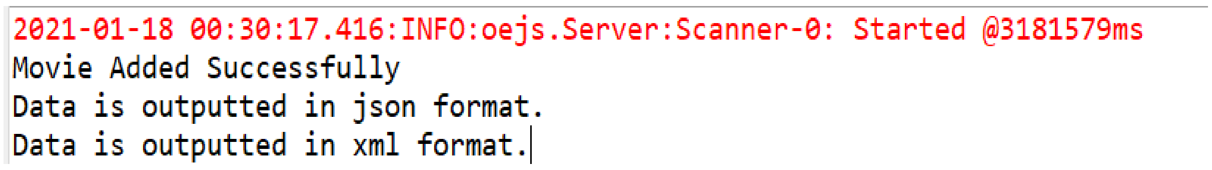


This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<films>
  ▼<headings>
     <heading>id</heading>
     <heading>title</heading>
     <heading>year</heading>
     <heading>director</heading>
     <heading>stars</heading>
     <heading>review</heading>
   </headings>
  ▼<film>
     <id>555</id>
     <title>hello</title>
     <year>2020</year>
     <director>testing</director>
     <star>testing</star>
     <review>testing phase</review>
   </film>
 </films>
```

Figure 16 shows the same film searched for in XML format.

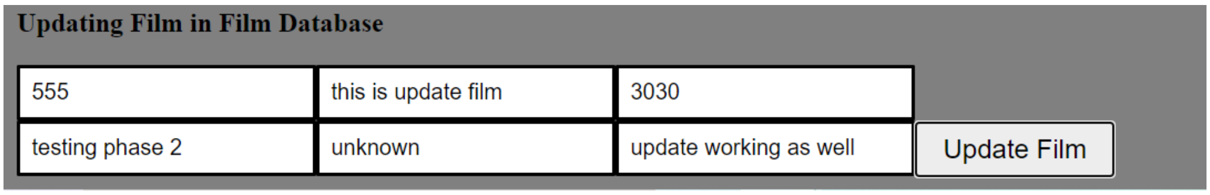
When you add a film in the database it gives this success message in eclipse as well that the film has been added.



```
2021-01-18 00:30:17.416:INFO:oejs.Server:Scanner-0: Started @3181579ms
Movie Added Successfully
Data is outputted in json format.
Data is outputted in xml format.
```

Update Film

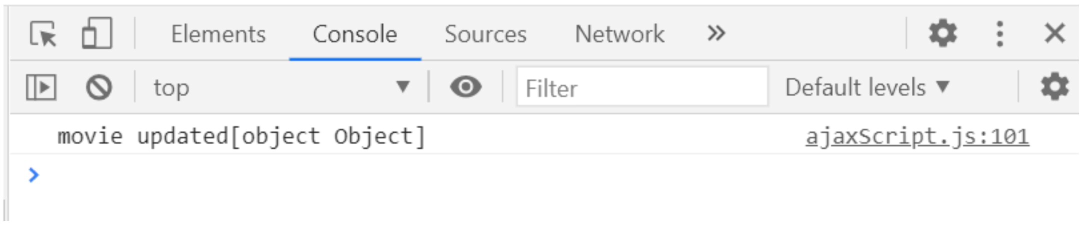
In this part I will try updating the same film by using the same ID. In this part the ID is supposed to stay the same because we are first getting the film by ID and then updating it using the below method. Fill out the fields as shown below or you may wish to change them and after done with this we press the Update Film button.


Updating Film in Film Database

555	this is update film	3030	
testing phase 2	unknown	update working as well	Update Film

Figure 17 shows when we update the same film using the same ID

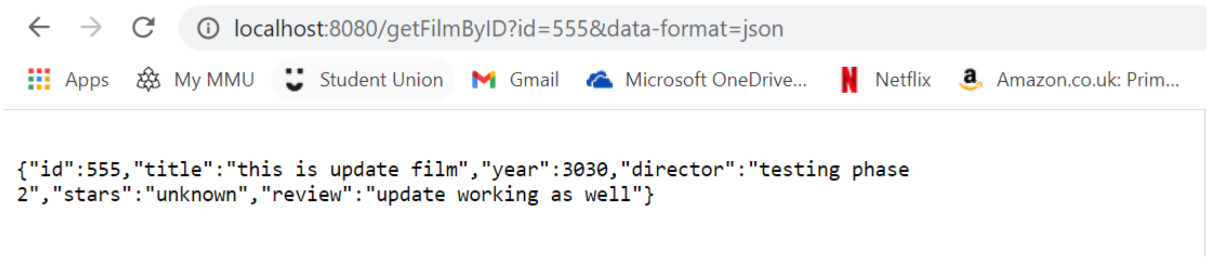
Once the button is pressed the console shows the following message that the film has been updated successfully into the database.



```
movie updated[object Object]                    ajaxScript.js:101
```

Figure 18 shows the console message for updating film

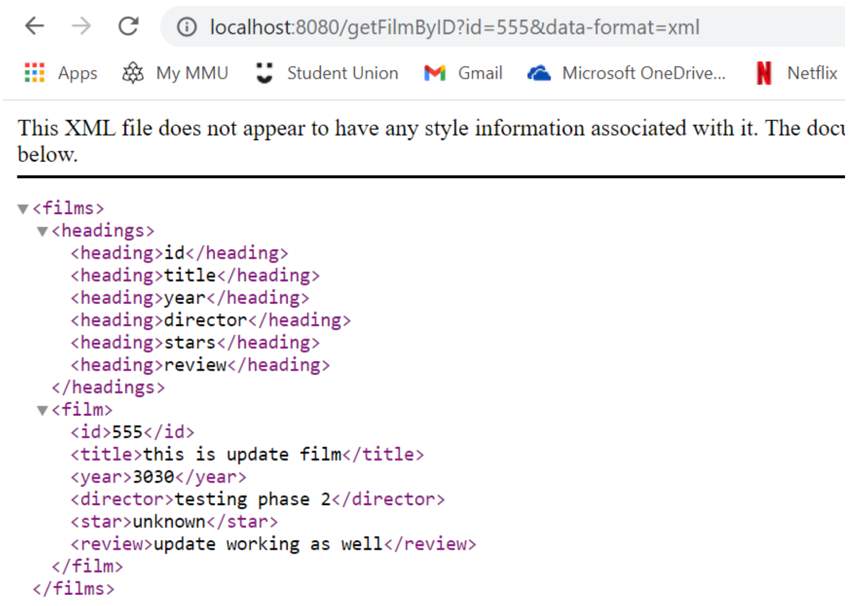
To test whether the update worked successfully we search for the same ID we used in the add film part and as we can see in the below figure that the film has been updated as we filled out the input fields in figure 17 and this means that the update works fine as well.



```
{"id":555,"title":"this is update film","year":3030,"director":"testing phase
2","stars":"unknown","review":"update working as well"}
```

Figure 19 shows update working in JSON format

For the same film when we search in the format of XML it gives the output as shown in the below figure. Moreover, you can search for this film by using the title as well that is “this is update film” and it will give the same output as shown below in XML format



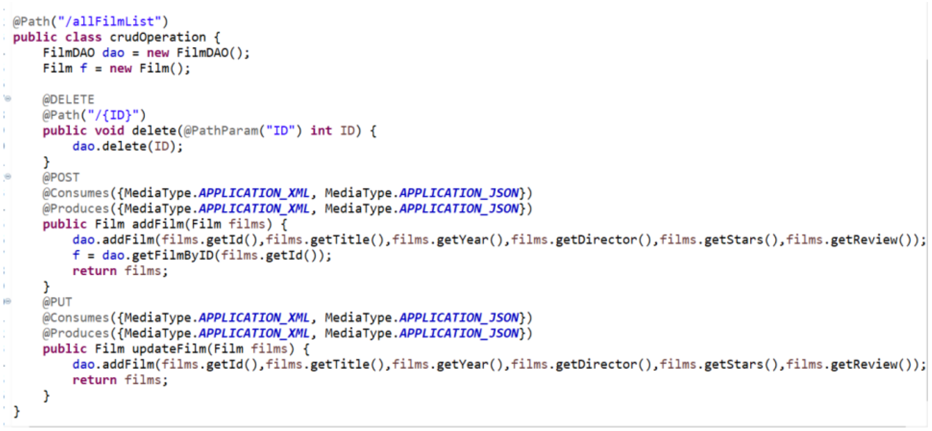
This XML file does not appear to have any style information associated with it. The docu
below.

```
▼<films>
  ▼<headings>
      <heading>id</heading>
      <heading>title</heading>
      <heading>year</heading>
      <heading>director</heading>
      <heading>stars</heading>
      <heading>review</heading>
    </headings>
  ▼<film>
      <id>555</id>
      <title>this is update film</title>
      <year>3030</year>
      <director>testing phase 2</director>
      <star>unknown</star>
      <review>update working as well</review>
    </film>
  </films>
```

Figure 20 shows update working in XML format

Rest jersey

I used the Jersey RESTful web service for doing my CRUD operations. Below is the code I implemented to achieve CRUD operations successfully running. The path was same for every crud operation and we just had to change the method that was DELETE, POST for adding film and PUT for updating film in database. This made me not go into creating different servlets for each crud operation and instead I did it all in one place. We also used MediaType so that it knows that its json or xml. In the PUT code we used get to get all the details about the film first and then update them accordingly. Updating and Adding a film to database is already shown above.

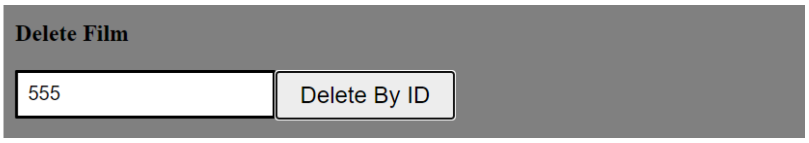


```java
@Path("/allFilmList")
public class crudOperation {
    FilmDAO dao = new FilmDAO();
    Film f = new Film();

    @DELETE
    @Path("/{ID}")
    public void delete(@PathParam("ID") int ID) {
        dao.delete(ID);
    }
    @POST
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public Film addFilm(Film films) {
        dao.addFilm(films.getId(),films.getTitle(),films.getYear(),films.getDirector(),films.getStars(),films.getReview());
        f = dao.getFilmByID(films.getId());
        return films;
    }
    @PUT
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public Film updateFilm(Film films) {
        dao.addFilm(films.getId(),films.getTitle(),films.getYear(),films.getDirector(),films.getStars(),films.getReview());
        return films;
    }
}
```

Figure 21 shows RESTful Jersey CRUD code

Delete Button Working:

Now for this part we will try deleting the same film that we added in the images above. We can only delete film by the specific ID. So the figure below shows that we added the ID into the field and we pressed the delete button and it successfully deletes the film.

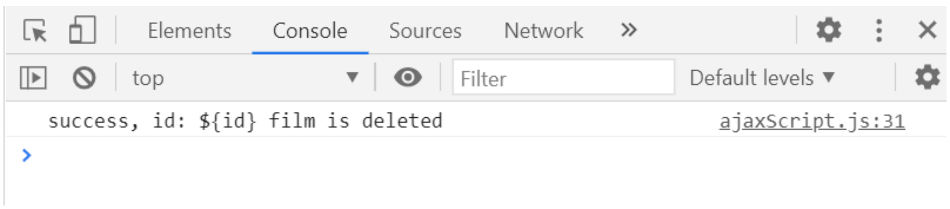


Delete Film

555

Figure 22 shows the delete film interface

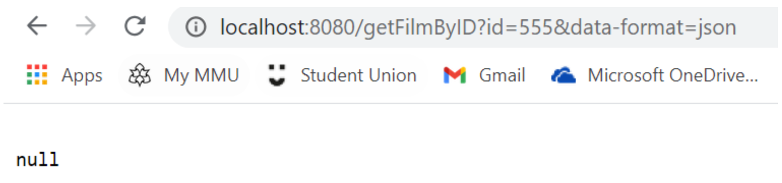
The below console output shows that the Film with that specific id was successfully deleted from the database and the console output was success.



```
success, id: ${id} film is deleted                    ajaxScript.js:31
```

Figure 23 shows the console output when delete operation is placed

To further test if it actually did delete it from the database we search for that exact film by using the ID above and when we search it shows as null which means that there is no data for that film with that specific id which in turn means that out delete button is working fine.



localhost:8080/getFilmByID?id=555&data-format=json

```
null
```

Figure 24 shows when the same id is searched but the result come as null

Coding practices

I used different good coding practices like I commented most of my code that I thought would help in future update if ever needed on this project. Indentation is a good coding practice as it allows the developer to easily figure out where the specific code ends using the curly brackets and I indented my code accordingly, so it was easier for me to understand where my code ends for every operation that I did in my project. I also named my folders and file correctly and any other developer who tries to look into my code will find everything well placed and can easily find using the names. My servlets are named according to what operation they are doing as getting all films is named as “getAllFilms” and getting film by id is named as “getFilmById” and the getting film by name is named as “getFilmByTitle”. This is a good coding practice because you know where your things are and what those things do. I made a separate folder of my ajax script and named it as JavaScript and inside a file named as ajaxScript.js that had all the stuff needed for AJAX. Then for the CRUD operation I made a separate package in the main java folder and named it as crud and inside there was a crudOperations class that had all the code for doing CRUD operations. My DAO was in db package named as FilmDAO. Every aspect of my code was separate as I used the MVC architecture that helped me a lot. All of my code was precise and easy to update. I also refactored the code at some points so that I don’t need to write up the code again and I used constructors as well to keep the code clean.

Conclusion

The report above has given a summary of what was done in the project that I created for web services assignment. There are screenshot evidence of my code working above. There is screenshot evidence of data added to the database as well when we search for that specific film by id or title. Finally, there is a screenshot of my project deployed into the Google app engine as well. Other Screenshots include my code working for adding film and updating and searching films by id or title and deleting film by id as well. The report also shows about good coding practices that I used while coding and the architecture I used in my project including DAO design pattern and MVC architecture.