Developing, Deploying, Publish and Finding your own Web Service

REPORT 1CWK100 – 18048968

Haris Rafiq
MANCHESTER METROPOLITAN UNIVERSITY | ENTERPRISE PROGRAMMING

CONTENTS

Criteria Met

Criteria	Completed
Criteria 1 Access to the data using simple http web service calls (10 marks)	Full implementation has been completed, CRUD implemented in a separate class, with all the methods working including persistence storage.
Criteria 2 Options to return the data in text, json (the default) or xml (10 marks)	Made use of libraries such as JAXB to format the given data to XML and GSON used to format the data to JSON. Proper separators (#) used to output the data as TEXT.
Criteria3 Google App Engine or Microsoft Azure to implement the application on a remote cloud-based server. (20 marks)	Google App Engine implemented, and all CRUD functionality fully tested on cloud database.
Criteria 4 A WSDL description of the interface to the web service (5 marks)	Full WSDL interface implemented and documented.
Criteria 5 Access to the data using REST type interaction (10 marks)	Full implementation working. Data transactions sent in correct RESTful way for all CRUD operations. These operations were tested using Postman and then implemented on front-end using AJAX requests.
Criteria 6 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 (15 marks)	JQUERY User Interface was used to create a responsive front-end interface for the user. JQUERY was mainly used for AJAX event handers and calls such as PUT for update, DELETE to delete, GET to request data and POST to create films.
Criteria 7 Critical analysis of your work and the techniques you have used. (30 marks)	This is the report with critical analysis for my work and the techniques I have used.

Introduction

Developing, deploying, publishing and finding your own web service is the name of this assignment that we were given within the Software Engineering, Enterprise Programming module. The goal of this assignment was to create an old movies presentation using JAXB and GSON libraries to convert the data, using CRUD functionalities to create, read, update and delete the data and manipulating it using AJAX requests so the user can manipulate the database easily using the implemented jQuery user interface.

I have uploaded the database to google cloud and tested it so whenever any user would want to instantiate the program, they can easily do so by a normal app engine server without having to worry about implementing their own database.

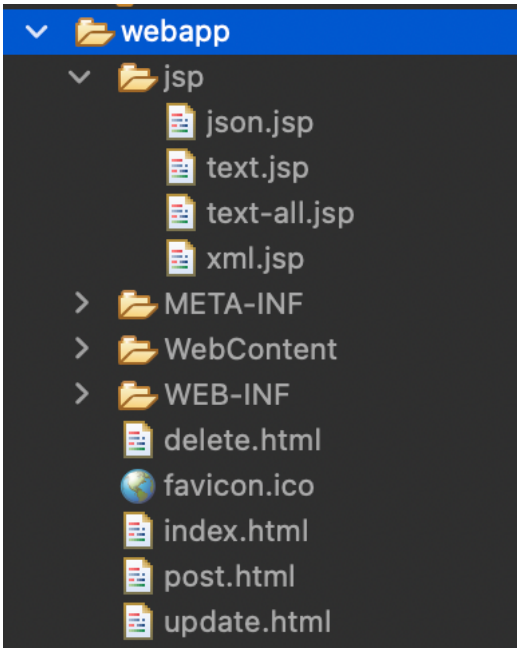
MVC Implementation

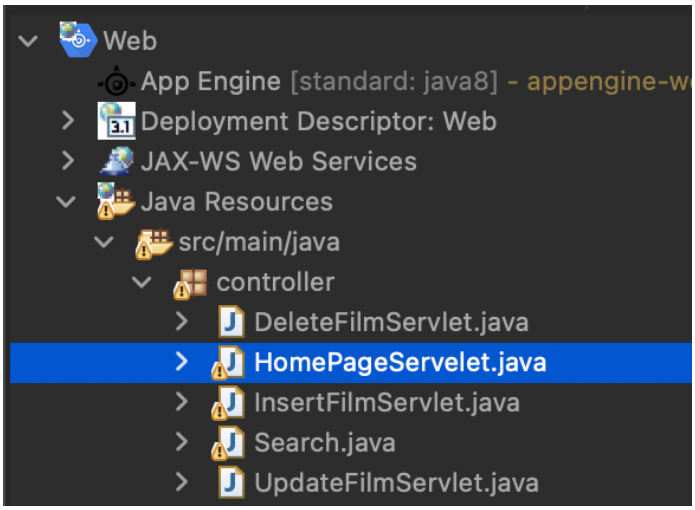
One of the SE techniques this project required us to demonstrate was the use of MVC. Which means within this project there are three main packages that I have used.

The main point of the model is to directly works with the database rather than data processing and user interface. I have used the Model package which has:

the Film class that contains the setters and getters in order to retrieve the specific information from the client. The Film DAO directly connects to the database and contains query functions to retrieve the data according to the functions. FilmInfo is used for the interface and the FilmStore model was used for implementing JAXB and converting the data to XML format.

The view within the MVC architecture has the job to display the user interface and it receives the data that interacts directly with the user. In my program the view would be considered the index.html file which contains code to structure the user interface. The JSP files such as text, xml and json.jsp which contain attributes so the servlet can manipulate them to output the data required from the user. I have also included a stylesheet to clean up the structure and add some colours to the view.



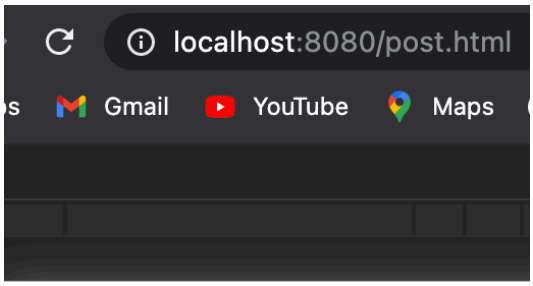


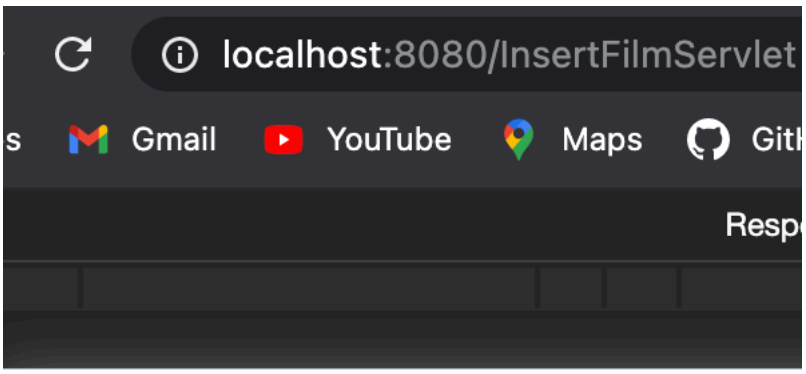
The controller is one of the most essential and important parts of MVC, within the controller we process the data before making changes to it within the model and then we output it from the request we get from the View. Within the Controller package we have files such as the HomePageServlet which is used to get all of the films from the database using the data-format chosen via a drobox provided in the front-end view. The Search servlet in controller is used to get individually chosen movies either via ID or name, again this can be searched by choosing the format you want use. There are also post requests within three servlets which get use a function from FilmDAO each and create, update or delete the film.

Whilst working on the MVC I have realised that it has been easier to debug and finding errors within my program because there are multiple layers of codes. However, it hasn't been easy as the MVC architecture has strict rules on methods.

Following examples of HTTP:

Post film non-RESTful using servlets





Create Film

Film ID: 12345

Film Title: test

Film Year: 1999

Film Director: Assignment

Film Stars: Admin

Film Review: Might work?

Submit Create a new movie

Successfully Inserted

```
1  use filmsDB;
2
3  select * from films where id = 12345;
4
5
```

100% 38:3

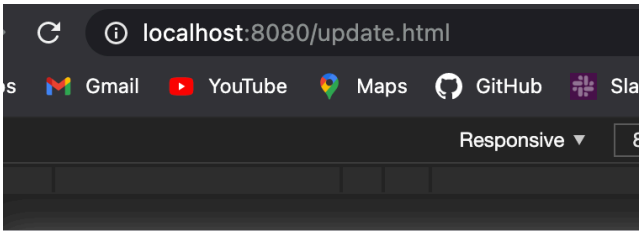
Result Grid Filter Rows: Search Ex

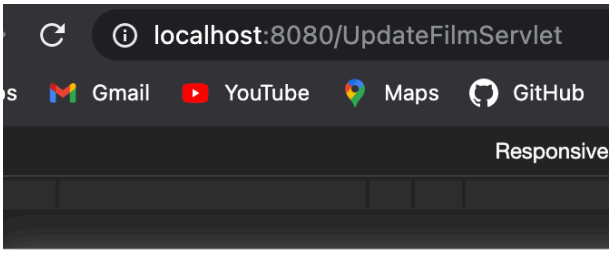
id	title	year	director	stars	review
12345	test	1999	Assignment	Admin	Might work?

You can see that once on the create film servlet you submit a form for the film you it will give you a message telling you the movie was created with the servlet function showed below. Later on, you can see that by checking on the database on MySQL you can see that the movie created is correct.

```
        try {

            FilmDAO dao = new FilmDAO();

    String filmId = request.getParameter("fId1");
    String filmName = request.getParameter("fTitle1");
    String filmYear = request.getParameter("fYear1");
    String filmDirector = request.getParameter("fDirector1");
    String filmStars = request.getParameter("fStars1");
    String filmReview = request.getParameter("fReview1");

    int fId= Integer.parseInt(filmId);
    int fYear= Integer.parseInt(filmYear);

    dao.postCreateFilm(fId, filmName, fYear, filmDirector, filmStars, filmReview);

    PrintWriter out = response.getWriter();
    out.println("Successfully Inserted");
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    doGet(request, response);
}
```

Update film non-RESTful using servlets





Update Film: choose ID to update the film

Film ID: 12345

Film Title: test 2 : the sequel

Film Year: 2021

Film Director: Nick

Film Stars: Kaleem

Film Review: This assignment deserve

Submit Update Movie

Successfully Updated

```
1  use filmsDB;
2
3  select * from films where id = 12345;
4
5
```

100% 38:3

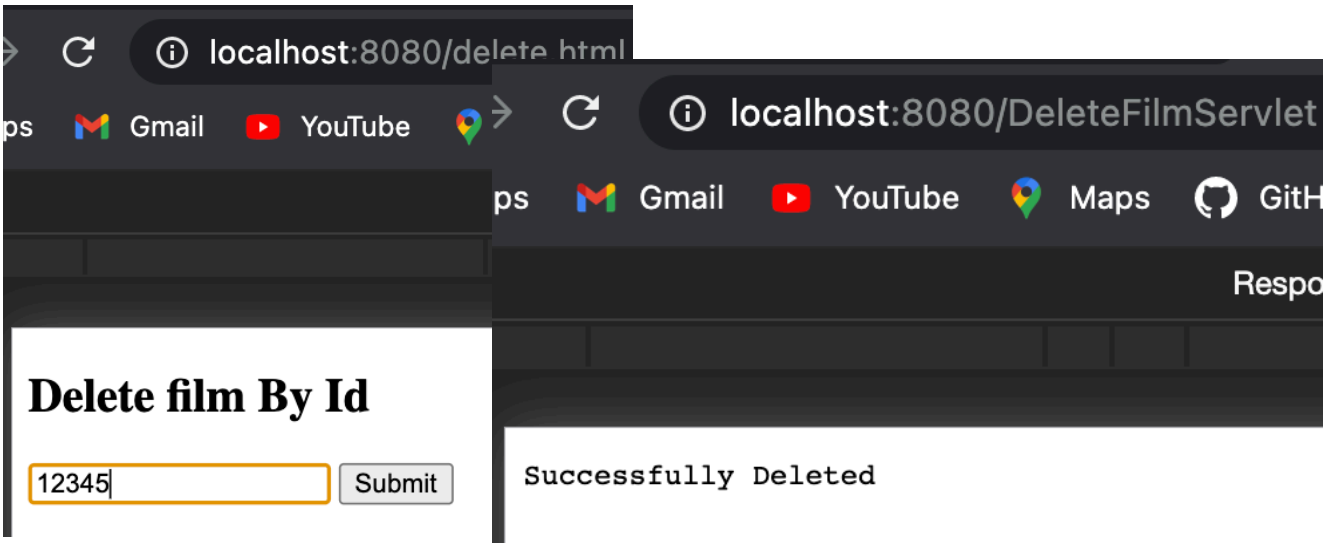
Result Grid | Filter Rows: Search | Export:

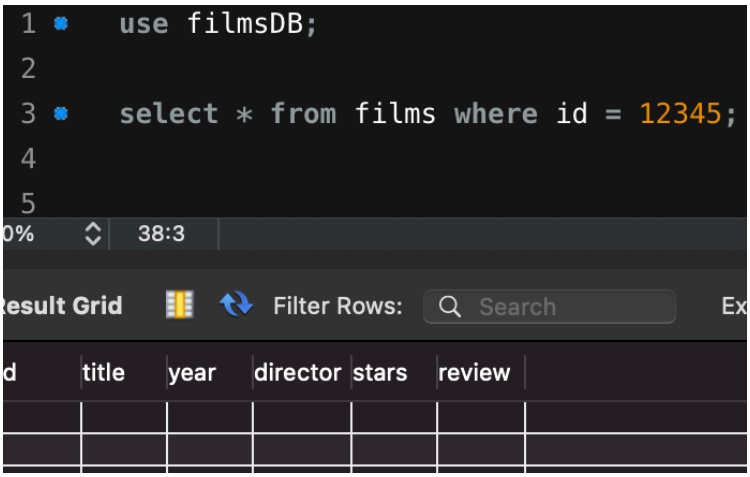
id	title	year	director	stars	review
12345	test 2 : the sequel	2021	Nick	Kaleem	This assignment deserves 100%

To update the movie, it is the similar concept as create, however when we get the update function, we have to keep in mind that the id parameter we give will be used to update the movie of the chosen id. You can see how I updated the movie we created earlier with the create functionality and then I used MySQL to ensure that it has in fact been created successfully.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) th
    // TODO Auto-generated method stub

        try {

            FilmDAO dao = new FilmDAO();

    String filmId = request.getParameter("uId");
    String filmName = request.getParameter("uTitle");
    String filmYear = request.getParameter("uYear");
    String filmDirector = request.getParameter("uDirector");
    String filmStars = request.getParameter("uStars");
    String filmReview = request.getParameter("uReview");

    int fId= Integer.parseInt(filmId);
    int fYear= Integer.parseInt(filmYear);

    dao.updateFilm(fId, filmName, fYear, filmDirector, filmStars, filmReview);

    PrintWriter out = response.getWriter();
    out.println("Successfully Updated");
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    doGet(request, response);
}
```

Delete film non-RESTful using servlets:



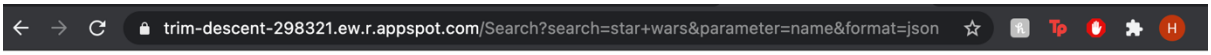


As you can see from the screenshots the delete film servlet lets me delete the movie by the id I have put in the form. Then it will give me a response confirming the delete is completed. Later on, I have checked if the movie we created has been deleted successfully and as you can see from the screenshot from MySQL it has.

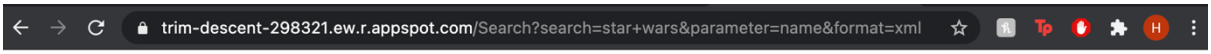
```
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
        // TODO Auto-generated method stub
        try {

            FilmDAO dao = new FilmDAO();

            String delete = request.getParameter("deleteFilm");

            int deleteId = Integer.parseInt(delete);

            dao.deleteFilmById(deleteId);

        PrintWriter out = response.getWriter();
        out.println("Successfully Deleted");
            }
            catch (Exception e) {
                e.printStackTrace();
            }


        doGet(request, response);
    }

}
```

GET/READ film by name and id with different formats servlet:

This image will come across if you press the grey “Search button” on main screen which will take you to a new page with the format you chose whether it is XML, JSON or TEXT.

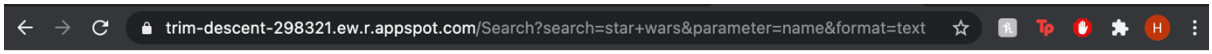


{"id":10954,"title":"STAR WARS","year":1978,"director":"GEORGE LUCAS","stars":"MARK HAMILL, HARRISON FORD","review":"The original, the one that started the whole ball rolling, and if you dont know the plot leave this website immediately! Essentially an old-fashion Saturday morning pictures episode in space, with the good guys taking on and defeating the baddies, led by the menacing, wheezing Darth Vader, using his Death Star to destroy entire planets. Also features Carrie Fisher with a really silly hairstyle."}



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

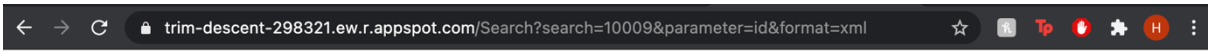

```
▼<film>
   <id>10954</id>
   <title>STAR WARS</title>
   <year>1978</year>
   <director>GEORGE LUCAS</director>
   <star>MARK HAMILL, HARRISON FORD</star>
   <review>The original, the one that started the whole ball rolling, and if you dont know the plot leave this website immediately!
   Essentially an old-fashion Saturday morning pictures episode in space, with the good guys taking on and defeating the baddies,
   led by the menacing, wheezing Darth Vader, using his Death Star to destroy entire planets. Also features Carrie Fisher with a
   really silly hairstyle.</review>
 </film>
```

10954#STAR WARS#1978#GEORGE LUCAS#MARK HAMILL, HARRISON FORD#The original, the one that started the whole ball rolling, and if you dont know the plot leave this website immediately! Essentially an old-fashion Saturday morning pictures episode in space, with the good guys taking on and defeating the baddies, led by the menacing, wheezing Darth Vader, using his Death Star to destroy entire planets. Also features Carrie Fisher with a really silly hairstyle.

As you can see the format chosen was JSON, which then I used GSON library to convert the java data object to JSON that was the first image, the second one shows how I converted data to XML using JAXB and the last one is plain text using # separators. You can clearly see if I used the id to search the movies in the link by checking the parameter or if I used the name as well as the format I chosen. The examples below are the same as above but will show the film searched by the id.

{"id":10009,"title":"A LEAGUE OF THEIR OWN","year":1992,"director":"PENNY MARSHALL","stars":"GEENA DAVIS, TOM HANKS","review":"Loosely based on the true story of the all-womens baseball league which was formed during the war while all the male players were fighting overseas. Concentrating on one team, the Rockford Peaches whose members include Davis, Madonna, and Lori Petty. Broken-down coach Tom Hanks has to knock them into some sort of shape."}



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


```
▼<film>
   <id>10009</id>
   <title>A LEAGUE OF THEIR OWN</title>
   <year>1992</year>
   <director>PENNY MARSHALL</director>
   <star>GEENA DAVIS, TOM HANKS</star>
   <review>Loosely based on the true story of the all-womens baseball league which was formed during the war while all the male
   players were fighting overseas. Concentrating on one team, the Rockford Peaches whose members include Davis, Madonna, and Lori
   Petty. Broken-down coach Tom Hanks has to knock them into some sort of shape.</review>
 </film>
```


10009#A LEAGUE OF THEIR OWN#1992#PENNY MARSHALL#GEENA DAVIS, TOM HANKS#Loosely based on the true story of the all-womens baseball league which was formed during the war while all the male players were fighting overseas. Concentrating on one team, the Rockford Peaches whose members include Davis, Madonna, and Lori Petty. Broken-down coach Tom Hanks has to knock them into some sort of shape.

GET/READ all films servlet:

I have managed to get all films with different formats, like before I used JAXB and GSON to convert the films into JSON format and XML as well as separators for all films in text. Below follow three figures of all films in different formats.

[Film [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.], Film [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.], Film [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.], Film [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.], Film [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 personal revenge. After forger Mark (Chow) is crippled and double crossed by an associate, cop Kit (Cheung) risks his badge to help him. Violent and gripping. Also available as a boxed set with a supplementary documentary and booklet.], Film [id=10006, title=A BRONX TALE, year=1993, director=ROBERT DE NIRO, stars=ROBERT DE NIRO, CHAZZ PALMINTERI, review=De Niros directorial debut, taken from co-star Palminteris own stage play. De Niro leads a cast of virtual unknown as a hard-working Bronx bus driver, alarmed at his sons fascination with the local underworld figures. But the lad is involved even deeper than he knows and soon hell have to confront the gangsters in an effort to win back his sons respect.], Film [id=10007, title=A CHILD LOST FOREVER, year=1993, director=CLAUDIA WEILL, stars=BEVERLY DANGELO, WILL PATTON, review=Another of the Odyssey labels seemingly endless supply of true-life weepies, this one involves a young woman (Beverly DAngelo from High Spirits) reluctantly giving up her child for adoption. Years later she discovers that it died in mysterious circumstances at the age of only three and decides to investigate.], Film [id=10008, title=A FEW GOOD MEN, year=1992, director=ROB REINER, stars=TOM CRUISE, JACK NICHOLSON, review=After the mysterious death of a young Marine in the barracks, a young navy Lawyer (Cruise) is called in to investigate. The further he delves into the mystery the more he is thwarted by the fearsome and unyielding Colonel Nathan Jessup (Nicholson), a man who knows the truth and is determined to conceal it....], Film [id=10009, title=A LEAGUE OF THEIR OWN, year=1992, director=PENNY MARSHALL, stars=GEENA DAVIS, TOM HANKS, review=Loosely based on the true story of the all-womens baseball league which was formed during the war while all the male players were fighting overseas. Concentrating on one team, the Rockford Peaches whose members include Davis, Madonna, and Lori Petty. Broken-down coach Tom Hanks has to knock them into some sort of shape.], Film [id=10010, title=A LITTLE PRINCESS, year=1995, director=ALFONSO CUARON, stars=LIESEL MATTHEWS, ELEANOR BRON, review=Enchanting family story, with a young shy girl being sent to an austere boarding school in New York after her father leaves to go to war. Initially she has a terrible time, being regarded as an outsider, but slowly realises her own self-worth and becomes liked and admired by both the staff and the pupils. Eleanor Bron plays the strict governess.], Film [id=10011, title=A LOW DOWN DIRTY SHAME, year=1994, director=KEENAN IVORY WAYANS, stars=KEENAN IVORY WAYANS, JADA PINKETT, review=Cool and sleek comedy thriller directed by and starring IN LIVING COLORs Wayans. Hes the Shame of the name, an ex-cop whose personal vendetta with a drug lord got him thrown off the force. Now a private eye hes given an opportunity to get back at the man who wrecked his career. Enjoyable fun with some gripping shoot-outs. Mrs Will Smith, Jada Pinkett co-stars.], Film [id=10012, title=A NIGHTMARE ON ELM ST. 4, year=1988, director=RENNY HARLIN, stars=ROBERT ENGLUND, RODNEY

Data Access Object (DAO) Pattern

The Data Access is used so that the client persistence layer of the software can be separated to the admin or the business layer. I used DAO patterns to ensure the complex code which involves CRUD operations would be hidden from the user.

```
12 public class Film {
13     int id;
14     String title;
15     int year;
16     String director;
17     String stars;
18     String review;
19 
20     public Film() {
21         super();
22     }
23 
24     public Film(int id, String title, int year, String director, String stars, String review) {
25         super();
26         this.id = id;
27         this.title = title;
28         this.year = year;
29         this.director = director;
30         this.stars = stars;
31         this.review = review;
32     }
33 
34     public int getId() {
35         return id;
36     }
37     public void setId(int id) {
38         this.id = id;
39     }
```

To start off I created packages following the MVC architecture. Within the model I defined the Film class which was used to implement as a domain model. In Film you can see that there are setters and getters that just store the information such as: id, title, year, director, stars, review; about specific movies within them.

```
public interface FilmInfo {
     public void postCreateFilm (int id, String title, int year, String director, String stars, String review);
     public void updateFilm (int id, String title, int year, String director, String stars, String review);
     public void deleteFilmById(int id);
     public Film getFilmByName(String name);
     public Collection getAllFilms();
    }
```

The Film Info interface was created to define an abstract API to perform Create, Read, Update and Delete operations. This class clearly demonstrates how the Film class is completely an independent domain compared to this persistence layer.

The next step was to connect the database. I initially tested the code on a mudfoot local server, however later on changed it to a cloud-based server where the database is later store.

```
public Film getFilmByName(String name){

            openConnection();
            oneFilm=null;
            // Create select statement and execute it
            try{
                String selectSQL = "select * from films where title= " + "'" + name + "'";
                ResultSet rs1 = stmt.executeQuery(selectSQL);
            // Retrieve the results
                while(rs1.next()){
                    oneFilm = getNextFilm(rs1);
                }

                stmt.close();
                closeConnection();
            } catch(SQLException se) { System.out.println(se); }

           return oneFilm;
       }
```

```java
public void updateFilm(int id, String title, int year, String director, String stars, String review){

     openConnection();
     oneFilm=null;
     // Create select statement and execute it
     try{

         String selectSQL = "UPDATE films\n"
                 + "SET id =" + id + ", title = '" + title + "'" + ", year = " + year + ", director ='"
                 + "WHERE id =" + id +";";

         PreparedStatement update = connection.prepareStatement(selectSQL);

         update.execute();


         stmt.close();
         closeConnection();
     } catch(SQLException se) { System.out.println(se); }

}
```

FilmDAO main purpose is to perform database related operations such as CRUD methods which from the screenshot you can tell I have implemented Create, Read, Update and Delete methods with the appropriate functionalities. There are 2 example screenshots to show one for fetching the data and one for posting it (UpdateFilm).

Within the controller we have the search servlet which lets us process the data and manipulate it in the way we want it to display to the user by retrieving the data from the FilmDAO. It is clearly proven that whilst we extract data by using FilmDAO all of the CRUD operations low level details are hidden from the controller and only called upon request.

```java
PrintWriter pw = response.getWriter();
FilmDAO dao = new FilmDAO();
Film film = new Film();
FilmStore filmStore = new FilmStore();


search = request.getParameter("search");
format = request.getParameter("format");
parameter = request.getParameter("parameter");



if("id".equals(parameter)){

    int id = Integer.parseInt(search);
    film = dao.getFilmByID(id);

}
else if("name".equals(parameter)) {

    film = dao.getFilmByName(search);

}


if(format.equals("json")) {

    Gson gson = new Gson();
    data = gson.toJson(film);
    address = "json";
    request.setAttribute("json", data);
    response.setContentType("application/json");


    }
else if(format.equals("xml")){

    request.setAttribute("film", film);
    response.setContentType("application/xml");
    address = "xml";
```

Within the screenshot you can see how we would use the Search servlet to process the data and how objects are persisted from FilmDAO.

GSON & JAXB (Formatting data)

This project required me to demonstrate if I am able to convert the default data given as an XML format and display it for the user. To complete this task, I made use of JAXB which provides a convenient way to convert or marshal java objects into XML and vice-versa.

The following are the annotations I used from JAXB:

@XmlRootElement: this annotation was used inside the FilmStore as well as Film model, this annotation allows me to specify the name of the root element of the XML using its name attribute in this case the root element was film.
@XmlType: allows us to define the order in which I want to lay out my XML fields, I personally chose to output the data as an order from id, title, year, director, stars, review and I applied this order throughout the project.
@XmlElement: is used when you define the actual xml element/tag you want to define and manipulate, in my case I wanted to work with the individual chosen film tags.

In order to convert the java objects from my HomePageServlet to XML I used Marshalling, which gives me the ability to convert a JAXB derived Java object tree into XML format.

To format java object I started off with JAXBContext object which prevents that any unnecessary space would be interpreted in any significance by accident. The next step for this was to use the marshal method, where we set `Marshaller.JAXB_FORMATTED_OUTPUT` to true so that we can achieve the JAXB output. Finally we will user the marshal method from the Marshaller object to use the filmStore class object and an output file which in my case I used PrintWriter in my servlet to display the XML generated parameters.

```
if(format.equals("json")) {
    Gson gson = new Gson();
    data = gson.toJson(allFilms);
    address = "jsp/json.jsp";
    response.setContentType("application/json");
    request.setAttribute("json", data);
    RequestDispatcher dispatcher =  request.getRequestDispatcher(address);
    dispatcher.forward(request, response);

}
else if(format.equals("xml")) {

     filmStore.setFilmList(allFilms);
        try {

            JAXBContext context = JAXBContext.newInstance(FilmStore.class);
            Marshaller m = context.createMarshaller();
            m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

            // Print to
            m.marshal(filmStore, pw);

            // Write to File

        }
        catch(Exception e){
            System.out.print(e);
        }
```

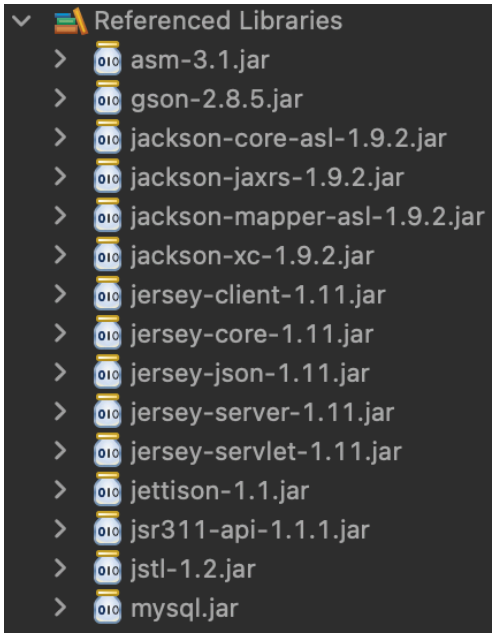
How I implemented GSON to convert JAVA objects and JSON and used JAXB for XML.

To convert Java Objects to JSON format as required I imported and added a build path to the GSON library. GSON simply required me to create a simple object and call in the method toJson() to convert the object which in this case was all films array from FilmDAO.

RESTFul Web Services (JERSEY)

REST is one of the SE techniques that I have learnt in this module and applied within this project. It is another architectural style that I have implemented which follows the HTTP protocol and Web Standards. RESTful Web Services has helped me define CRUD operations within my code such as (POST, GET, PUT, DELETE). RESTFul webservices allows me to define a base URI for its services such as manipulating data formats to JSON, XML or TEXT etc.

The REST client library I have chosen is JERSEY. To use JERSEY, I imported all the provided jar files and then build-path them within my project.



The next step was to define the jersey servlet dispatcher, to do that I edited the code inside my web.xml, mainly the parameter

jersey.config.server.provider.packages which allows us to search which package Jersey will look for the web service classes with properties that are appointed to my resource class within the resources package. The Rest URL is the base link where my application will be placed.

```
<?xml version="1.0" encoding="utf-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
         version="3.1">


           <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

 <servlet>
 <servlet-name>Jersey REST Service</servlet-name>
 <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer
 </servlet-class>

 <init-param>
     <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
     <param-value>true</param-value>
</init-param>


     <init-param>
     <param-name>com.sun.jersey.config.property.packages</param-name>
     <param-value>Resources</param-value>
     </init-param>
 <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
 <servlet-name>Jersey REST Service</servlet-name>
 <url-pattern>/rest/*</url-pattern>
 </servlet-mapping>


</web-app>
```

Within the Resources class there is an @GET annotation which registers the class, using @Produces and @Consumes notations which will then deliver the and send the requested format which can be either JSON or XML. I used @GET so that I can call all of the movies within an array either in xml or json based on format.

```
@Path("/films")

public class resources {


//  FilmDAO dao = new FilmDAO();
    Film film = new Film();

    @GET
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public List<Film> getAllFilms() {


    List<Film> films = new ArrayList<Film>();

    films = FilmDAO.getSingletonCam().getAllFilms();

    return films;
    }
```

The next step was to create a REST client using one of the Jersey Libraries, after done so I created CRUD functions within the FilmDAO class which helped me post data to manipulate by using rest annotations for example:

@PathParam annotation was used to define that the id is inserted as parameter that we get directly from DAO by using Singletons. For example, I needed the path to film Id to delete the film by id as the parameter for the function required and id to delete that specific film from the database. The same situation occurred with updating the film.

```
@POST
@Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public Film postCreateFilm(Film f){

    FilmDAO.getSingletonCam().postCreateFilm(f.getId(), f.getTitle(), f.getYear(), f.getDirector(), f.getStars(), f.getReview());
    //dao.postCreateFilm();

    film = FilmDAO.getSingletonCam().getFilmByID(f.getId());
    //film = dao.getFilmByID(f.getId());

    return film;

}
```

The annotation @POST was used to call REST services via Jersey and to post a new film where the function postCreateFilm was used.

```
@PUT
@Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public Film updateFilm(Film f){

    FilmDAO.getSingletonCam().updateFilm(f.getId(), f.getTitle(), f.getYear(), f.getDirector(), f.getStars(), f.getReview());

    film = FilmDAO.getSingletonCam().getFilmByID(f.getId());

    return film;

}
```

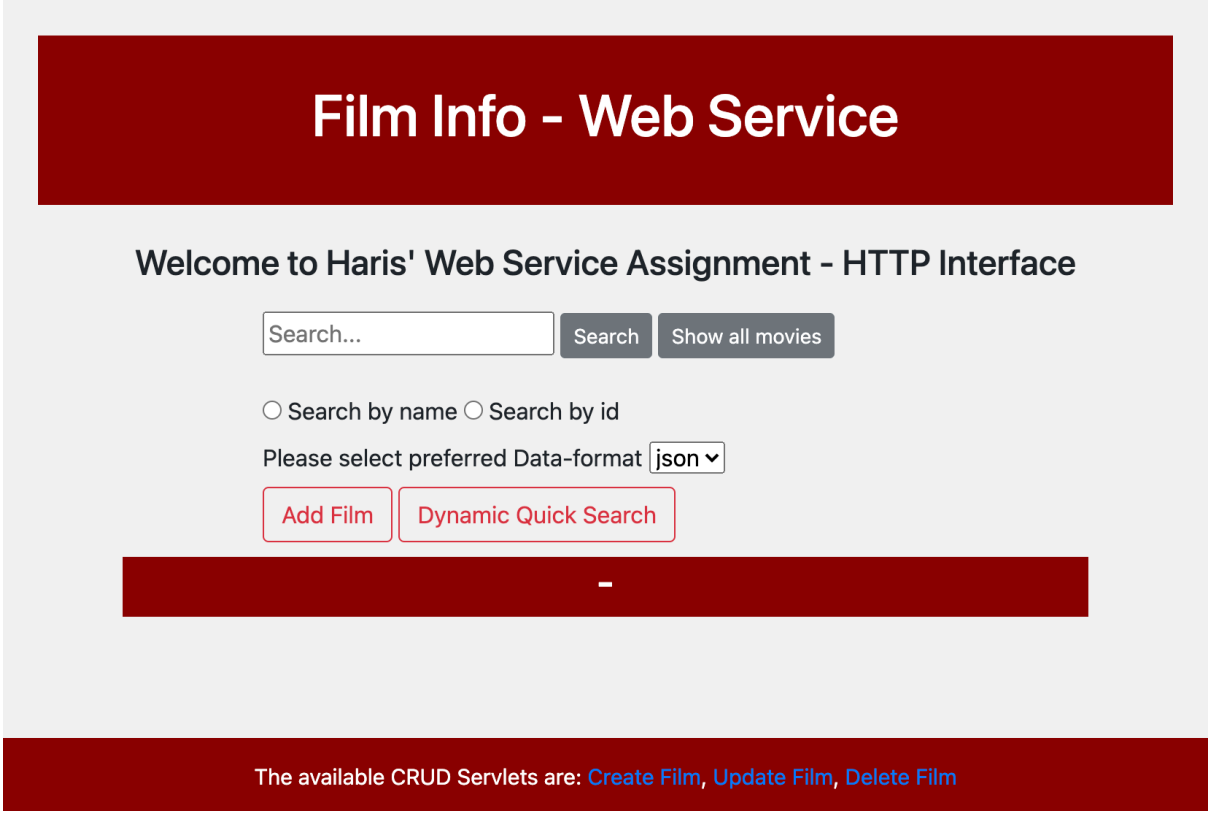
The annotation @PUT was used to call REST services via Jersey and to update a film using the id where the function updateFilm was executed.

```
@DELETE
@Path("{filmId}")
public void deleteFilmById(@PathParam("filmId") int id){

    FilmDAO.getSingletonCam().deleteFilmById(id);
    film = FilmDAO.getSingletonCam().getFilmByID(id);

}
```

The annotation @DELETE was used to call REST services via Jersey and to delete a film using the id specified within the @Path parameter where the function deleteFilmById was executed.

AJAX, jQuery front-end & HTTP

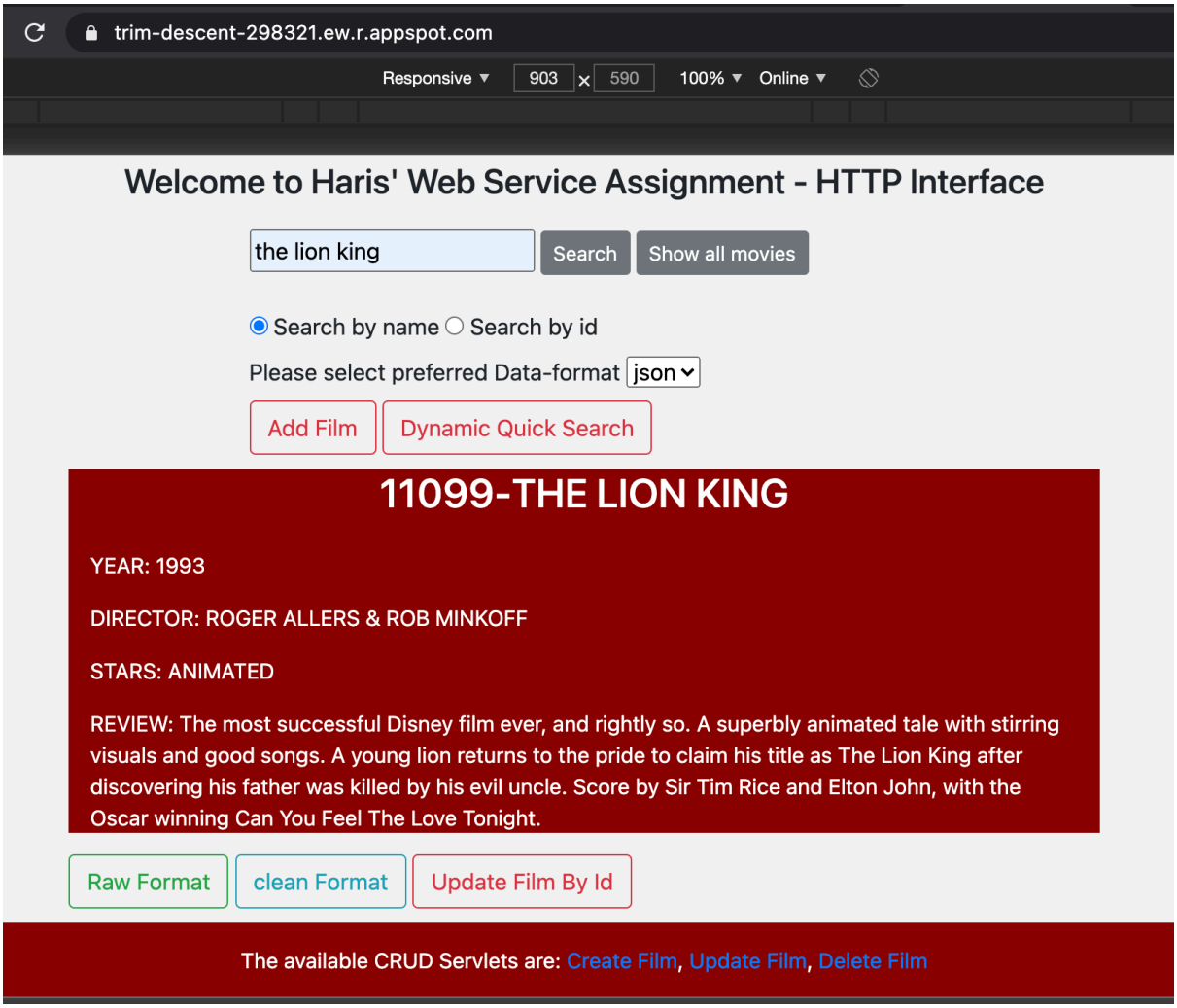
To start designing and creating my front-end I only used the index.html file with no styling at all. The basic functionality worked, and the servlet loaded perfectly. After making sure all of the functionality is working correctly, I created a scripts folder which contained the jQuery and AJAX calls folder and a stylesheet folder which only contained a CSS style folder in order to format my page. I also used bootstrap to style some of my buttons and add toggle effects.



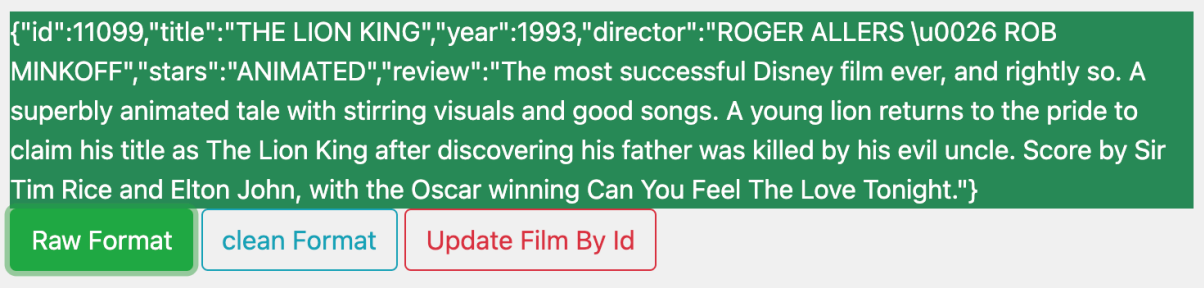
This the simple front-end design, the Search and Show all movies will redirect to either the specific film searched or all of the films' servlet. This will occur once the format has been chosen and if searching a specific movie, you can choose between searching it by id if you remember or simply by the film name. As you can see in the footer there are servlets with POST functionalities, these were created so CRUD functionalities can work with or without RESTful services.

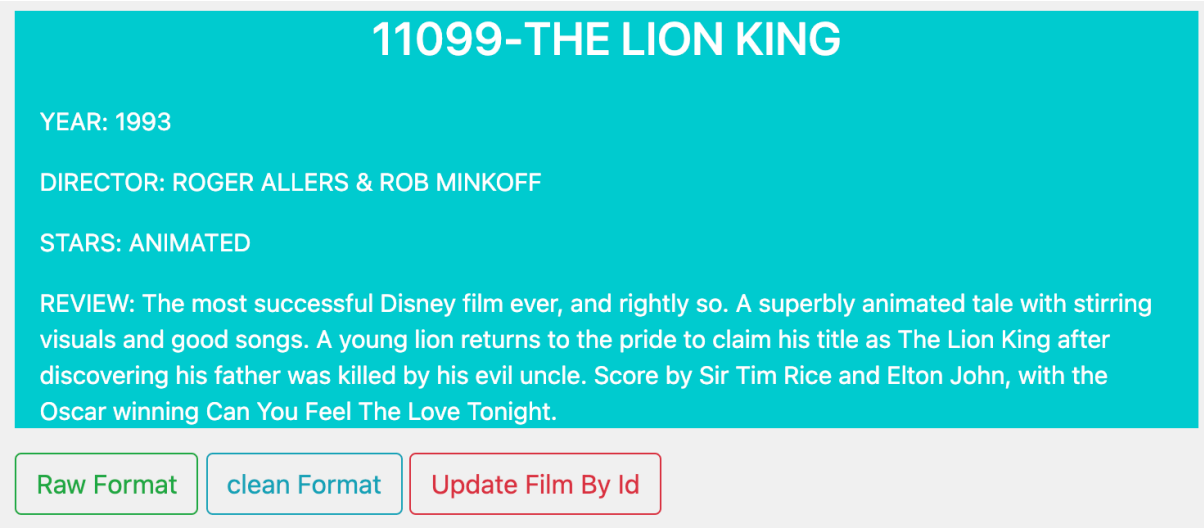
The purpose of ajax is to output data into the page without reloading the screen. As you can see, I have created a Dynamic quick search button. This button is completely different from servlets methods as it will just use AJAX and jQuery to output the film, we search for choosing whichever format we choose. This is done by using GET AJAX methods and some simple jQuery tricks. The reason I decided to use AJAX within my project was because you can conveniently request JSON, XML, and TEXT from a remote server using both HTTP Post and HTTP Get.

To start off with AJAX I used $.get() method, which allowed me to GET movies and load the film data from a server using HTTP GET request. So, if you choose to retrieve data via a dynamic search using ajax you can choose between any formats as they have been implemented and parsed correctly by using AJAX.



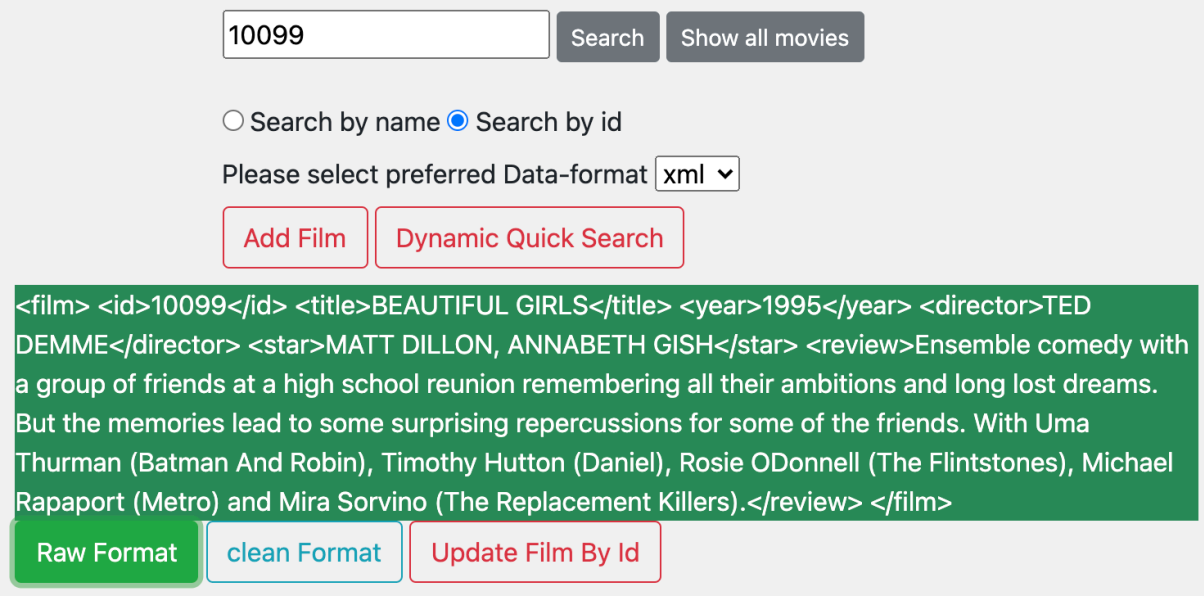
Once you click the Dynamic Quick search with the desired format you can see the movie you chose via ID or Name. The console will also log the format you have chosen to ensure that it is in fact returning the movie via that specific format. After pressing the quick search, three more buttons will appear, one to update the film which will have the value of the film that you chose the search and will also have a delete function, more on that later. You can swap between raw format or clean format examples between xml, json and TEXT below with respective console logs.

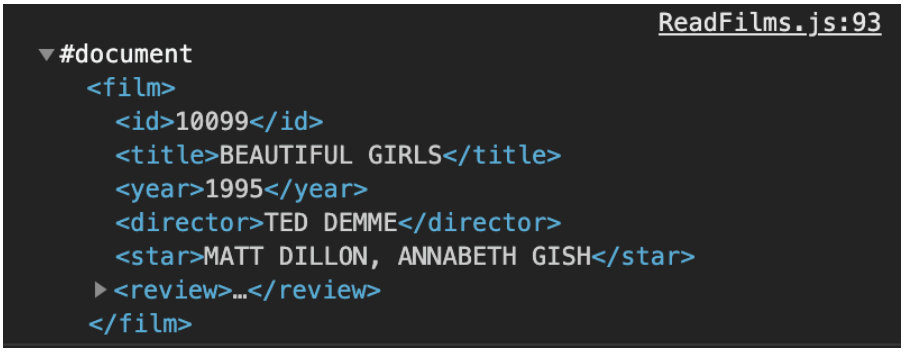


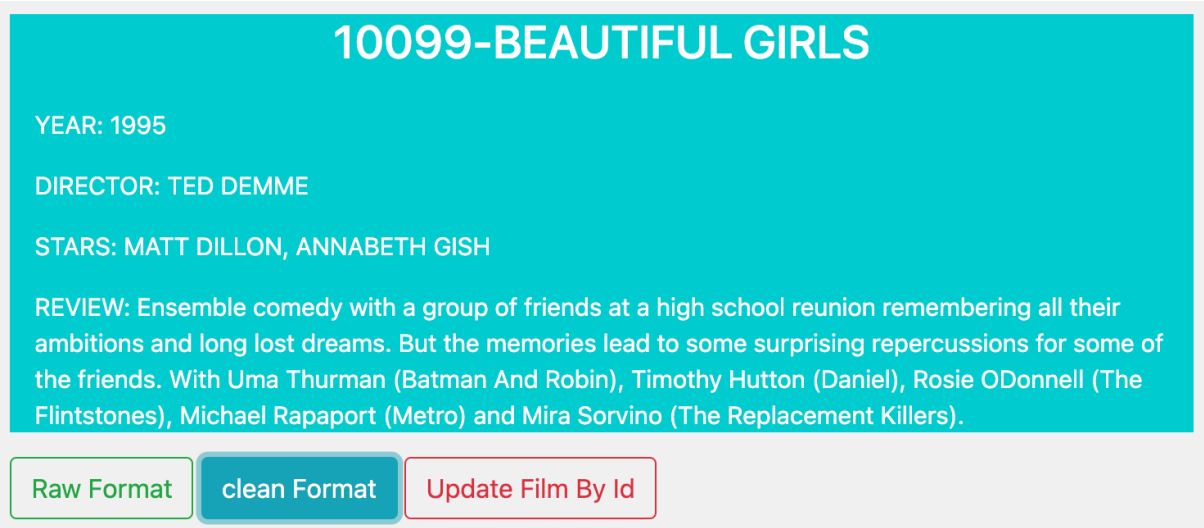


Json as raw and the console log the last figure is switching back to clean format.

JSON.parse() is the method I used to interpret the data to JSON and display it. It took the data parameter from AJAX

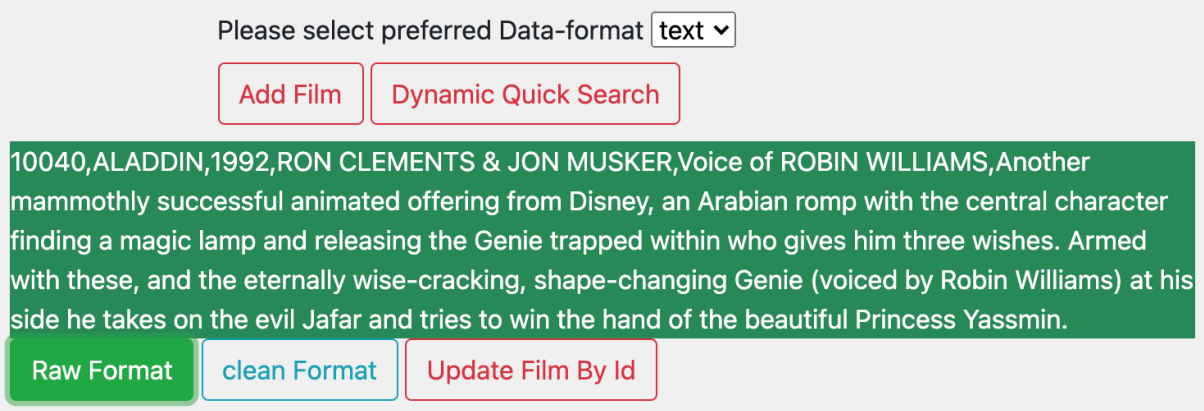


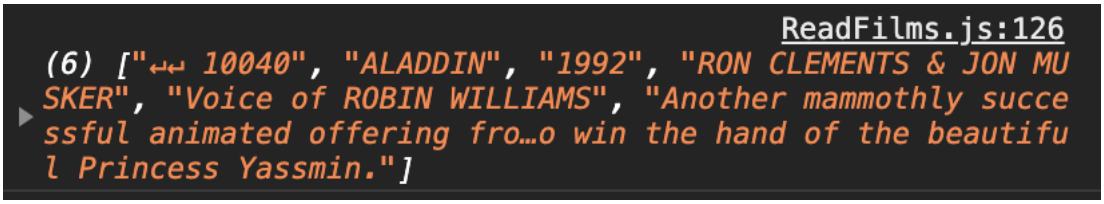


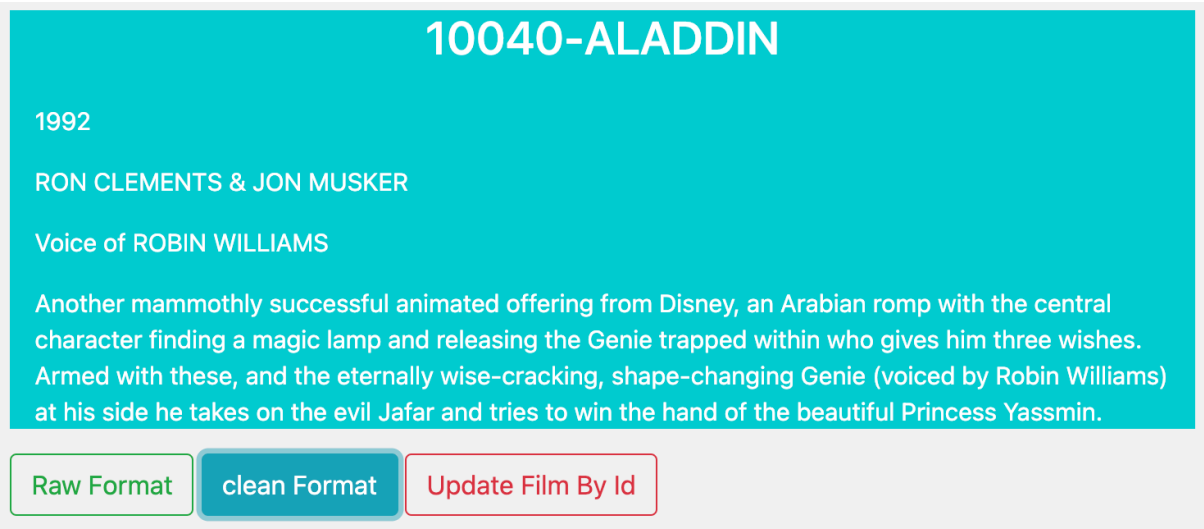


XML as raw and the console log the last figure is switching back to clean format.

XML was slightly more complicated because you had to convert each XML element separately, the first step was to convert the data to xml string, then I had to get each node separately and parse the string to the format using the DOM parser.





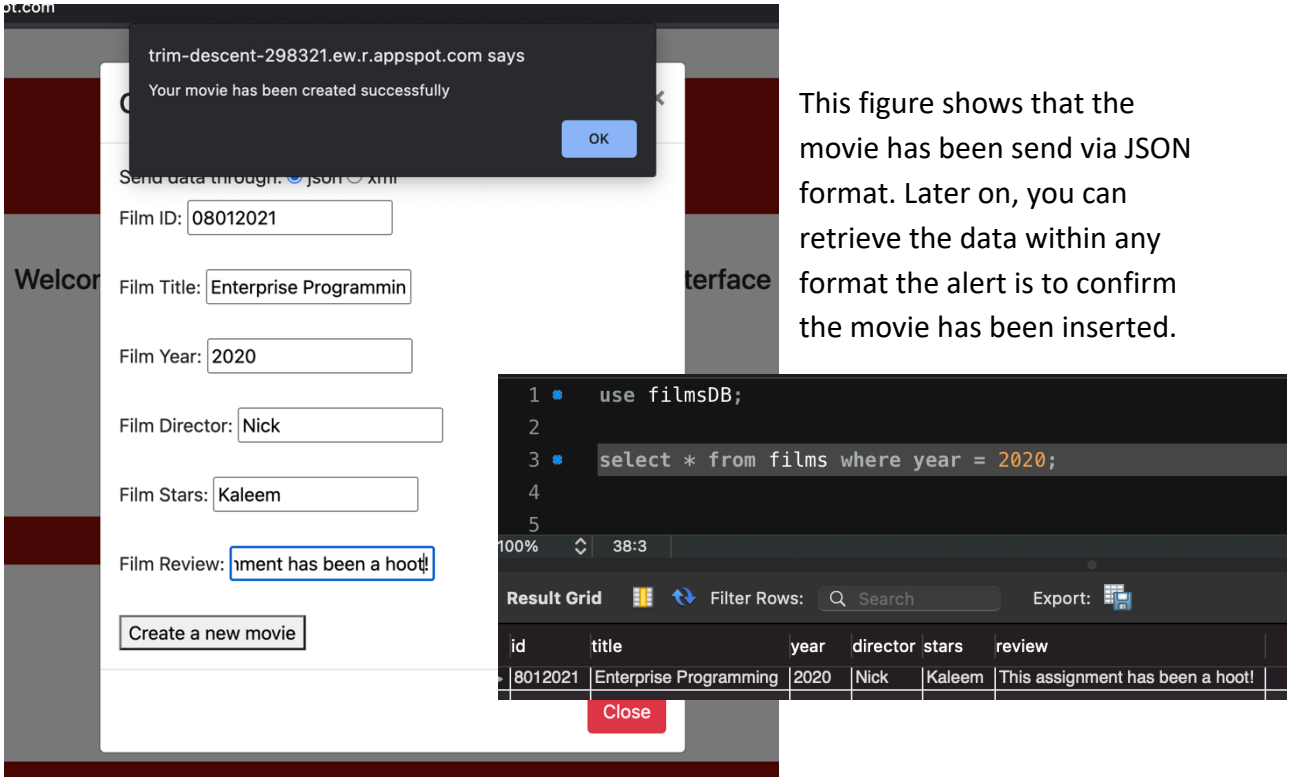


TEXT as raw and the console log the last figure is switching back to clean format.

To output the data as text I used the inbuilt split function which allowed me to use the # separators to split strings within a single array.

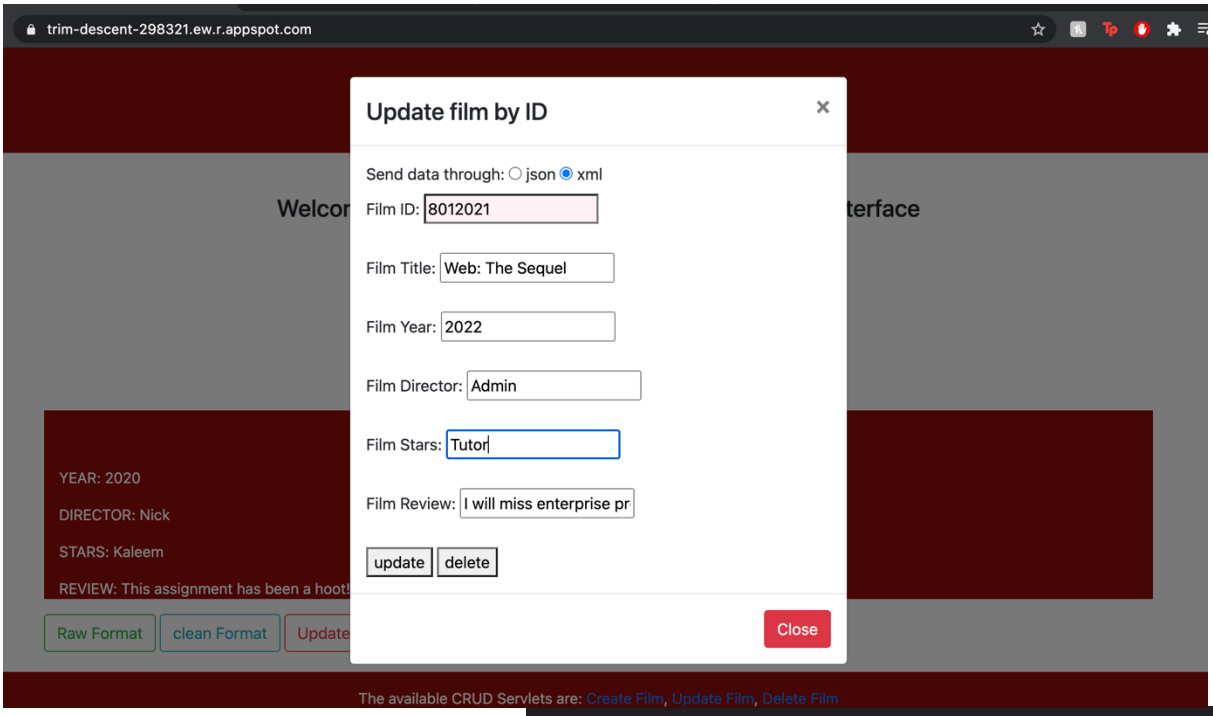
CRUD functionalities using AJAX and jQuery

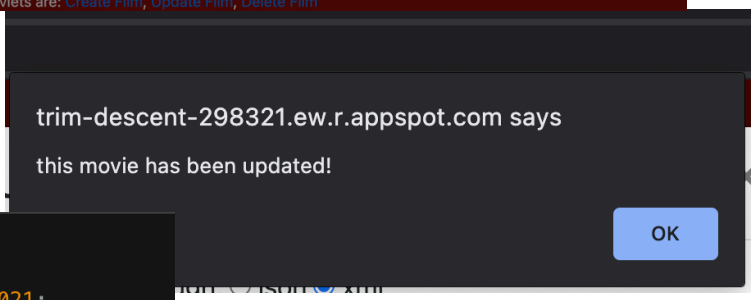
I created a separate script and linked inside my index for the CRUD functionalities, specifically the post functionalities. The first one I used is POST to create brand new film. Within the main page if you click on Add Film you will get a pop up with a form. Within this form you can create a film of your choice with the details based on id, title and etc. You can send this data off the data using either the JSON format or XML the data will be parsed according to your choice.

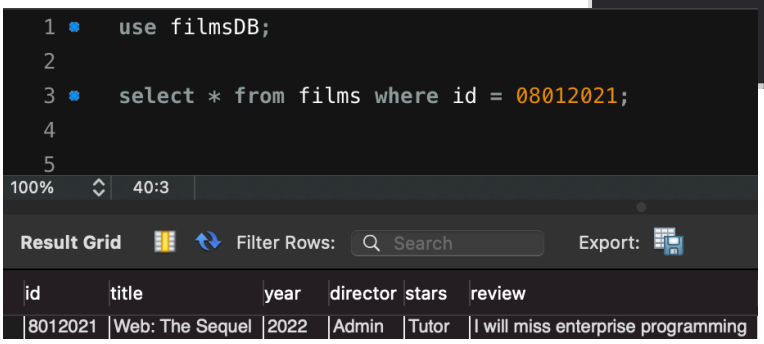


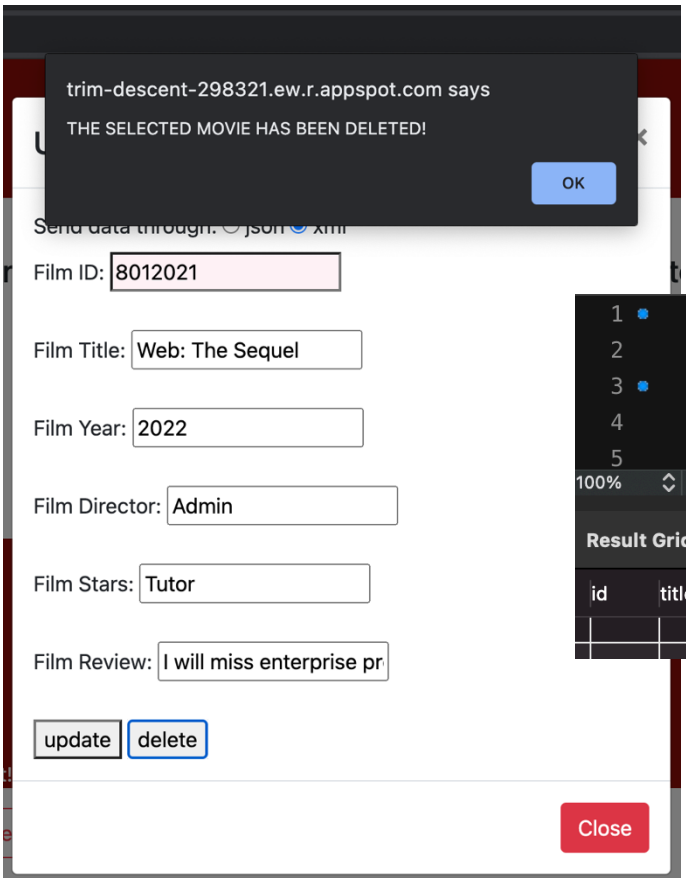
This figure shows that the movie has been send via JSON format. Later on, you can retrieve the data within any format the alert is to confirm the movie has been inserted.

A similar approach was taken to Update. Instead of using type POST I used PUT in order to perform and update from the CRUD operation. The values to update a movie will only show after you search for a movie from the dynamic search. It will take the values from the search and add it to a form where the id cannot be varied. Here you will have the option to simply Just DELETE the film you have searched or update it directly from the form and send it again as JSON or XML. To send this data off I made sure I set the content type based upon the format chosen.

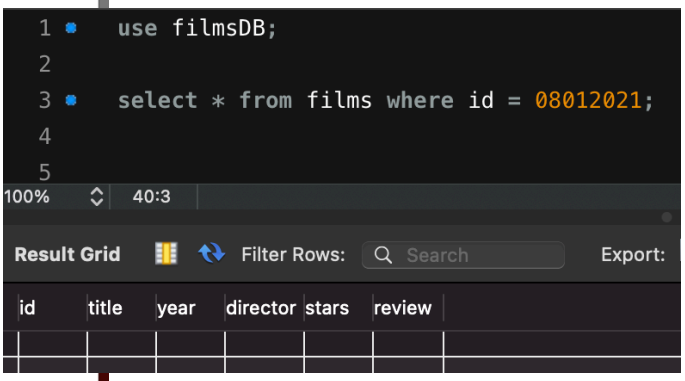








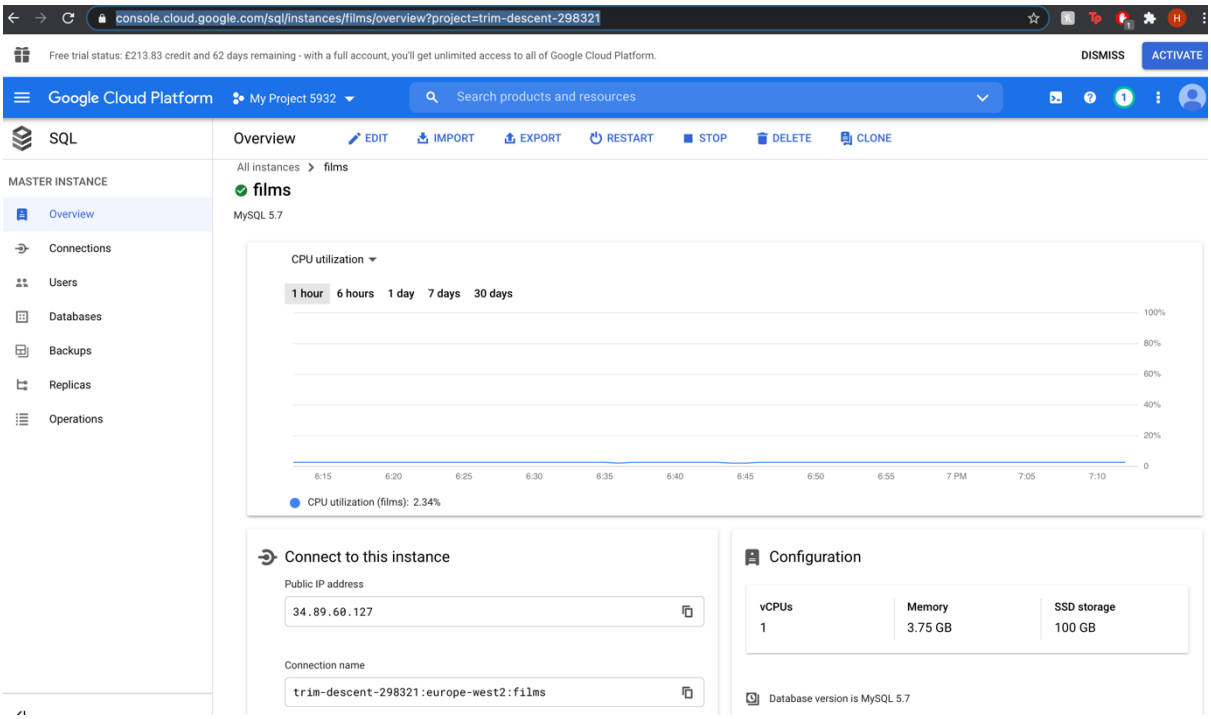
Proof that DELETE function operates by deleting the movie we created.



Google Cloud

I have uploaded my program remotely on a google cloud platform as you can see both the database and the project are hosted by Google Cloud. You can simply access the website using the following link: https://trim-descent-298321.ew.r.appspot.com/

```
String Ip_Instance = "34.89.60.127";
String databaseName = "filmsDB";
String instanceConnectionName = "trim-descent-298321:europe-west2:films";
String gcpUrl = String.format("jdbc:mysql://%s/%s?cloudSqlInstance=%s", Ip_Instance,databaseName,instanceConnectionName );
String user = "root";
String password = "12345";
```

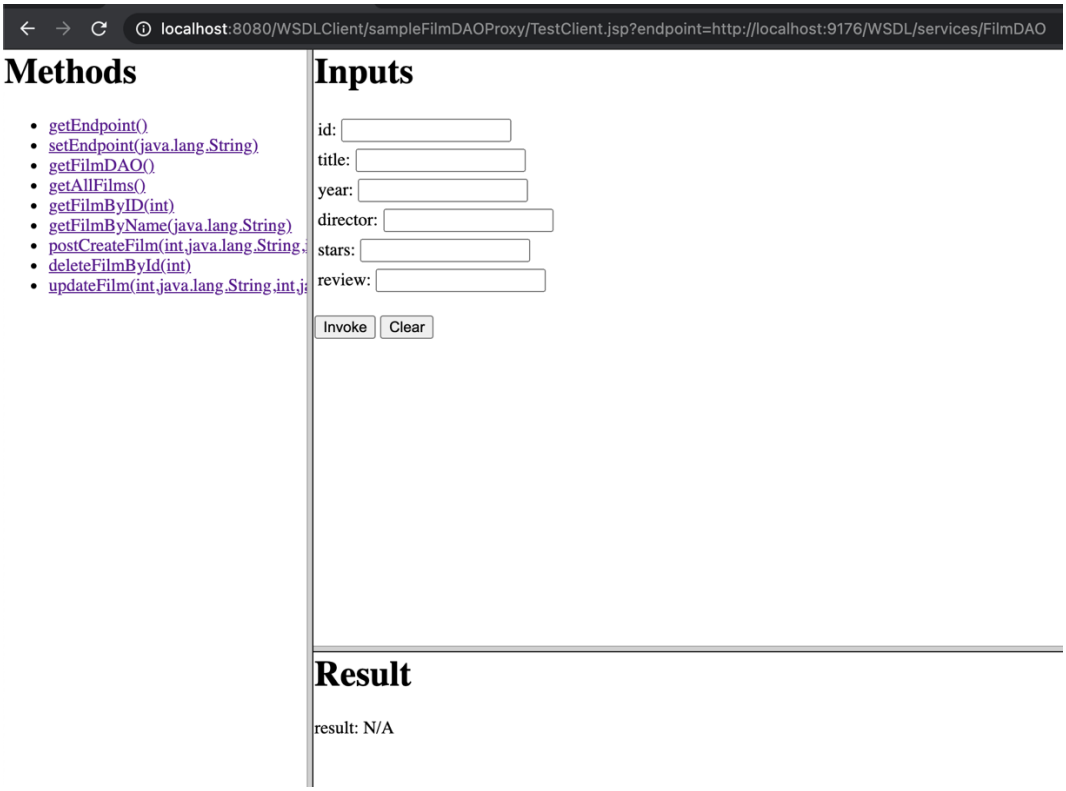


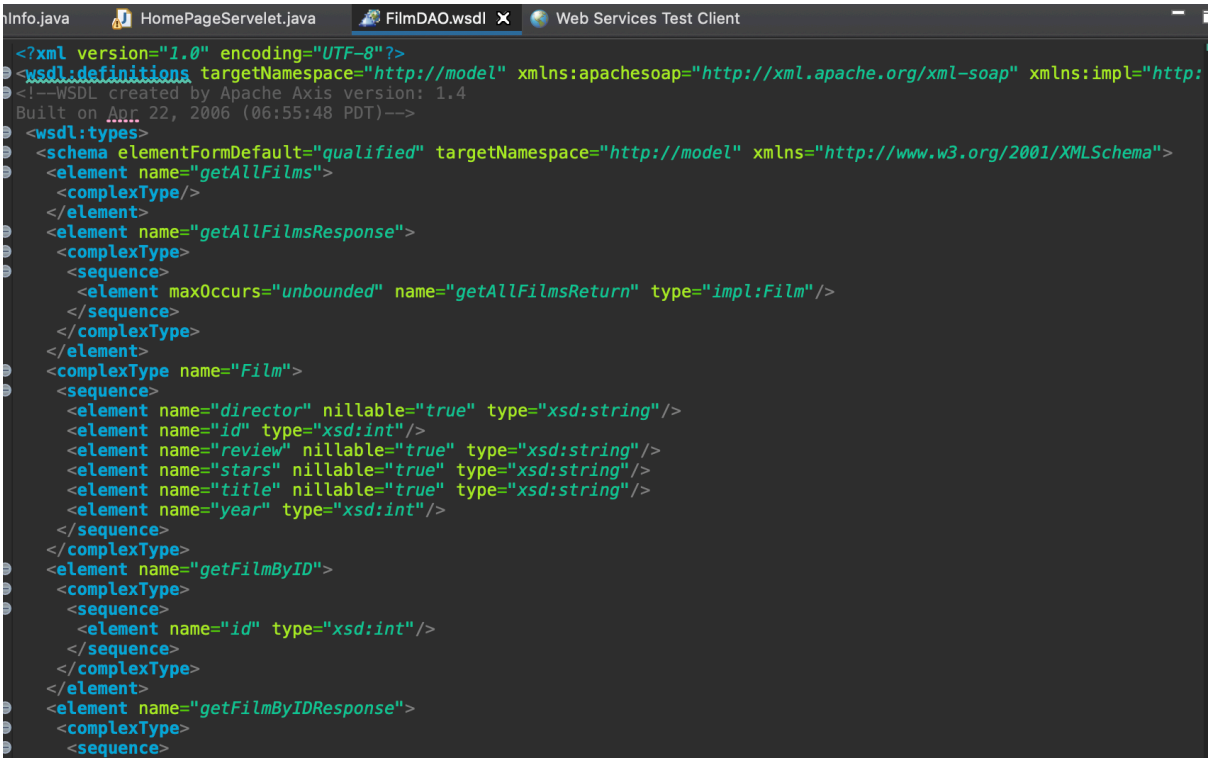

```
Beginning deployment of service [default]...
╔═══ Uploading 5 files to Google Cloud Storage ═══╗
File upload done.
Updating service [default]...
.....................................................................
Setting traffic split for service [default]...
.....................................done.
Deployed service [default] to [https://trim-descent-298321.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=trim-descent-298321
```

The last figure shows that the application has been deployed/uploaded successfully.

WSDL (SOAP)documentation

WSDL automated provides abstract operations and bound it to a concrete network protocol. Uploaded WSDL and the this is the front-end generated with all the main functionalities, all of these have been tested and validated. WSDL also gives us a message which defines the protocol.





Specific SE techniques utilised

To complete this project, I tried my best to use the best coding practice and SE techniques learnt throughout the past 2 years within this software engineering course.

Singleton Design Pattern

One of the design patterns that I chose to use are singletons. This is one of the most used design patterns of java and helped me create a connection pool, instead of creating a new connection object within every class where I used FilmDAO to call CRUD operations.

```
private static FilmDAO singletonCam;

public FilmDAO() {

    super();
}

public static synchronized FilmDAO getSingletonCam() {

    //FilmDAO singleton object
        if (singletonCam == null) {
            singletonCam = new FilmDAO();

        }

        return singletonCam;
    }
```

I first created single object using the factory method afterwards I called the singleton instance withing my resources for REST implementation without having the need to create an object to use the functions within my FilmDAO class.

Code refactoring and repetition

It is always a good idea to go over your code again once you have tested the functionality so that your code can feel, look and preform at its optimal. One example of refactoring my code is making use of singletons as before that I created an object for every function, I used in RESTful resources class however instead of keep making object I chose to use singletons which would help me not **repeat** my code over and over.

Indentation and structure

As you can tell by the MVC architecture screenshots I have structured my code following a specific pattern which would save me time once I would have to look for a specific servlet or file in which I have an error or would need to refactor. As you can see it is also important to indent the code that you write because once too many lines of codes are written it can be hard to find a specific line with an error whilst debugging so I always make sure my code is indented in a way where I can find what I'm looking for.

```
if(format.equals("json")) {
    Gson gson = new Gson();
    data = gson.toJson(allFilms);
    // convert the arraylist to JSON
    address = "jsp/json.jsp";
    response.setContentType("application/json");
    request.setAttribute("json", data);
    RequestDispatcher dispatcher =  request.getRequestDispatcher(address);
    dispatcher.forward(request, response);

}
else if(format.equals("xml")) {


     response.setContentType("text/xml");
     filmStore.setFilmList(allFilms);
        try {

            JAXBContext context = JAXBContext.newInstance(FilmStore.class);
            Marshaller m = context.createMarshaller();
            m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

            // Print to
            m.marshal(filmStore, pw);

            // Write to File

        }
        catch(Exception e){
            System.out.print(e);
        }



}
else if(format.equals("text")){
```

Appropriate naming conventions

All of the variables, IDs, classes within my code have been written with a purpose to it, if anyone would read my code, they would understand what the purpose is of most variable or classes because the names given all had a purpose to them.

```
  <input type="radio" name="parameter" id="name" value="name" required>
  <label>Search by name</label>
  <input type="radio" name="parameter" id="id" value="id" required>
  <label>Search by id</label>

  <br>

  <label>Please select preferred Data-format</label>

<select name="format" id="format">
    <option value="json" id= "json">json</option>
    <option value="xml" id="xml">xml</option>
    <option value="text" id="text">text</option>
</select>
```

Conclusion

In conclusion, in this report I have included every part of assignment I have worked on, I have analysed each individual part from simple servlet request to working with CRUD functionalities and AJAX. I have given proof of work and evidence on how I completed my assignment in as much detailed as I could. I also gave screenshots of the database uploaded to google cloud and the link to the full application where anyone can fully test the code.

References citation:

Moodle, Enterprise Programming section as a whole + example of last year work

W3schools.com

baeldung.com