1/8/2021

Developing, Deploying, Publishing and Finding your own Web Service

Proof of work

Nathan Ainsley
18028669

Contents

Sections Completed 2
Introduction 3
HTTP Interface Implementation 3
REST Implementation.......... 6
SOAP (WSDL) Implementation.......... 10
HTTP Interface in the cloud 11
Ajax and Front-End.......... 12

Sections Completed

Component	**Work done**
Criteria 1 Access to the data using simple http web service calls (10 marks)	Full implementation working, Persistence storage and all CRUD methods working.
Criteria 2 Options to return the data in text, json (the default) or xml (10 marks)	All formatting included. Use of standard libraries to create formatting for JSON (GSON) and XML (JAXB). Proper separators used for TEXT format. Capable of handling range of objects.
Criteria3 Google App Engine or Microsoft Azure to implement the application on a remote cloud-based server. (20 marks)	Fully tested and demo'd CRUD operations Server connected to a cloud database, connection ip must be authorised.
Criteria 4 A WSDL description of the interface to the web service (5 marks)	Attempt at WSDL but not fully implemented
Criteria 5 Access to the data using REST type interaction (10 marks)	Full implementation working. Data transactions send in correct RESTful way for all CRUD operations
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 used to make AJAX calls and for event-handlers for the front end. Mustache.js was used for templating of results to the user.
Criteria 7 Critical analysis of your work and the techniques you have used. (30 marks)	Separate report for critical analysis submitted

Introduction

This assignment for the Enterprise Programming unit requires the creation of a web service from scratch. The webservice should be deployed to the cloud and be accessible from anywhere in the world.

The Aim of this report is to show the working implementation of the access methods to the service. These being HTTP interface, REST, WSDL and cloud. To demonstrate the functionality screen shots of the project and the database will be provided at every point.

Please use Tomcat9.0 with access to port 8080 for localhost implementations.

HTTP Interface Implementation

My HTTP Interface implementation uses the MVC framework, as such it has 3 java packages;

- Model: Contains 3 java classes
  - Film.java – Class for the film object
  - FilmDAOHib.java – Data Access object using Hibernate to access the database. Contains all the crud operations that access the database directly
  - FilmArray.java – Class for Film Array object for XML generation
- Controller: Contains 5 java servlets that interact with the Model and the View
  - AllFilms.java – Servlet that can return all the records in the database
  - DeleteFilm.java – Servlet that can be used to delete film in database
  - InsertFilm.java – Servlet that can be used to insert a film into the database
  - Search.java – Servlet that can be used to search for a film in the database with multiple different parameter options
  - UpdateFilm.java – Servlet that can be used to update the data of a film in the database.
- View: Contains 1 java class to format data types
  - Formatter.java – Class to format the data returned in the AllFilms.java and Search.java servlets into either Json, Xml or String

The default data return type is JSON however can be selected as either XML or string with either AllFilms?Format={type} or Search?searchval={}&format={type} where searchval is either;

- filmname
- filmid
- filmyear
- filmdirector
- filmstars

Key	**Values**	**HTTP Method**	**Servlets**
Format	Json(Default)/Xml/String Effects the returned format of data when provided	Get	/AllFilms /Search
Search Value	filmname/filmid/filmyear /filmdirector/filmstars	Get	/Search

	Providing a different one will change the search type and return a different value. Providing more than one will cause an error		
Id	Id of film record	Get	/DeleteFilm
Film Object	Film object of film record	Post	/UpdateFilm /InsertFilm

once the project has been imported these servlets can be interacted with via the front-end design which will be talked about later in the report.

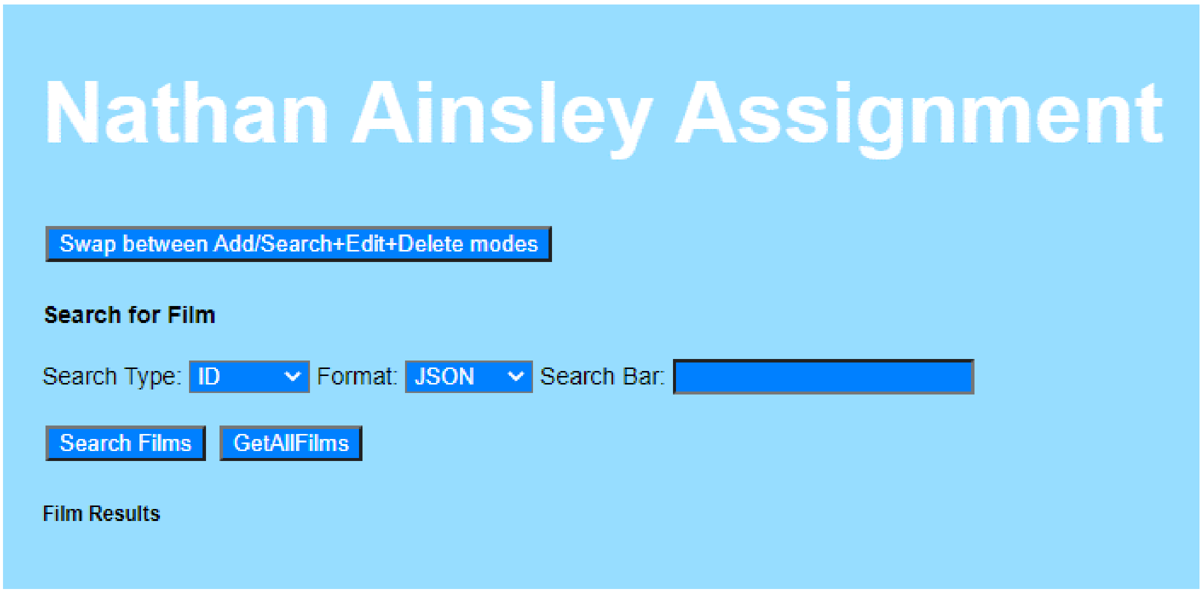
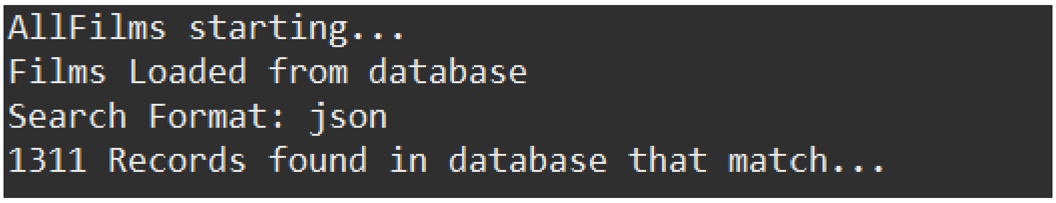


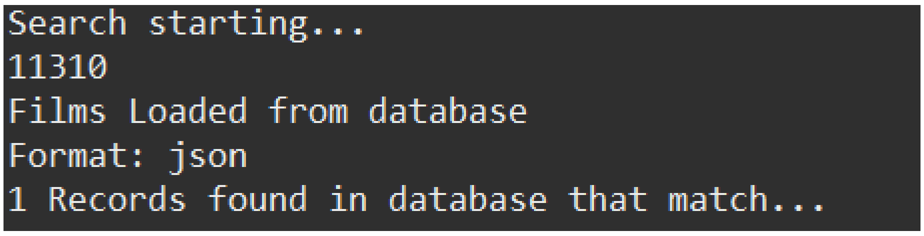
Figure 1.0 Shows the GUI of the Front end design to interact with the HTTP interface

For debug purposes when a servlet is called by the application it will print out to console what it is currently doing. This helped in flattening out bugs and helps to show the application working.



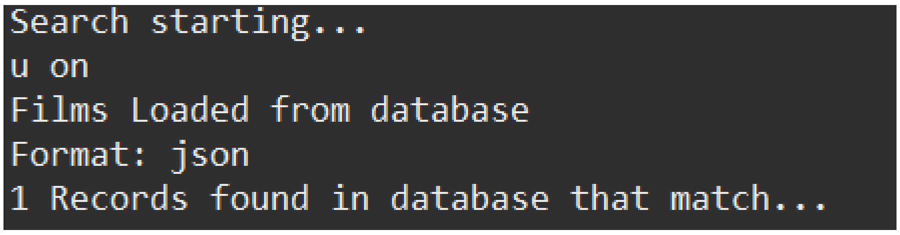
```
AllFilms starting...
Films Loaded from database
Search Format: json
1311 Records found in database that match...
```

Figure 2 shows the log in eclipse when the GetAllFilms button was pressed in the GUI



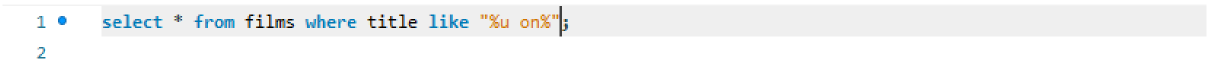
```
Search starting...
11310
Films Loaded from database
Format: json
1 Records found in database that match...
```

Figure 3 Shows the Search function running with the parameter passed being filmid=11310

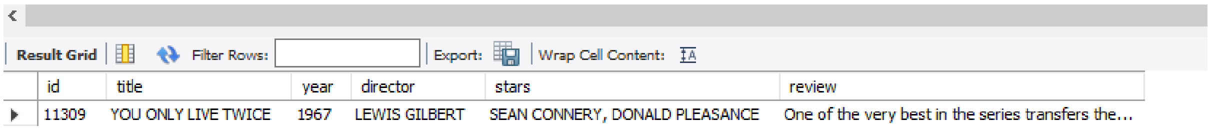


```
Search starting...
u on
Films Loaded from database
Format: json
1 Records found in database that match...
```

Figure 4 Shows a search using the string 'u on'



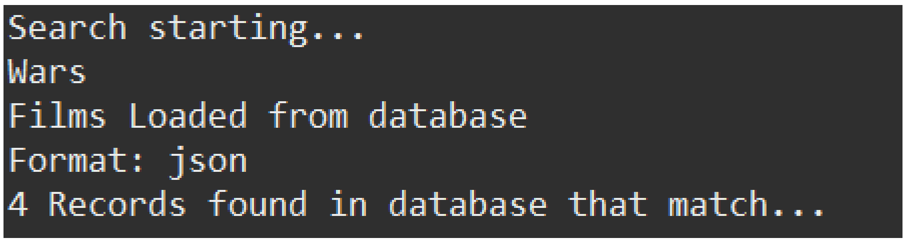
```
select * from films where title like "%u on%";
```



id	title	year	director	stars	review
11309	YOU ONLY LIVE TWICE	1967	LEWIS GILBERT	SEAN CONNERY, DONALD PLEASANCE	One of the very best in the series transfers the...

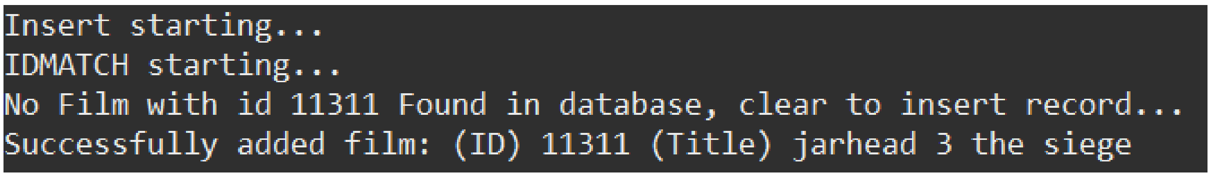
Figure 5 Shows the result of the string search

Because the search functions use the operation Like instead of a simple = the servlets are able to deal with the title, year, director or stars being partial searches as seen in Figure 4 Shows a search using the string 'u on'.



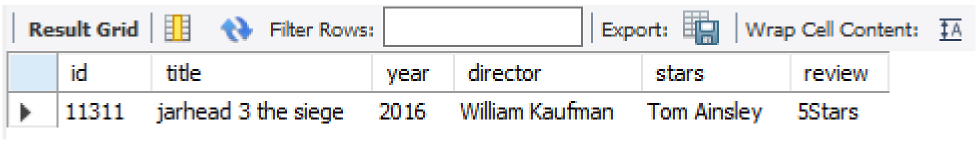
```
Search starting...
Wars
Films Loaded from database
Format: json
4 Records found in database that match...
```

Figure 6 Shows the Search function running with the parameter passed being filmName=Wars



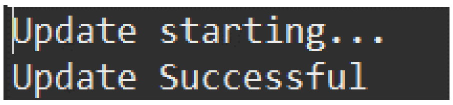
```
Insert starting...
IDMATCH starting...
No Film with id 11311 Found in database, clear to insert record...
Successfully added film: (ID) 11311 (Title) jarhead 3 the siege
```

Figure 7 Shows the adding of a film with the ID 11311 to the database



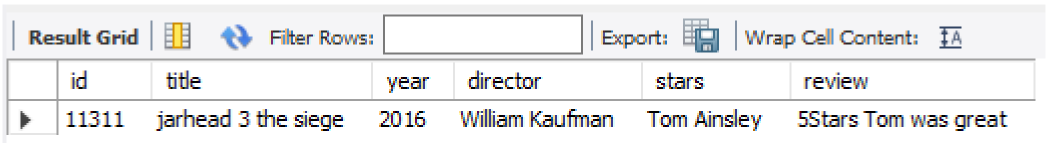
id	title	year	director	stars	review
11311	jarhead 3 the siege	2016	William Kaufman	Tom Ainsley	5Stars

Figure 8 Shows the film has been added to the database



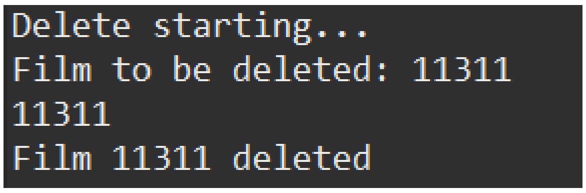
```
Update starting...
Update Successful
```

Figure 9 Shows the console log from the updating of a record showing that it started and was Successful



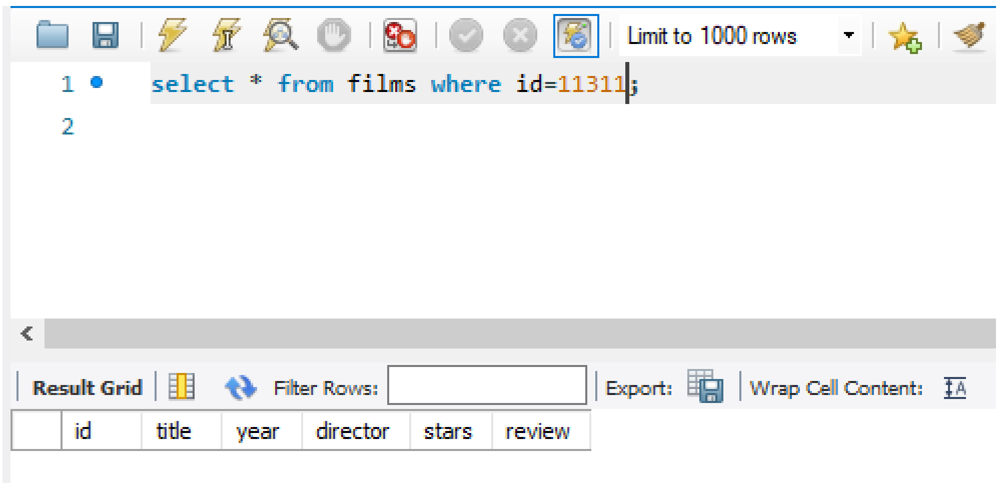
id	title	year	director	stars	review
11311	jarhead 3 the siege	2016	William Kaufman	Tom Ainsley	5Stars Tom was great

Figure 10 Shows that the review for the film has now been updated after the update was ran



```
Delete starting...
Film to be deleted: 11311
11311
Film 11311 deleted
```

Figure 11 Shows the Console log when the delete button is pressed



```sql
select * from films where id=11311;
```

id	title	year	director	stars	review

Figure 12 Shows that the Sql query returns empty because the film with that id was just deleted

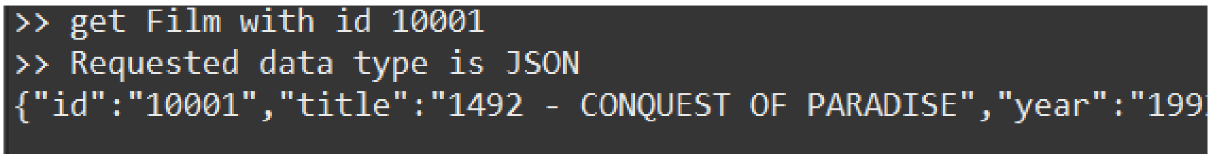
REST Implementation

My REST implementation contains 3 packages within the project. Those being;

- rest.services.film.dao – Contains the Data accessor object
- rest.services.film.model – Contains the Film object and Film Array Object
- rest.services.film.resources – Contains the java files that operate the CRUD operations via the rest methods

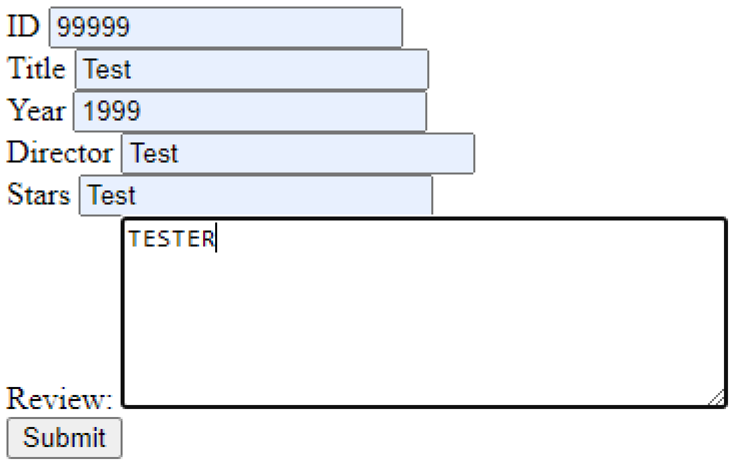
Resource Paths	**Request Type**	**Return**
rest.18028669.services/rest/films	Get	Returns all the films in the database to screen
rest.18028669.services/rest/films/count	Get	Returns the number of records in the database
rest.18028669.services/rest/films/{id}	Get	Returns specific record with the id supplied
rest.18028669.services/rest/films	Post	Adds film object to the database
rest.18028669.services/rest/films	Put	Updates film record with object
rest.18028669.services/rest/films/{id}	Delete	Deletes film record from database

Using a standalone java application, I tested the functionality of the REST operations.



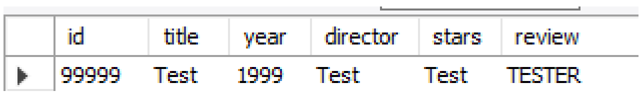
```
>> get Film with id 10001
>> Requested data type is JSON
{"id":"10001","title":"1492 - CONQUEST OF PARADISE","year":"199
```

Figure 13 Shows the REST interface returning a result for a search



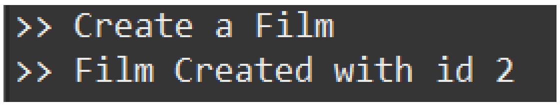
ID	99999
Title	Test
Year	1999
Director	Test
Stars	Test
Review:	TESTER

Figure 14 Shows the create_film.html page that allows you to create a film with an interface



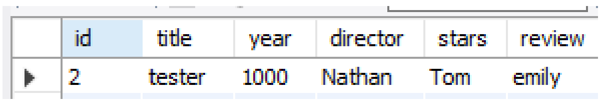
id	title	year	director	stars	review
99999	Test	1999	Test	Test	TESTER

Figure 15 Shows the Sql result when querying the mudfoot database that the REST service is linked to



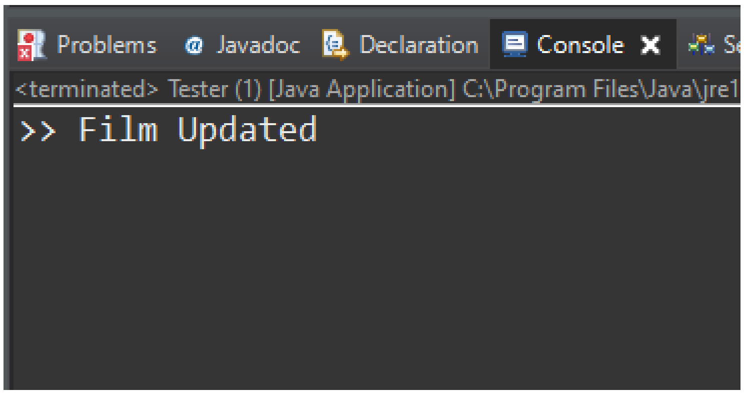
```
>> Create a Film
>> Film Created with id 2
```

Figure 16 Shows the return from the standalone java function showing that a film with the id 2 was created



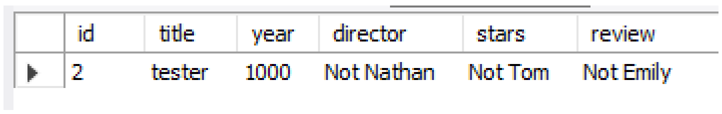
id	title	year	director	stars	review
2	tester	1000	Nathan	Tom	emily

Figure 17 Shows the film created with the code in the standalone java application



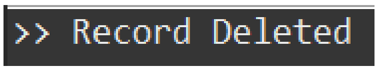
```
<terminated> Tester (1) [Java Application] C:\Program Files\Java\jre1
>> Film Updated
```

Figure 18 After running the update method to update the record created with id two a confirmation is given



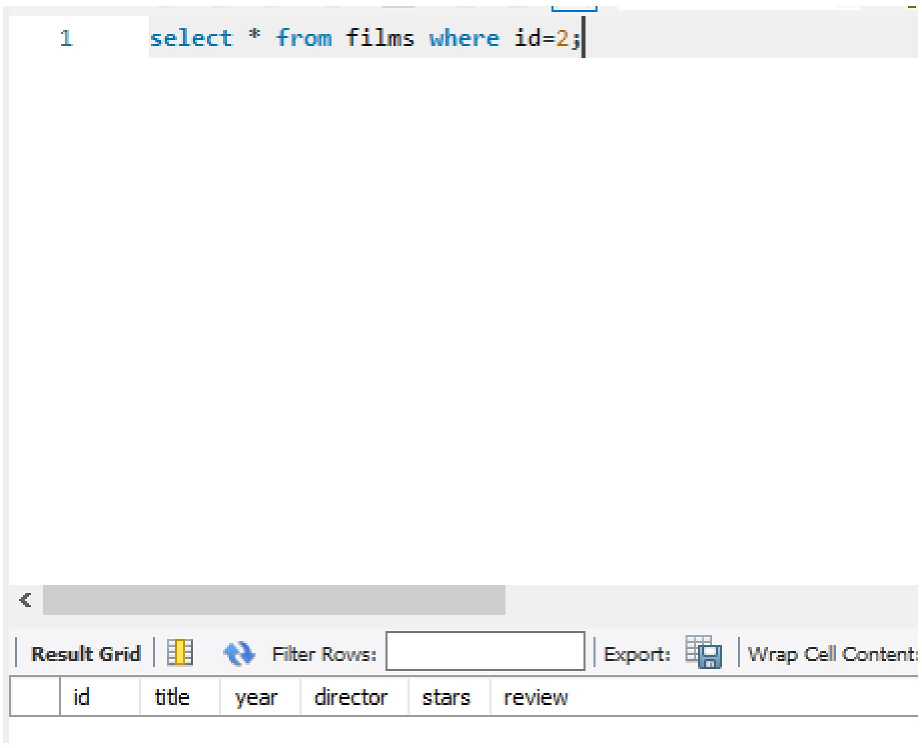
id	title	year	director	stars	review
2	tester	1000	Not Nathan	Not Tom	Not Emily

Figure 19 Shows that the record with ID has now been updated.



```
>> Record Deleted
```

Figure 20 Shows the result after calling the delete rest method

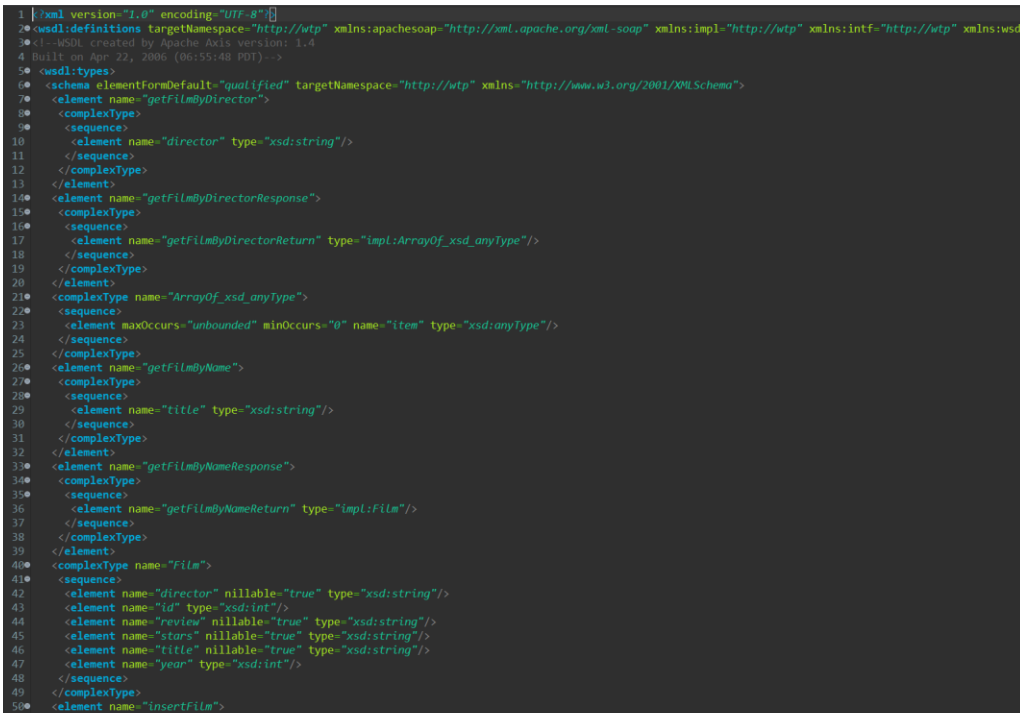


```
select * from films where id=2;
```

id	title	year	director	stars	review

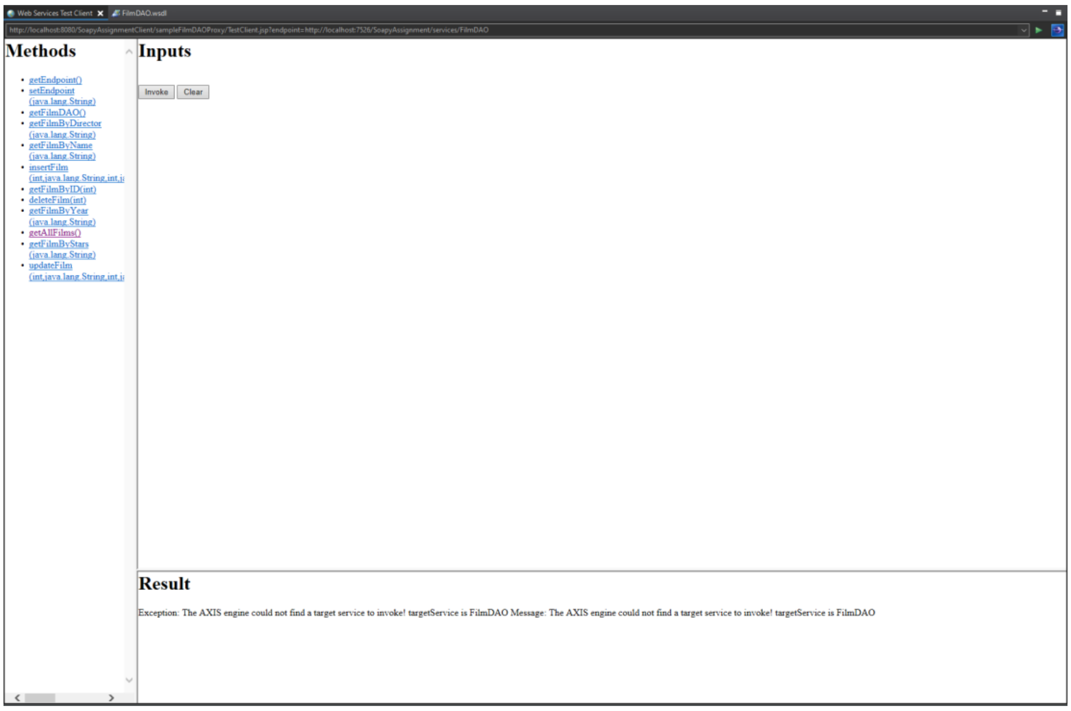
Figure 21 Shows that after the delete has been ran, searching for the record in the database returns nothing as the record has been deleted

SOAP (WSDL) Implementation



```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://wtp" xmlns:apachesoap="http://xml.apache.org/xml-soap" xmlns:impl="http://wtp" xmlns:intf="http://wtp" xmlns:wsd
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
 <wsdl:types>
  <schema elementFormDefault="qualified" targetNamespace="http://wtp" xmlns="http://www.w3.org/2001/XMLSchema">
   <element name="getFilmByDirector">
    <complexType>
     <sequence>
      <element name="director" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="getFilmByDirectorResponse">
    <complexType>
     <sequence>
      <element name="getFilmByDirectorReturn" type="impl:ArrayOf_xsd_anyType"/>
     </sequence>
    </complexType>
   </element>
   <complexType name="ArrayOf_xsd_anyType">
    <sequence>
     <element maxOccurs="unbounded" minOccurs="0" name="item" type="xsd:anyType"/>
    </sequence>
   </complexType>
   <element name="getFilmByName">
    <complexType>
     <sequence>
      <element name="title" type="xsd:string"/>
     </sequence>
    </complexType>
   </element>
   <element name="getFilmByNameResponse">
    <complexType>
     <sequence>
      <element name="getFilmByNameReturn" type="impl:Film"/>
     </sequence>
    </complexType>
   </element>
   <complexType name="Film">
    <sequence>
     <element name="director" nillable="true" type="xsd:string"/>
     <element name="id" type="xsd:int"/>
     <element name="review" nillable="true" type="xsd:string"/>
     <element name="stars" nillable="true" type="xsd:string"/>
     <element name="title" nillable="true" type="xsd:string"/>
     <element name="year" type="xsd:int"/>
    </sequence>
   </complexType>
   <element name="insertFilm">
```

Figure 22 Shows the attempt at the implementation of the design view of the WSDL generated from a bottom up approach



http://localhost:8080/SoapyAssignmentClient/sampleFilmDAOProxy/TestClient.jsp?endpoint=http://localhost:7526/SoapyAssignment/services/FilmDAO

Methods

- getEndpoint()
- setEndpoint (java.lang.String)
- getFilmDAO()
- getFilmByDirector (java.lang.String)
- getFilmByName (java.lang.String)
- insertFilm (int,java.lang.String,int,j
- getFilmByID(int)
- deleteFilm(int)
- getFilmByYear (java.lang.String)
- getAllFilms()
- getFilmByStars (java.lang.String)
- updateFilm (int,java.lang.String,int,j

Inputs

Result

Exception: The AXIS engine could not find a target service to invoke! targetService is FilmDAO Message: The AXIS engine could not find a target service to invoke! targetService is FilmDAO

Figure 23 Shows the front end design generated from the attempt at WSDL

HTTP Interface in the cloud

As per criteria 3 of the assignment the HTTP implementation of the project was uploaded to the google cloud platform. This was then connected to a Sql database instance also running on the google cloud.

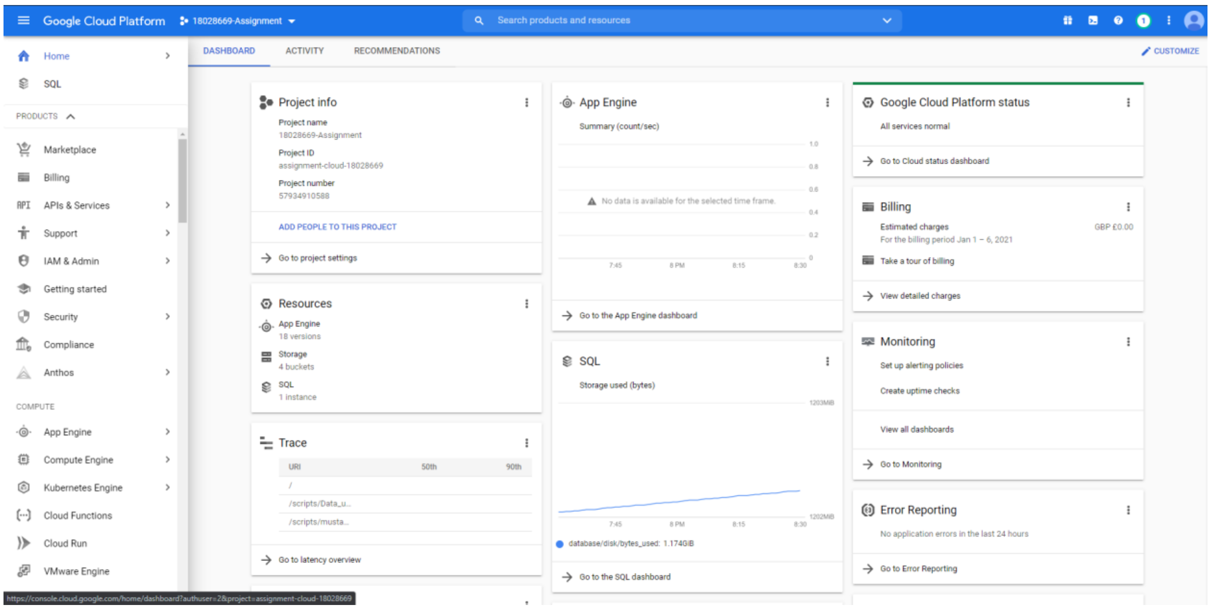
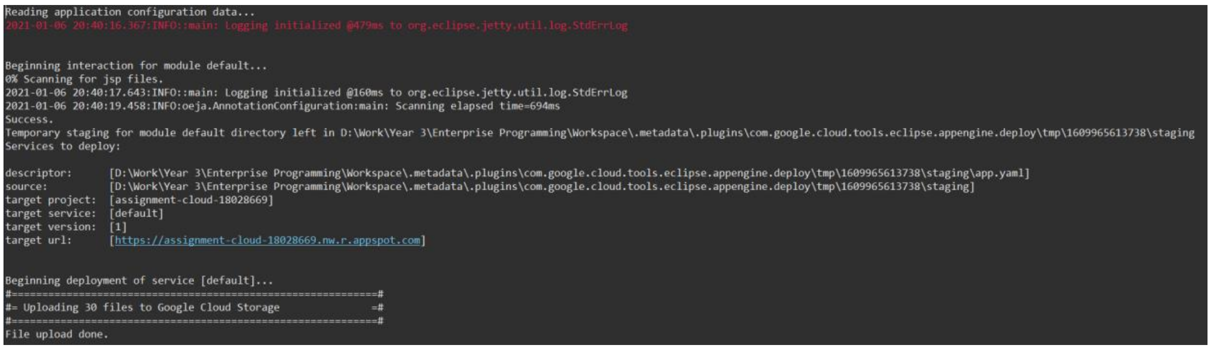


Figure 24 shows the dashboard for both the Sql database and the application



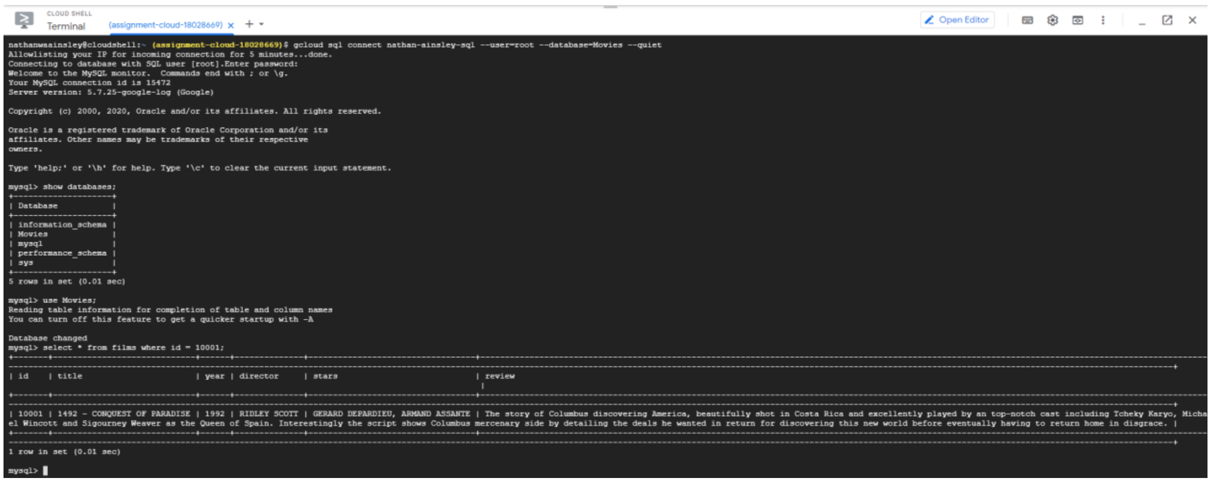
```
Reading application configuration data...
2021-01-06 20:40:16.367:INFO::main: Logging initialized @479ms to org.eclipse.jetty.util.log.StdErrLog


Beginning interaction for module default...
0% Scanning for jsp files.
2021-01-06 20:40:17.643:INFO::main: Logging initialized @160ms to org.eclipse.jetty.util.log.StdErrLog
2021-01-06 20:40:19.458:INFO:oeja.AnnotationConfiguration:main: Scanning elapsed time=694ms
Success.
Temporary staging for module default directory left in D:\Work\Year 3\Enterprise Programming\Workspace\.metadata\.plugins\com.google.cloud.tools.eclipse.appengine.deploy\tmp\1609965613738\staging
Services to deploy:

descriptor:      [D:\Work\Year 3\Enterprise Programming\Workspace\.metadata\.plugins\com.google.cloud.tools.eclipse.appengine.deploy\tmp\1609965613738\staging\app.yaml]
source:          [D:\Work\Year 3\Enterprise Programming\Workspace\.metadata\.plugins\com.google.cloud.tools.eclipse.appengine.deploy\tmp\1609965613738\staging]
target project:  [assignment-cloud-18028669]
target service:  [default]
target version:  [1]
target url:      [https://assignment-cloud-18028669.nw.r.appspot.com]


Beginning deployment of service [default]...
#============================================================#
#= Uploading 30 files to Google Cloud Storage              =#
#============================================================#
File upload done.
```

Figure 25 Shows the eclipse log of uploading an update for the app to the cloud



```
nathanwaainsley@cloudshell:~ (assignment-cloud-18028669)$ gcloud sql connect nathan-ainsley-sql --user=root --database=Movies --quiet
Allowlisting your IP for incoming connection for 5 minutes...done.
Connecting to database with SQL user [root].Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15472
Server version: 5.7.25-google-log (Google)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| Movies             |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql> use Movies;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select * from films where id = 10001;
+-------+-----------------------------+------+--------------+-----------------------------------+----------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------+
| id    | title                       | year | director     | stars                             | review
                                                                                                  |
+-------+-----------------------------+------+--------------+-----------------------------------+----------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------+
| 10001 | 1492 - CONQUEST OF PARADISE | 1992 | RIDLEY SCOTT | GERARD DEPARDIEU, ARMAND ASSANTE | The story of Columbus discovering America, beautifully shot in Costa Rica and excellently played by an top-notch cast including Tcheky Karyo, Micha
el 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. |
+-------+-----------------------------+------+--------------+-----------------------------------+----------------------------------------
---------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.01 sec)

mysql> 
```

Figure 26 Shows the connection to the google cloud Sql server via the google cloud shell online

The application running on the cloud is the same as that of the basic HTTP interface and has the front-end design as seen in the next section.

The cloud application can be reached at https://assignment-cloud-18028669.nw.r.appspot.com/ however the Sql server will most likely be stopped to prevent the occurring of unwanted charges.

Ajax and Front-End

The interface for both the Cloud and regular HTTP interface is the same as it's the same application. The front-end design of this application as seen bellow in Figure 27 Front-End Design shows the search interface of the application.

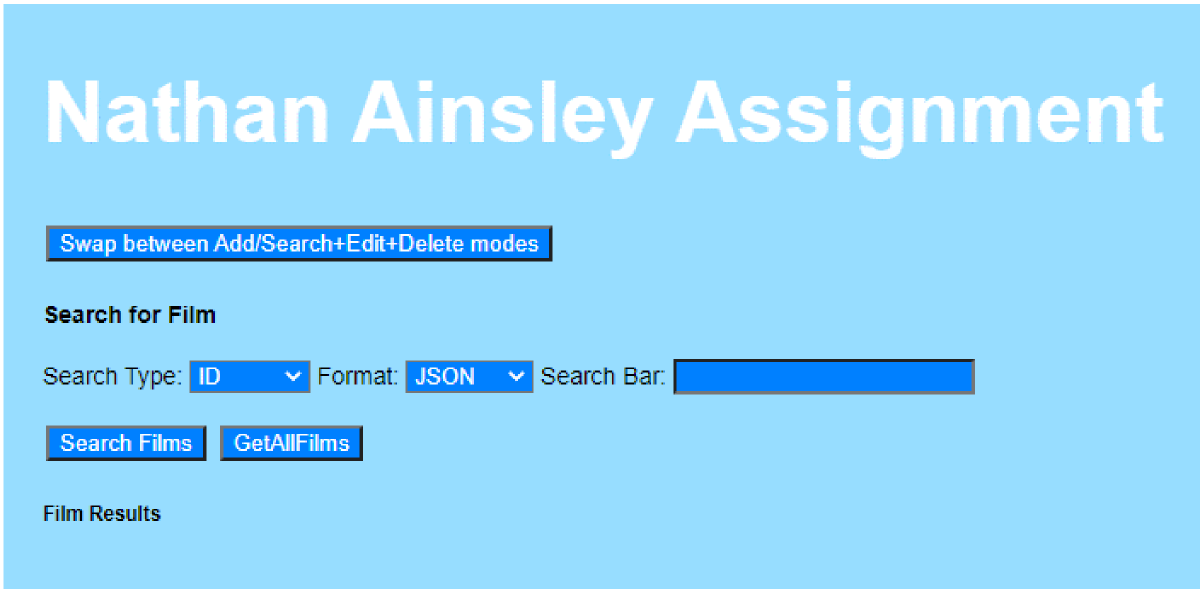


Figure 27 Front-End Design

The top button will swap the interface between the adding and Deleting mode.

Bellow the mode switch button is the search interface, as seen bellow in Figure 28 Search Type Drop Down there is a drop down menu to select the type of search. This dictates the search type.

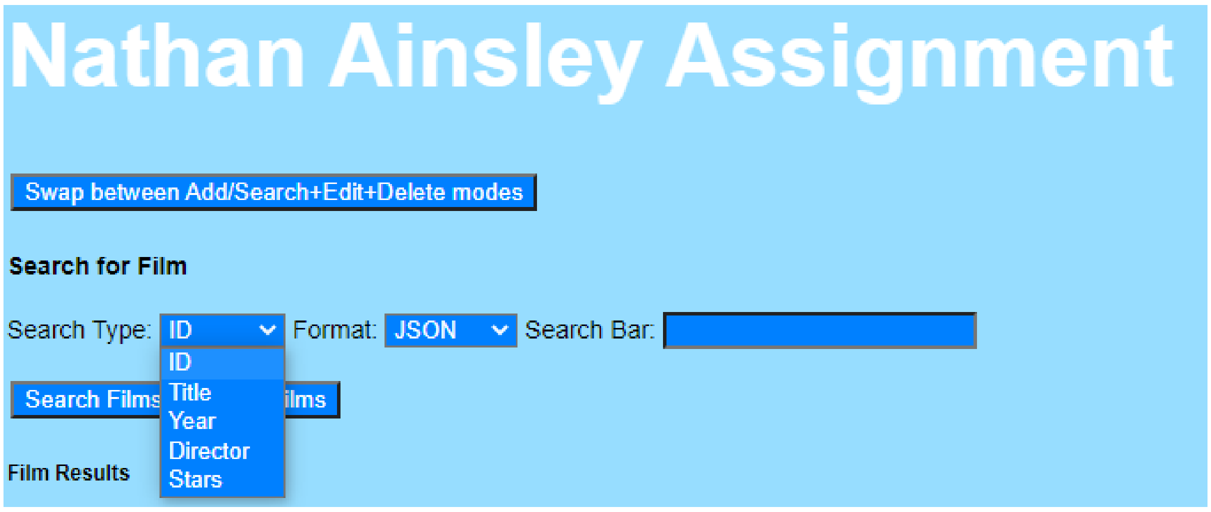


Figure 28 Search Type Drop Down

Again in Figure 29 Format Drop Down there is a second drop down box to this time select the format that the data would be returned in. The combination of the two dropdown boxes doesn't matter and can be mixed and matched in every combination.

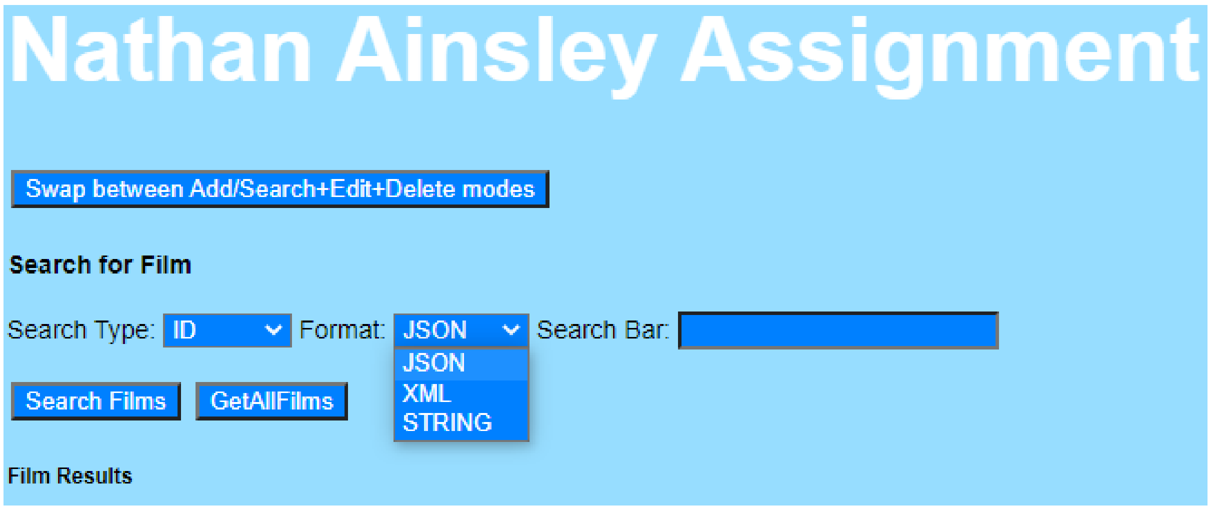
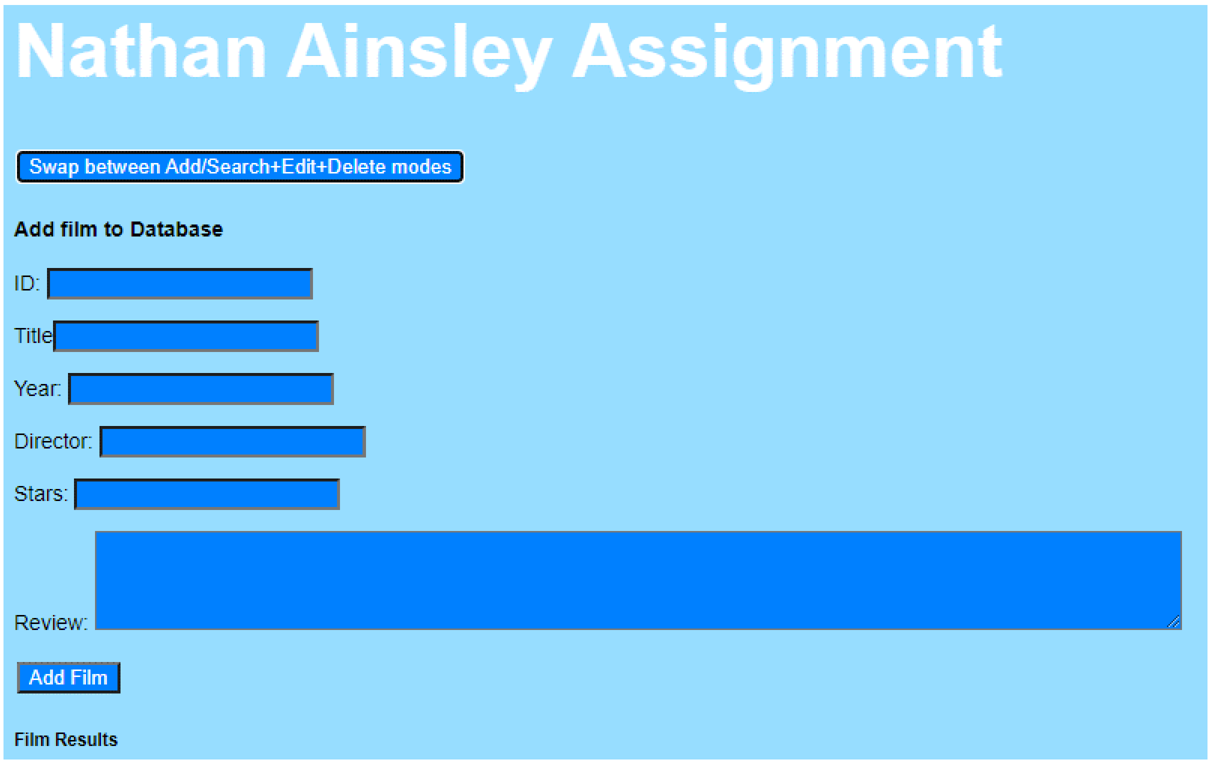


Figure 29 Format Drop Down

Next there is the search bar which your search result will be entered. It is important that you give a search query that matches the search type selected otherwise no result would be returned.

Bellow the search configuration settings are the search buttons. The left one will run a search using the configuration settings set above it. The second button, the one on the right will return every film in the database to the application.



Nathan Ainsley Assignment

Swap between Add/Search+Edit+Delete modes

Add film to Database

ID:

Title

Year:

Director:

Stars:

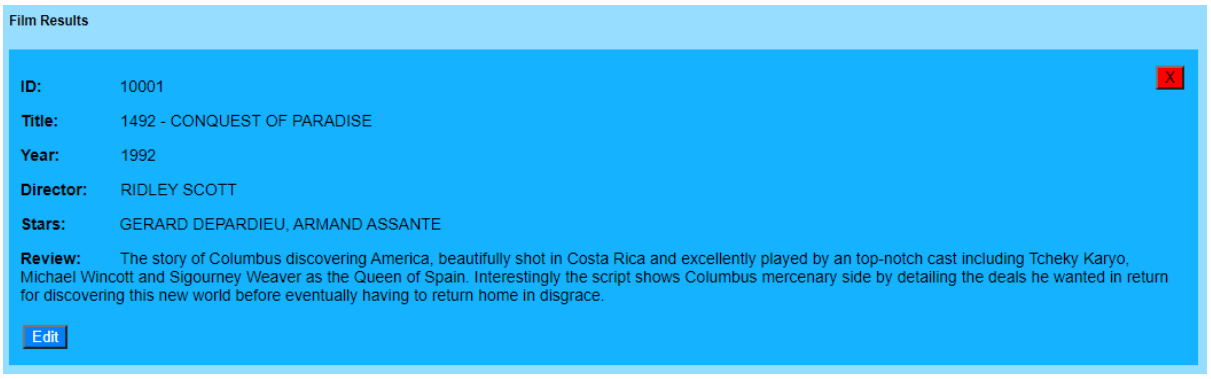
Review:

Add Film

Film Results

Figure 30 After pressing the Mode Swap button this is the add screen shown

Shown above in Figure 30 After pressing the Mode Swap button this is the add screen shownis the add mode for the application. Entering values into these boxes and pressing the button will attempt to add a record into the database. The servlets will run a search to validate if the ID used is unique, if not you would be asked to use a different ID.



Film Results

X

ID: 10001

Title: 1492 - CONQUEST OF PARADISE

Year: 1992

Director: RIDLEY SCOTT

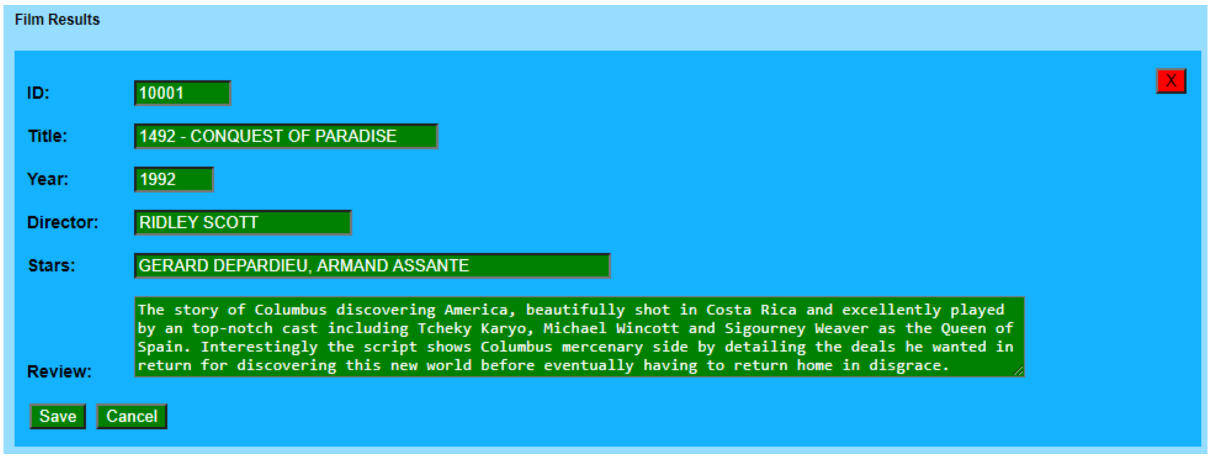
Stars: GERARD DEPARDIEU, ARMAND ASSANTE

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.

Edit

Figure 31 Shows the template created for the result using Mustache.js

As seen above in Figure 31 Shows the template created for the result using Mustache.js results are displayed in individual boxes. These boxes allow me to do some powerful operations thanks to Mustache.js however this is covered in the critical analysis.
When the Edit button is pressed the box becomes an edit box for the record with the existing values loaded into the inputs and text areas.



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.

Figure 32 Shows the edit mode of a single film

When the edit button is pressed the box turning into edit mode which makes it look like Figure 32 Shows the edit mode of a single film. Pressing the cancel button will revert the box to its state in Figure 31 Shows the template created for the result using Mustache.js and no changes will be saved. However, if the Save button is pressed then the values of all the boxes will be saved. This means that if you don't change the values in the boxes nothing will be changed as the data will be updated with itself.

In the top corner of each of the result boxes is a small red box with an X in it, this is the delete button. Pressing that button will delete the record within that box from the database. This means to edit or delete a record you first must search for the record and then perform the action on it saving time and resources having to search then run a different operation on a different page.

Conclusion

This report has shown the work done towards the assignment and gives an understanding implementation created. To-do this I showed screenshots of the implementations in operation, eclipse logs and MySQL workbench queries.

For a more detailed explanation of how to invoke or operate the services please refer to the readme file within the submission folder.