1/8/2021

Nathan Ainsley

18028669

**Developing, Deploying, Publishing and Finding your own Web Service**

Proof of work

Contents

[Sections Completed 2](#_Toc60861916)

[Introduction 3](#_Toc60861917)

[HTTP Interface Implementation 3](#_Toc60861918)

[REST Implementation 6](#_Toc60861919)

[SOAP (WSDL) Implementation 10](#_Toc60861920)

[HTTP Interface in the cloud 11](#_Toc60861921)

[Ajax and Front-End 12](#_Toc60861922)

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

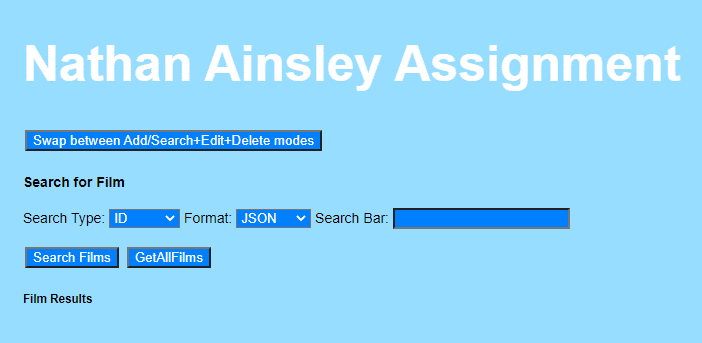
once the project has been imported these servlets can be interacted with via the front-end design which will be talked about later in the report.  


Figure .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.

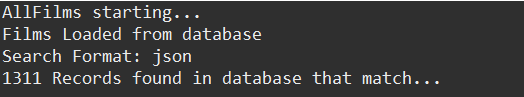


Figure shows the log in eclipse when the GetAllFilms button was pressed in the GUI

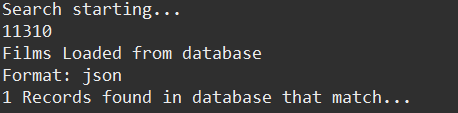


Figure Shows the Search function running with the parameter passed being filmid=11310

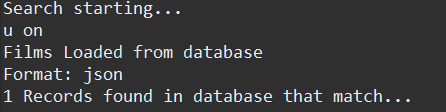


Figure Shows a search using the string 'u on'

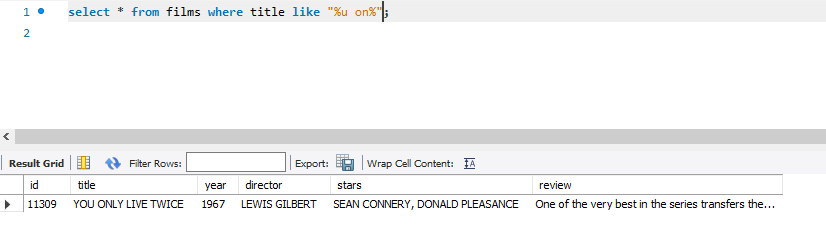


Figure Shows the result of the string search

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'.

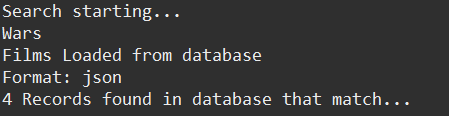


Figure Shows the Search function running with the parameter passed being filmName=Wars

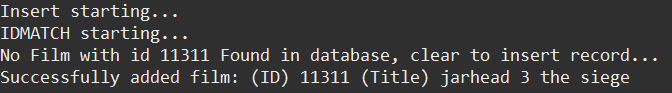


Figure Shows the adding of a film with the ID 11311 to the database



Figure Shows the film has been added to the database



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

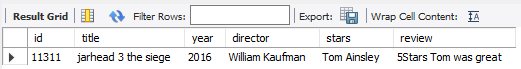


Figure Shows that the review for the film has now been updated after the update was ran

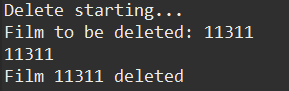


Figure Shows the Console log when the delete button is pressed

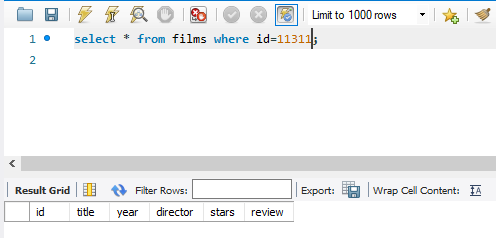


Figure Shows that the Sql query returns empty because the film with that id was just deleted

# 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.

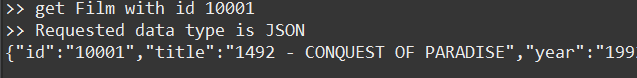


Figure Shows the REST interface returning a result for a search

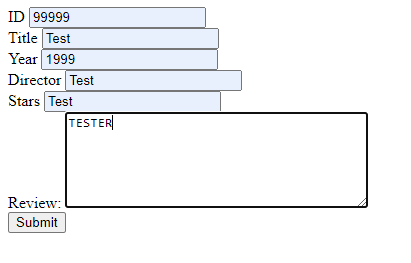


Figure Shows the create\_film.html page that allows you to create a film with an interface



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



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



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

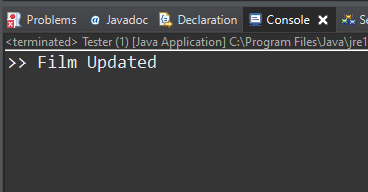


Figure After running the update method to update the record created with id two a confirmation is given



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



Figure Shows the result after calling the delete rest method

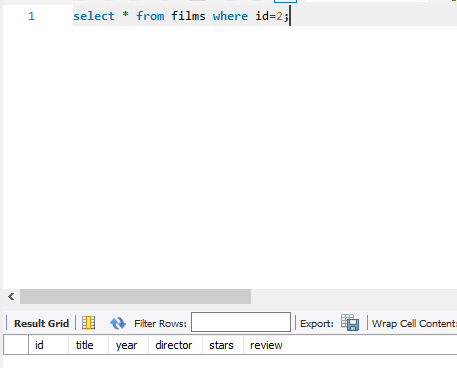


Figure Shows that after the delete has been ran, searching for the record in the database returns nothing as the record has been deleted

# SOAP (WSDL) Implementation

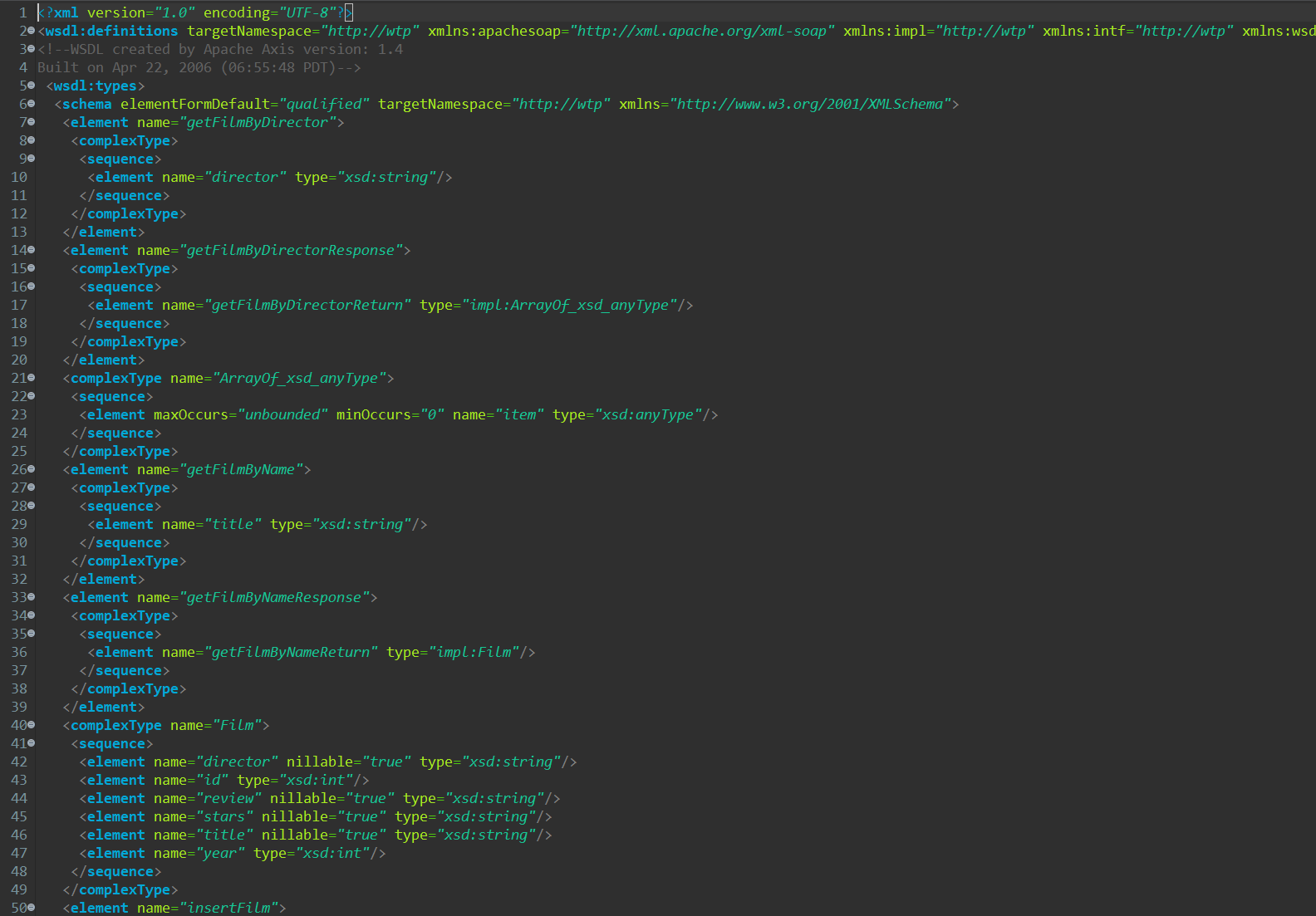


Figure Shows the attempt at the implementation of the design view of the WSDL generated from a bottom up approach

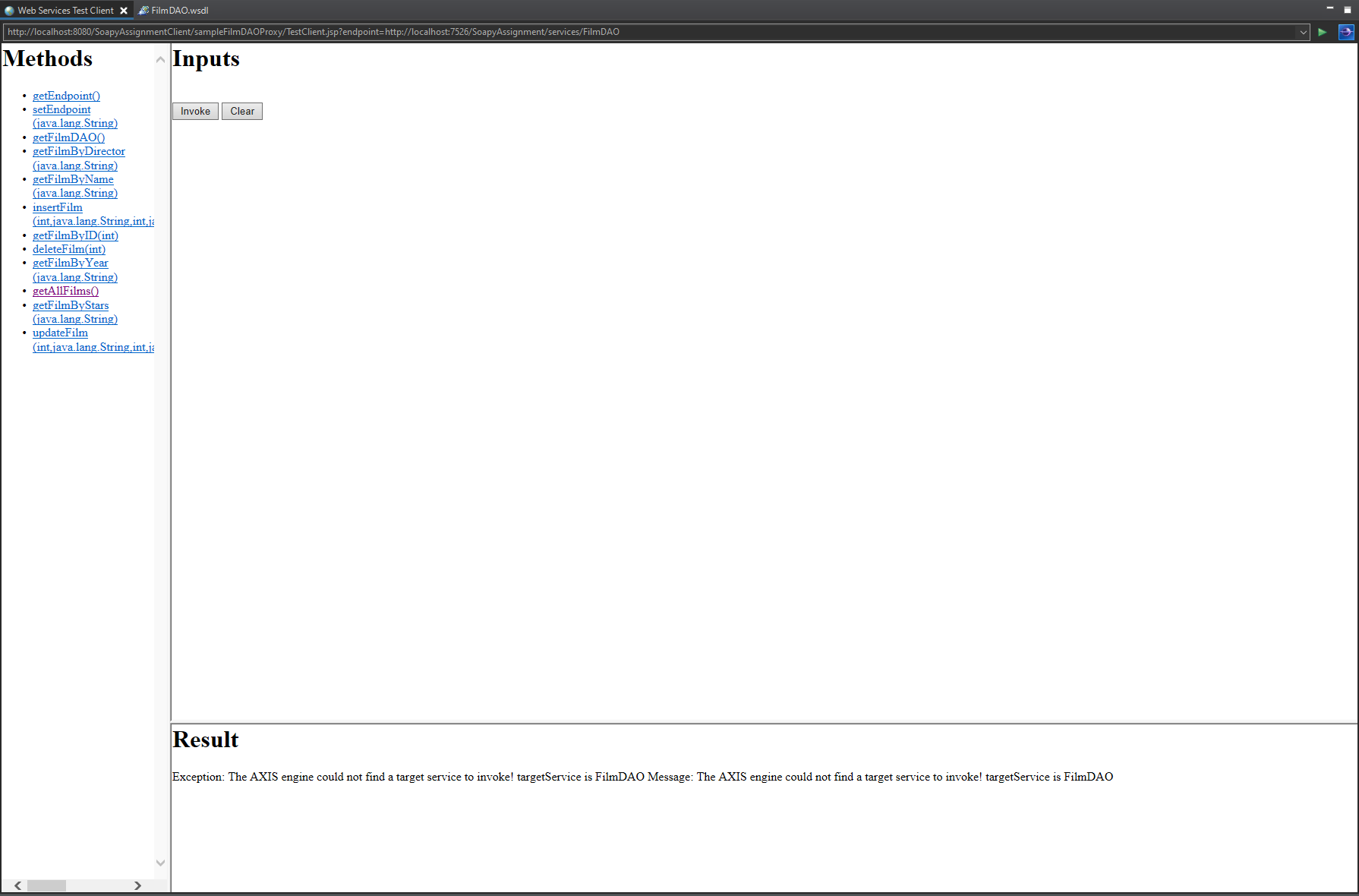


Figure Shows the front end design generated from the attempt at WSDL

# 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.

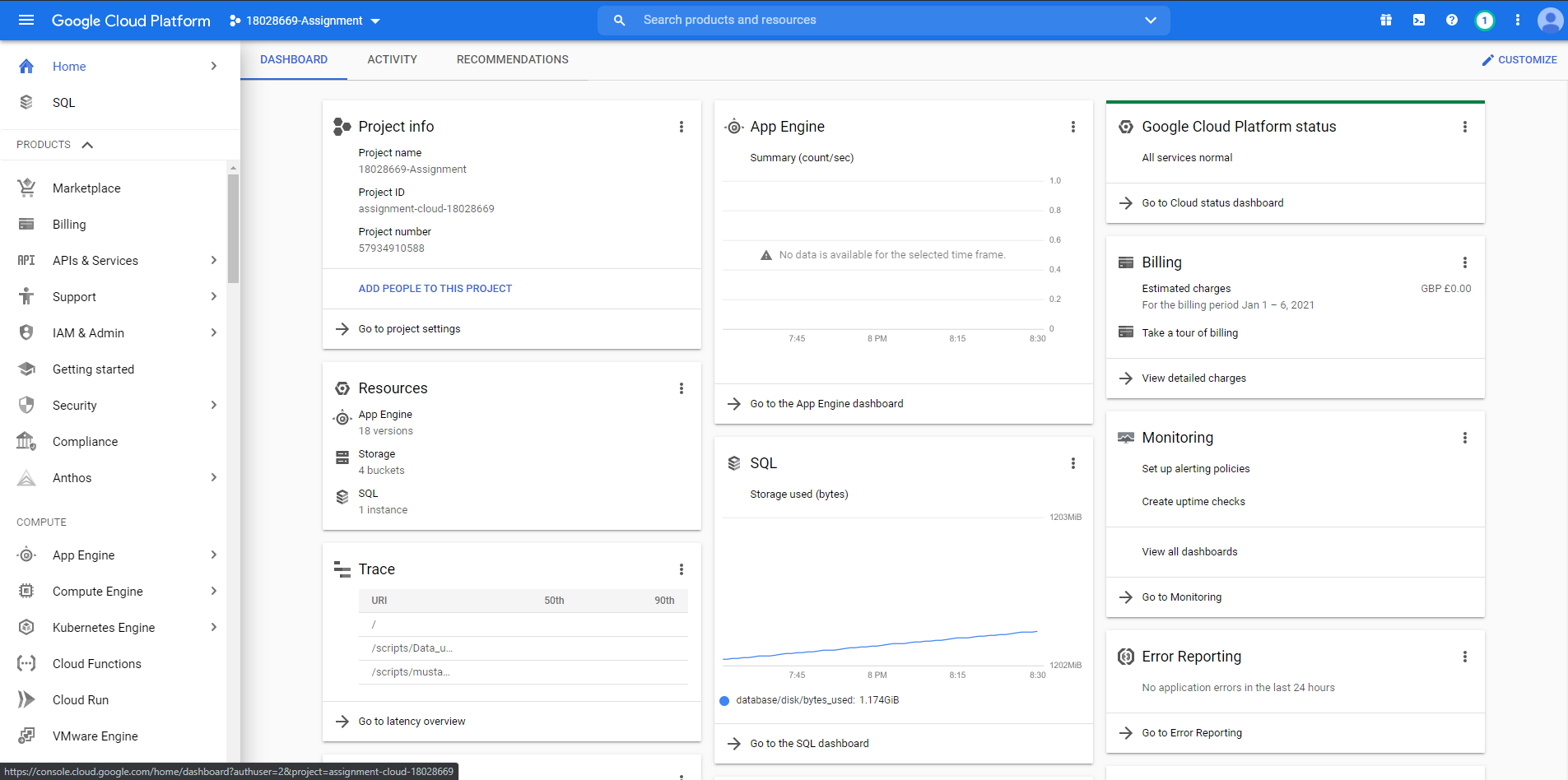


Figure shows the dashboard for both the Sql database and the application

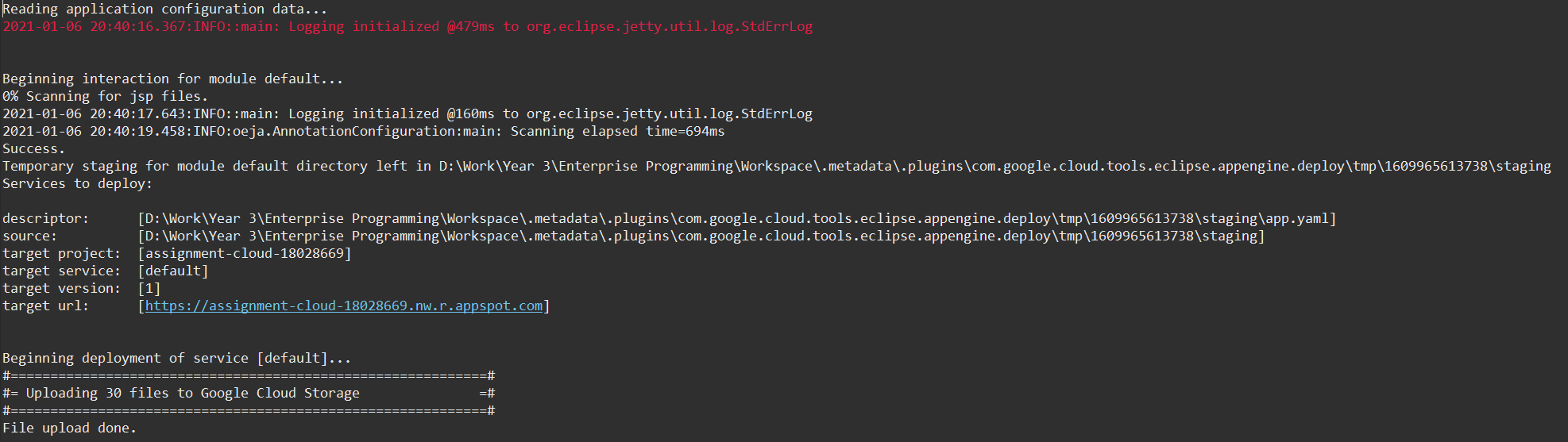


Figure Shows the eclipse log of uploading an update for the app to the cloud

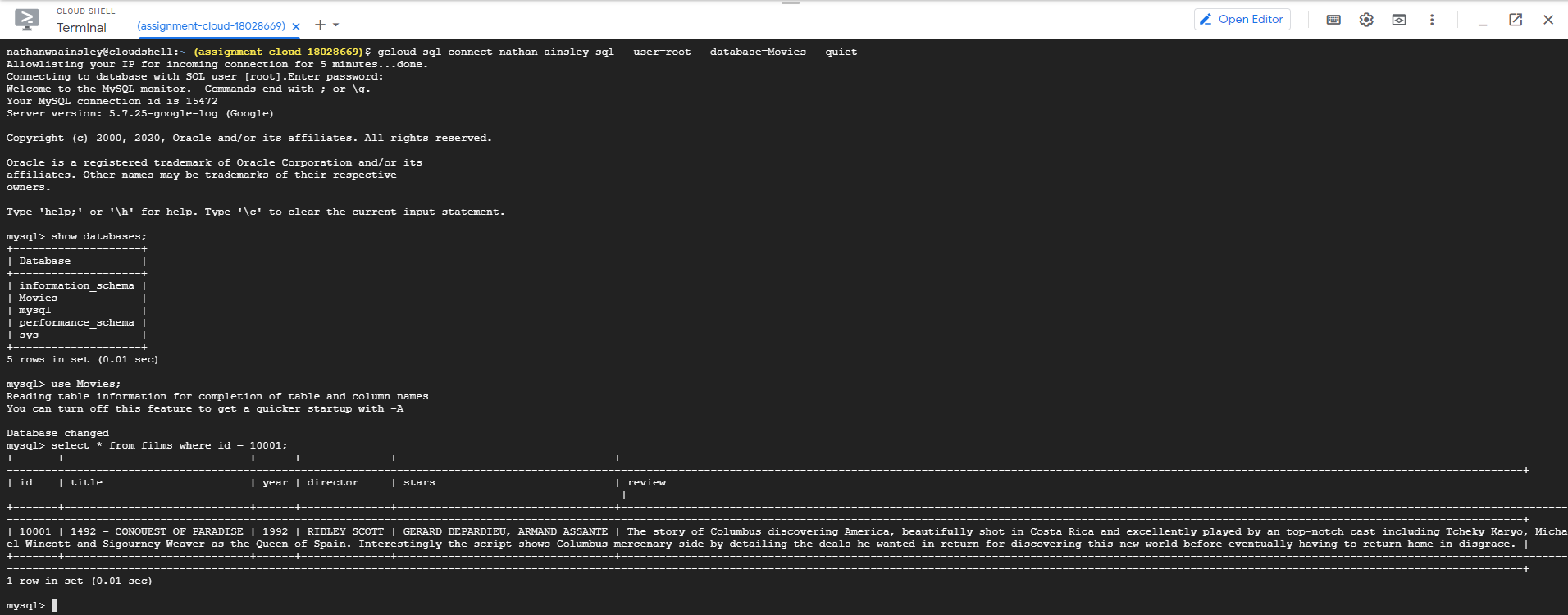


Figure Shows the connection to the google cloud Sql server via the google cloud shell online

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

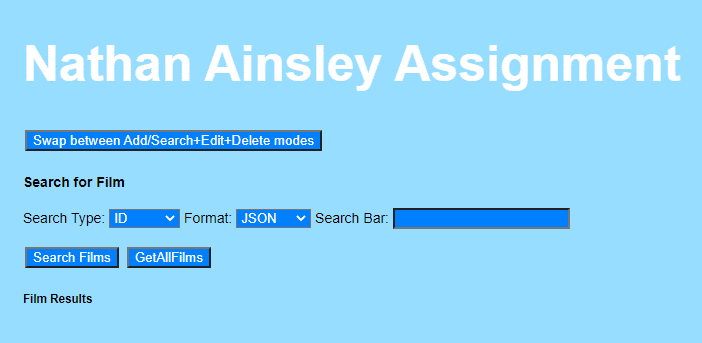
The interface for both the Cloud and regular HTTP interface is the same as it’s the same application. The front-end design of this application as seen bellow in Figure 25 Front-End Design shows the search interface of the application. 

Figure 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 26 Search Type Drop Down there is a drop down menu to select the type of search. This dictates the search type.

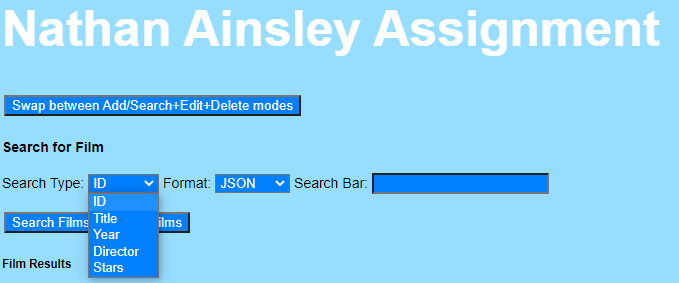


Figure Search Type Drop Down

Again in Figure 27 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.

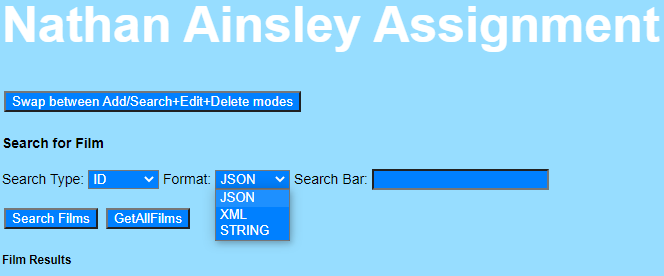


Figure Format Drop Down

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.

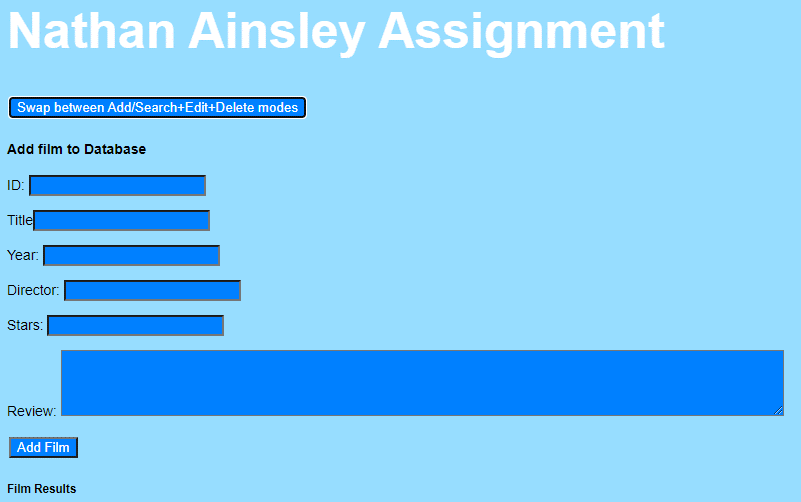


Figure After pressing the Mode Swap button this is the add screen shown

Shown above in Figure 28 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.

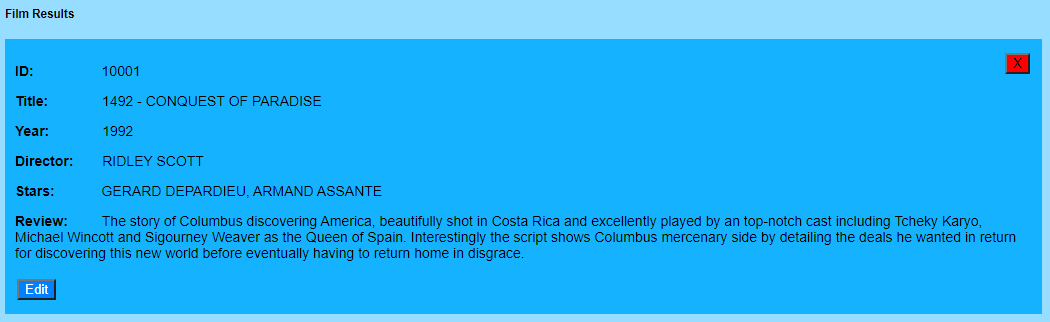


Figure Shows the template created for the result using Mustache.js

As seen above in Figure 29 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.

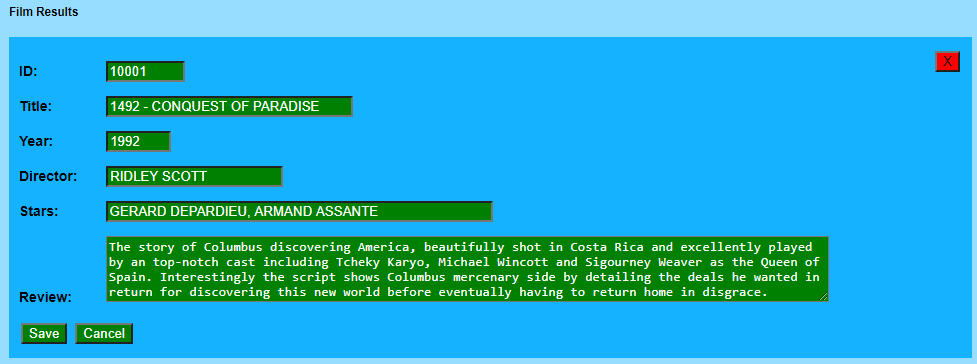


Figure Shows the edit mode of a single film

When the edit button is pressed the box turning into edit mode which makes it look like Figure 30 Shows the edit mode of a single film. Pressing the cancel button will revert the box to its state in Figure 29 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.