

## Introduction

This project is about creating a films web service for other software developers. The task involves the usage of different techniques used widespread, by many organisations such as Facebook, IBM and Microsoft . This project achieves the same functionality (dealing with the database) using different technology such as MVC (Model View Controller), WSDL and Restful Service. Moreover, the focus of this assignment is also on advanced design patterns as they greatly aid readability, usability and reliability.

## Coding Techniques

### If statements

```
if (films != null)
{
    String allFilmsJSON = gson.toJson(films);
    response.getWriter().println(allFilmsJSON);
}
```

*if (variable.equals("literal")) – (Avoided)*

```
if ("xml".equals(type))
{
    response.setContentType("text/xml");
    outputPage = "/WEB-INF/results/film_xml.jsp";
}
```

If statements are used throughout, to assure the data was available before it's sent to a different file or used by another variable. Also, it helped dealing with null pointer exceptions because navigating the issue was made extremely easy by the if statements. On the other hand, it did make the code bloated. A good alternative to that is switch statements.

Switch statements are usually more efficient than if statements and help greatly with readability too. Furthermore, switch statements run much faster than the equivalent logic coded with many if-else statements; this is because the compiler knows that the case constants consists of same types and must be compared for equality whereas with if-else statements, the compiler knows nothing.

Switch statements can also allow the whole project to run faster as it does not have to evaluate every single previous condition first. This could have helped in my project as the whole CRUD process could have worked a lot faster; also, readability and the overall code quality could have been improved. On the other hand, one flaw of switch statements is restricted conditions. i.e. the conditions are based on a single value, integer or a string whereas if-else statements can test the condition with different conditions and ranges of value.

### System.out.println

In my opinion, a software related product cannot be completed without any use of messages as they play an important role to debug errors at an early age. It's also the simplest and the most efficient way to print messages. However, the drawback of using it is that statements are temporary, so when a console is cleared all the messages are removed. Custom messages also affect the reusability as one developer's message may cause conflict with another. A better alternative would be a PrintWriter as it can write to

other sources like `HttpResponse`, whereas `System.out.println` can only write to the console. Moreover, `PrintWriter` also enables default encoding. For example, `PrintStream(OutputStream ops, Boolean autoFlush, String encoding)`;

## Access to Data Using HTTP Web Calls

To access the data firstly a database must be created, which is done in MySQL Workbench. The database consists of a table names 'films' which contains all the film data. To access the data in Eclipse, some client-side must be added. Firstly 'Film.java' is created to represent the film with its given attributes. In the 'FilmDAO.java' class, the connection is made to the database using unique mySQL username and passwords. Several methods are also created such as `insertFilm` and `deleteFilm`, where SQL queries to access the data. Using the MVC design-pattern

```
public class ControllerDAO {
    public static void main(String[] args) throws SQLException {
        //RETRIEVE ALL FILMS
        FilmDAO dao = new FilmDAO();
        ArrayList<Film> allFilms = dao.getAllFilms();
        System.out.println("Text: " + allFilms + "\n");
        Gson gson = new Gson();
        String allFilmsJSON = gson.toJson(allFilms);
        System.out.println("\nJSON: " + allFilmsJSON + "\n");

        //SEARCH FILM BY Name
        String search = "Bang Bang";
        Film oneFilm = dao.getFilmByName(search);
        System.out.println("Film by Name: " + oneFilm);

        //SEARCH FILM BY Id
        int id = 1;
        Film searchFilm = dao.getFilmById(id);
        System.out.println("Film by Id: " + searchFilm);

        //INSERT
        Film in = new Film(0, "Dabangg 3", 2019, "Salman Khan", "249mins", "comedy movie. Exceeded the expectations.");
        dao.insertFilm(in);
        System.out.println("Film Inserted: " + in);

        //DELETE
        int filmId = 6;
        dao.deleteFilm(filmId);
        System.out.println("Film Deleted, Film ID: " + filmId);

        //UPDATE
        Film update = new Film(4, "Dabangg 4", 2019, "Salman Khan", "249mins", "comedy movie. Exceeded the expectations.");
        dao.updateFilm(update);
        System.out.println("Film Updated: " + update + "\n");
    }
}
```

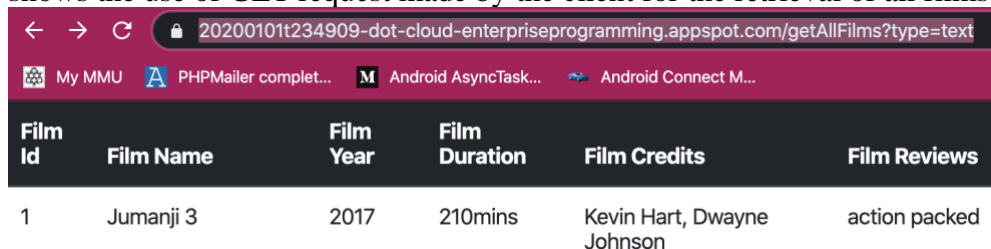
```
//gets all the films from the database
public ArrayList<Film> getAllFilms()
{
    System.out.println("\nRetrieving all Films ...\n");
    ArrayList<Film> filmsList = new ArrayList<>();
    openConnection();
    String selectFilmNames = "SELECT * FROM Films";
    try {
        ResultSet rs = statement.executeQuery(selectFilmNames);
        System.out.println("selectFilmNames Query - Status = Successfull\n");
        while (rs.next()) {
            Film film = getFilm(rs);
            filmsList.add(film);
        }
        statement.close();
        closeConnection();
    } catch (SQLException se) {
        se.printStackTrace();
        System.out.println("selectFilmNames Query - Status = Failed");
    }
    return filmsList;
}

//used to hold incoming data
private Film getFilm(ResultSet rs) {
    Film filmRs = null;
    try {
        filmRs = new Film(rs.getInt("filmId"), rs.getString("filmName"), rs.getInt("filmYear"),
            rs.getString("filmCredits"), rs.getString("filmDuration"), rs.getString("filmReview"));
    } catch (SQLException se) {
        se.printStackTrace();
        se.getMessage();
        System.out.println("Error occurred. Please check: getFilm.");
    }
    return filmRs;
}
```

approach, all this is stored in the Model. In the Controller, methods from the 'FilmDAO' will be executed. For the data to be accessed, `doGet` and `doPost` methods are put into place. `doGet` allows parameters to be passed in the URL however `doPost` parameters are sent separately. The CRUD methods are administered in the `FilmDAO` class and are invoked as servlets in the controller.

## Http, SOAP and Rest:

For the first part of the project, Http is used to transfer data over a network. The whole process is executed when a request is made to the URL that is associated to the server. Once the connection is established, different requests such as DELETE is used to delete data from the database. i.e. Figure 1 shows the use of GET request made by the client for the retrieval of all films.



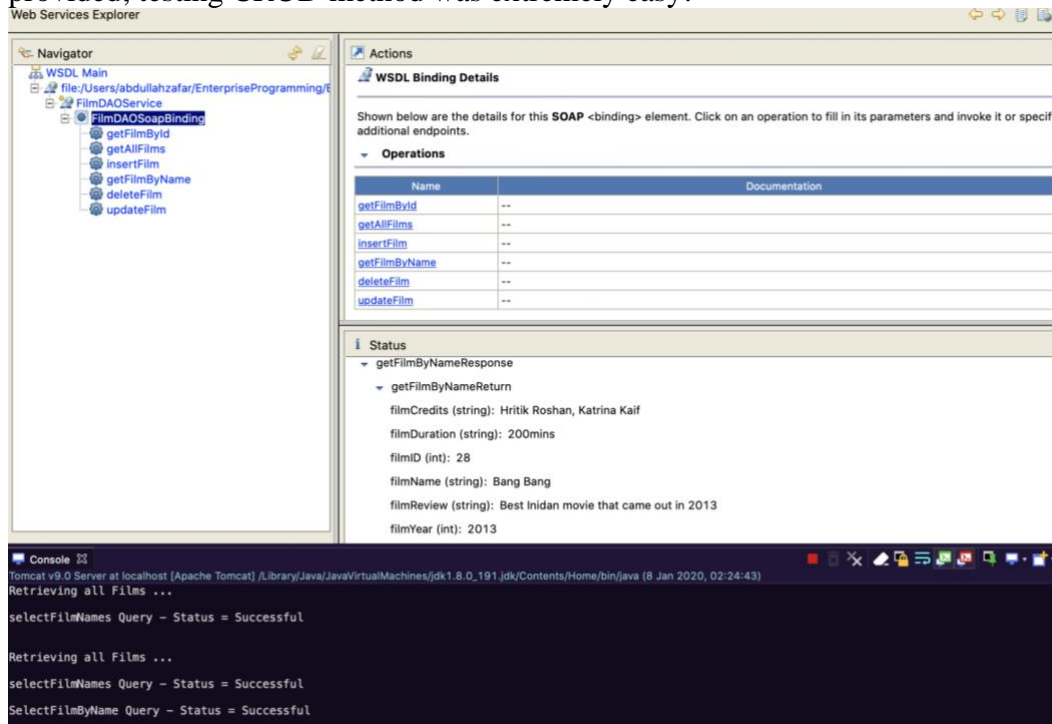
Film Id	Film Name	Film Year	Film Duration	Film Credits	Film Reviews
1	Jumanji 3	2017	210mins	Kevin Hart, Dwayne Johnson	action packed

Figure 1- Image showing the use of GET to retrieve data

There are many advantages of using Http, one of which is that the server doesn't need to know anything about the client; hence any technology can be used to respond to Http Requests. The major disadvantage of Http is that it's not a reliable protocol; therefore, it doesn't guarantee a response being delivered. This also means that it is possible to deal with problems such as system failures which are both expensive to resolve and time consuming.

## WSDL:

WSDL is used to describe network services in an XML format. WSDL includes a binding for Http GET and POST requests in order to interact between a Web Browser and a Web Site (Citeseerx.ist.psu.edu, 2020). This part of the project was based heavily on the DAO as the web service was generated by Eclipse; however, the functionality came mainly from the DAO and film model. With the interface provided; testing CRUD method was extremely easy.



## API:

API in simple terms is tools for building an application. In the project, jQuery library was used for the retrieval of data from the database and to submit the http requests without having to refresh the browser. API helped mainly with the functionality as it provided the building blocks; which made the whole process of development speedy. A slight disadvantage of using the API was that a descent amount of knowledge was required not just to get the API to work but to use the correct method required for the completion of the project. The use of API's affects the future proofing of the project as API's change all the time and backward compatibility is not feasible. i.e. non-updated API can cause a project to not work once the grace period is over.

The project is heavily based on jQuery UI as it provides the interface to test the functionality of the CRUD method. JQuery eased the whole process by providing the api which acted well as a template to fulfil all the requirements of the project. On the other hand, better API's such as DataTables could have been implement for more functionality and for an enhanced modern look to the assignment. Furthermore, DataTables also provides functionality for the basic, yet extremely common features like Ordering, paging and advanced searching; which unfortunately, jQuery UI does not.

In addition, DataTables provide a detailed description to prevent security threats like SQL injection, Cross Site request and privilege escalation. This feature is vital for any product as it provides a sense of relief to the users. In comparison, jQuery lacks this feature which means more time is required to find the adequate code to implement security features; which works for their API.

## Google App Engine – Cloud Database

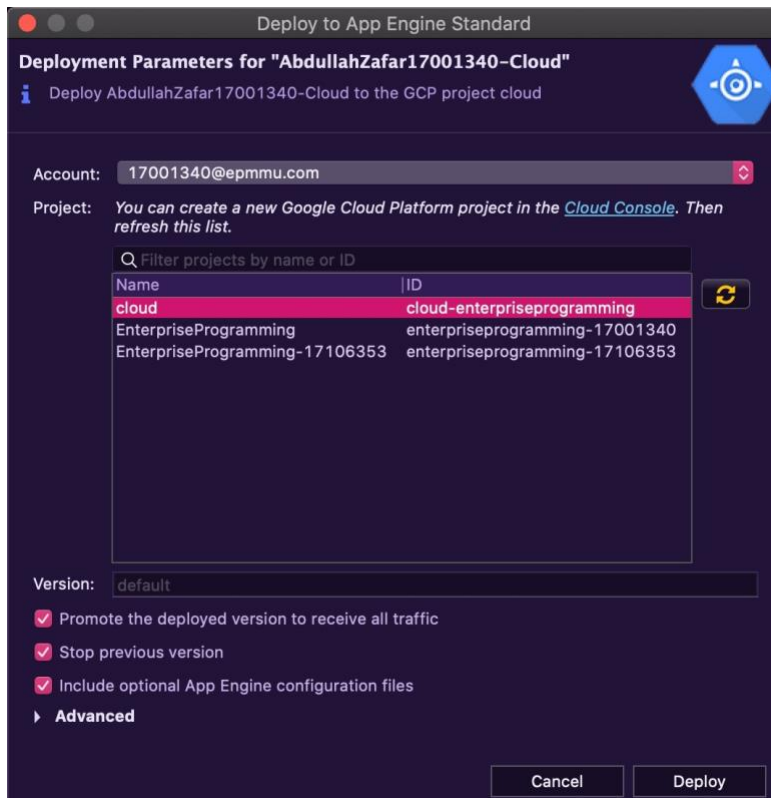


Figure 2 - Deploy Google App Engine

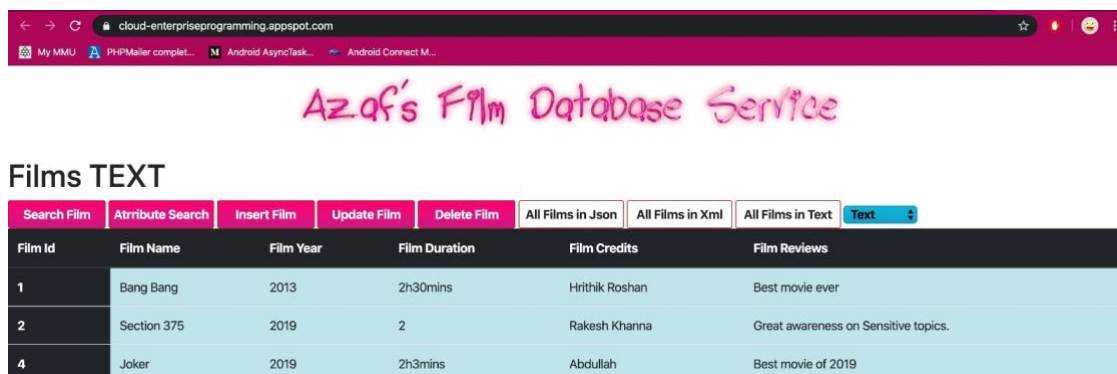


Figure 3 - Google App Engine

Figure 2 above shows the program running on Google App Engine. The engine is able to make use of all CRUD methods and retrieve the data as well as editing it. The URL is compiled when creating the Google Project and creating a client name.

## General Principles:

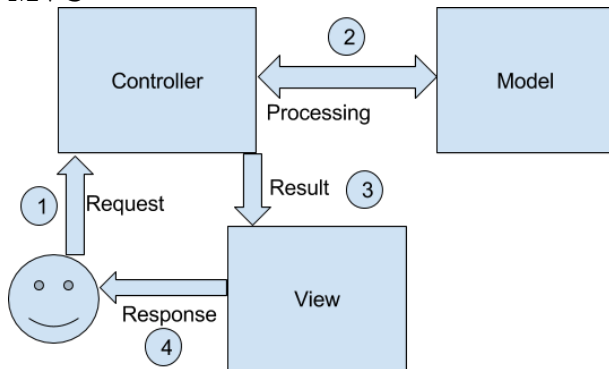
### Comments-

Comments are an essential part of any software related products as they explain the programmers coding decisions. They are added not just to improve the readability but to explain the logic behind the code rather than the code itself. Comments can also be used in pseudocode form which excludes the worry of correct language syntax; allowing the programmer to concentrate on the logic.

## Design Pattern:

Design Pattern is an ideal repeatable solution which prevents frequently occurring problems in a project's design. Nowadays, design pattern is a common technique used by almost every software developer as they don't just help others understand the code, but also explains the foundation and provides a template; which speeds up the process and aids in any further development required in the future for a new developer.

## MVC



*Figure 4-MVC Application Architecture (O’Keeffe and Ó Cinnéide, 2020)*

An example of a design pattern I used for my project is MVC. Figure 1 shows the structure used to implement MVC.

M- Model represents a java object

V- View presents the data the model contains whenever requested

C- Controller is responsible for updating the view whenever the data changes and handles any requests made by the user.

MVC ensures future proofing as it works irrespective of the device. This is beneficial as the complete consistency is assured throughout the development process; the product can be catered according to the customers’ needs. MVC is one of the top priorities in software development as it enables modularity and makes the code more readable and manageable. Furthermore, in my project MVC was the main root, as it helped with the testing before the project was finally deployed to google cloud; this not just prevented unnecessary deployment but also helped with time management.

### Data Accessor Object

DAO was one of the most important factors of the assignment. DAO when used with JDBC allows the execution of queries, manipulation of data stored in a database locally or on the cloud and reduces data redundancy. On the contrary, DAO has some disadvantages; for example, it forces developers to use multiple queries to retrieve specific information which can be achieved easily using a single SQL set operation.

### Data Transfer Object

DTO carries data between different processes to reduce the number of method calls. DTO is commonly used when a request is made by the client to the server; the server then communicates with the database and populates the object which is then retrieved and displayed to the client. i.e. `getAllFilms()` allows a client to retrieve all films without having to request each attribute using multiple calls which can be very costly.

In hindsight, time management was an issue that affected the amount of design patterns implemented into my work. I would have liked to use connection pooling as it allows the connection to be reused rather than it being recreated whenever it’s requested. Furthermore, it provides benefits like improved connection due to minimized stale connections.



Another design pattern that I would definitely implement would be Singleton because The Singleton Design Pattern aims to keep a check on initialization of objects of a particular class by making sure of that only one instance of the object exists throughout the Virtual Machine. In addition, a Singleton class also provides one unique access point to the object so that each subsequent call to the access point returns only that particular object.

The lack of design patterns, refactoring and composition can result in major risks i.e. a complete redevelopment of the entire project which is both costly and time consuming.

### Refactoring:

```
1 package Utils;
2
3 import javax.servlet.http.HttpServletResponse;
4
5 public class util {
6
7     public static String formatType(HttpServletResponse response, String format) {
8
9         String outputPage;
10
11         if("xml".equals(format)) {
12             response.setContentType("text/xml");
13             outputPage = "/WEB-INF/results/film_xml.jsp";
14         }
15         else if("json".equals(format)) {
16             response.setContentType("text/json");
17             outputPage = "/WEB-INF/results/film_json.jsp";
18         }
19         else {
20             response.setContentType("text/html");
21             outputPage = "/WEB-INF/results/film_text.jsp";
22         }
23
24         return outputPage;
25     }
26 }
27 }
```

Figure 5- Code used to show refactoring

```
1 package Servlet;
2
3 import java.io.IOException;
4
5 @WebServlet("/getAllFilms")
6 public class getAllFilms extends HttpServlet {
7     private static final long serialVersionUID = 1L;
8
9     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
10
11         FilmDAO filmdao = new FilmDAO();
12         ArrayList<Film> allFilms = filmdao.getAllFilms();
13         System.out.println("Array List: " + allFilms);
14         String format = request.getParameter("format");
15         String outputPage = util.formatType(response, format);
16         RequestDispatcher rd = request.getRequestDispatcher(outputPage);
17         rd.include(request, response);
18     }
19
20     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
21         doGet(request, response);
22     }
23 }
24 }
```

Figure 6- Use of utils to show correct format

Refactoring is a term used to improve the design of a program without altering its behaviour (JournalDev, 2020). Refactoring increases the maintainability and helps improve the design; hence reducing the cost. In my project, refactoring was used to avoid the repetition of a condition used by all the servlets. This not just improved readability but improved the overall quality of the code too by reducing the complexity. Using the correct technique also aided with debugging and helped to extend and maintain the code.

### Conclusion:

The project allowed us to anticipate how the programme will be used in the real world and gave us a better understanding of its features and functionality. The different types of methods to access and invoke data such as RESTful, SOAP/WSDL and through Google App Engine.

Using RESTful to interact with the data is very beneficial. The majority of API's with the variety of different user interfaces use RESTful, meaning RESTful is a lot more fluid than SOAP as it can be accessed on a variety of different platforms.

SOAP and WSDL rely heavily on XML, meaning the documents are very heavy due to the document passing of a lot of XML. It also means that in SOAP, a lot more code is used than actually needed.

The use of Google App Engine is also extremely beneficial as it allows multiple access to a project, where the project can be altered by different users at the same time. Although this maybe a benefit, some may experience trouble with this as different people have different coding standards and techniques and it might make the code a little hard to understand due to different approaches.

#### **References:**

O’Keeffe, M. and Ó Cinnéide, M. (2020). *Search-based refactoring for software maintenance*.

JournalDev. (2020). *MVC Design Pattern - JournalDev*. [online] Available at:  
<https://www.journaldev.com/16974/mvc-design-pattern> [Accessed 10 Jan. 2020].

Citeseerx.ist.psu.edu. (2020). [online] Available at:  
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.460.467&rep=rep1&type=pdf> [Accessed 10 Jan. 2020].