

Critical Analysis

Introduction

The purpose of this report is to describe in detail the techniques and practices used during the practical section of the assignment. This assignment has mostly been about showing the various different ways that a program can be done. During the practical section of this assignment, this was shown through the development of client and locally based software as well as server and cloud-based software. This report will also attempt to detail how each of the techniques and practices were implemented and why they were chosen over an alternative.

Techniques

Many different techniques can be used during software development. These can include the various design patterns, refactoring and even code documentation.

Design Patterns

Design Model

During software development, programmers will often use a design model during production. These models lay out how development will be organised. One of the most common of these is the waterfall method. This sequential approach takes the developer through requirements, design, implementation, verification to maintenance. Working down from one to the other.

Although a popular method, it was decided that the waterfall method may not fit the assignment well. After some research, I decided for the assignment that the best option was the incremental build model. Moving from design, implementation to incremental testing, with small amounts added each time. This model permitted each section of the assignment to be pushed a little bit at a time without too many concerns.

MVC

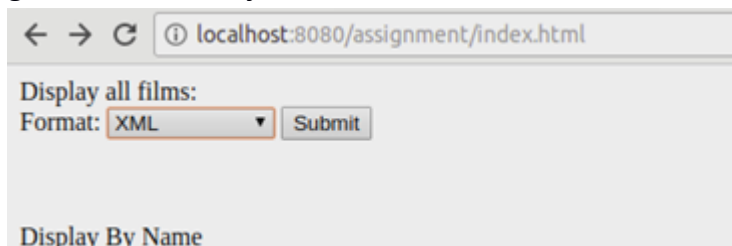
The Model-View-Controller (MVC) is an example of one of the techniques that has been implemented into the practical section of the assignment. Referring to an architectural design pattern used with user interfaces. The MVC splits an application into three individual sections. Figure one shows how MVC is used to split the output of the web projects to show three possible formats of code. One in JSON, another in XML and the last in a normal string text form. This was done by entering 'format=json' line into the URL by the user with JSON being switched for xml. String was also implemented but mostly used as a default until JSON was implemented for this feature.

```
// MVC Pattern
request.setAttribute("films", films);
String format = request.getParameter("format");
String outputPage;
if ("xml".equals(format)) {
    response.setContentType("text/xml");
    outputPage = "/WEB-INF/results/films-xml.jsp";
} else if ("json".equals(format)) {
    response.setContentType("application/json");
    outputPage = "/WEB-INF/results/films-json.jsp";
} else if (format == null){
    response.setContentType("application/json");
    outputPage = "/WEB-INF/results/films-json.jsp";
} else {
    response.setContentType("text/plain");
    outputPage = "/WEB-INF/results/films-string.jsp";
}

RequestDispatcher dispatcher =
    request.getRequestDispatcher(outputPage);
dispatcher.include(request, response);
```

[Figure 1]

I was also later able to implement a system with buttons and dropdown boxes that permitted the project to auto add these format choices to the URL and update the page. This had the same effect as manually inputting the format choice to the URL, however it also made the option to do so clearer as well as make the feature more accessible to those who do not know the code and in general make the system easier to use. This feature is shown working in figures 2 and 3.



[Figure 2]



[Figure 3]

Builder

One of the major features of the project is that of builders. These method patterns are designed to clean up code by removing the need to specify an exact class. Instead, objects are built which directly reference to the original class. Rather than writing the code out multiple times, builders allow the user to create multiple versions of one by using objects.

In the assignment, I use a builder to control the Films. This allows a film object to be set. By running code through objects to the factory class, the assignment is able to drop large amounts of unnecessary code. Objects can also be set with variables.

```
////// Auto generate id ////  
  
// Take all the films and put them into an arraylist  
ArrayList<Film> flist = dao.getAllFilms();  
  
// Counts the arraylist  
int count = flist.size();  
count = count+1;  
  
// Uses the count of the number of rows to auto set the id  
String id = String.valueOf(count);  
  
//////  
  
// Gets the other parameters from what is entered.  
String title = request.getParameter("title");  
String year = request.getParameter("year");  
String director = request.getParameter("director");  
String stars = request.getParameter("stars");  
String review = request.getParameter("review");  
  
// Create a Film object  
Film f1 = new Film(id,title,year,director,stars,review);
```

[Figure 4]

Insert a film
Insert a film:
Title: THE WRATH OF KHAN Year: 1982 Director: Nicholas Meyer Stars: Shatner, Leonard Nimoy Review: The second Star Trek m

[Figure 5]

```
{["id":"1312","title":"STAR TREK II: THE WRATH OF KHAN","year":"1982","director":"Nicholas Meyer","stars":"William Shatner, Leonard Nimoy","review":"The second Star Trek move is swift, droll and adventurous, not to mention appealingly gadget-happy"]}
```

[Figure 6]

1 • select * from films where title like "%khan%"

id	title	year	director	stars	review
1312	STAR TREK II: THE WRATH OF KHAN	1982	Nicholas Meyer	William Shatner, Leonard Nimoy	The second Star Trek move is swift, droll and ad...

[Figure 7]

Dependency Injection

Dependency Injection refers to a technique where objects are used to supply dependencies for another object. In the assignment, this technique links and works closely with the factory class. In the assignment, I use variables in an object to allow a user to enter data through elements on a webpage. Figures 4 to 7 show how this was implemented and tested. In the assignment, this system works with the getters and setters on the "Film" class to tidy up the program and allow the developer to call the class and its functions as many times as they wish without rewriting the code. Instead all they need to do is create a new object which will then connect with the Film class.

```

14 String id;
15 String title;
16 String year;
17 String director;
18 String stars;
19 String review;
20
21 public String getId() {
22     return id;
23 }
24 public void setId(String id) {
25     this.id = id;
26 }
27 public String getTitle() {
28     return title;
29 }
30 public void setTitle(String title) {
31     this.title = title;
32 }
33 public String getYear() {
34     return year;
35 }
36 public void setYear(String year) {
37     this.year = year;
38 }
39 public String getDirector() {
40     return director;
41 }
42 public void setDirector(String director) {
43     this.director = director;
44 }
45 public String getStars() {
46     return stars;
47 }
48 public void setStars(String stars) {
49     this.stars = stars;
50 }

```

[Figure 8]

Refactoring

During the assignment, I have always moved any code that may become useful later on into a method. For example, in the "Film" class, I have added a function called "toString()". The purpose of this method is to convert the JSON from an object into MySQL ready to be entered into the database. This code could have been typed multiple times, however it was later decided that the function would be better as it would clean the assignment of unnecessary repeating code and allow easy calling of that function on any string [Figure 9].

```

16 }
17 @Override
18 public String toString() {
19
20     return "\"" + id + "\", \"" + title + "\", \"" + year + "\", \"" + director
21         + "\", \"" + stars + "\", \"" + review + "\"";
22
23 }
24 }

```

[Figure 9]

DAO

The data access object (DAO) is a highly useful object that permits that provides connection with a database or similar through the use of an abstract interface. Through the assignment, I have created two DAOs.

The first DAO is set to connect to the Mudfoot server. This DAO is able to connect directly to a MySQL database hosted on the Mudfoot server [Figure 9]. Once here the DAO will wait for further instructions. The developer can then activate individual functions in the DAO which will then send

MySQL queries and updates directly to the database and return any data if necessary. Figures 10 and 11 show how after initiating the DAO, the program then calls the “getAllFilms” function and stores the result in an ArrayList. This function when triggered will activate its MySQL which will return all films currently stored in the database before then sending that data back to the ArrayList ready to be used.

After this was implemented, I then reused the DAO for implementation with AJAX. AJAX is highly useful as, when implemented correctly, it can speedup calls between the newly built servlets for each command in the DAO and the webpage.

The second DAO is a bit different but is designed to do more or less the same. As the Google Cloud app setup does not use SQL, some changes had to be made to adapt the DAO to the necessary settings. In addition to changes to the DAO, I have also had to make some changes to the way the DAO is called. Similarly to the AJAX, this system calls servlets which then control the adding and removing from the datastore [Figure 12]. One issue that had to be resolved however was that Google prefers an ID to be a long and the JSP took in a string. In order to fix this issue, the assignment was fitted with a number of checks which will confirm that no number has been entered and remove any none number before then parsing the string into a long, ready to be entered into the datastore.

```

1 package assignmentServ;|
2 import java.sql.Connection;|
9
10
11 public class FilmDAO {
12
13     Film oneFilm = null;
14     Connection conn = null;
15     Statement stmt = null;
16
17     public FilmDAO() {}
18
19
20 private void openConnection(){
21     // MySQL driver
22     try{
23         Class.forName("com.mysql.jdbc.Driver").newInstance();
24     } catch (Exception e) { System.out.println(e); }
25
26     // Connection to mudfoot database
27     try{
28         conn = DriverManager.getConnection
29         ("jdbc:mysql://mudfoot.doc.stu.mmu.ac.uk:3306/????????user=??????&password=??????"); [Figure 10]
30         stmt = conn.createStatement();
31     } catch (SQLException se) { System.out.println(se); }
32     }
33 private void closeConnection(){
34     try {
35         conn.close();
36     } catch (SQLException e) {
37         e.printStackTrace();
38     }
39 }
40
41

```

```
FilmDAO dao = new FilmDAO();|
```

```
ArrayList<Film> films = dao.getAllFilms(); [Figure 11]
```

```

// Gets all films
public ArrayList<Film> getAllFilms(){

    ArrayList<Film> allFilms = new ArrayList<Film>();
    openConnection();

    // Create select statement and execute it
    try{
        String selectSQL = "select * from films";
        ResultSet rs1 = stmt.executeQuery(selectSQL);
        // Retrieve the results
        while(rs1.next()){
            oneFilm = getNextFilm(rs1);
            allFilms.add(oneFilm);
        }

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

    return allFilms;
}

```

[Figure 12]

```

1 package assignment;
2
3 import java.util.List;
4
5 public enum FilmDAO {
6     INSTANCE;
7
8     public List<Film> listFilms() {
9         EntityManager em = EMFService.get().createEntityManager();
10        // read the existing entries
11        Query q = em.createQuery("select m from Todo m");
12        List<Film> todos = q.getResultList();
13        return todos;
14    }
15
16    public void add(String id, String title, String year, String director, String stars, String review) {
17        synchronized (this) {
18            EntityManager em = EMFService.get().createEntityManager();
19            Film newFilm = new Film(id, title, year, director, stars, review);
20            em.persist(newFilm);
21            em.close();
22        }
23    }
24
25    public List<Film> getFilms(String userId) {
26        EntityManager em = EMFService.get().createEntityManager();
27        Query q = em.createQuery("select t from Todo t");
28        q.setParameter("userId", userId);
29        List<Film> films = q.getResultList();
30        return films;
31    }
32
33    public void remove(long id) {
34        EntityManager em = EMFService.get().createEntityManager();
35        try {
36            Film todo = em.find(Film.class, id);
37            em.remove(todo);
38        } finally {
39            em.close();
40        }
41    }
42 }

```

[Figure 13]

Libraries and the APIs

Using libraries is a highly useful technique when programming. The JSON library is one of the libraries that has been used in order to provide a product that does the job but remains simple and easy to use. Using the JSON library's functions means that the assignment is not using developer made functions which would make the code unnecessarily long and untidy. Instead, the code calls functions in the JSON library. Thereby doing the same job but far more cleanly.

The assignment also uses the servlet-api library which allows servlets to be implemented and used without the need to build them manually. This again lowers the amount of unnecessary code while still meeting the objective.

While building the assignment, a number of difficulties arose surrounding the AJAX. The largest of which was that each function had been built manually. Only later was it discovered that this could have been done through jQuery. Though the AJAX works fine as it is, the code is rather long and messy. If jQuery had been used however, functions could be called from the library, rather than manually entered. This would have saved a large amount of time as well as made the code cleaner and easier to read.

Documentation

Ensuring the code is readable is one of the most important parts of software development. If badly written, programmers that may wish to use or read your code. If very badly written the original writer themselves would be unable to understand what is written.

In order, therefore, to make certain that code is clear and understandable, documentation is required. In software development, the primary form of documentation of code is through developer comments. These comments can sit next to the code in the compiler and yet code or other things said in these will not be added to the program. Java, however, has another feature that permits further documentation of a program. When added, Javadoc can generate API documentation in a HTML format from a program's source code.

In the assignment, I have used both of these features. In order to ensure that I have a good understanding of what the code is doing, as well as better clarify what does what, I have implemented numerous comments. I have also added Javadoc documentation in order to describe in greater detail what could not be written in comments.

Another problem that can come up during programming is that variables can be given inappropriate names. When initially working on the assignment, many variables in my code were given names that were very similar to others or had nothing to do with what they did. Although the variables worked, I decided this was unclear and difficult to follow. Therefore, I later went through the code and began changing variable and object names so that they were better fitted for purpose. These features are now far clearer and make greater sense when reading the code.

Cloud Computing

In recent days, cloud computing has become more frequently used and more popular. Two of the largest tech giants in the world, Google and Amazon have both built their own cloud service system in order to take advantage of this fast moving section of the computing industry.

In this assignment, one of the sections was to implement the application onto a Google cloud server. By using the Google App Engine and Eclipse's Google Web Application the application has

successfully been rebuilt to use the Google Cloud systems. In order for this to be done however, the Google section's servlets had to be rebuilt and the DAO completely redesigned. The new DAO is set to send the new film data through to the Google Cloud, but in order to do this, the queries themselves had to be edited to work with the datastores used by Google.

What makes good code?

Though "good" is a subjective term, there are numerous practices and techniques that programmers, both amateur and professional abide by in order to provide the highest quality of code possible.

Naming

One major requirement for good code is, as touched on earlier, good naming of variables and objects. If variables and objects are not well named, they are likely to confuse whomever is reading the code. This could even lead to the developer themselves becoming confused about what they are doing. However, another good practice is to stick to naming conventions regarding the way a variable is written. In the assignment, variables and objects all use the "camelCase" style. This allows anyone reading the code to easily recognise what is a variable and what isn't. Even permitting one to determine what is a variable and what is a function.

Another major important practice is that of commenting and documentation. Using Javadoc to document the code has enabled the ability for a user to use the Javadoc reader. The reader will ignore the code of the document and instead just return the Javadoc data. As Javadoc returns in a format similar to HTML, the code is easy to read and neat. It is also better than having to read through the code looking for Javadoc messages. Especially if the reader is not a programmer.

Commenting is just as, if not more important than using Javadoc, mostly as Javadoc, as the name suggests, is designed only for the Java language. During the assignment, comments are frequent and are designed to ensure that whomever is reading is certain about what is doing what. They are also occasionally there to split up the code into sections in order to tidy up the code and make it easier to follow.

Proof assignment works:

Eclipse Console-Based:

The aim of this first section was to build the basic application and have it run in a local controller. I also implemented a scanner to aid in testing. The program triggers the correct functions when activated. I also created an object in the code which would be automatically added to the code when the insertFilm function is triggered. This function also has a system which will check to confirm if the film already exists by comparing the entry's id to what is already in the database. If the file does exist, the SQL exception is caught and an error message is sent to the console. If there is no error, the sytem assumes there are no issues and adds the film.

```
Problems Javadoc Declaration Console Servers Development Mode
Controller (4) [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (12 Jan 2018, 21:05:04)
Do you wish to get all data? 1 for yes, 2 for no
```

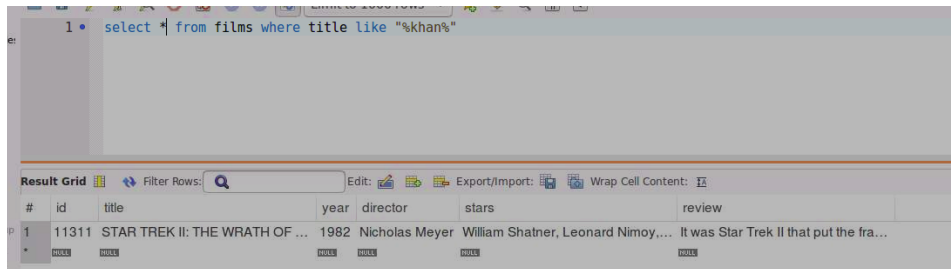
```
Problems Javadoc Declaration Console Servers Development Mode
Controller (4) [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (12 Jan 2018, 21:25:14)
"11271", "MAX MASK", "1997", "SERGIO STIVALETTI", "ROBERT HOSSEIN, ROMINA MONDELLO", "Produced by Italian horror icon Dario Argento, this is a remake of the classi
"11272", "WAXWORKS", "1986", "ANTHONY HICKOX", "ZACK GALLIGAN, DEBORAH FOREMAN", "Deeply black and very gory comedy with a group of teenagers accepting a bet to sp
"11273", "WAXWORK II: LOST IN TIME", "1992", "ANTHONY HICKOX", "ZACK GALLIGAN, MONICA SCHNARRE", "Hilarious sequel with a truly inventive script and very little to
"11274", "WAYNES WORLD", "1992", "PENELOPE SPHERIS", "MIKE MYERS, DANA CARVEY", "Myers and Carvey recreate their roles from TVs Saturday Night Live as Wayne and
"11275", "WAYNES WORLD 2", "1993", "STEPHEN SURJIK", "MIKE MYERS, DANA CARVEY", "The brainless ones strike again, this time turning into concert promoters, hoping
"11276", "WEDLOCK", "1990", "LEWIS TEAGUE", "RUTGER HAUER, MIMI ROGERS", "Clever sci-fi actioneer with Hauer as a master jewel thief caught and imprisoned in a hi-
"11277", "WEEKEND AT BERNIES", "1989", "TED KOTCHEFF", "ANDREW MCCARTHEY, TERRY KISER", "Andrew McCarthy and Jonathan Silverman are two young employees whose plan
"11278", "WEEKEND AT BERNIES 2", "1992", "ROBERT KLANE", "ANDREW MCCARTHEY, TERRY KISER", "Not a brilliant sequel. The two protagonists from the first film (McCart
"11279", "WELCOME TO SARAJEVO", "1997", "MICHAEL WINTERBOTTOM", "STEPHEN DILLANE, WOODY HARRELSON", "Powerful drama set in the Bosnian conflict and loosely based o
"11280", "WES CRAVENS NEW NIGHTMARE", "1994", "WES CRAVEN", "ROBERT ENGLUND, MIKO HUGHES", "Brilliant spin on the NIGHTMARE ON ELM STREET series with the real actr
"11281", "WHATS EATING GILBERT GRAPE", "1993", "LASSE HALLSTROM", "JOHNNY DEPP, LEONARDO DI CAPRIO", "Odd and often enchanting family saga revolving around Johnny
"11282", "WHEN A MAN LOVES A WOMAN", "1994", "LUIS MANDOKI", "MEG RYAN, ANDY GARCIA", "Solid performances from the two leads as Andy Garcia discovers his wives sec
"11283", "WHEN HARRY MET SALLY", "1989", "ROB REINER", "BILLY CRYSTAL, MEG RYAN", "Brilliantly well-observed adult comedy following the characters of the title, tw
"11284", "WHEN SATURDAY COMES", "1996", "MARIA GIESE", "SEAN BEAN, EMILY LLOYD", "Sean Bean (a real-life Blades fanatic) is a brewery worker who dreams of playing
"11285", "WHEN THE BOUGH BREAKS", "1993", "MICHAEL COHN", "ALLY WALKER, MARTIN SHEEN", "Very scary horror thriller with a serial killers crimes coming to light aft
"11286", "WHEN THE BULLET HITS THE BONE", "1995", "DAMIAN LEE", "JEFF WINCOTT, MICHELLE JOHNSON", "Jeff Wincott (Mission Of Justice / Martial Outlaw) plays a doctr
"11287", "WHILE YOU WERE SLEEPING", "1995", "JON TURTLETAUB", "SANDRA BULLOCK, BILL PULLMAN", "Charming and clever romantic comedy with Sandra Bullock spotting her
"11288", "WHITE MANS BURDEN", "1995", "DESMOND NAKANO", "JOHN TRAVOLTA, SIDNEY POITIER", "Considered and committed drama which fluffs a lot of the major issues. Ne
"11289", "WHITE MEN CANT JUMP", "1992", "RON UNDERWOOD", "WESLEY SNIPES, WOODY HARRELSON", "Wesley and Woody are street basketball plays, constantly trying to out-
"11290", "WHITE SANDS", "1992", "ROGER DONALDSON", "WILLEM DAFOE, MICKEY ROURKE", "Intelligent thriller, bolstered by very fine performances. Dafeo is a small towr
"11291", "WHITE SQUALL", "1998", "RIDLEY SCOTT", "JEFF BRIDGES, CAROLINE GOODALL", "Ridley Scott (Alien / Black Rain) turns his attentions to a true story in which
"11292", "WHITE TIGER", "1995", "ROBERT MARTIN", "GARY DANIELS, GARY HIROYUKI-TAGAWA", "British martial arts champ Gary Daniels takes the lead in this beat-em-up.
"11293", "WHO FRAMED ROGER RABBIT?", "1988", "ROBERT ZEMECKIS", "BOB HOSKINS, CHRISTOPHER LLOYD", "Amazing blend of animation and live-action, set in a world where
"11294", "WHOS HARRY CRUMB?", "1989", "PAUL FLAHERTY", "JOHN CANDY, JEFFREY JONES", "Candy gets most of the best lines in this average comedy, playing a disastrous
"11295", "WHOS THE MAN", "1992", "TED DEMME", "DOCTOR DRE, ED LOVER", "Two bumbling friends join the police force and get involved in a murder case in this comedy.
"11296", "WILD BILL", "1995", "WALTER HILL", "JEFF BRIDGES, ELLER BARKIN", "Plodding and overlong film retelling of the Wild Bill Hickock story, helped by Jeff Bri
"11297", "WILD CACTUS", "1992", "JAG MUNDHRA", "INDIA ALLEN, JAGD NAUGHTON", "A married couple (India Allen and American Werewolf In London star David Naughton) a
"11298", "WILD ORCHID", "1990", "ZALMAN KING", "MICKEY ROURKE, CARRE OTIS", "Mickey Rourke hoped to repeat the success of 91/2 Weeks with this erotic drama. Carre
"11299", "WILD ORCHID 2: TWO SHADES OF BLUE", "1991", "ZALMAN KING", "WENDY HUGHES, TOM SKERRITT", "A sequel in name only, although directed by the same man. A yc
"11300", "WILLOW", "1988", "RON HOWARD", "VAL KILMER, JOANNE WHALLEY", "Fantasy adventure produced by George Lucas, creator of Star Wars. Nothing was skimmed on th
"11301", "WITCH HUNT", "1994", "PAUL SCHRADER", "DENNIS HOPPER, JULIAN SANDS", "A sequel to the clever Cast A Deadly Spell with Dennis Hopper taking over from Fred
"11302", "WITHIN THE ROCK", "1996", "GARY J. TUNICLIFFE", "XANDER BERKELEY, CAROLINE BARCLAY", "Alien goes underground in this average sci-fi thriller helped by sc
"11303", "WITHNAIL & I", "1986", "BRUCE ROBINSON", "RICHARD E. GRANT, PAUL MC GANN", "Comedy which has developed a definite cult following over the years. The titl
"11304", "WIZARDS OF THE LOST KINGDOM", "1985", "HECTOR OLIVERA", "BO SVENSON, VIDAL PETERSON", "Fantasy adventure with a young boy teaming up with an over-the-hil
"11305", "WOLF", "1994", "MIKE NICHOLS", "JACK NICHOLSON, MICHELLE PFEIFFER", "Jack Nicholson finds himself changing after being bitten by a wolf on a country road.
"11306", "WYATT EARP", "1993", "LAWRENCE KASDAN", "KEVIN COSTNER, DENNIS QUAYD", "Three hours long but always gripping version of the life of Wyatt Earp. It shows
"11307", "YEAR OF THE DRAGON", "1985", "MICHAEL CIMINO", "MICKEY ROURKE, JOHN LONE", "Tough and violent underworld saga based on the true life story of New York cc
```

```
Do you wish to search by name? 1 for yes, 2 for no
1
Enter film to be searched for:
star
"10377", "FIST OF THE NORTH STAR", "1996", "TONY RANDEL", "GARY DANIELS, COSTAS MANDYLOR", "Live-action adaptation of the famous Japanese Manga. British martial ar
"10742", "ONE TOUGH BASTARD", "1995", "KURT WIMMER", "BRIAN BOSWORTH, BRUCE PAYNE", "Somewhat indelicate title but definitely appropriate, considering the characte
"10949", "STAR TREK: FIRST CONTACT", "1996", "JONATHAN FRAKES", "PATRICK STEWART, JONATHAN FRAKES", "One of the best of the Star Trek movies, an action-packed tale
"10950", "STAR TREK: GENERATIONS", "1994", "DAVID CARSON", "PATRICK STEWART, WILLIAM SHATNER", "An astral shift in time brings Captains Picard and Kirk together in
"10951", "STAR TREK III: THE SEARCH FOR SPOCK", "1984", "LEONARD NIMOY", "WILLIAM SHATNER, DEFOREST KELLEY", "Another excellent big-screen adventure for the crew c
"10952", "STAR TREK IV: THE VOYAGE HOME", "1986", "LEONARD NIMOY", "WILLIAM SHATNER, LEONARD NIMOY", "One of the better entries in the big-screen Star Trek series.
"10953", "STAR TREK VI: THE UNDISCOVERED COUNTRY", "1991", "NICHOLAS MEYER", "WILLIAM SHATNER, LEONARD NIMOY", "One of the better outings for the Enterprise, with
"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 pl
"10955", "STARGATE", "1994", "ROLAND EMMERICH", "KURT RUSSELL, JAMES SPADER", "Spader is the scientist who is attached to Kurt Russells elite commando team as the
Would you like to input a film? 1 for yes, 2 for no
```

```
"10955", "STARGATE", "1994", "ROLAND EMMERICH", "KURT RUSSELL, JAMES SPADER", "Spader is the scientist who is attached to Kurt Russells elite commando team as the
Would you like to input a film? 1 for yes, 2 for no
1
"11311", "STAR TREK II: THE WRATH OF KHAN", "1982", "Nicholas Meyer", "William Shatner, Leonard Nimoy, DeForest Kelley", "It was Star Trek II that put the franchis
```

```
No
Would you like to input a film? 1 for yes, 2 for no
1
"11311" , "STAR TREK II: THE WRATH OF KHAN" , "1982" , "Nicholas Meyer" , "William Shatner, Leonard Nimoy, DeForest Kelley" , "It was Star Trek II that put the franchi
Error. The film may already exist
```

After a rerun:



The screenshot shows a database query interface. At the top, a SQL query is entered: `select * from films where title like "%khan%"`. Below the query, a "Result Grid" displays the results. The grid has columns: #, id, title, year, director, stars, and review. One row is visible with the following data: # 1, id 11311, title STAR TREK II: THE WRATH OF KHAN, year 1982, director Nicholas Meyer, stars William Shatner, Leonard Nimoy, DeForest Kelley, and review It was Star Trek II that put the franchise on the map.

#	id	title	year	director	stars	review
1	11311	STAR TREK II: THE WRATH OF KHAN	1982	Nicholas Meyer	William Shatner, Leonard Nimoy, DeForest Kelley	It was Star Trek II that put the franchise on the map

Google Cloud:

This next section was all about implementing the Google Cloud version of the program. To do this, I created a new Google Web Application and connected it to the Google Appengine. This section of the project also required updating various classes. The DAO required a near complete redesign in order to properly implemented the add and remove functions required.

Film

You have a total number of 0 Film.

ID	Title	Year	Director	stars	review	clear
----	-------	------	----------	-------	--------	-------

New film

id:

Title:

Year:

Director:

Stars:

Review:

Film

You have a total number of 1 Film.

ID	Title	Year	Director	stars	review	clear
1	STAR TREK II: THE WRATH OF KHAN	1982	Nicholas Meyer	William Shatner Leonard Nimoy	Cool	Done

New film

id:

Title:

Year:

Director:

Stars:

Review:

Film

You have a total number of 1 Film.

ID	Title	Year	Director	stars	review	clear
1	STAR TREK II: THE WRATH OF KHAN	1982	Nicholas Meyer	William Shatner Leonard Nimoy	Cool	Done

New film

id	<input type="text" value="2"/>
Title	<input type="text" value="STAR WARS"/>
Year	<input type="text" value="1977"/>
Director	<input type="text" value="George Lucas"/>
Stars	<input type="text" value="Mark Hamill Alec Guinness"/>
Review	<input type="text" value="Less Cool"/> x
<input type="button" value="Create"/>	

Film

You have a total number of 2 Film.

ID	Title	Year	Director	stars	review	clear
1	STAR TREK II: THE WRATH OF KHAN	1982	Nicholas Meyer	William Shatner Leonard Nimoy	Cool	Done
2	STAR WARS	1977	George Lucas	Mark Hamill Alec Guinness	Less Cool	Done

New film

id	<input type="text"/>
Title	<input type="text"/>
Year	<input type="text"/>
Director	<input type="text"/>
Stars	<input type="text"/>
Review	<input type="text"/>
<input type="button" value="Create"/>	

Film

You have a total number of 1 Film.

ID	Title	Year	Director	stars	review	clear
2	STAR WARS	1977	George Lucas	Mark Hamill Alec Guinness	Less Cool	Done

New film

id	<input type="text"/>
Title	<input type="text"/>
Year	<input type="text"/>
Director	<input type="text"/>
Stars	<input type="text"/>
Review	<input type="text"/>
<input type="button" value="Create"/>	

Ajax:

The last part of the assignment that was successfully implemented is the AJAX. Using the same basic classes as the controller version, this section is able to trigger newly build insert, getall and getByName servlets which take in a user chosen input and return data from the database where the input matches a title. This part also allows this data to be displayed in one of four ways. Standard GET and POST requests as well as setting the format as JSON and String. The first two also display as JSON as JSON is set as the default option for the format. Though an XML format section was attempted, it was unsuccessful.

← → ↻ ⓘ localhost:8080/assignment/ajax.html ☆ 666 ⋮

searching for TITLE GET

khan

SearchFilm

[{"id":"11311","title":"STAR TREK II: THE WRATH OF KHAN","year":"1982","director":"Nicholas Meyer","stars":"William Shatner, Leonard Nimoy, DeForest Kelley","review":"It was Star Trek II that put the franchise on the right track as the spirit of Gene Roddenberry's 60s television series was harnessed to spectacular effect."}]

searching for TITLE POST

searchTitlePost

[{"id":"11311","title":"STAR TREK II: THE WRATH OF KHAN","year":"1982","director":"Nicholas Meyer","stars":"William Shatner, Leonard Nimoy, DeForest Kelley","review":"It was Star Trek II that put the franchise on the right track as the spirit of Gene Roddenberry's 60s television series was harnessed to spectacular effect."}]

searching for TITLE POST json

searchTitlePostJSON

[{"id":"11311","title":"STAR TREK II: THE WRATH OF KHAN","year":"1982","director":"Nicholas Meyer","stars":"William Shatner, Leonard Nimoy, DeForest Kelley","review":"It was Star Trek II that put the franchise on the right track as the spirit of Gene Roddenberry's 60s television series was harnessed to spectacular effect."}]

searching for TITLE POST plain

searchTitlePostPLAIN

["11311", "STAR TREK II: THE WRATH OF KHAN", "1982", "Nicholas Meyer", "William Shatner, Leonard Nimoy, DeForest Kelley", "It was Star Trek II that put the franchise on the right track as the spirit of Gene Roddenberry's 60s television series was harnessed to spectacular effect."]