REPORT

Enterprise Programming

Joe Allen (20097029)

Design Patterns

I have used a singleton pattern when instantiating my filmDAO objects. I did this because there was no need to create more than one in order for it to serve its purpose (to give access to database operations using JDBC). The constructor in my filmDAO class was made private, and instantiated as seen below:

Text

Description automatically generated



Setting it up like this saves the CPU space that would be required if I were to create a new filmDAO object each time it was required. I used the filmDAO within each servlet method and accessed the relevant method(s) based on the nature of the request i.e. doGet uses getAllFilms(), getFilmByID() and getFilmByName().

Text

Description automatically generatedFurthermore, I have used Data Transfer Objects (DTOs) to simply hold data within my project, for example when creating film objects.

Here, I am using the ‘this’ keyword within the constructor to make establishing Film objects straightforward; by passing in the attributes as parameters.

I have made use of getters and setters for each attribute to function as accessors to film data upon request.

Another design pattern I made use of within the project is connection pooling. This allowed me to find a connection to my database, stored on Mudfoot, through the use of a connection URL, the database credentials and the JDBC connection driver.

Refactoring code is increasingly important as the scale of a project increases, with the increased memory usage. I have refactored my code to combat this by making use of functions for operations that are regularly repeated. A good example of this would be where I have used a modal to display a ‘loading’ message to the user. I have created a ‘closeLoadingModal()’ to save writing the same lines of code over and over again.

Text

Description automatically generated

RESTful Web Service

I used the jQuery and Bootstrap libraries in CSS/JavaScript to provide more functionality and design features. jQuery has allowed me to access event listens and interact with html elements i.e. removeClass, setting display, updating values etc, as well as the ability to make AJAX requests which I used when obtaining film objects to display in forms (i.e. the ‘update film’ form already containing the previous values.)

Text

Description automatically generated

Text

Description automatically generated

Here is an example of a modal I have used. The ‘modal fade’ class gives the modal a fade transition onto the page. The content of the form is contained within the body, with action buttons in the footer.

Text

Description automatically generated

Here is a modal I created to display a success message upon completion of an operation. The default message within modal-body is changed using JavaScript if this modal is used for something else i.e updating a record or deleting.

Text

Description automatically generated

This is a modal that displays a spinner wheel to communicate to the user that the resource is loading, whilst the request happens in the background. The “spinner” class comes from the Bootstrap library.

Text

Description automatically generated

Text

Description automatically generated

**JavaScript**

This method initialises the address and format parameter for a GET request, submits request using ajaxResult method then uses a responseHandler to run the correct method based on the format the user selected.

**Text

Description automatically generated**

ajaxResult initiates a GET request to the server, passing the desired data format in the “Accept” header.

Text

Description automatically generated

The ‘findHandler’ method takes the format and returns the method to be used to construct the data in that format.

Text

Description automatically generated

JAXB is used to convert the list of Film objects into an XML structure. Using a library like this or GSON for JSON format allows complex operations to be performed with a few lines of code and saved me from having to write the functions manually.

Text

Description automatically generated

Similarly to XML, I used the GSON library to convert Film objects to a JSON format:

Text

Description automatically generated

For data in string format, I used “rawData.split(‘[\\n](file:///\\n)’)” to separate each film object, and then used the # symbol as a delimiter to split each attribute.

Text

Description automatically generated

Sets the headings of the table:

Text

Description automatically generated

Initialises the film table:

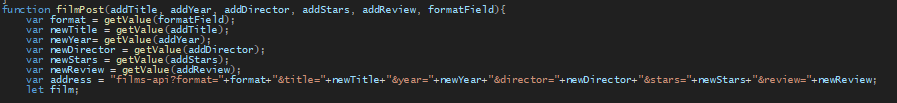
Text

Description automatically generated

Inserts data into a html element i.e. film table, textbox values for update form etc.



Establishes conditions of POST request, sets parameters etc.



Validation of user inputs:

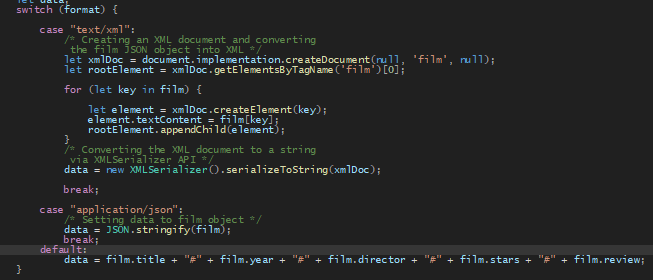
Text

Description automatically generated

Text

Description automatically generatedGraphical user interface, text

Description automatically generatedIf all the validation tests are passed, open POST request:

Switch statement determines how the data returned by the server is formatted:

Text

Description automatically generatedIf HTTP request returns status code 200 & readyState of 4, remove pre-existing tables and display/remove modals. If request returns an error, alert the user.

filmPut method – sets up address and parameters for a PUT request:

Text

Description automatically generated

Text

Description automatically generatedValidation checks for PUT request. If passed, create a Film object, and pass it to ajaxPut:

Text

Description automatically generatedSimilarly to ajaxPost, ajaxPut converts the film object into the relevant format (XML, JSON, String) to send to the server.

Text

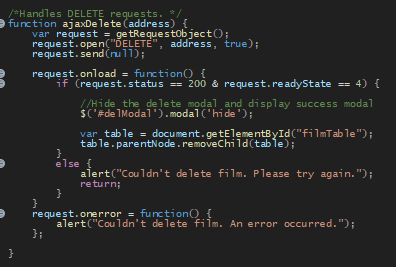
Description automatically generatedIf request is successful, show success message and clear the form. If not, alert the user.

Method to initiate DELETE request

Text

Description automatically generated

Opens DELETE request, success modal displayed upon correct network response:



This method gets the browser-specific request object, and covers all browsers (Internet Explorer 5 and 6 covered in the ‘else if’ statement:

Text

Description automatically generated

Text

Description automatically generatedThe following method uses jQuery to listen for a search request using ID. A GET request is made within the corresponding function to return a film object if the entered ID can be matched to one. If a record is found, a new table is created containing the record. If not, the user is alerted.

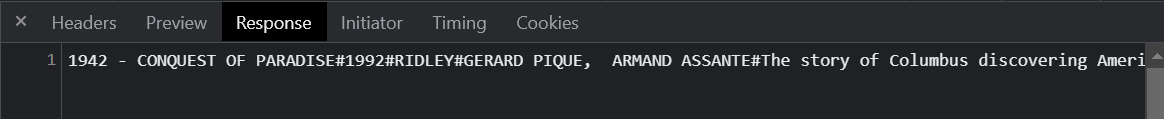
Text

Description automatically generatedA similar search function, but for searching by film title as opposed to the ID. This is important because the title of a film is memorable unlike its ID number which is exclusive to this dataset.

NETWORK REQUESTS:

Graphical user interface, text

Description automatically generatedGET

Text

Description automatically generatedText

Description automatically generated

POST

A screenshot of a computer

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generatedText

Description automatically generated

PUT

GET request made initially to obtain current film data;

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

Parameters in URL (Bad practice – definitely a shortcoming)

Text

Description automatically generated

DELETE

Graphical user interface, text

Description automatically generatedGraphical user interface, application

Description automatically generatedSame as with PUT, a GET request is made to get the film name for display purposes

Text

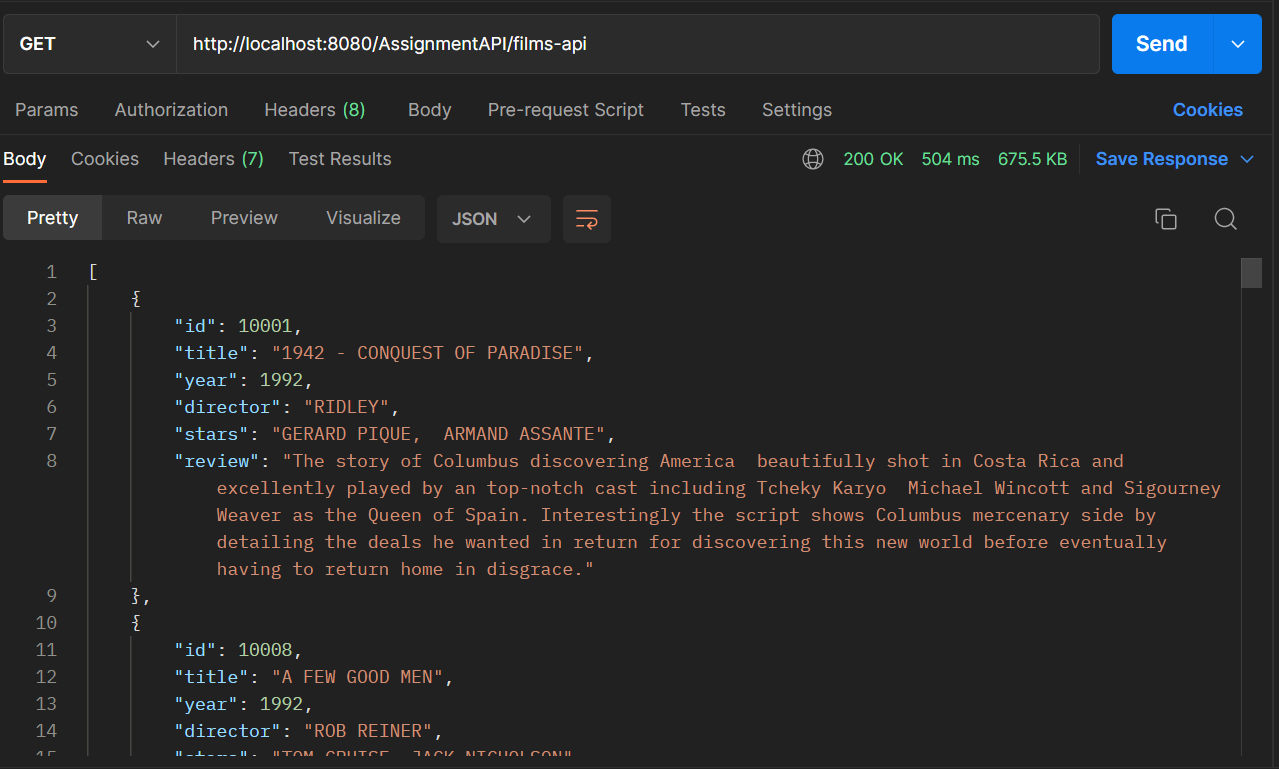
Description automatically generated

POSTMAN screenshots

GET - XML

A screenshot of a computer

Description automatically generated with medium confidence

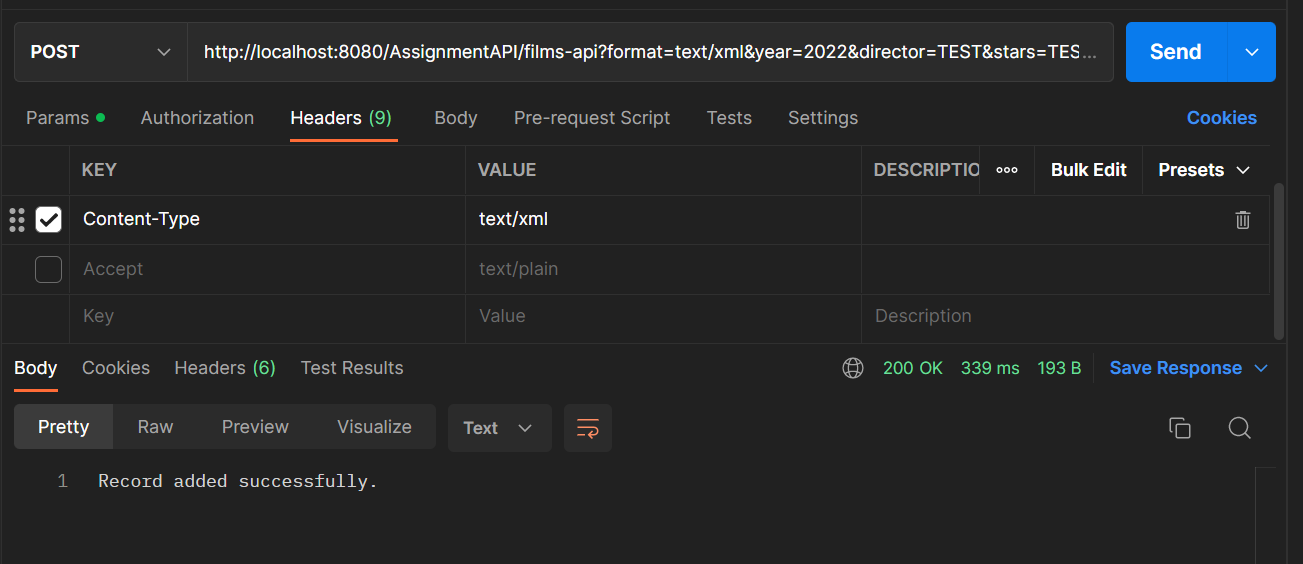
GET - JSON

GET - STRING

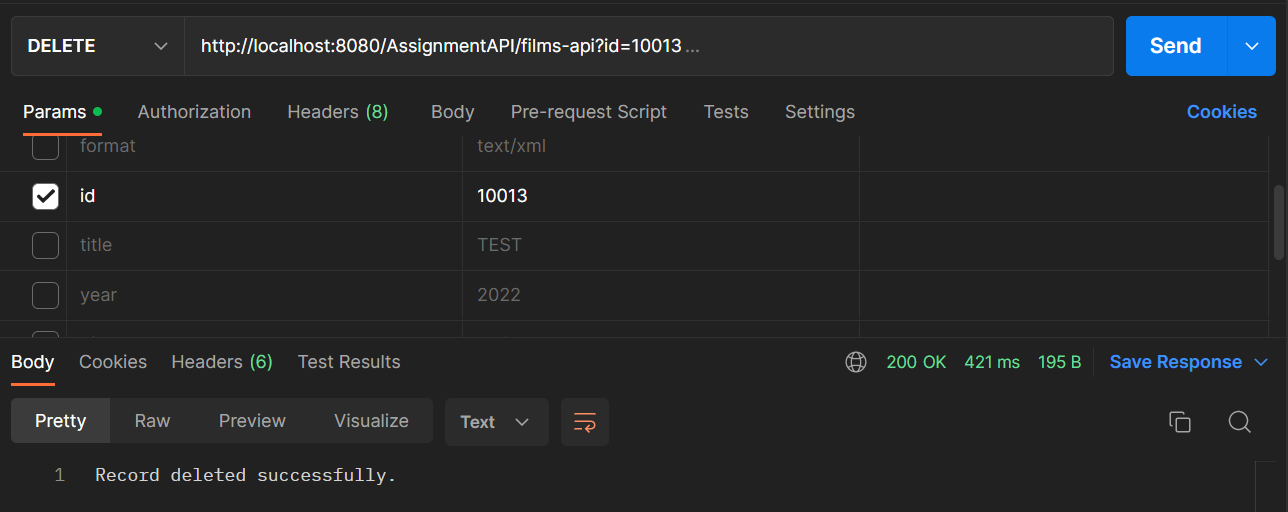
Text

Description automatically generated

POST



DELETE



FRONT-END

Graphical user interface

Description automatically generated

Shape, rectangle

Description automatically generated

A picture containing chart

Description automatically generated

A picture containing calendar

Description automatically generated

“Add Film” modal:

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

“Update film” modal:

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

“Delete film” modal:

Graphical user interface, text

Description automatically generated

Search by ID – modifies the table to show the matching record:

Text

Description automatically generated with low confidence

Search by title – modifies the table to show records with a matching substring:

A picture containing table

Description automatically generated

FilmDAO

Establishing database connection using JDBC driver and connection url + credentials:

Text

Description automatically generated

Gets the next film in a resultSet:

Text

Description automatically generated

Gets ALL films:

Text

Description automatically generated

Search using ID:

Text

Description automatically generated

Search by title:

Text

Description automatically generated

Inserts new film:

Text

Description automatically generated

Updates an existing film using ID:

A screenshot of a computer

Description automatically generated with medium confidence

Deletes film:

Text

Description automatically generated

Cloud Implementation

After many hours of messing about with various errors trying to set my database up with AWS, from “incorrect driver used” to “incorrect character set index” I did not have enough time to fix the problems and get it up and running. The process was more complicated than I thought it would be, and I should have allocated more time to get this up and running.

MVC Application

HomeController – displays “home.jsp” page:

A screenshot of a computer

Description automatically generated with medium confidence

Home.jsp – uses JSTL and Java beans to inject film records onto the page in a table.

Text

Description automatically generated

InsertController – doGet loads Insert.jsp, doPost gets Film object and inserts into the database.

Text

Description automatically generated

Graphical user interface, text

Description automatically generatedInsert.jsp

UpdateController – similar to InsertController but includes ID within film object:

Text

Description automatically generated

Update.jsp – uses JSTL + JBeans to display data about the film being updated. This saves the user from re-entering data = user-friendly

Graphical user interface, text

Description automatically generated

DeleteController – doGet retrieves film data to display in Delete.jsp page, doPost takes ID parameter, conducts deletion, and redirects the user to the homepage.

Text

Description automatically generated

Delete.jsp – carries ID value through a hidden element to be accessed in the DAO.

A screenshot of a computer

Description automatically generated with medium confidence

FilmDAO – All the same methods as the DAO used in the RESTful web service, apart from getFilmByTitle(), which does not exist in the MVC.

Challenges faced:

Throughout the course of this project, there have been many challenges encountered. I would say the biggest challenge has been the complexity but also the depth of the material that we have learnt. I have personally found it a challenge to come in each week and grasp the topic of that week straight away meaning that I have had to go back through a lot of the content during the creation of the assignment to refresh my knowledge. As a result, there was a lot of back and forth and less time spent strictly on implementing code to fulfil the objectives, which has been disrupting. However, I know that with repetition comes experience and knowledge, and I hope that when it comes to working on web projects in the future, the groundwork that has been done during this project will be put to use. I have certainly learnt the foundational knowledge of web services by using HTTP requests and servlets for the first time, along with the anatomy of network requests and how data can be sent in different formats from client-to-server and vice versa.

Secondly, a good portion of my development has come from my own research which has meant learning completely new concepts, for example the use of Bootstrap to use modals in my JSP page and the jQuery library in JavaScript. There was a lot of trial and error whilst learning, which evaluated my perseverance and became time consuming. It came to significant effect, however, as it made using JavaScript a lot simpler due to the quick and easy manipulation of DOM elements that jQuery offers.

There were some things I tried to do that I failed to be able to implement due to the complexity or because I managed my time poorly and did not leave enough time to play around and try to solve any issues. For example, I tried to implement the DataTables plugin for jQuery to add filtering and pagination to my film table. I tried as many different ways as possible to make it work but it proved more difficult than first seemed and was more of a desirable feature than a compulsory one, so I decided against using it. Another example was trying to set up my database on the cloud. Dealing with the aforementioned complications when doing this proved too time-taxing with the time I had left. I even tried multiple platforms: first with Google Cloud and then AWS, but each had their own complications which I could not justify trying to fix when I still had bits of key functionality to work on.

Future Work

I plan to continue working on this assignment past this unit as it is something that can be used in my portfolio when applying for roles in the industry. Here is a list of the improvements I plan to make:

* Make the database live on the cloud through AWS/Google Cloud
* Improve the security of my HTTP requests by including data in the body of the request rather than the URL
* Make the table more interactive and user-friendly by using an API like DataTables
* Add a ‘rating’ column that shows an average star rating
* Add user functionality to allow for publicly accessible reviews
* Add a ‘genre’ column and allow searching by genre
* Searching & sorting by star rating, genre, title, year
* Make design improvements to page structure, colour scheme etc.