

**ENTERPRISE PROGRAMMING ASSIGNMENT**

**Student Name: ABDULLA IBRAHEM**

**Student ID: 19027109**

**Unit Tutor: Mohammed Kaleem**

**CRITICAL ANALYSIS**

The Data Access Object (DAO) pattern is used to separate the logic for accessing the database from the rest of the application. It provides an abstract interface for the database operation. I also used the Open-Close Principal pattern is used to separate the code for connecting to the database from the code for querying the database. This allows the connection code to be reused without modification, while the querying code can be changed to suit different needs. I used the Single Responsibility Principle pattern to separate the various responsibilities of the class into different methods, such as the Open Connection, Close Connection, GetAllFilm, GetFilmByID and newFilm.

Text, application

Description automatically generated

The Factory Method pattern can be seen in the way the getNextFilm method is used to instantiate new film objects from the ResualtSet. It’s Worth nothing that this code is using the old version of JDBC, it is recommended to use the new version which is JDBC 4.0 and above.

There are a few ways to optimize the code in the class DAO, instead of opening and closing the database connection every time a method is called, a connection pool be used to maintain a pool of open connections. This will significantly improve the performance of the application by reducing the time and resources needed to open and close connections. I use a try-catch block to close the connection and statement objects. By using try-with-resources the connection and statement object can be automatically closed without the need for an explicit close () call. Instead of concatenating strings to create the SQL statements, Prepared Statements should be used. This will prevent SQL injection attacks and will also improve the performance of the application by reducing the need to parse the SQL statement every time.

The code to open and close the connection creates the statement, retrieves the results that can be extracted to a separate method, and then called in other methods. The correct SQL data type should be used for each column in the databases. For example, if a column is data, it should be of type DATE in the databases and not a VARCHAR. Instead of using SELECT \* in the query, only the required fields should be selected. This will reduce the amount of data retrieved from the databases, which can improve performance.

**MVC Design Patterns:**

I used MVC (Model-View-Controller) design pattern in my MVC application which organizes the codebase into three distinct sections the model handles the business logic and interacts with the database using CRUD operations CREATE, RETRIEVE, UPDATE and DELETE. The Film DAO Data Access Object classes act as DB, whereas the Film DAO class contains methods such as getAllFilms, getFilmByTitle, deleteFilm, insertFilm and updateFilm, which contain SQL query language to interact with the film database. The controller acts as the middleman, implementing the model and then doing something with the data retrieved by the model. For example, the populate table. Java servlet class uses the getAllFilms() method from the DAO to retrieve a list of films from the database, convert it to a JSON array and serve it to the web browser. The controllers are represented by the JSP pages which include films.JSP, add film. JSP, delete film. JSP and many more retrieve the JSON array and display it in a table format for the user to view in a readable form. Using the MVC pattern has many benefits, such as keeping the codebase organized and easy to understand, it also makes it easy to maintain and extend the application, as each part has low dependency on one another, which makes it easy to develop the application quickly or work on it with multiple developers. The potential disadvantage of this pattern is that the controller is dependent on the model, but this doesn’t affect the architecture and it’s relatively easy to make modifications the separation of responsibilities makes it easier to apply modifications for current or feature use.

Text

Description automatically generated This is the Film Controller class that handles HTTP requests and responses for a web application. The class extends the HttpServelt class and is annotated with the WebServelt annotation, specifying the URL pattern “/Film” that this servlet will handle. The do Get () method is overridden to handle the GET requests and retrieves a list of films from the DAO class and sets them as an attribute of the request object.

And this is the delete Film class responsible for deleting film from the database, do Post method retrieves the id of the film to be deleted from the request’s parameters, creates a new Film DAO object, and calls the delete Film method of the DAO passing the film object as a parameter. This will delete the film from the database.

Text

Description automatically generated

The screenshot below shows the class getFilmByName it uses a Film DAO object to access a database and retrieve the film information. The servlet retrieves the film name and format from the request parameters and then calls the getFilmByName () method of the Film DAO to get a list of films with that name.

Text

Description automatically generated

This is the updated class that handles requests to update a film record in a database. When I submit a form to update a film the servlet retrieves the form data and create a new Film object with the updated data. It uses a Film DAO object to update the film record in the database. The servlet then redirects the user back to the list of films.

Text

Description automatically generated

And this is the add new film class, the servlet is invoked when a user submits a form to add a new film. When the servlet Is called, it retrieves the form data from the request object and create a new film object with the form data. Then, it uses the film DAO class to save the new film to the database. Finally, it redirects the user to a page that displays all films.

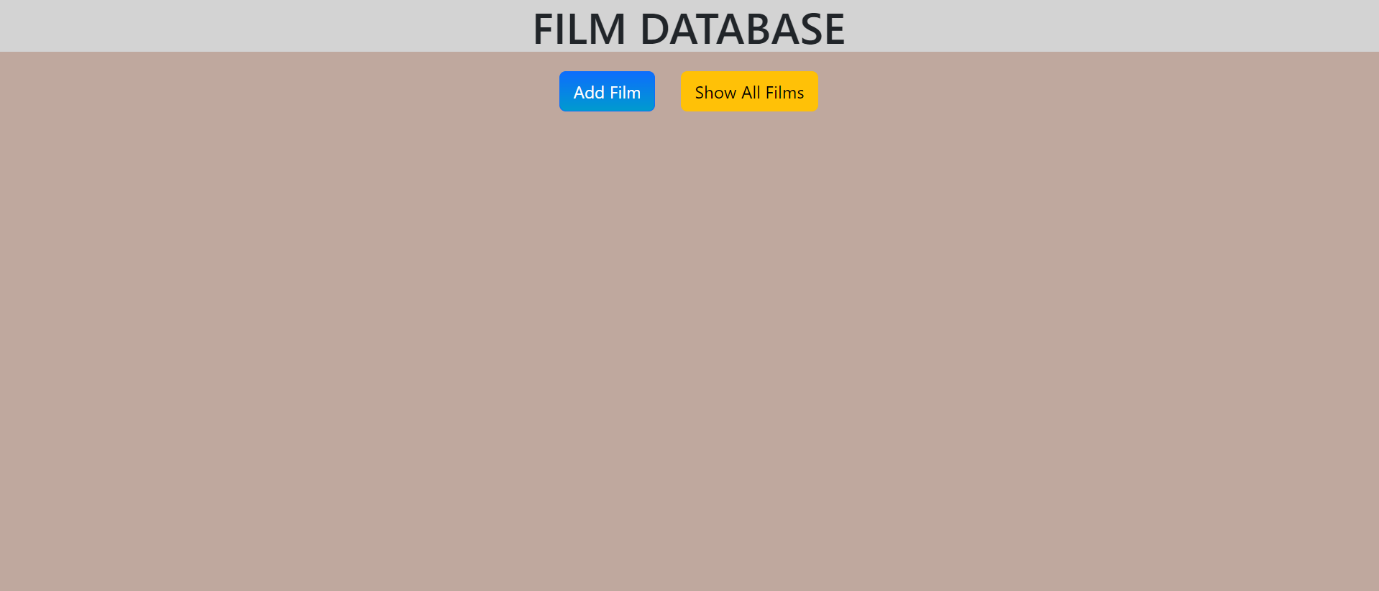
Text

Description automatically generated

Finally, these are all files that I have set up to come up with the final product of the MVC application, including the Java Controller, JSP pages, film DB, models, and the style sheet CSS.

Text

Description automatically generated with medium confidence

Timeline

Description automatically generated

**Restful Web service**

the Restful handle HTTP GET requests, the servlet retrieves a list of films from a Film DAO object, and then formats the list in one of three ways depending on the Content-Type header of the GET requests: application/XML, application/JSON or text/plain. If the Content-Type header is header is application/XML, the servlet uses JAXB to convert the list of films to XML and sends it to the client. If the Content-Type header is application/JSON, the servlet uses GSON to convert the list of films to JSON and sends it to the client. If the Content-Type header is text/plain or is not provided, the servlet sends the list of films to the client in plain text format.

In the screenshots below I provide the API test results in different formats, XML, JSON and text/plain using this API link: <http://localhost:8080/Myrestfulapi/Myrestfulapi>

Text

Description automatically generated

**Graphical user interface, text, application, email

Description automatically generatedGET request in JSON format**

**GET request in XML format**

Graphical user interface, text, application, email

Description automatically generated

**And finally, the POST request in JSON, XML, and text/pain**

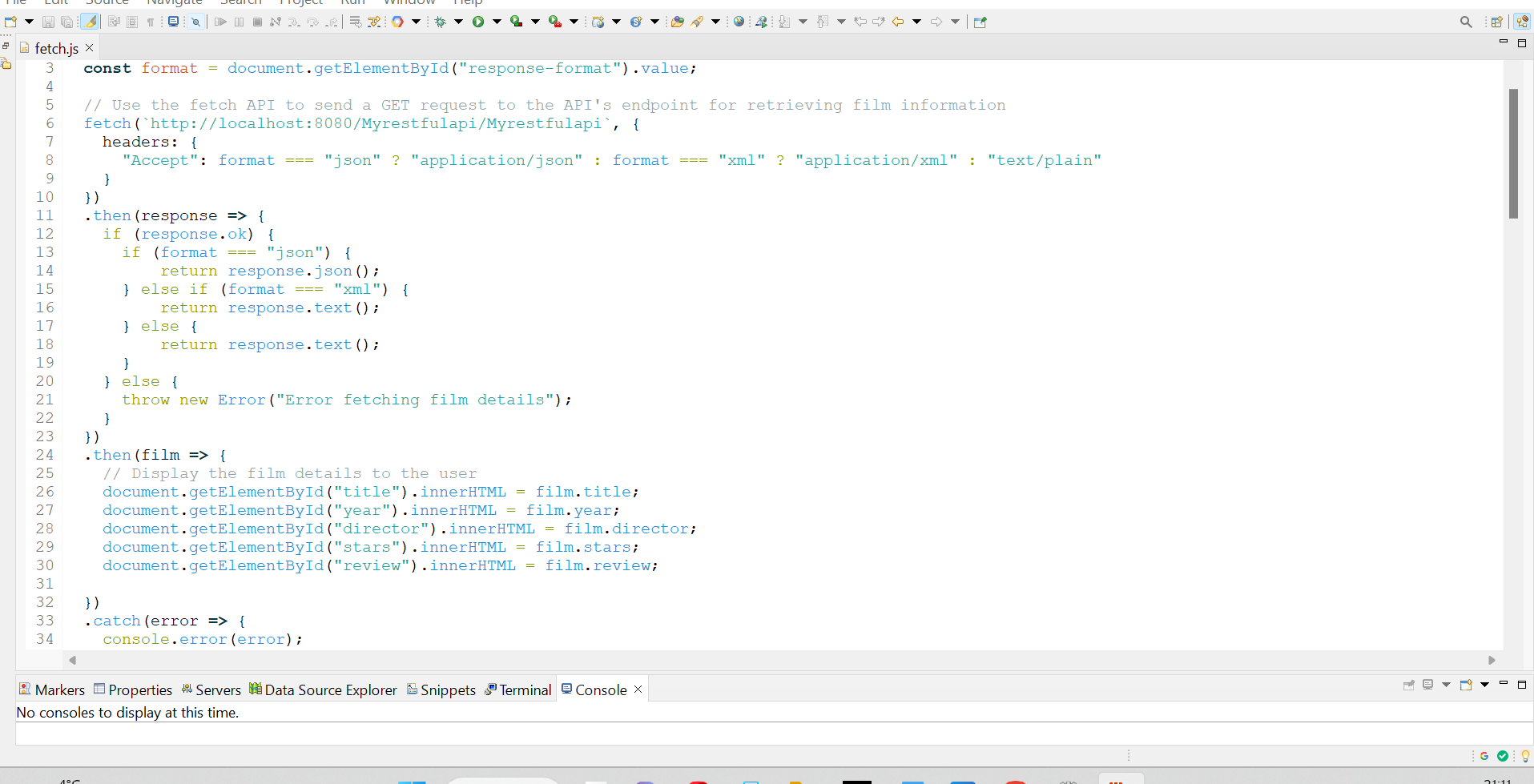
Graphical user interface, text, application, email

Description automatically generated

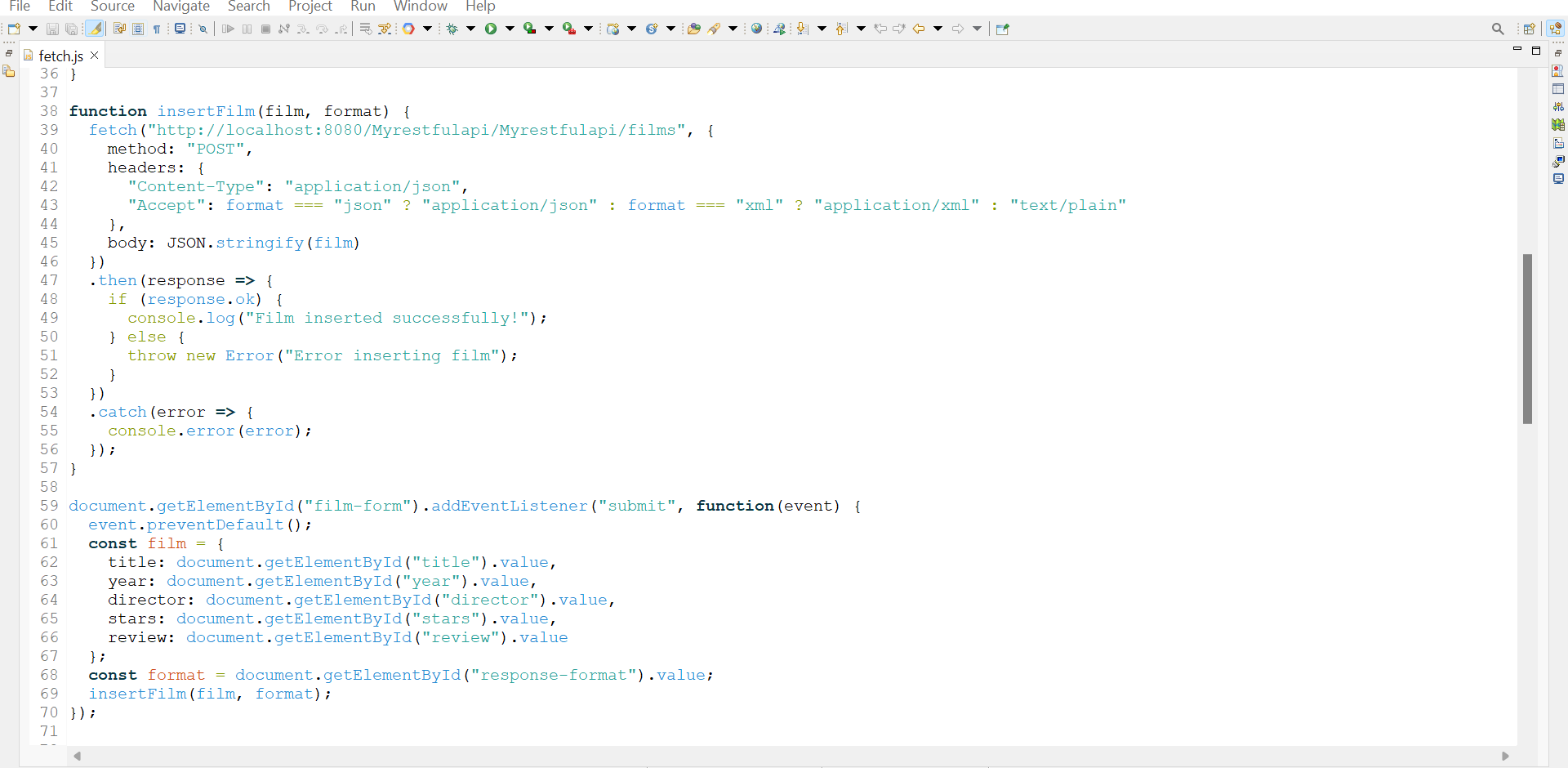
The API link: <http://localhost:8080/Myrestfulapi/Myrestfulapi> you can use this link to test on Postman GET, POST, PUT and DELETE.

**JavaScript Front End Client**

The client-side JavaScript script part, I have attempted it to make it work. The script file that interacts with API to fetch, insert, update, and delete film information. The script uses the fetch API to send GET, POST, PUT and DELETE requests to the API’S endpoint, and it uses the values of from elements on the HTML page to send data along with the requests. The script also uses the response from the API to display film information on HTML Page, and it logs messages to the console to indicate the success or failure of each request. However, I did not manage to make it display on HTML page to come up with a final JavaScript client-side application, but I learn a lot from this challenge.

This is the GET method using JavaScript

INSERT method



UPDATE method

Graphical user interface, text, application, email

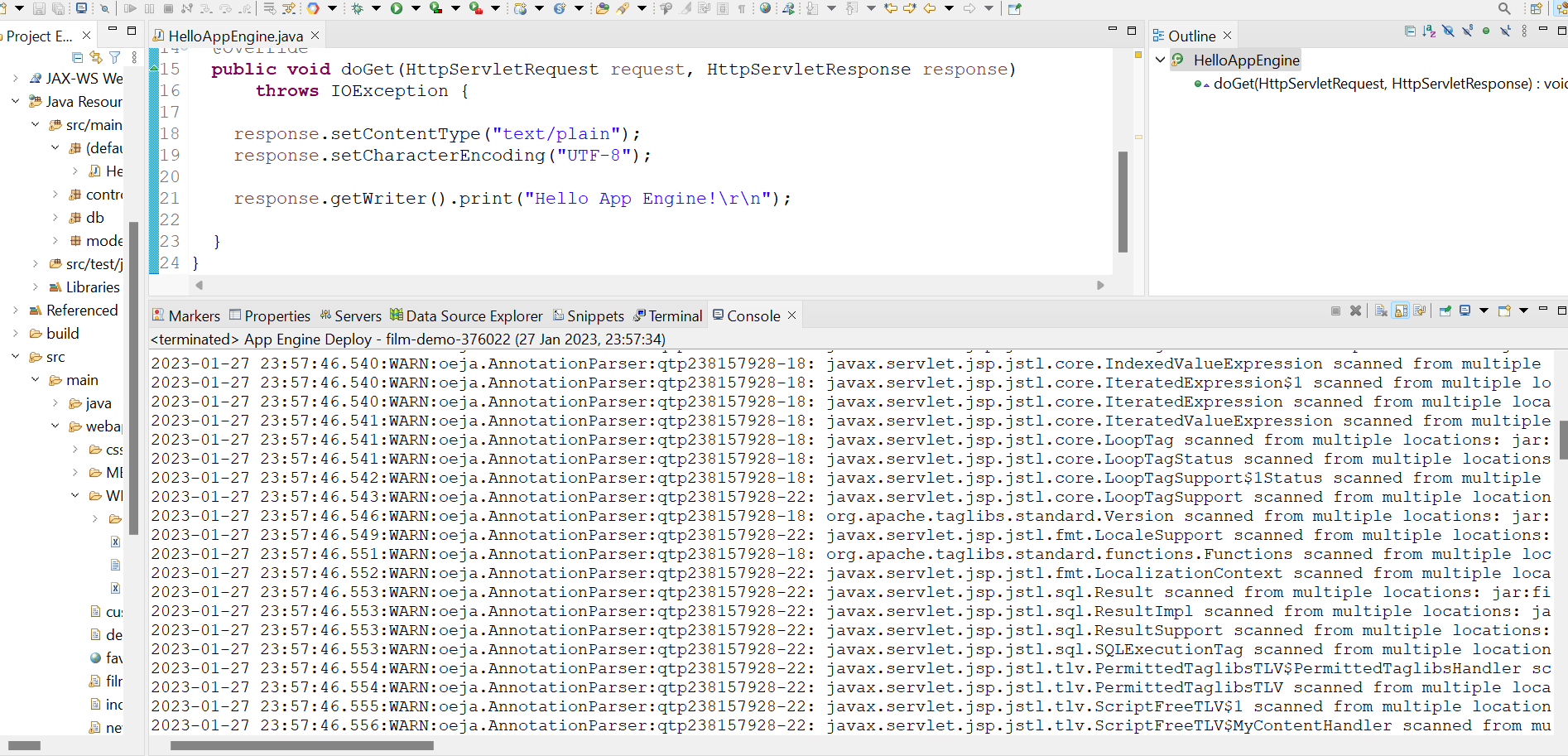
Description automatically generated

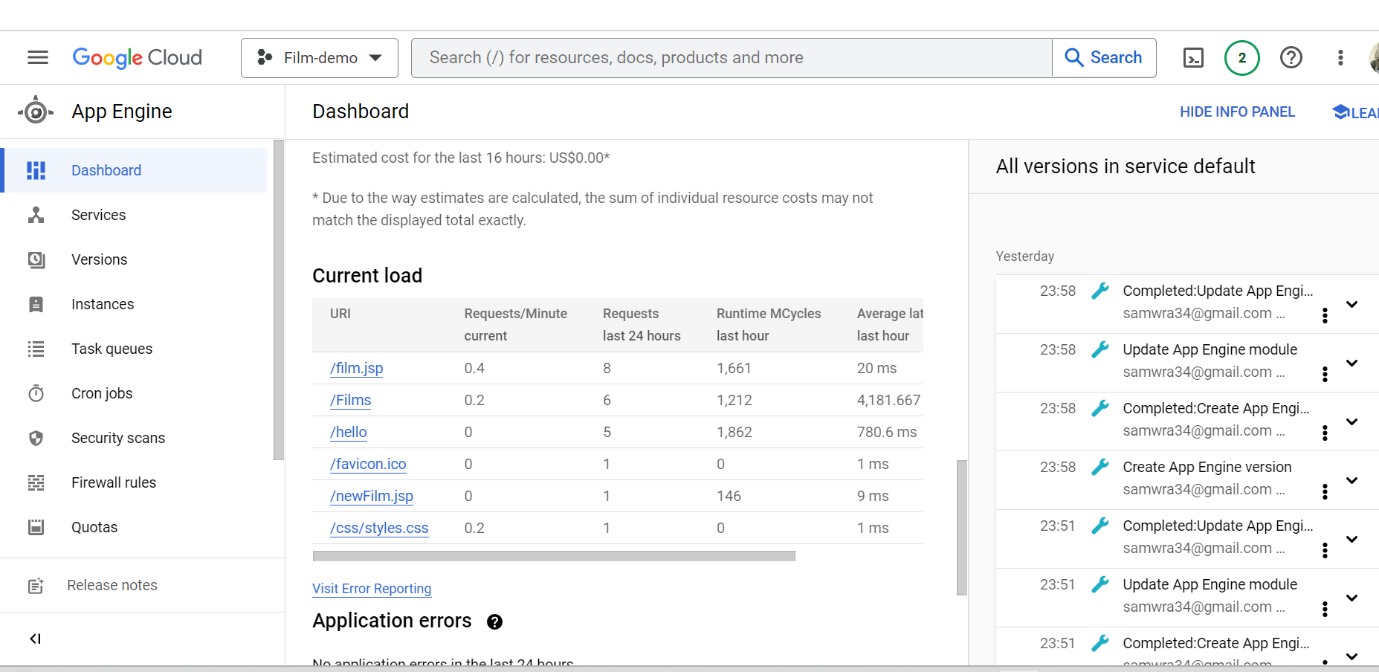
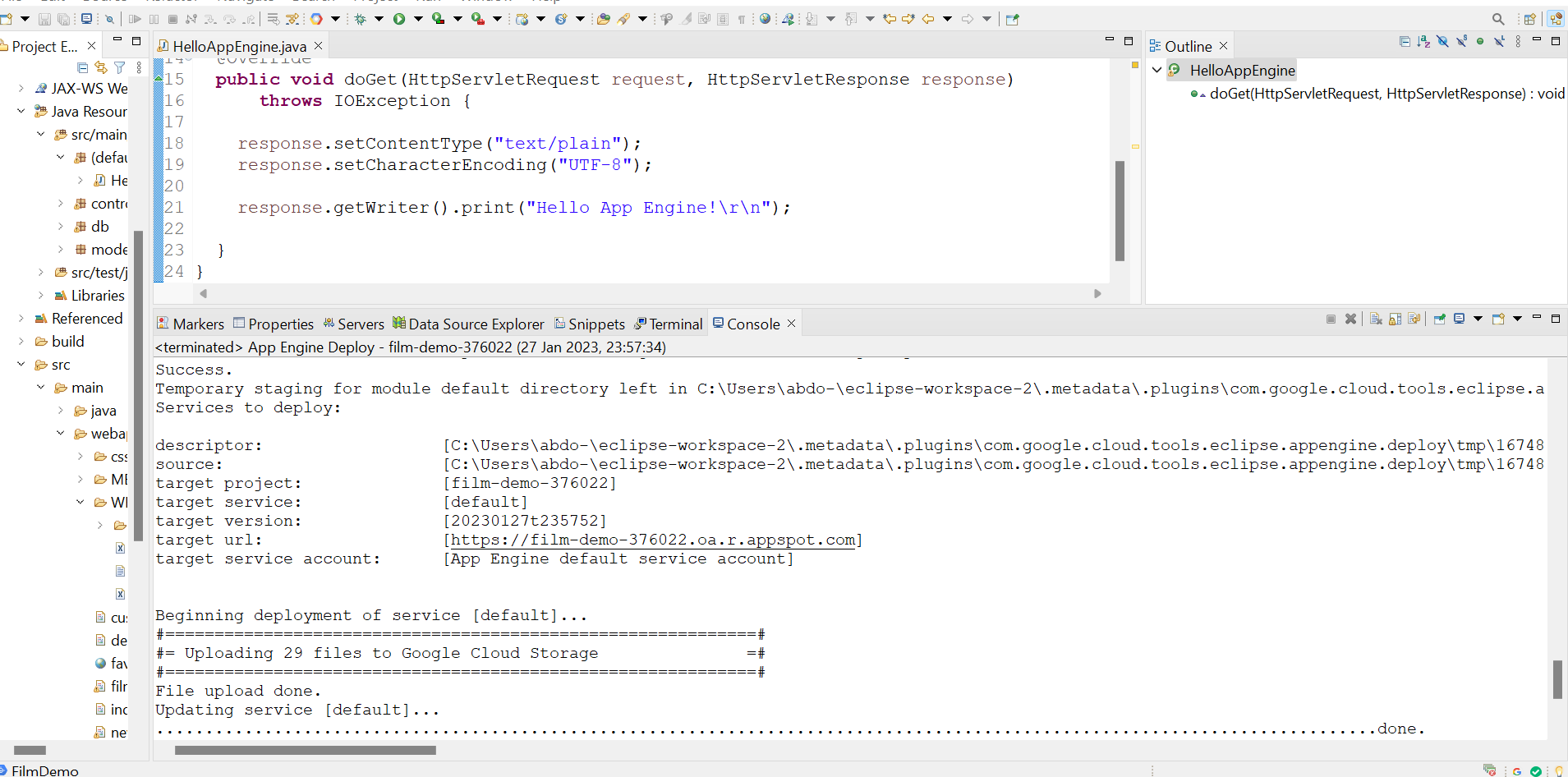
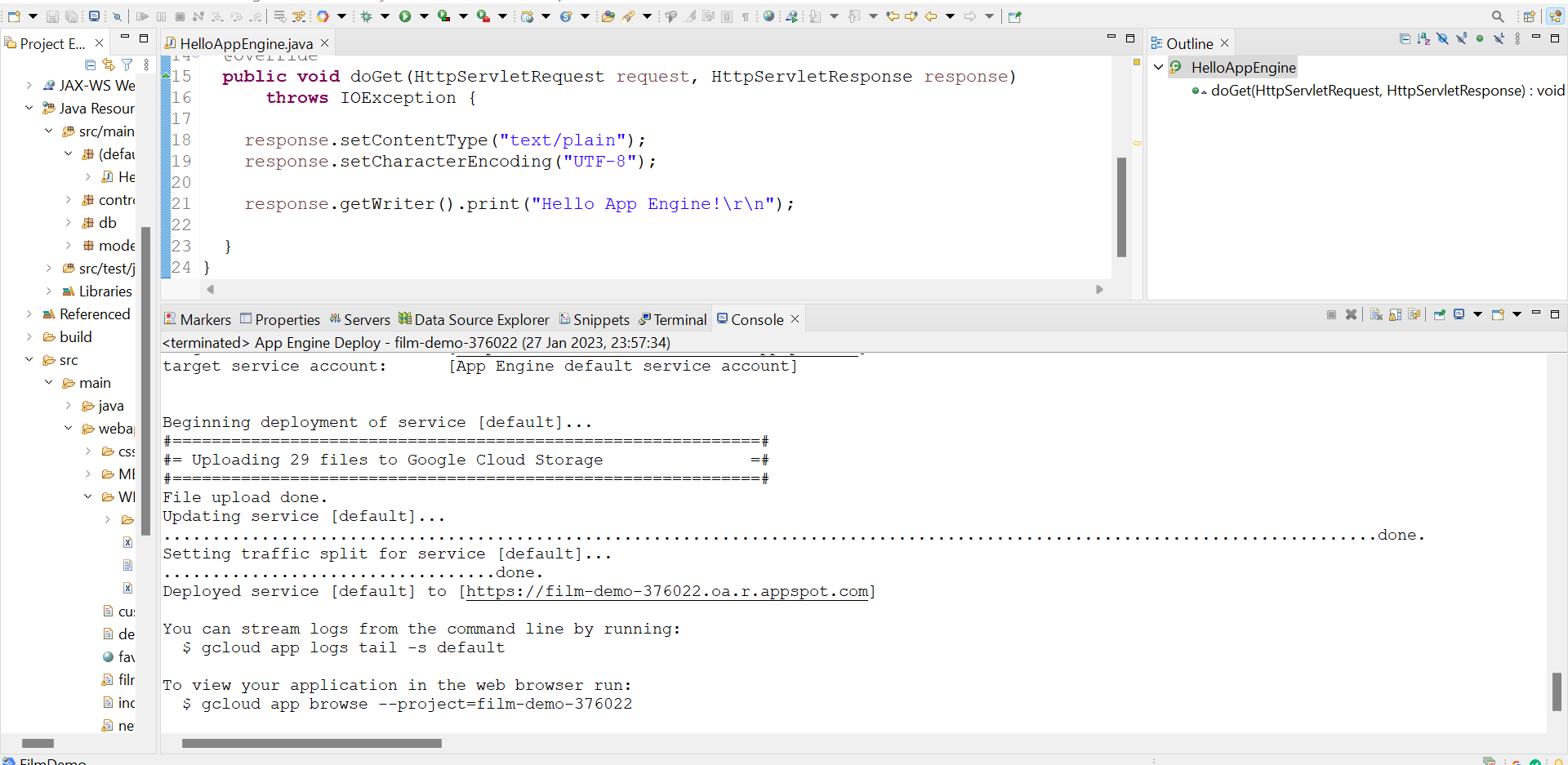
Graphical user interface, text, application

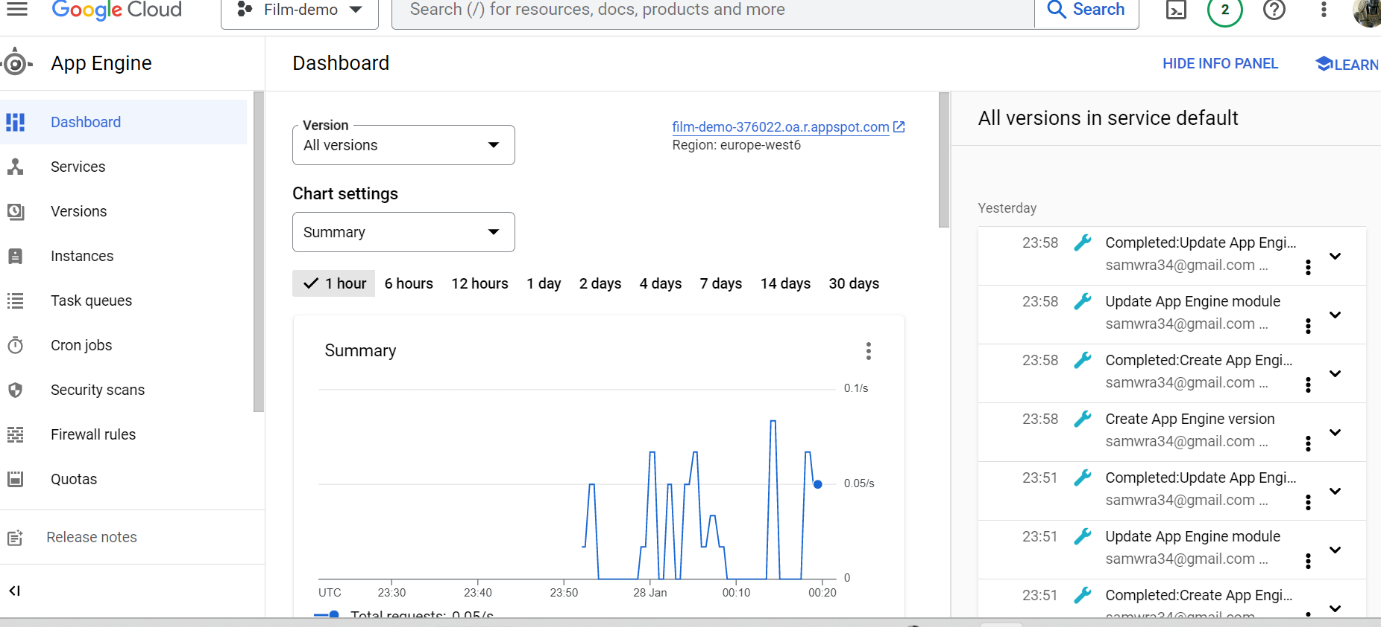
Description automatically generatedAnd finally DELETE method

**Film database in cloud**

I deployed my java MVC application in google cloud and it is live I have tested all functionality and it is working perfectly. In below figure I provided screenshot of what I have done while deploying the project.







This is my project in google could platform after I deployed

Text

Description automatically generated with medium confidence

And finally, this my project link and its live now live now you can test using below link

Cloud link <https://film-demo-376022.oa.r.appspot.com/film.jsp>