Software Requirement System

Movie Recommendation System

Prepared By:

Nur Mizan Qistina Binti Muhammad Fauzi

Date:

3rd February 2023

Table of Contents

TABLE OF FIGURES

System Feature List

1.0. This system provides various features for the users.

1.1. Login and Registration

- For both of these features, the system provided validation for both login control and registration control.
- The validation includes existed username, password verification and email input.

1.2. Admin

- Admin are allowed to add new users as both Admin and User in Admin Dashboard.
- This part also provides the validation same as registration.
- Admin can view all movies in the database.

1.3. User

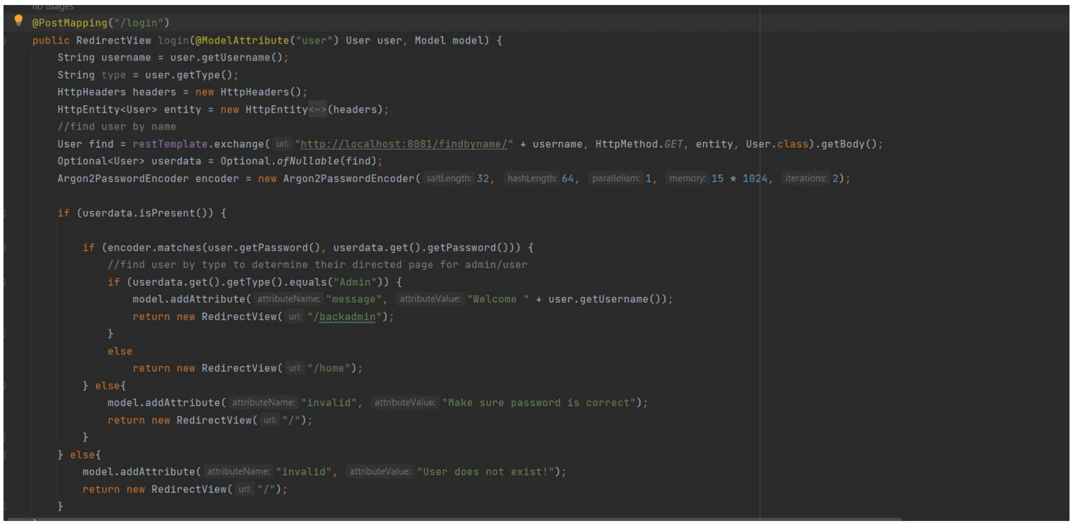
- User are allowed to login and redirected to their homepage.
- User can view movies.
- View recommended movies.
- Add movie to favourite.
- Rate movie.

1.1.1. Login and Registration

Login:

This validation will occur when user click on the login button and the username input is already existed in the database. The system will retrieve the username from database for comparison to the input and allow continue if exist. Then the system will compare the user type of that specific username and redirect the admin to the admin page and user to their homepage.

The password was stored as hash password in the database for security purpose, therefore check for comparison with the hashed password and user input is also done to allow login.



```
no usages
@PostMapping("/login")
public RedirectView login(@ModelAttribute("user") User user, Model model) {
    String username = user.getUsername();
    String type = user.getType();
    HttpHeaders headers = new HttpHeaders();
    HttpEntity<User> entity = new HttpEntity<~>(headers);
    //find user by name
    User find = restTemplate.exchange( url: "http://localhost:8081/findbyname/" + username, HttpMethod.GET, entity, User.class).getBody();
    Optional<User> userdata = Optional.ofNullable(find);
    Argon2PasswordEncoder encoder = new Argon2PasswordEncoder( saltLength: 32,  hashLength: 64,  parallelism: 1,  memory: 15 * 1024,  iterations: 2);

    if (userdata.isPresent()) {

        if (encoder.matches(user.getPassword(), userdata.get().getPassword())) {
            //find user by type to determine their directed page for admin/user
            if (userdata.get().getType().equals("Admin")) {
                model.addAttribute( attributeName: "message",  attributeValue: "Welcome " + user.getUsername());
                return new RedirectView( url: "/backadmin");
            }
            else
                return new RedirectView( url: "/home");
        } else{
            model.addAttribute( attributeName: "invalid",  attributeValue: "Make sure password is correct");
            return new RedirectView( url: "/");
        }
    } else{
        model.addAttribute( attributeName: "invalid",  attributeValue: "User does not exist!");
        return new RedirectView( url: "/");
    }
```

Figure 1: Login validation

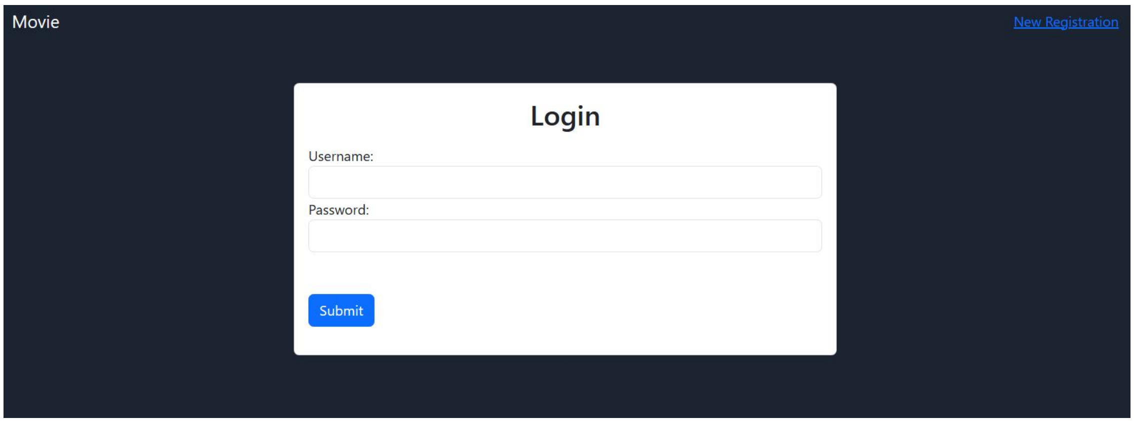
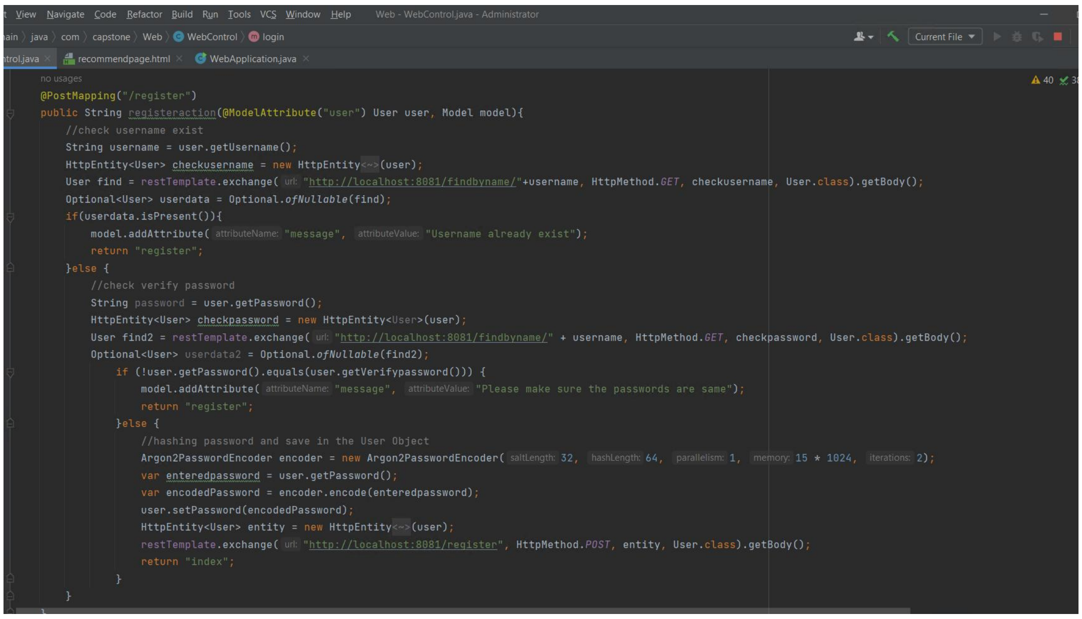


Figure 2: Login interface

Registration:

This feature happens when a user tried to create a new account. The validation includes existed username checker, hashing the password and password verification. With the same algorithm, the system will retrieve the username and password from database for checking and save the hashed password in the database. The user will be directed to the login page for login.


View Navigate Code Refactor Build Run Tools VCS Window Help Web - WebControl.java - Administrator
java 〉 com 〉 capstone 〉 Web 〉 WebControl 〉 login
Current File
recommendpage.html
WebApplication.java
no usages

```java
@PostMapping("/register")
public String registeraction(@ModelAttribute("user") User user, Model model){
    //check username exist
    String username = user.getUsername();
    HttpEntity<User> checkusername = new HttpEntity<~>(user);
    User find = restTemplate.exchange( url: "http://localhost:8081/findbyname/"+username, HttpMethod.GET, checkusername, User.class).getBody();
    Optional<User> userdata = Optional.ofNullable(find);
    if(userdata.isPresent()){
        model.addAttribute( attributeName: "message",  attributeValue: "Username already exist");
        return "register";
    }else {
        //check verify password
        String password = user.getPassword();
        HttpEntity<User> checkpassword = new HttpEntity<User>(user);
        User find2 = restTemplate.exchange( url: "http://localhost:8081/findbyname/" + username, HttpMethod.GET, checkpassword, User.class).getBody();
        Optional<User> userdata2 = Optional.ofNullable(find2);
            if (!user.getPassword().equals(user.getVerifypassword())) {
                model.addAttribute( attributeName: "message",  attributeValue: "Please make sure the passwords are same");
                return "register";
            }else {
                //hashing password and save in the User Object
                Argon2PasswordEncoder encoder = new Argon2PasswordEncoder( saltLength: 32,  hashLength: 64,  parallelism: 1,  memory: 15 * 1024,  iterations: 2);
                var enteredpassword = user.getPassword();
                var encodedPassword = encoder.encode(enteredpassword);
                user.setPassword(encodedPassword);
                HttpEntity<User> entity = new HttpEntity<~>(user);
                restTemplate.exchange( url: "http://localhost:8081/register", HttpMethod.POST, entity, User.class).getBody();
                return "index";
            }
    }
```

Figure 3: Registration system

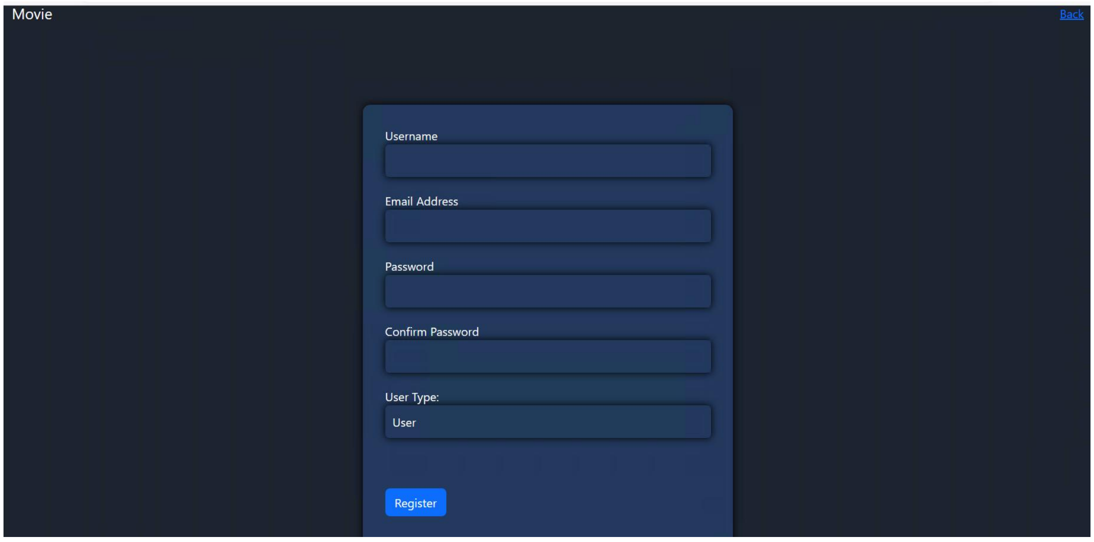


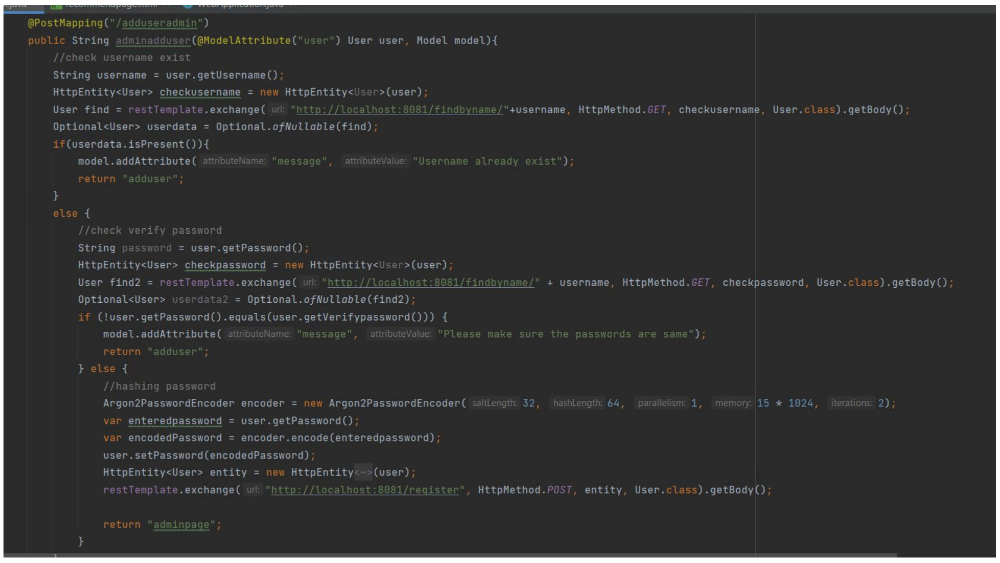
Figure 4: Registration Interface

1.1.1. Admin

This feature allows admin to add users and view all movies.

Add users:

This feature is exactly the same as registration but the admin extra option is to choose the user as admin or user using drop down button. The data will be stores in the database. Upon successful task, admin will be redirected to the admin dashboard page.



```java
@PostMapping("/adduseradmin")
public String adminadduser(@ModelAttribute("user") User user, Model model){
    //check username exist
    String username = user.getUsername();
    HttpEntity<User> checkusername = new HttpEntity<User>(user);
    User find = restTemplate.exchange( url: "http://localhost:8081/findbyname/"+username, HttpMethod.GET, checkusername, User.class).getBody();
    Optional<User> userdata = Optional.ofNullable(find);
    if(userdata.isPresent()){
        model.addAttribute( attributeName: "message",  attributeValue: "Username already exist");
        return "adduser";
    }
    else {
        //check verify password
        String password = user.getPassword();
        HttpEntity<User> checkpassword = new HttpEntity<User>(user);
        User find2 = restTemplate.exchange( url: "http://localhost:8081/findbyname/" + username, HttpMethod.GET, checkpassword, User.class).getBody();
        Optional<User> userdata2 = Optional.ofNullable(find2);
        if (!user.getPassword().equals(user.getVerifypassword())) {
            model.addAttribute( attributeName: "message",  attributeValue: "Please make sure the passwords are same");
            return "adduser";
        } else {
            //hashing password
            Argon2PasswordEncoder encoder = new Argon2PasswordEncoder( saltLength: 32,  hashLength: 64,  parallelism: 1,  memory: 15 * 1024,  iterations: 2);
            var enteredpassword = user.getPassword();
            var encodedPassword = encoder.encode(enteredpassword);
            user.setPassword(encodedPassword);
            HttpEntity<User> entity = new HttpEntity<~>(user);
            restTemplate.exchange( url: "http://localhost:8081/register", HttpMethod.POST, entity, User.class).getBody();

            return "adminpage";
        }
```

Figure 5: Admin add user

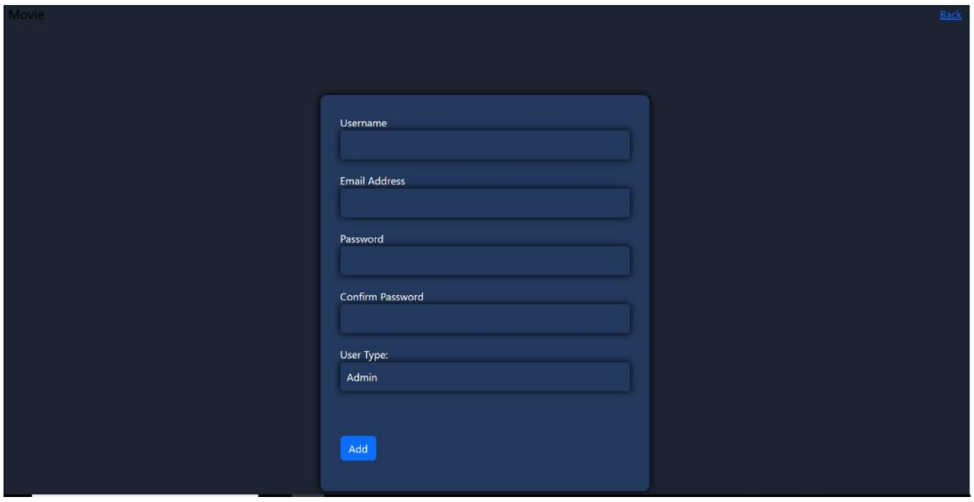
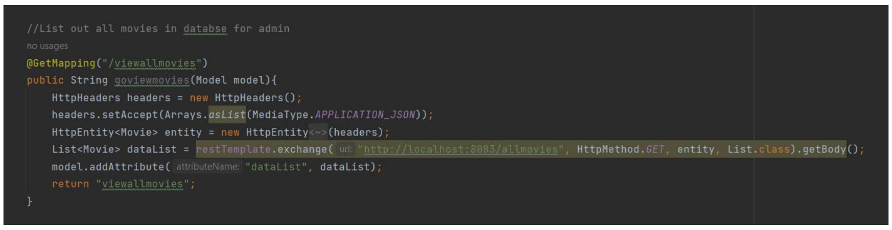


Figure 6: add user interface

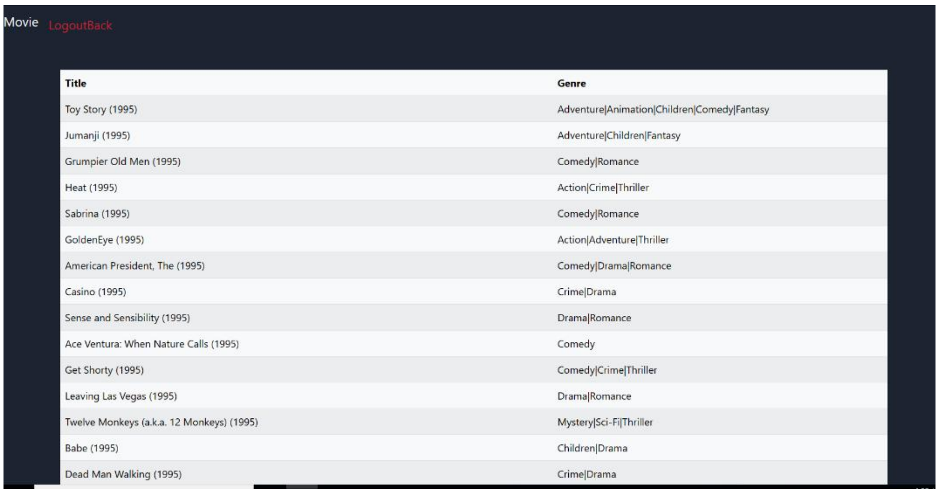
View all movies:

This feature allow admin to click view all movies button and see all movies from database in a table.



```java
//List out all movies in databse for admin
no usages
@GetMapping("/viewallmovies")
public String goviewmovies(Model model){
    HttpHeaders headers = new HttpHeaders();
    headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
    HttpEntity<Movie> entity = new HttpEntity<~>(headers);
    List<Movie> dataList = restTemplate.exchange( url: "http://localhost:8083/allmovies", HttpMethod.GET, entity, List.class).getBody();
    model.addAttribute( attributeName: "dataList", dataList);
    return "viewallmovies";
}
```

Figure 7: View all movies for Admin



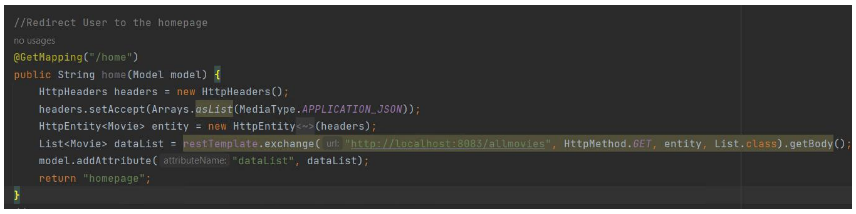
Movie LogoutBack

Title	Genre
Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
Jumanji (1995)	Adventure|Children|Fantasy
Grumpier Old Men (1995)	Comedy|Romance
Heat (1995)	Action|Crime|Thriller
Sabrina (1995)	Comedy|Romance
GoldenEye (1995)	Action|Adventure|Thriller
American President, The (1995)	Comedy|Drama|Romance
Casino (1995)	Crime|Drama
Sense and Sensibility (1995)	Drama|Romance
Ace Ventura: When Nature Calls (1995)	Comedy
Get Shorty (1995)	Comedy|Crime|Thriller
Leaving Las Vegas (1995)	Drama|Romance
Twelve Monkeys (a.k.a. 12 Monkeys) (1995)	Mystery|Sci-Fi|Thriller
Babe (1995)	Children|Drama
Dead Man Walking (1995)	Crime|Drama

Figure 8: All movies interface

1.1.2. User Homepage:

This feature allows users to view all movies in their homepage. This feature displays the movies with the posters.



```java
//Redirect User to the homepage
no usages
@GetMapping("/home")
public String home(Model model) {
    HttpHeaders headers = new HttpHeaders();
    headers.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
    HttpEntity<Movie> entity = new HttpEntity<~>(headers);
    List<Movie> dataList = restTemplate.exchange( url: "http://localhost:8083/allmovies", HttpMethod.GET, entity, List.class).getBody();
    model.addAttribute( attributeName: "dataList", dataList);
    return "homepage";
}
```

Figure 9: User homepage

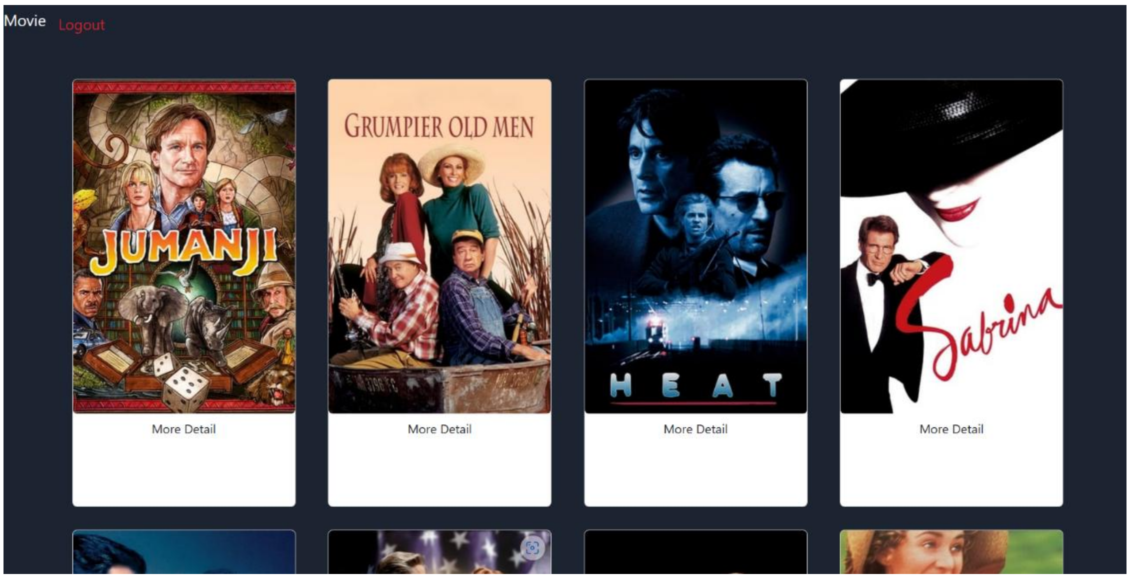
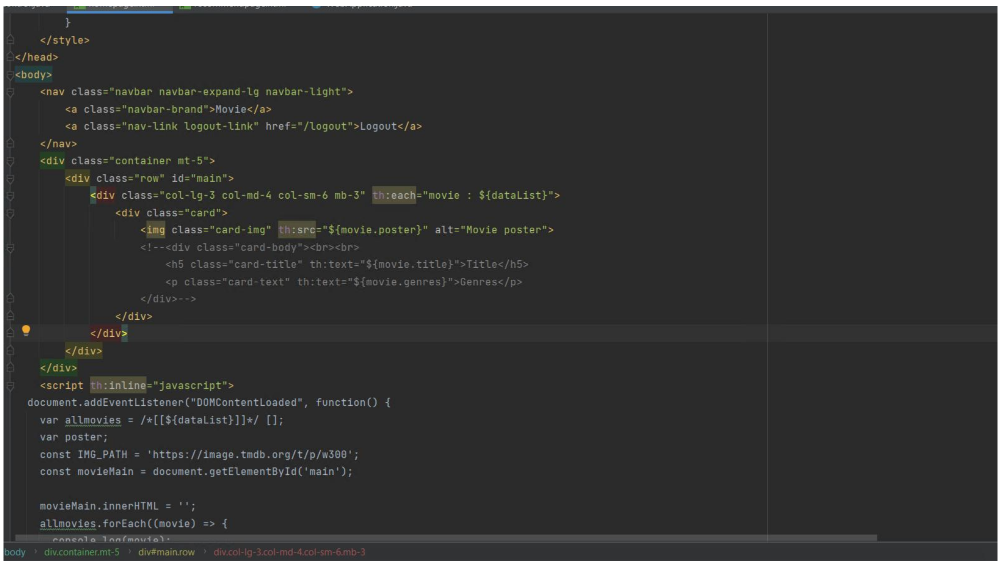


Figure 10: User homepage interface



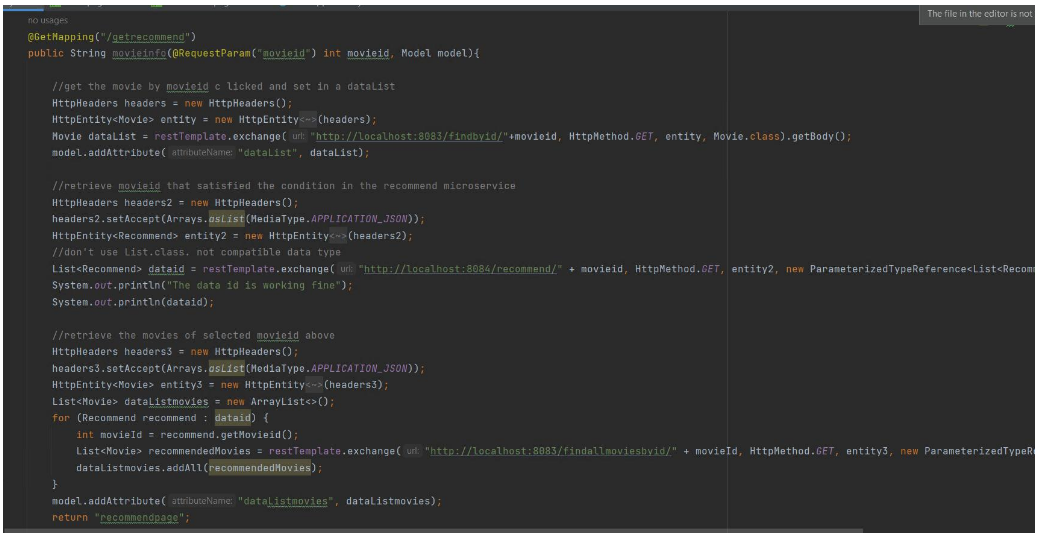
```
        }
    </style>
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light">
        <a class="navbar-brand">Movie</a>
        <a class="nav-link logout-link" href="/logout">Logout</a>
    </nav>
    <div class="container mt-5">
        <div class="row" id="main">
            <div class="col-lg-3 col-md-4 col-sm-6 mb-3" th:each="movie : ${dataList}">
                <div class="card">
                    <img class="card-img" th:src="${movie.poster}" alt="Movie poster">
                    <!--<div class="card-body"><br><br>
                        <h5 class="card-title" th:text="${movie.title}">Title</h5>
                        <p class="card-text" th:text="${movie.genres}">Genres</p>
                    </div>-->
                </div>
            </div>
        </div>
    </div>
    <script th:inline="javascript">
  document.addEventListener("DOMContentLoaded", function() {
    var allmovies = /*[[${dataList}]]*/ [];
    var poster;
    const IMG_PATH = 'https://image.tmdb.org/t/p/w300';
    const movieMain = document.getElementById('main');

    movieMain.innerHTML = '';
    allmovies.forEach((movie) => {
      console.log(movie);
body › div.container.mt-5 › div#main.row › div.col-lg-3.col-md-4.col-sm-6.mb-3
```

Figure 11: HTML file for homepage

Recommended Movies:

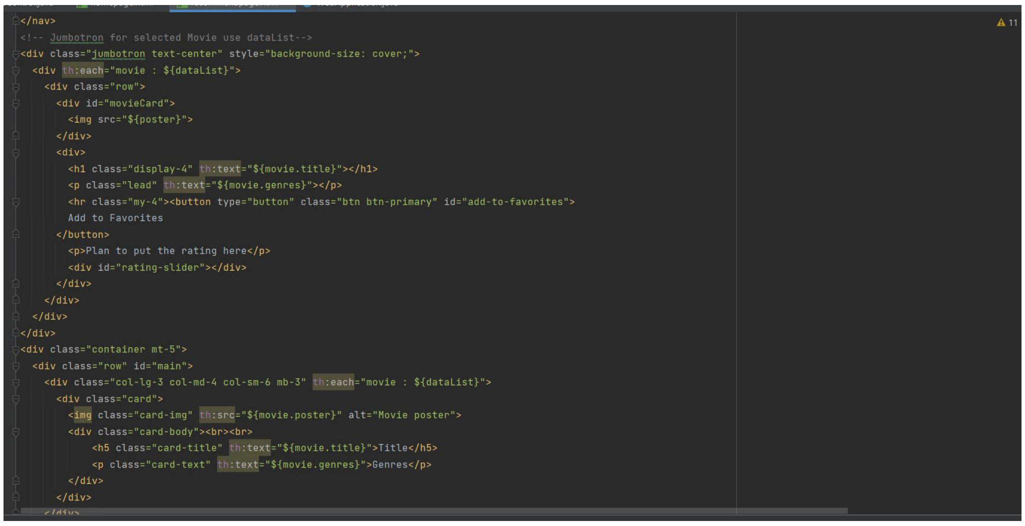
Upon clicking a movie poster, user will be redirected to a recommendation movie page where a list of recommended movies is displayed. This feature will first get the movie details from the clicked movie using findbyid method and then stored in a dataList. Then this dataList is then use to be display at the top of recommendation page.

Next step, the system retrieves all movieids that satisfied the condition of the query to be chosen as recommendation. Then the system will stores the results in dataid list and be used to loop each movieid and store the details of each movie in a new list. This list then will be used to displayed at the recommendation page.



```
no usages
@GetMapping("/getrecommend")
public String movieinfo(@RequestParam("movieid") int movieid, Model model){

    //get the movie by movieid c licked and set in a dataList
    HttpHeaders headers = new HttpHeaders();
    HttpEntity<Movie> entity = new HttpEntity<~>(headers);
    Movie dataList = restTemplate.exchange( url: "http://localhost:8083/findbyid/"+movieid, HttpMethod.GET, entity, Movie.class).getBody();
    model.addAttribute( attributeName: "dataList", dataList);

    //retrieve movieid that satisfied the condition in the recommend microservice
    HttpHeaders headers2 = new HttpHeaders();
    headers2.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
    HttpEntity<Recommend> entity2 = new HttpEntity<~>(headers2);
    //don't use List.class. not compatible data type
    List<Recommend> dataid = restTemplate.exchange( url: "http://localhost:8084/recommend/" + movieid, HttpMethod.GET, entity2, new ParameterizedTypeReference<List<Recom
    System.out.println("The data id is working fine");
    System.out.println(dataid);

    //retrieve the movies of selected movieid above
    HttpHeaders headers3 = new HttpHeaders();
    headers3.setAccept(Arrays.asList(MediaType.APPLICATION_JSON));
    HttpEntity<Movie> entity3 = new HttpEntity<~>(headers3);
    List<Movie> dataListmovies = new ArrayList<>();
    for (Recommend recommend : dataid) {
        int movieId = recommend.getMovieid();
        List<Movie> recommendedMovies = restTemplate.exchange( url: "http://localhost:8083/findallmoviesbyid/" + movieId, HttpMethod.GET, entity3, new ParameterizedTypeR
        dataListmovies.addAll(recommendedMovies);
    }
    model.addAttribute( attributeName: "dataListmovies", dataListmovies);
    return "recommendpage";
```

Figure 12: Recommendation control



```
</nav>
<!-- Jumbotron for selected Movie use dataList-->
<div class="jumbotron text-center" style="background-size: cover;">
  <div th:each="movie : ${dataList}">
    <div class="row">
      <div id="movieCard">
        <img src="${poster}">
      </div>
      <div>
        <h1 class="display-4" th:text="${movie.title}"></h1>
        <p class="lead" th:text="${movie.genres}"></p>
        <hr class="my-4"><button type="button" class="btn btn-primary" id="add-to-favorites">
        Add to Favorites
      </button>
        <p>Plan to put the rating here</p>
        <div id="rating-slider"></div>
      </div>
    </div>
  </div>
</div>
<div class="container mt-5">
  <div class="row" id="main">
    <div class="col-lg-3 col-md-4 col-sm-6 mb-3" th:each="movie : ${dataList}">
      <div class="card">
        <img class="card-img" th:src="${movie.poster}" alt="Movie poster">
        <div class="card-body"><br><br>
            <h5 class="card-title" th:text="${movie.title}">Title</h5>
            <p class="card-text" th:text="${movie.genres}">Genres</p>
        </div>
      </div>
```

Figure 13: HTML file for recommendation page

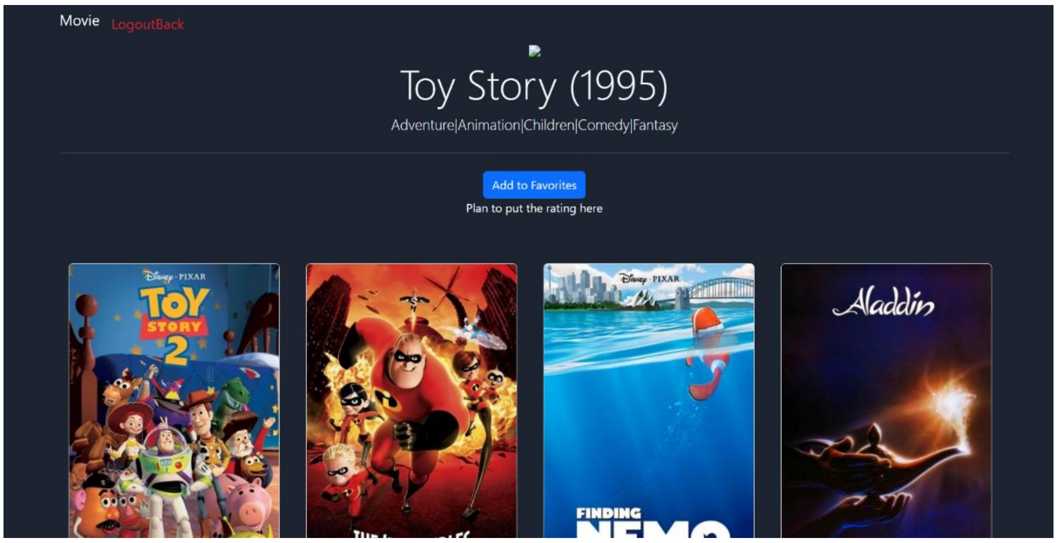
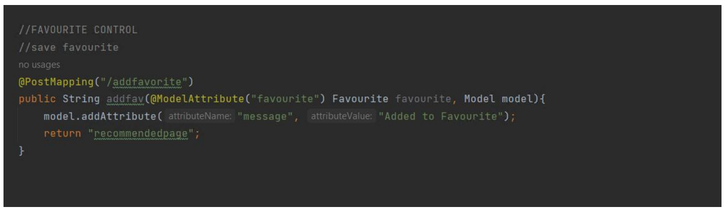

Movie
LogoutBack
Toy Story (1995)
Adventure|Animation|Children|Comedy|Fantasy
Add to Favorites
Plan to put the rating here

Figure 14: Recommendation Page Interface

Add Movie to Favourite:

This feature is for user to add the movie to their favourite movie list.

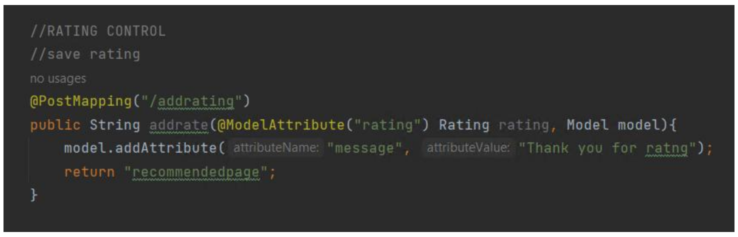


```
//FAVOURITE CONTROL
//save favourite
no usages
@PostMapping("/addfavorite")
public String addfav(@ModelAttribute("favourite") Favourite favourite, Model model){
    model.addAttribute( attributeName: "message",  attributeValue: "Added to Favourite");
    return "recommendedpage";
}
```

Figure 15: Save Favourite control

Rate movie:

User also can rate the movie within a scale of 1 to 5. This will be recorded for future references to upgrade recommendation system.



```
//RATING CONTROL
//save rating
no usages
@PostMapping("/addrating")
public String addrate(@ModelAttribute("rating") Rating rating, Model model){
    model.addAttribute( attributeName: "message",  attributeValue: "Thank you for ratng");
    return "recommendedpage";
}
```

Figure 16: Save Rating Control

2.0. Diagrams

2.1. Login

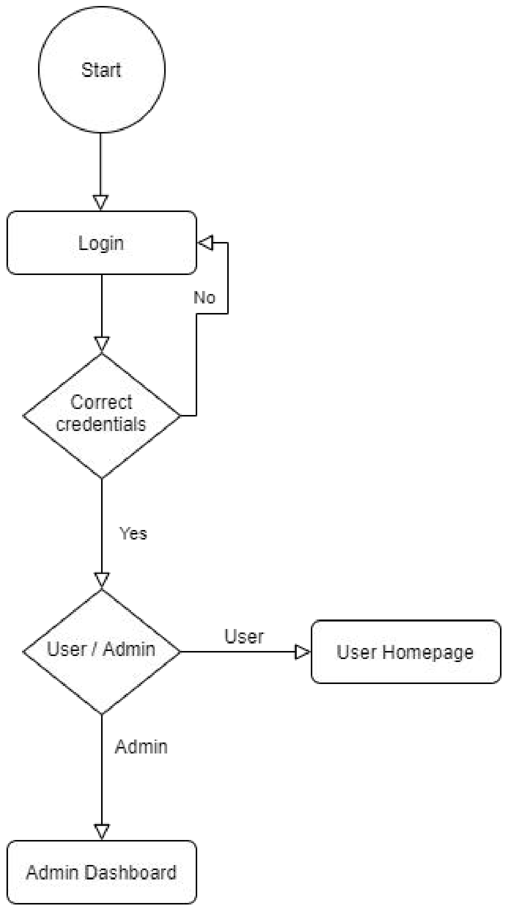


Figure 17: Login flowchart

The flowchart above is for the login page. The user will do the login by entering the credentials. The credentials then are checked and will redirected back to login page if the credentials are wrong. If the credentials are correct, the system will check for user type admin / user and redirect the user to their respective page.

2.2. Admin page

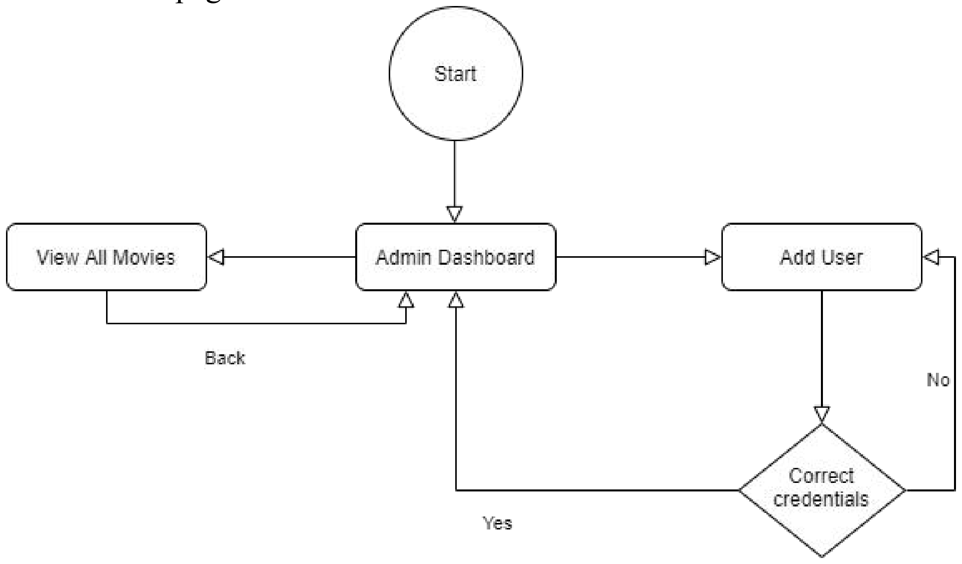


Figure 18: Admin flowchart

This flowchart is for admin that are able to choose to view all movies or add user in the dashboard. Admin can always logout and back to the dashboard anytime they want. If admin failed to enter correct credential during add user, the admin will be directed to the user page again.

2.3. User page

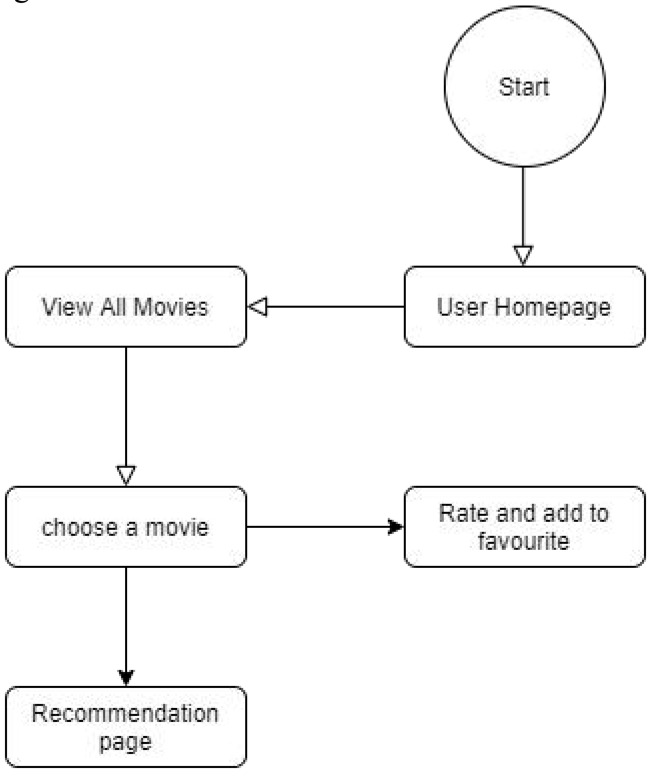


Figure 19: User Flowchart

This flowchart shows user interface where user can click on any movies and be recommended any movies within the same similarity rating of the selected movies.

2.4. Full Flowchart

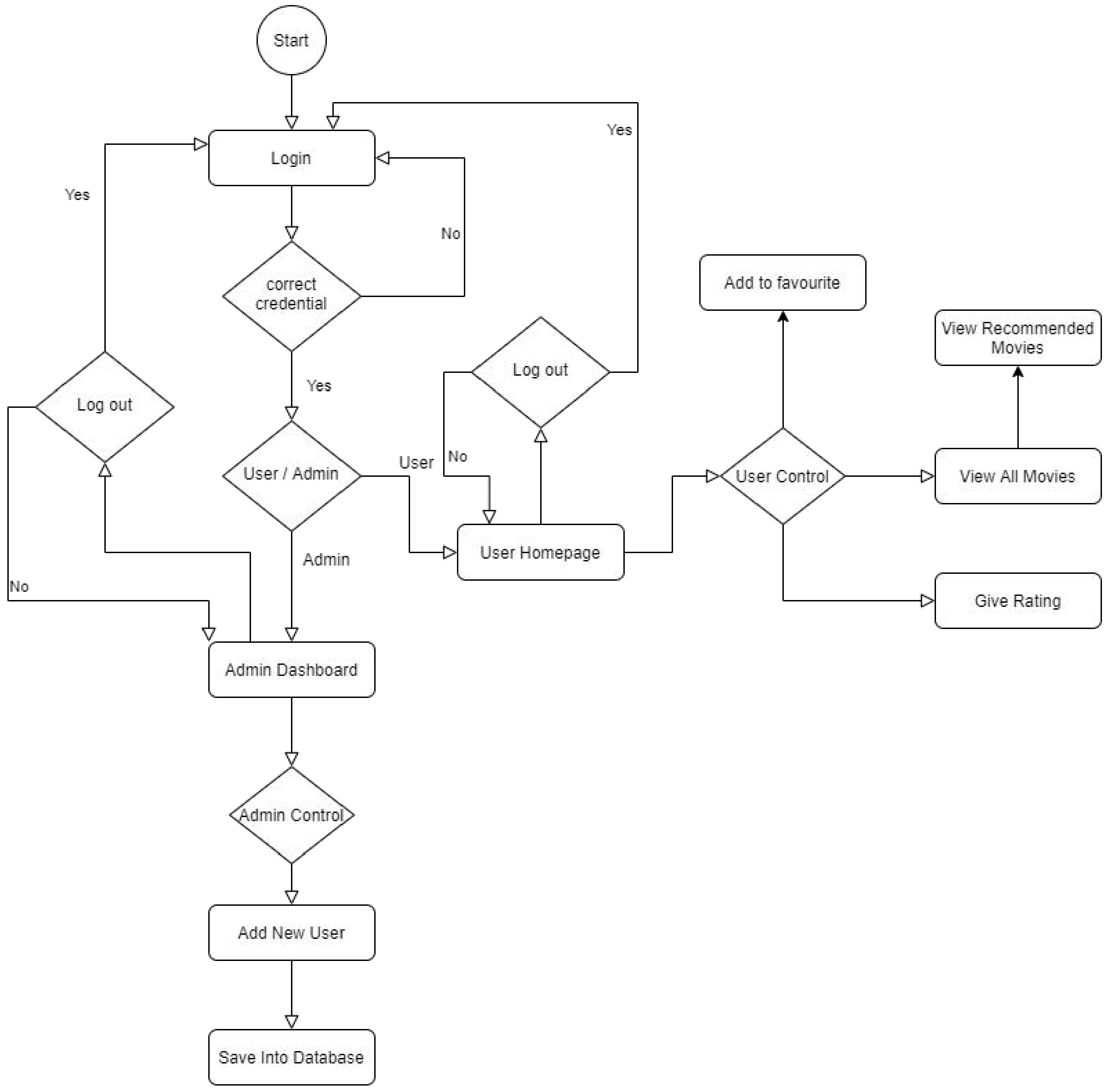


Figure 20: Full flowchart of the system

Above is the full flowchart for the system starting from login to each admin and user interface.

3.0. Database system (DB2)

3.1. Entity Relationship Diagram

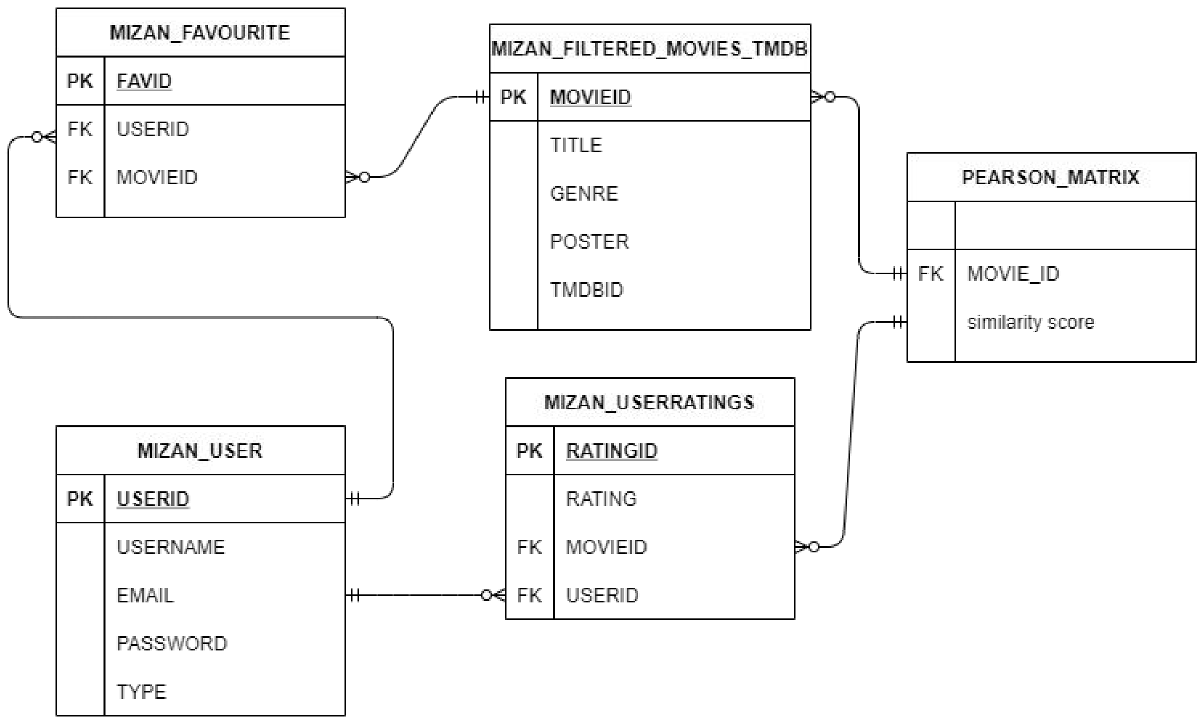
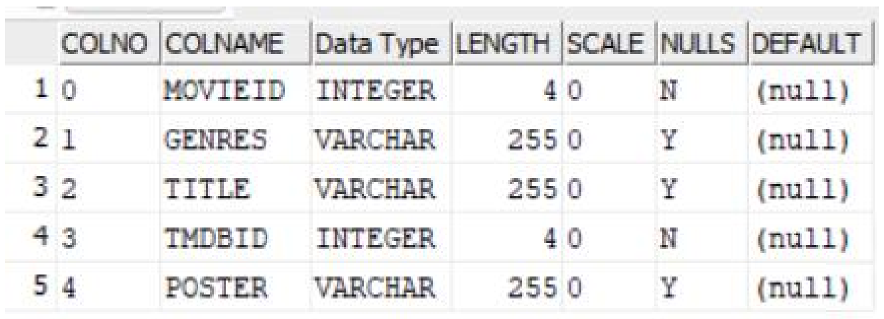


Figure 21: Entity Relationship Diagram

3.2. Tables

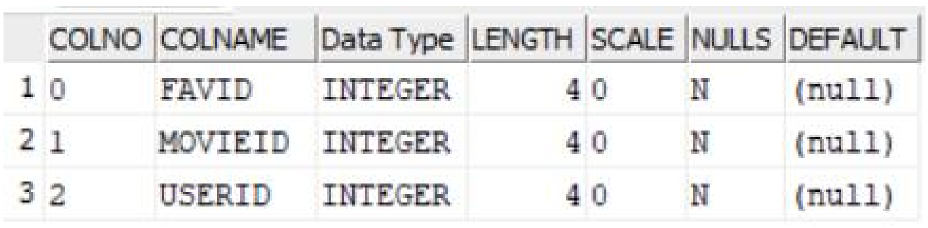
3.2.1. MIZAN_FILTERED_MOVIES_TMDB



	COLNO	COLNAME	Data Type	LENGTH	SCALE	NULLS	DEFAULT
1	0	MOVIEID	INTEGER	4	0	N	(null)
2	1	GENRES	VARCHAR	255	0	Y	(null)
3	2	TITLE	VARCHAR	255	0	Y	(null)
4	3	TMDBID	INTEGER	4	0	N	(null)
5	4	POSTER	VARCHAR	255	0	Y	(null)

Figure 22: Movie table

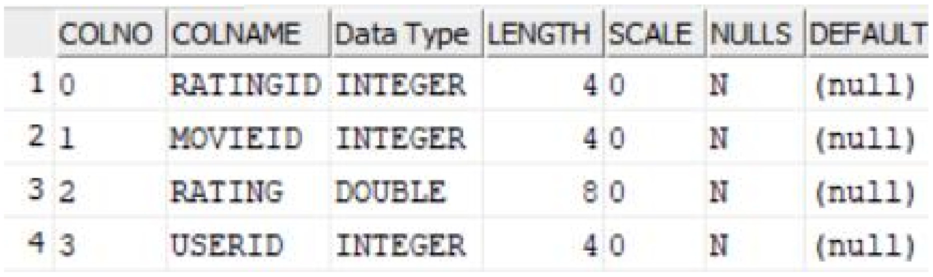
3.2.2. MIZAN_FAVOURITE



	COLNO	COLNAME	Data Type	LENGTH	SCALE	NULLS	DEFAULT
1	0	FAVID	INTEGER	4	0	N	(null)
2	1	MOVIEID	INTEGER	4	0	N	(null)
3	2	USERID	INTEGER	4	0	N	(null)

Figure 23: Favourite Table

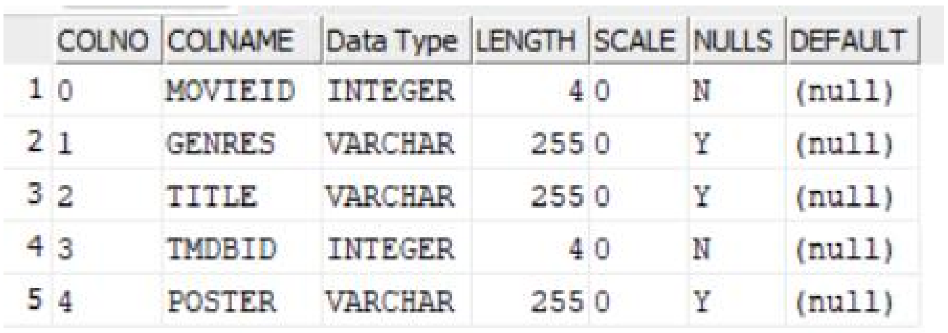
3.2.3. MIZAN_USERRATING



	COLNO	COLNAME	Data Type	LENGTH	SCALE	NULLS	DEFAULT
1	0	RATINGID	INTEGER	4	0	N	(null)
2	1	MOVIEID	INTEGER	4	0	N	(null)
3	2	RATING	DOUBLE	8	0	N	(null)
4	3	USERID	INTEGER	4	0	N	(null)

Figure 24: Rating Table

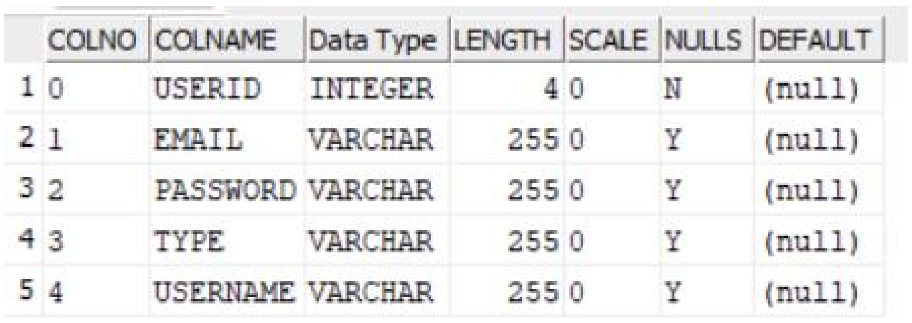
3.2.4. MIZAN_FILTERED_MOVIE_TMDB



	COLNO	COLNAME	Data Type	LENGTH	SCALE	NULLS	DEFAULT
1	0	MOVIEID	INTEGER	4	0	N	(null)
2	1	GENRES	VARCHAR	255	0	Y	(null)
3	2	TITLE	VARCHAR	255	0	Y	(null)
4	3	TMDBID	INTEGER	4	0	N	(null)
5	4	POSTER	VARCHAR	255	0	Y	(null)

Figure 25: Movie Table

3.2.5. MIZAN_USER



	COLNO	COLNAME	Data Type	LENGTH	SCALE	NULLS	DEFAULT
1	0	USERID	INTEGER	4	0	N	(null)
2	1	EMAIL	VARCHAR	255	0	Y	(null)
3	2	PASSWORD	VARCHAR	255	0	Y	(null)
4	3	TYPE	VARCHAR	255	0	Y	(null)
5	4	USERNAME	VARCHAR	255	0	Y	(null)

Figure 26: User Table

4.0. Microservice Architecture

4.1. Definition

Microservice architecture is an architecture software design that allows developer or programmer to separate big or large application into several small services that is independent. These services connect and communicate with each other using API. This also allows each microservice to be control by itself using Rest API.

Each microservice has its own table in database which ensure systematic and clean system. If one service broke down or failed, it won't affect the whole system. This will reduce time wasting and resource wasting greatly. Especially in a team development, this can save so many times by having different person do different microservices.

4.2. Implementation

In this recommendation system, Microservice architecture also have been applied by having 6 different services which are User-service, Movie-service, Recommend-service, Rating-service and Favourite-service for the back end and Web-service for the front end. By doing this, the database also properly sorted and have oriented. These services are connected through Eureka where the connection can easily be managed and handled for the communication between the services.

5.0. Pearson’s Correlations

5.1. Definition

The linear correlation between two continuous variables can be calculated by Pearson's correlation, also referred to as Pearson's correlation coefficient. It represents the degree and direction of the link between two variables and spans from -1 to 1. Strong positive correlations are represented by values of 1, strong negative correlations by values of -1, and no correlation by values of 0. The calculation of Pearson's correlation, which is frequently used in statistics to assess the degree of a relationship between two variables, is made by dividing the covariance of the two variables by the sum of their standard deviations.

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Where,

r = Pearson Correlation Coefficient

x_i = x variable samples	y_i = y variable sample
$\bar{x}$ = mean of values in x variable	$\bar{y}$ =mean of values in y variable

Figure 27: Pearson’s Correlation Formula

5.2. Implementation

In this project, the Pearson’s Correlations is used to calculate the similarity scores between all the movies in the database based on the given ratings by the users. This implementation resulted in negative correlation for the movies that have very low similarities in their rating and positive correlation for the other way round. When the number are closer to 1, then the movies are almost identical between the two.

Firstly, the dataset of rating from rating table are retrieved and stored in a matrix. Here, the row is the userid and the column is the movieid.

Then the calculation started by assuming x is userid and y is movieid. The calculation is done by the formula in figure 27 above.

Lastly, the result is stored in a table and pivoted so the matrix will have the same row and column for comparison. The result should have diagonally 1 value because it is compared with each other which should be similar.