Software Requirements Specification

for

<Movie Recommendation System (MRS)>

Prepared by <Suren Raj Suresh>

<Collabera>

<7 February 2023>

Table of Contents

Table of Contents .... **ii**

Revision History .... Error! Bookmark not defined.

1. Introduction **3**

1.1 Purpose 3

1.2 Document Conventions 3

1.3 Intended Audience and Reading Suggestions 3

1.4 Project Scope 3

2. Overall Description **4**

2.1 Product Perspective 4

3. Feature Lists **5**

3.1 Authentication........5

3.2 View User (User Controller)........9

3.3 Search (User Controller)........14

3.4 Recommended Movies (User Controller)........16

3.5 Add Rating (User Controller)........17

3.6 View User (Admin Controller)........19

3.7 View Movie (Admin Controller)........20

3.8 Add User (Admin Controller)........21

3.9 Add Movie (Admin Controller)........23

4. Entity Relation Diagram **25**

5. Pearson Correlation........ **25**

1. Introduction

1.1 Purpose

A recommender system primary objective is to enhance the user experience by making personalized recommendation based on the user's past behavior which can be based on the clicks or search history. Based on the stated source, the system can identify patterns and predicts future interests. With this, the system tends to help to increase user engagement. The user also benefits from reduced search time. From a business perspective, the use of the system can drive sales or content consumption. As a result, businesses benefit from increased customer satisfaction, loyalty, and revenue.

1.2 Intended Audience and Reading Suggestions

The intended audience for recommender system can be offered a highly personalized experience based on their individual preferences and behaviors. The system can also be used in a different way based on the context. For example, in a commercial setting, such as an e-commerce website looking for product recommendations. In a content- based environment, like a news website, the intended audience is a reader seeking personalized news suggestions. Additionally, there is a more general audience, such as movie-goers, who can benefit from movie recommendations.

1.3 Project Scope

The scope of recommendation systems for a movie website focuses on providing personalized and relevant suggestions based on user preferences, viewing history and overall trens. This includes: personalized recommendations, new releases and upcoming movie suggestions, and collection-based recommendations. The main objective of a recommendation system for movies is to provide personalized movie recommendation to users based on their preferences and viewing history, in order to enhance their experience and satisfaction with the movie platform. The goal is to help users discover new and interesting movies that they may not have otherwise found as well as, provide relevant and personalized movie suggestions to increase user engagement and satisfaction.

Recommendation systems for a movie website bring several benefits to both the website and its users. Firstly, by providing personalized and relevant movie suggestions, the recommendation system can increase user engagement and satisfaction. This can result in users spending more time on the site and having a better overall experience, leading to increased loyalty and repeat visits. Another benefit recommendation systems is improved content discovery. With the vast amount of content available on movie websites, it can be challenging for users to find new and interesting movies. Recommendation systems can help discover new titles they may not have otherwise found, increasing their chances of them finding something they like and improving their overall experience.

2. Overall Description

2.1 Product Perspective

A movie recommender system is a tool designed to help users discover new and relevant movies based on their individual preferences and viewing history.

Key features of a movie recommender system include:

1. Personalized recommendations: The system should use advanced algorithms (Pearson Correlation) to understand the user's preferences and recommend movies based on their clicks and ratings.
2. User-friendly interface: The system should have a simple, intuitive, and easy-to-use interface that allows users to search for movies.
3. Up-to-date movie database: The system should have a comprehensive database of movies and TV shows that is regularly updated with new releases and popular titles.
4. User reviews and ratings: The system should allow users to rate movies and leave reviews, providing valuable insight and feedback to other users.

The overall goal of a movie recommender system is to make movie discovery a delightful and personalized experience for users. By combining advanced algorithms, user-friendly interfaces, and a rich database of movies, the system should help users find the perfect movie to watch, every time.

3. Feature Lists

The following are the list of features available in this movie recommender system. Detailed explanation would be stated below.

Authentication

- Validation via JWT Token

User

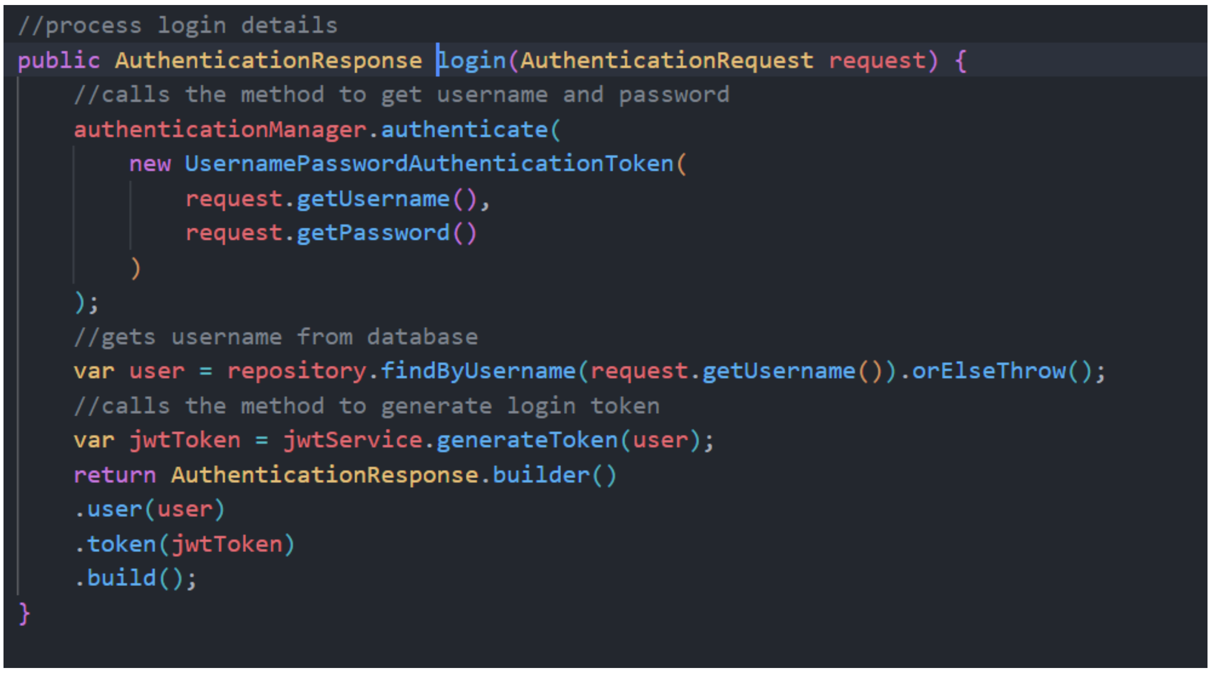
- View Movies
- Get recommended movies (uses the recommender system, which will be explained in chapter 3)
- Add movies to favorite
- Add rating to movies
- Search movie list

Admin

- View movie list
- Add user (normal/admin user)
- Add movies

3.1 Authentication (Auth Controller)

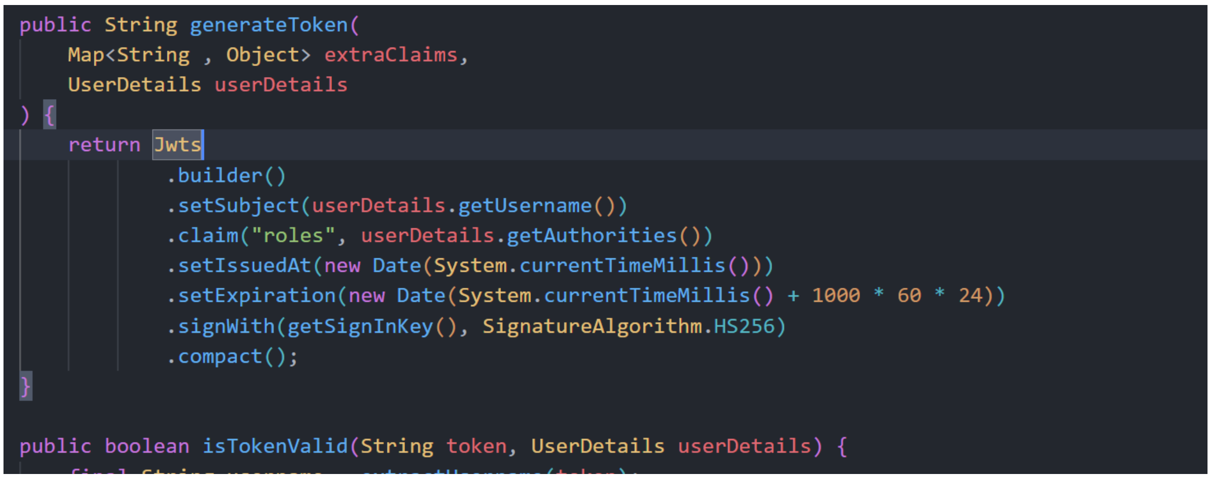
The authentication for the system uses JWT token validation to validates each user and grant them the stated token. The token for now is saved in browser's local storage which is then used throughout the web pages. The following figure 3.1.1 shows the source code where user details is processed after the form submission.



```
//process login details
public AuthenticationResponse login(AuthenticationRequest request) {
    //calls the method to get username and password
    authenticationManager.authenticate(
        new UsernamePasswordAuthenticationToken(
            request.getUsername(),
            request.getPassword()
        )
    );
    //gets username from database
    var user = repository.findByUsername(request.getUsername()).orElseThrow();
    //calls the method to generate login token
    var jwtToken = jwtService.generateToken(user);
    return AuthenticationResponse.builder()
    .user(user)
    .token(jwtToken)
    .build();
}
```

Figure 3.1.1: Source code for JWT Token Builder

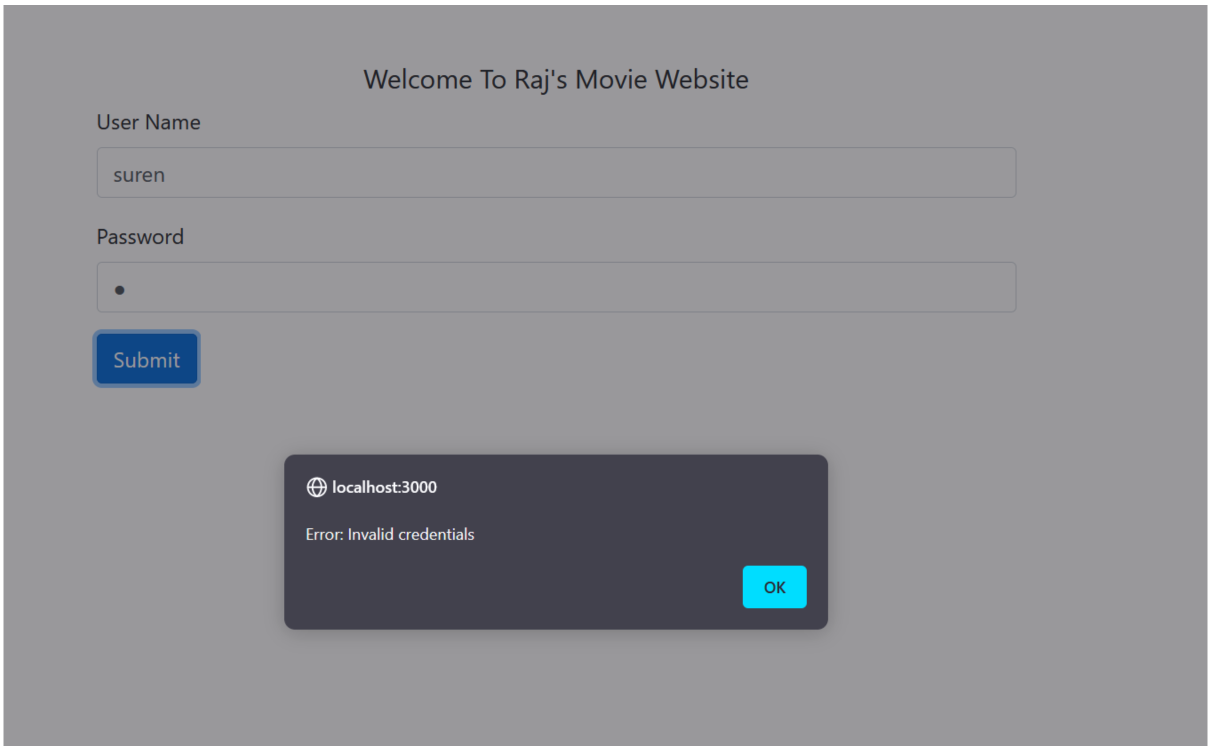
Based on the figure, the login and register method both calls for the ***generateToken()*** method to build the JWT token with given details. The method is as shown in Figure 3.1.2.



```
public String generateToken(
    Map<String , Object> extraClaims,
    UserDetails userDetails
) {
    return Jwts
            .builder()
            .setSubject(userDetails.getUsername())
            .claim("roles", userDetails.getAuthorities())
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + 1000 * 60 * 24))
            .signWith(getSignInKey(), SignatureAlgorithm.HS256)
            .compact();
}

public boolean isTokenValid(String token, UserDetails userDetails) {
```

Figure 3.1.2: generateToken() method

Based on the figure, beside the user details, token expiry time is also added for enhance security. Figure 3.1.3 shows the system message when user input the wrong credentials.


Welcome To Raj's Movie Website

User Name

suren

Password

●

Submit

localhost:3000

Error: Invalid credentials

OK

Figure 3.1.3: Wrong credentials prompt

Figure 3.1.4 shows the flow chart for login.

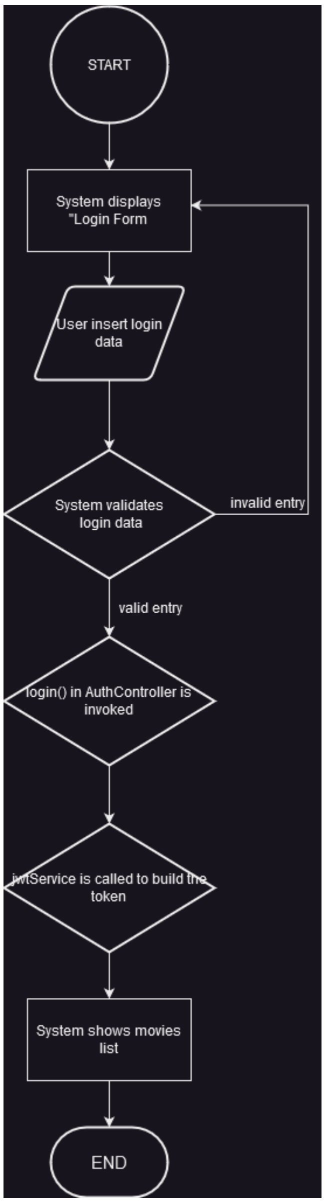

START
System displays "Login Form
User insert login data
System validates login data
invalid entry
valid entry
login() in AuthController is invoked
jwtService is called to build the token
System shows movies list
END

Figure 3.1.4: Login Flow Chart

For admin, the flow is the same except upon successful login, the system redirects the admin to the dashboard.

3.2 View Movies (User Controller)

This feature for user to view all the movies available within this movie recommender system. It is as shown in Figure 3.2.1.

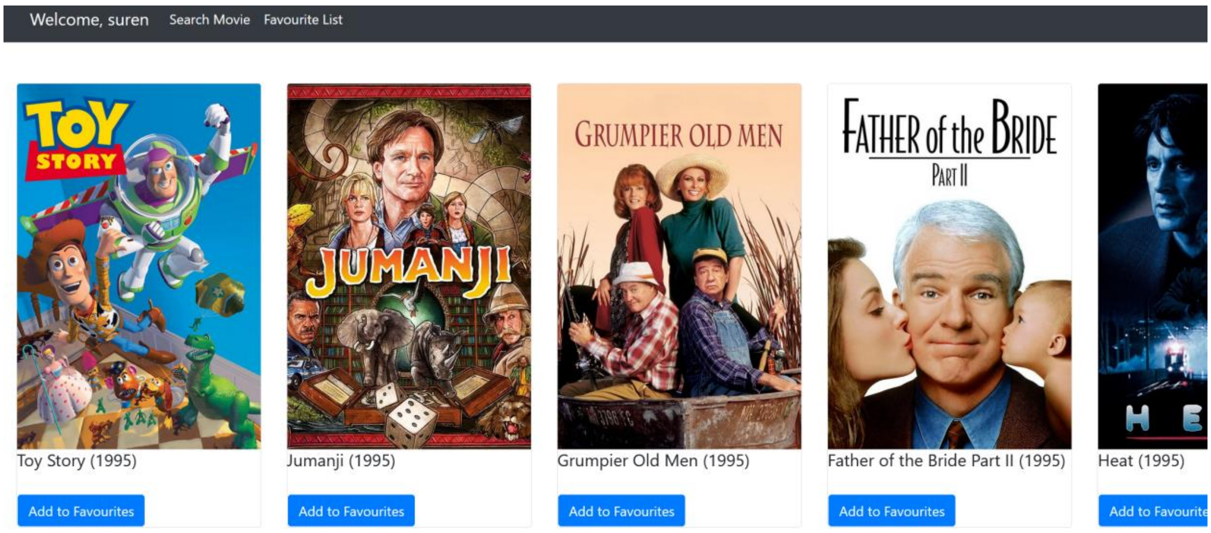
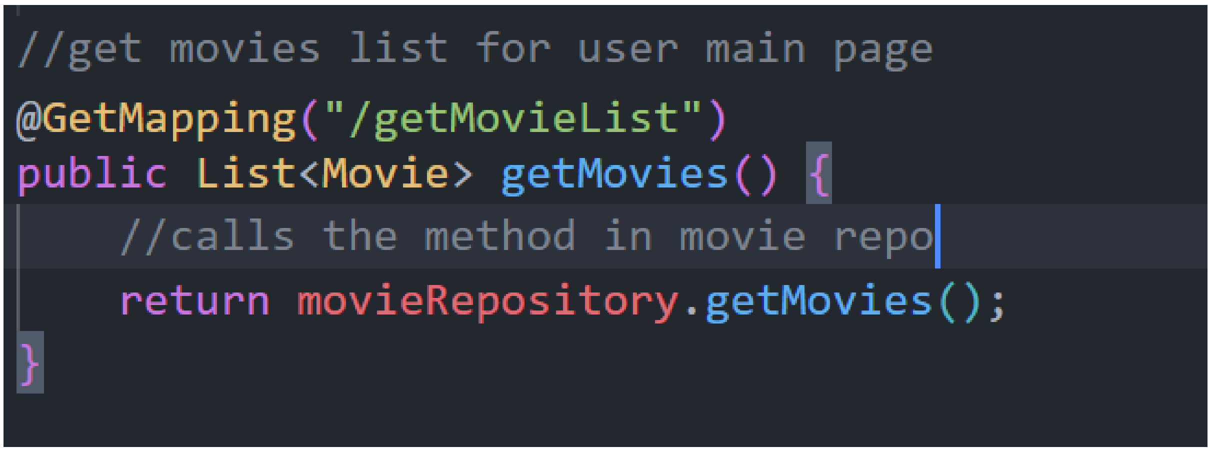


Figure 3.2.1: User Movie List

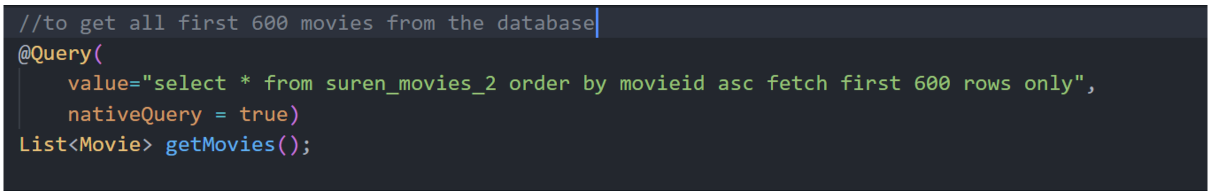
The list of movies is built by fetching two different API's. One would be the TMDB API to get the movie poster and the other one would the endpoint to retrieve movie information such as the title as shown in the figure. Figure 3.2.2 shows the source code responsible for this movie list feature.



```
//get movies list for user main page
@GetMapping("/getMovieList")
public List<Movie> getMovies() {
    //calls the method in movie repo
    return movieRepository.getMovies();
}
```

Figure 3.2.2: User Movie List Endpoint

The stated endpoint main purpose is to fetch the movie data that would be displayed to the user. The ***getMovies()*** method helps to communicate with database to get the requested data. The source code is as shown in Figure 3.2.3.



```
//to get all first 600 movies from the database
@Query(
    value="select * from suren_movies_2 order by movieid asc fetch first 600 rows only",
    nativeQuery = true)
List<Movie> getMovies();
```

Figure 3.2.3: Query for user movie list

Flow chart for retrieving movie list is as shown below.

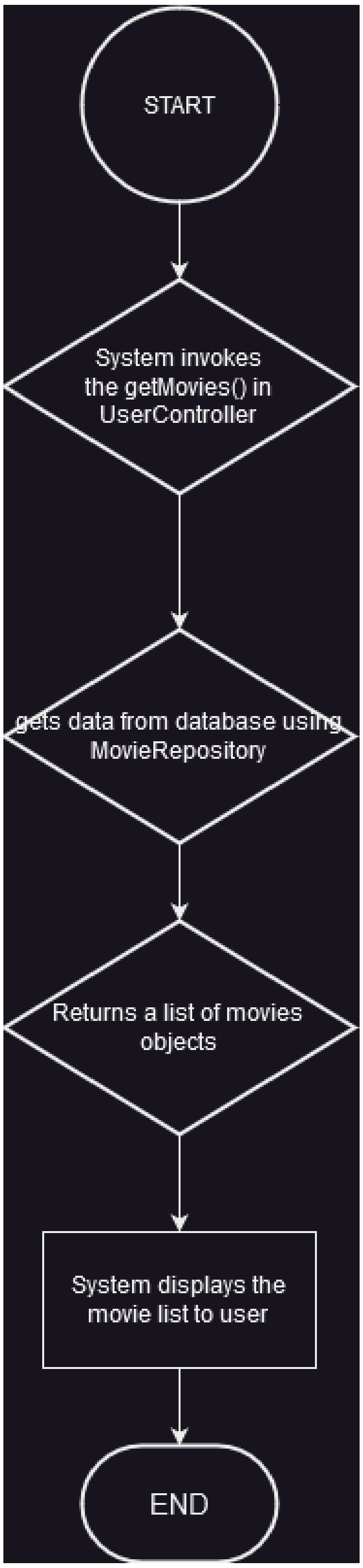
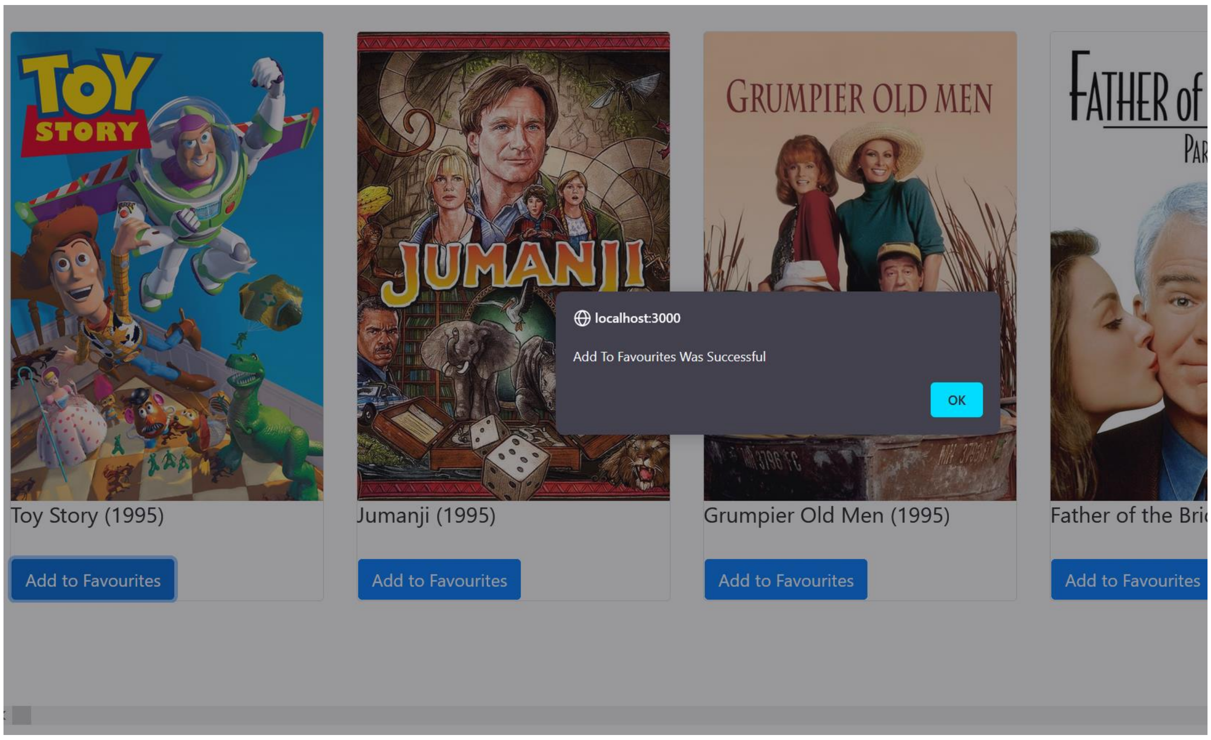

START
System invokes the getMovies() in UserController
gets data from database using MovieRepository
Returns a list of movies objects
System displays the movie list to user
END

Figure 3.2.4: Movie List Flow Chart

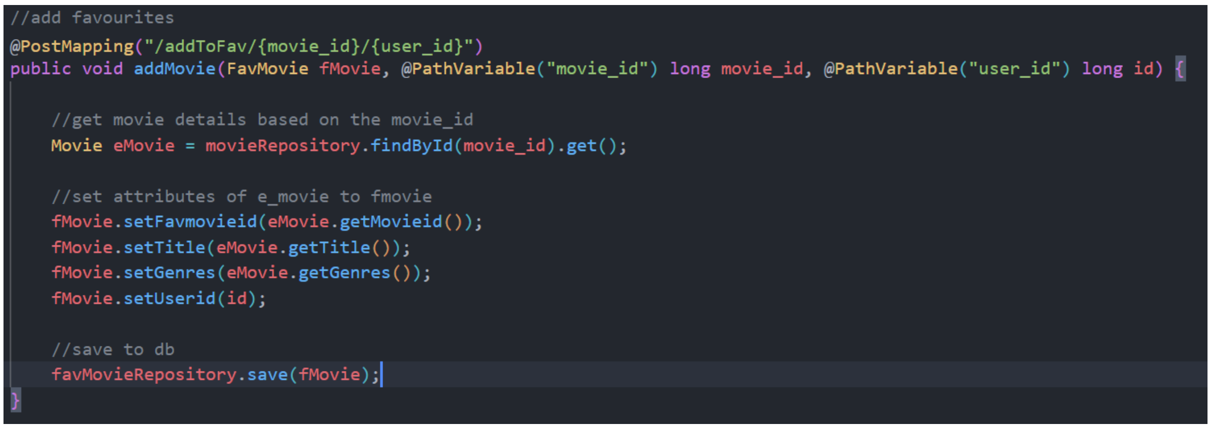
In the movie list page, user can add the movies to their favorite list by clicking on the “Add to Favorites”. Once clicking the button, the system will prompt a success message as shown in the Figure 2.2.5.



Add To Favourites Was Successful

Figure 3.2.5: Add to Favorite Success Message

The function responsible for above feature is as shown below.



```
//add favourites
@PostMapping("/addToFav/{movie_id}/{user_id}")
public void addMovie(FavMovie fMovie, @PathVariable("movie_id") long movie_id, @PathVariable("user_id") long id) {

    //get movie details based on the movie_id
    Movie eMovie = movieRepository.findById(movie_id).get();

    //set attributes of e_movie to fmovie
    fMovie.setFavmovieid(eMovie.getMovieid());
    fMovie.setTitle(eMovie.getTitle());
    fMovie.setGenres(eMovie.getGenres());
    fMovie.setUserid(id);

    //save to db
    favMovieRepository.save(fMovie);
}
```

Figure 3.2.6: Add to Favorite

Flow chart for add to favorite is as shown below.

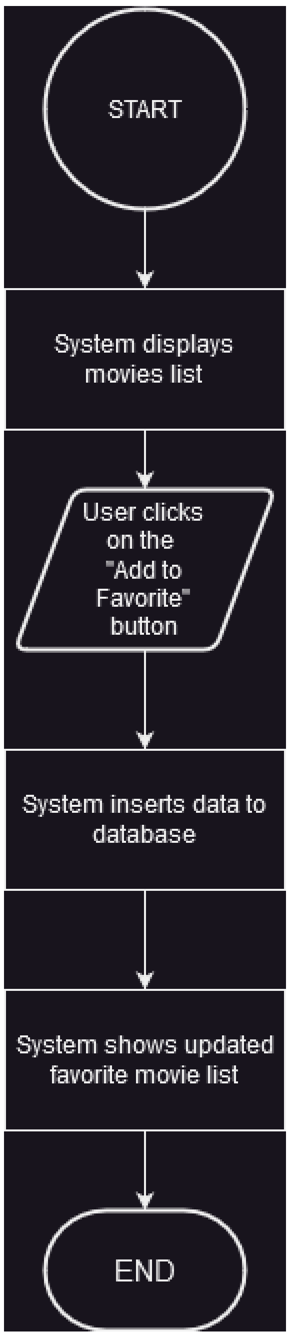

START
System displays movies list
User clicks on the "Add to Favorite" button
System inserts data to database
System shows updated favorite movie list
END

Figure 3.2.7: Add To Favorite Flow Chart

3.3 Search Movie (User Controller)

User can use the search function to search for movies.

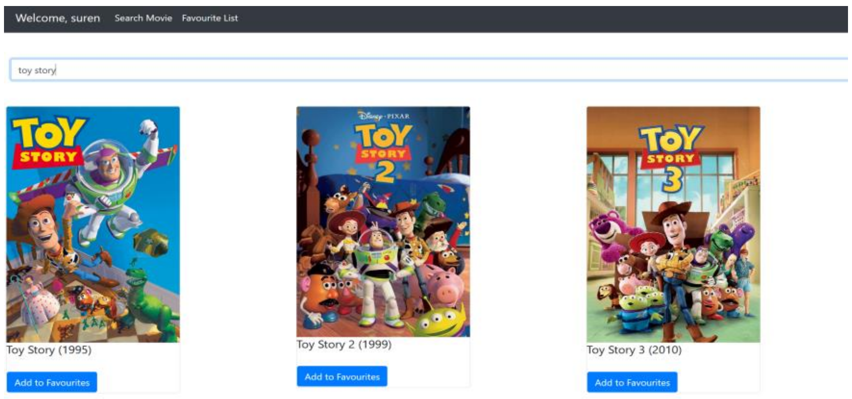
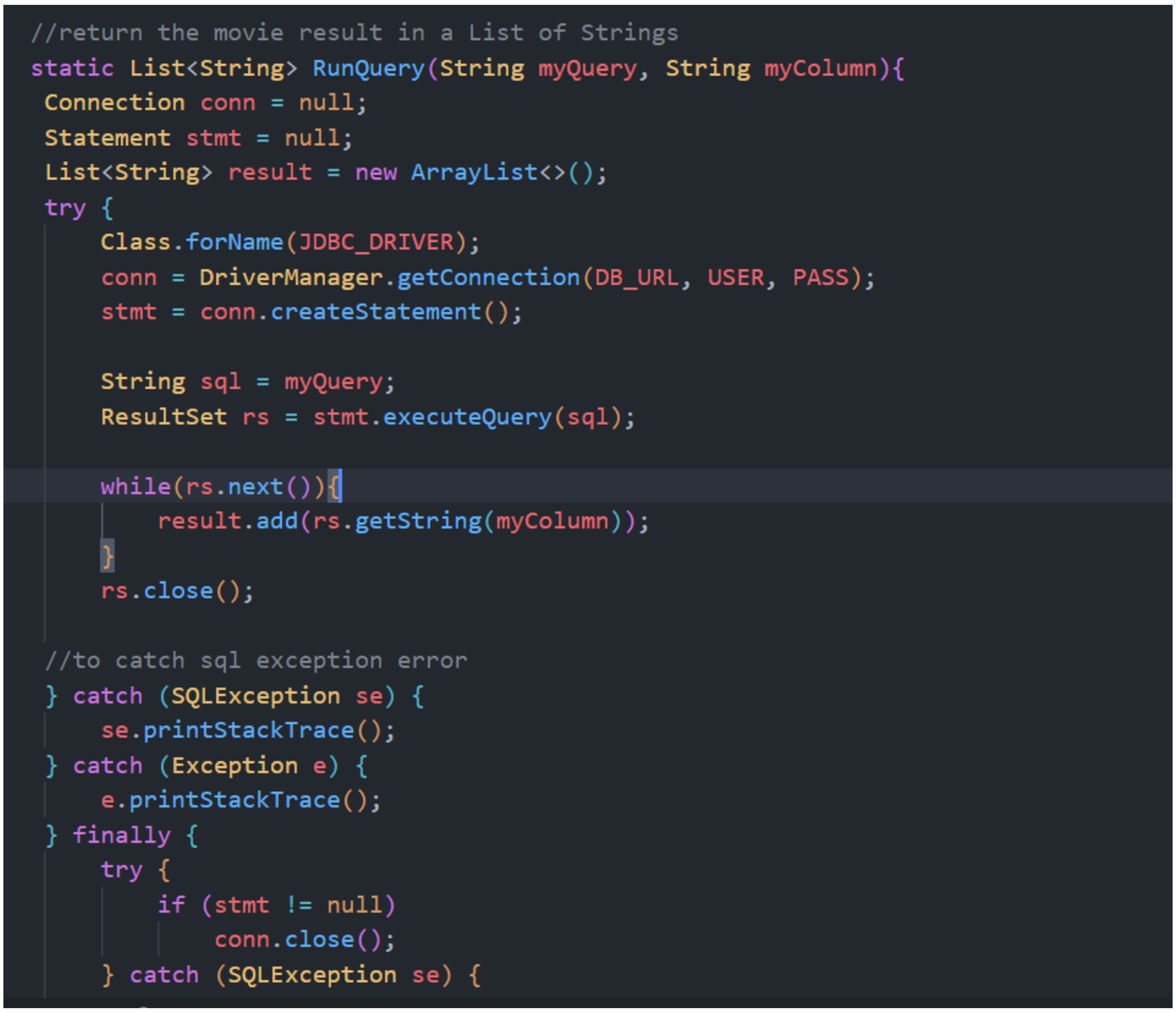


Figure 3.3.1: Search function

The source code for the search function is as shown below.



```
//return the movie result in a List of Strings
static List<String> RunQuery(String myQuery, String myColumn){
 Connection conn = null;
 Statement stmt = null;
 List<String> result = new ArrayList<>();
 try {
     Class.forName(JDBC_DRIVER);
     conn = DriverManager.getConnection(DB_URL, USER, PASS);
     stmt = conn.createStatement();

     String sql = myQuery;
     ResultSet rs = stmt.executeQuery(sql);

     while(rs.next()){
         result.add(rs.getString(myColumn));
     }
     rs.close();

 //to catch sql exception error
 } catch (SQLException se) {
     se.printStackTrace();
 } catch (Exception e) {
     e.printStackTrace();
 } finally {
     try {
         if (stmt != null)
             conn.close();
     } catch (SQLException se) {
```

Figure 3.3.2: Search Source Code

The flow chart for search function is shown below.

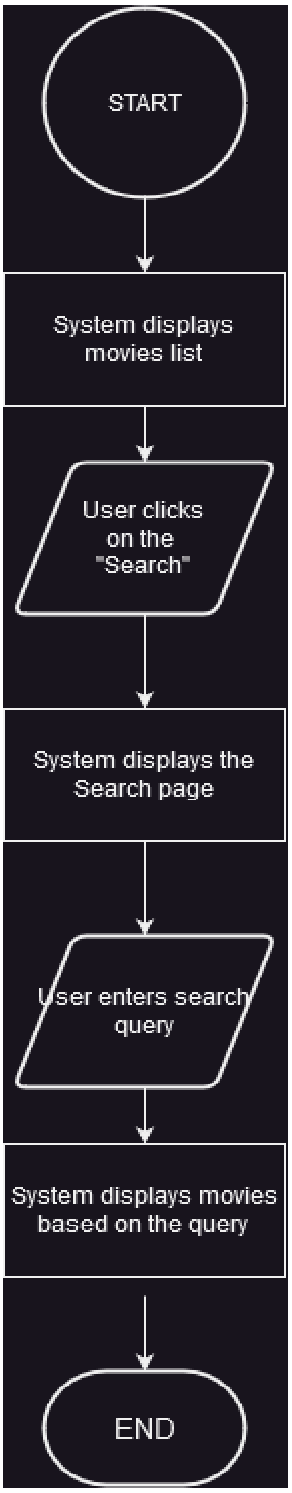

START
System displays movies list
User clicks on the "Search"
System displays the Search page
User enters search query
System displays movies based on the query
END

Figure 3.3.3: Search flow chart

3.4 Recommended movies (User Controller)

Recommended movies is shown to user whenever they click on any of the movie poster as shown in Figure 3.4.1.

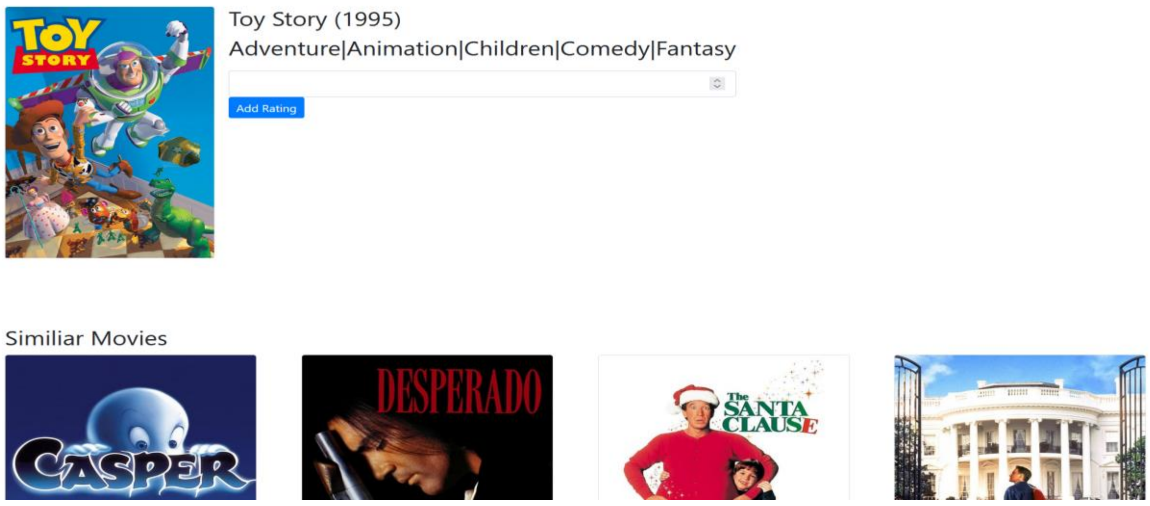
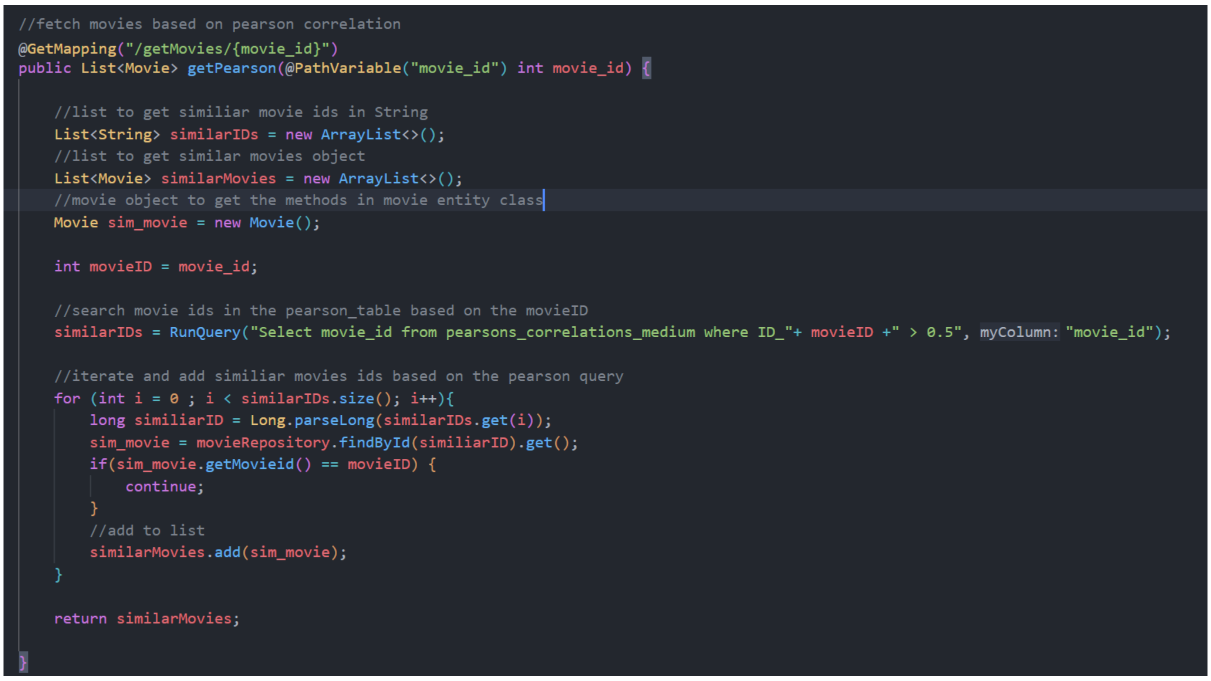


Figure 3.4.1: Recommended movies list.

The recommended movies are fetched using the function as shown in Figure 3.4.2.



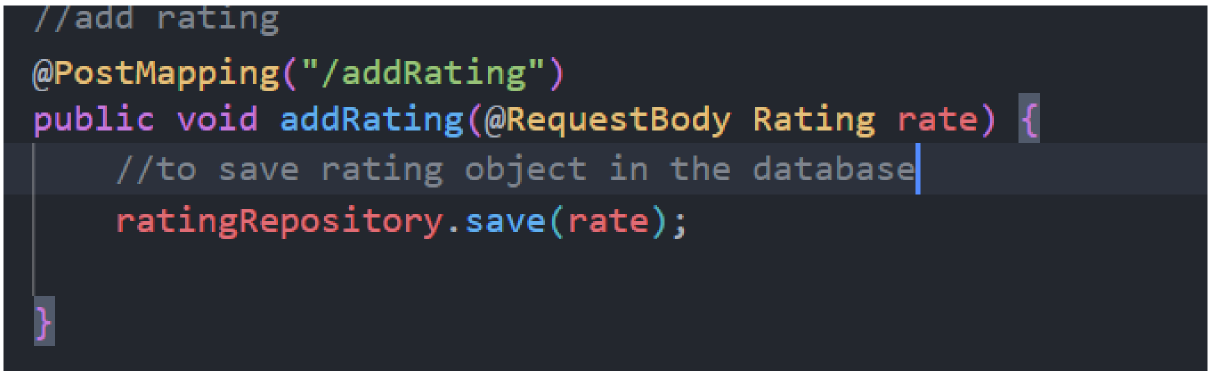
```
//fetch movies based on pearson correlation
@GetMapping("/getMovies/{movie_id}")
public List<Movie> getPearson(@PathVariable("movie_id") int movie_id) {

    //list to get similiar movie ids in String
    List<String> similarIDs = new ArrayList<>();
    //list to get similar movies object
    List<Movie> similarMovies = new ArrayList<>();
    //movie object to get the methods in movie entity class
    Movie sim_movie = new Movie();

    int movieID = movie_id;

    //search movie ids in the pearson_table based on the movieID
    similarIDs = RunQuery("Select movie_id from pearsons_correlations_medium where ID_"+ movieID +" > 0.5", myColumn: "movie_id");

    //iterate and add similiar movies ids based on the pearson query
    for (int i = 0 ; i < similarIDs.size(); i++){
        long similiarID = Long.parseLong(similarIDs.get(i));
        sim_movie = movieRepository.findById(similiarID).get();
        if(sim_movie.getMovieid() == movieID) {
            continue;
        }
        //add to list
        similarMovies.add(sim_movie);
    }

    return similarMovies;

}
```

Figure 3.4.2: Recommender Function

The shown function uses the **Pearson Correlation** to get movies that are similar to the movie clicked by the user. The explanation for how Pearson Correlation works would be stated in section 4.

3.5 Add Rating (User Controller)

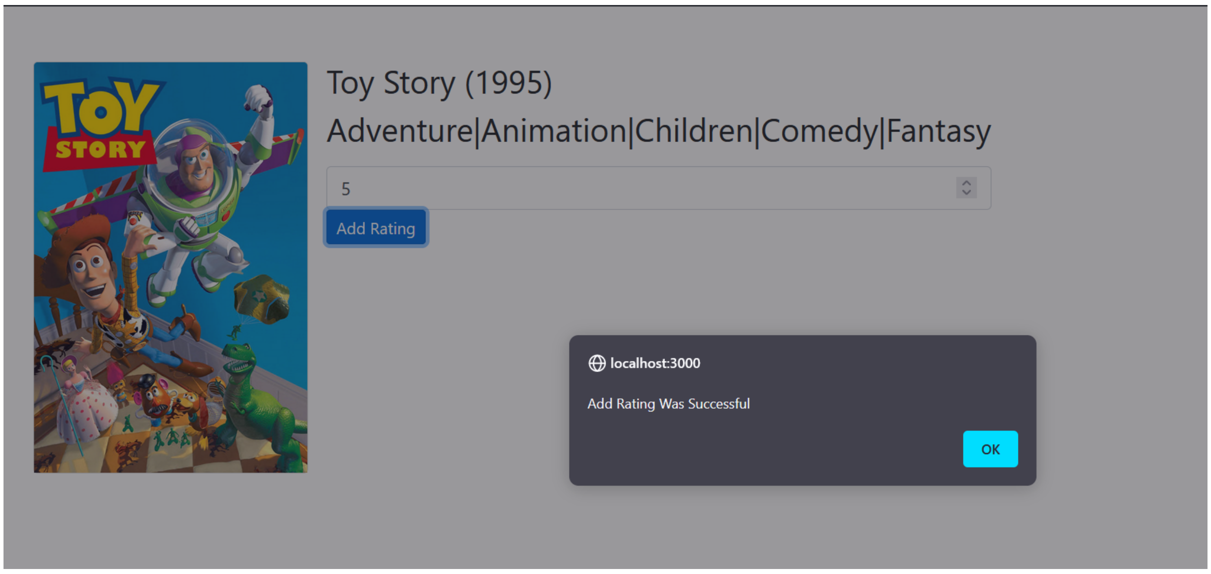
Based on figure 3.5.1, the user also can add rating to the movie via the rating form. The method to process the rating is as shown in Figure 3.5.1.



```
//add rating
@PostMapping("/addRating")
public void addRating(@RequestBody Rating rate) {
    //to save rating object in the database
    ratingRepository.save(rate);

}
```

Figure 3.5.1: addRating() method

Upon successful submission, the system displays the success message as shown in Figure 3.5.1.



Toy Story (1995)
Adventure|Animation|Children|Comedy|Fantasy

5

Add Rating

localhost:3000
Add Rating Was Successful

OK

Figure 3.5.1: Success message after adding rating.

The flow chart for the rating feature is as shown below.

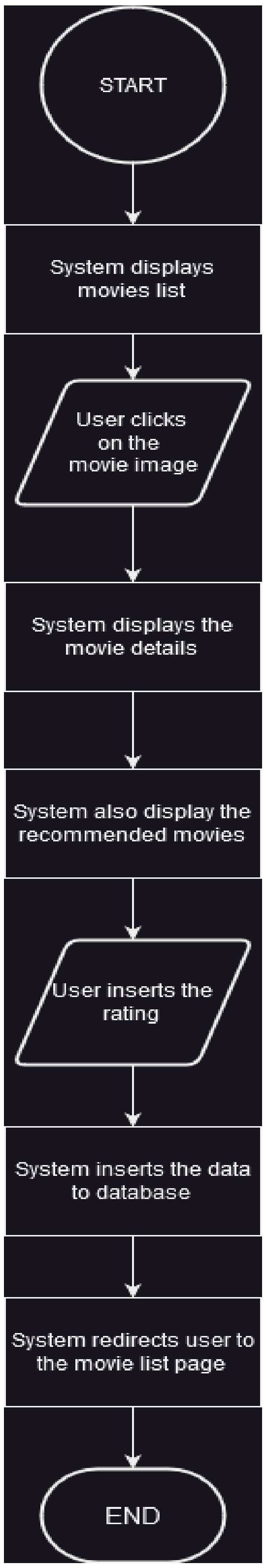
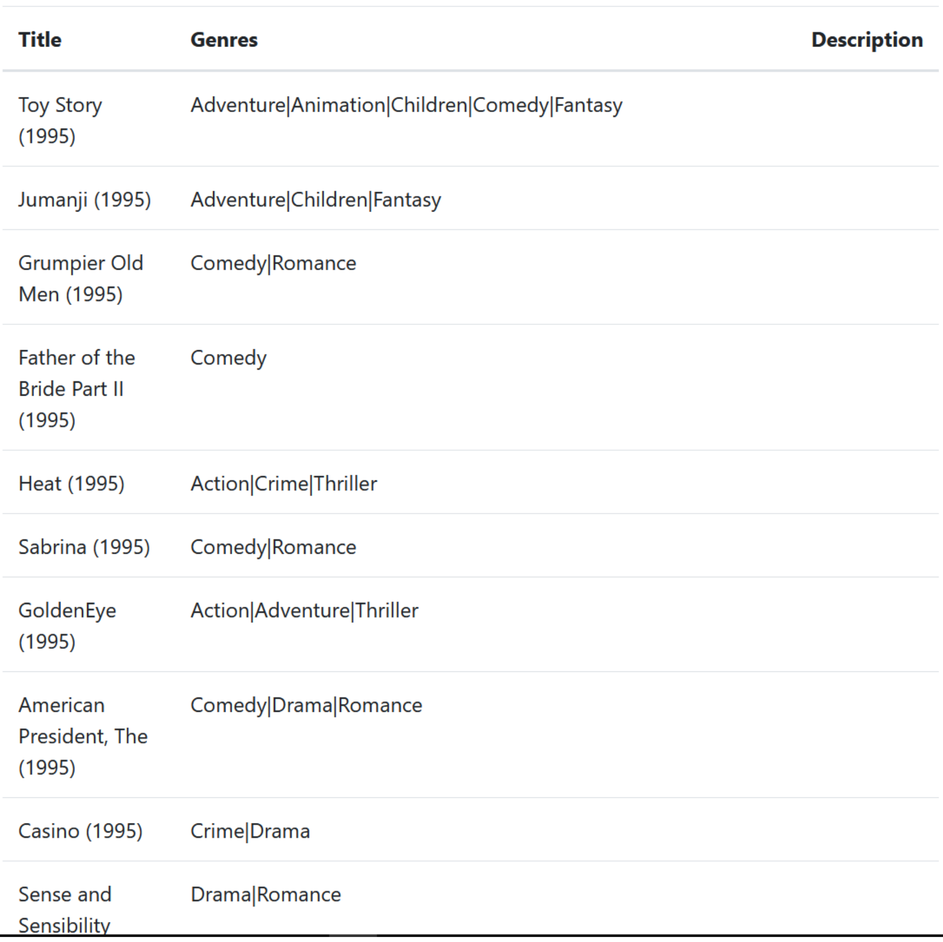

START
System displays movies list
User clicks on the movie image
System displays the movie details
System also display the recommended movies
User inserts the rating
System inserts the data to database
System redirects user to the movie list page
END

Figure 3.5.1: Movie Details Flow Chart

3.6 Admin View Movie (Admin Controller)

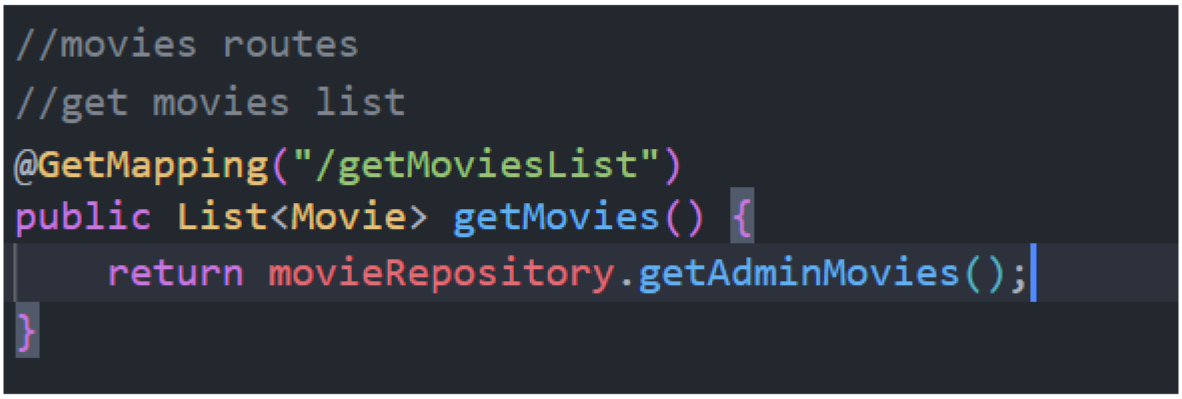
Admin can use this feature to view the movie list as shown in Figure 3.6.1.



Title	Genres	Description
Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	
Jumanji (1995)	Adventure|Children|Fantasy	
Grumpier Old Men (1995)	Comedy|Romance	
Father of the Bride Part II (1995)	Comedy	
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	Drama|Romance	

Figure 3.6.1: Admin View Movie List

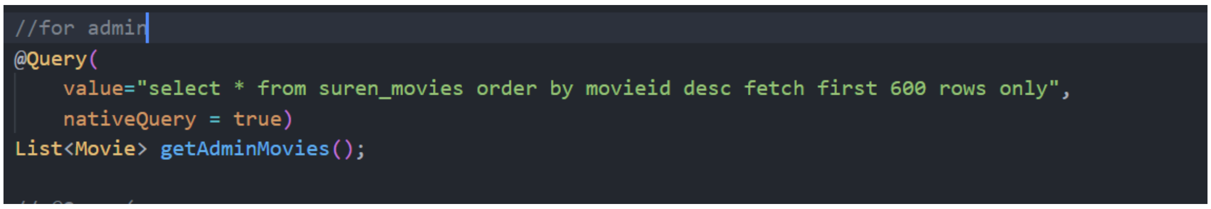
The following figure 3.5.2 is the source code responsible to fetch the movies data.



```
//movies routes
//get movies list
@GetMapping("/getMoviesList")
public List<Movie> getMovies() {
    return movieRepository.getAdminMovies();
}
```

Figure 3.6.2: getMovies()

The method calls for the ***getAdminMovies()*** which then retrieve movie list from the database.

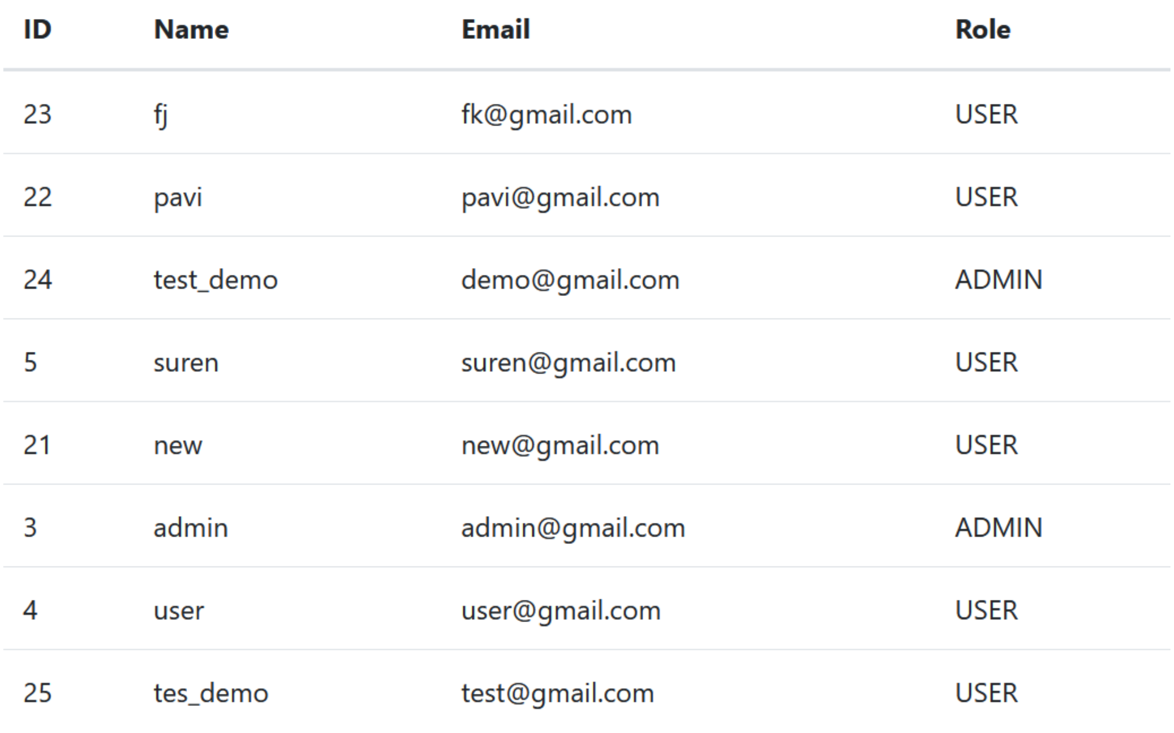


```
//for admin
@Query(
    value="select * from suren_movies order by movieid desc fetch first 600 rows only",
    nativeQuery = true)
List<Movie> getAdminMovies();
```

Figure 3.6.3: getAdminMovies()

3.7 Admin View User (Admin Controller)

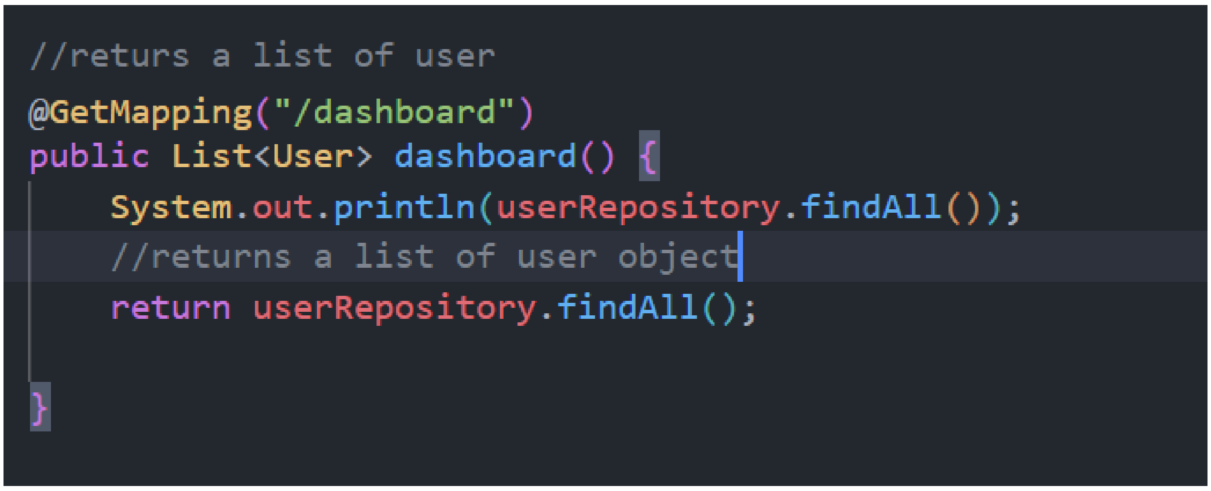
Admin can use this function to view all the available user in the system which includes both normal and admin user as shown in Figure 3.7.1.



ID	Name	Email	Role
23	fj	fk@gmail.com	USER
22	pavi	pavi@gmail.com	USER
24	test_demo	demo@gmail.com	ADMIN
5	suren	suren@gmail.com	USER
21	new	new@gmail.com	USER
3	admin	admin@gmail.com	ADMIN
4	user	user@gmail.com	USER
25	tes_demo	test@gmail.com	USER

Figure 3.7.1: Admin View User

The following source codes are responsible to fetch all the user's data from the database.

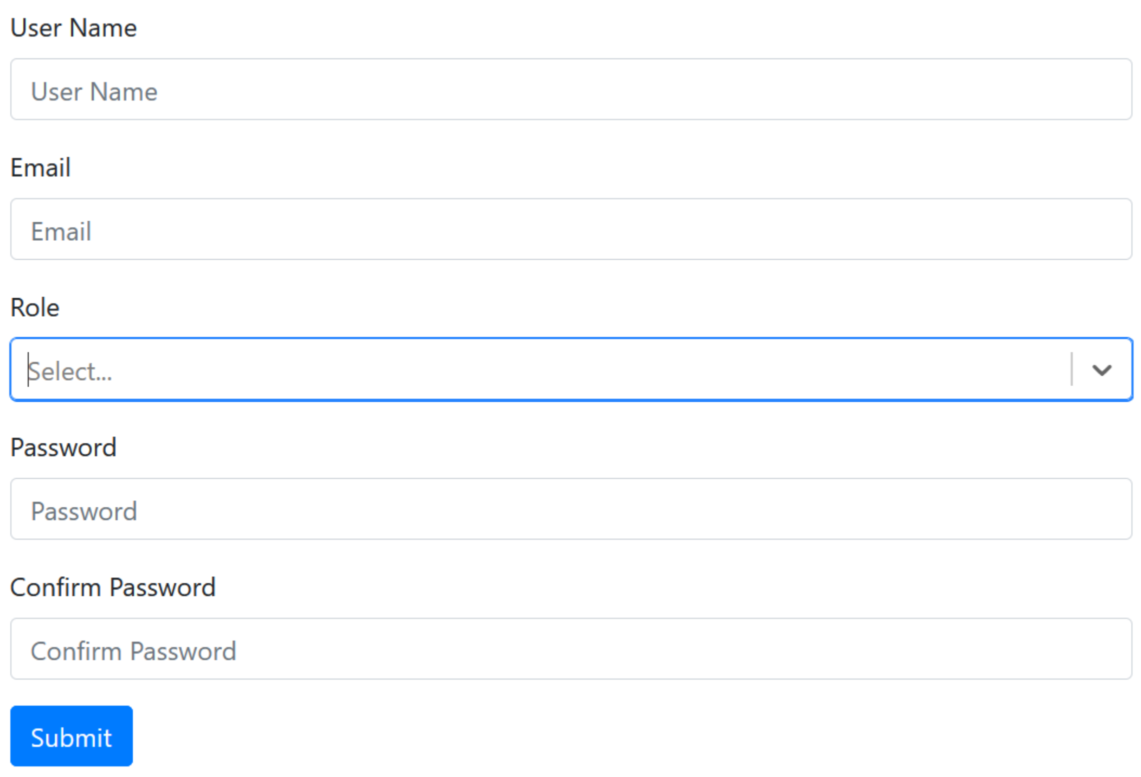


```
//returs a list of user
@GetMapping("/dashboard")
public List<User> dashboard() {
    System.out.println(userRepository.findAll());
    //returns a list of user object
    return userRepository.findAll();

}
```

Figure 3.7.2: dashboard()

3.8: Admin Add User (AuthController)

Admin can this feature to add new user to the system. Two types of user can be added, user and admin. The form is as shown in Figure 3.8.1.



User Name

User Name

Email

Email

Role

Select...

Password

Password

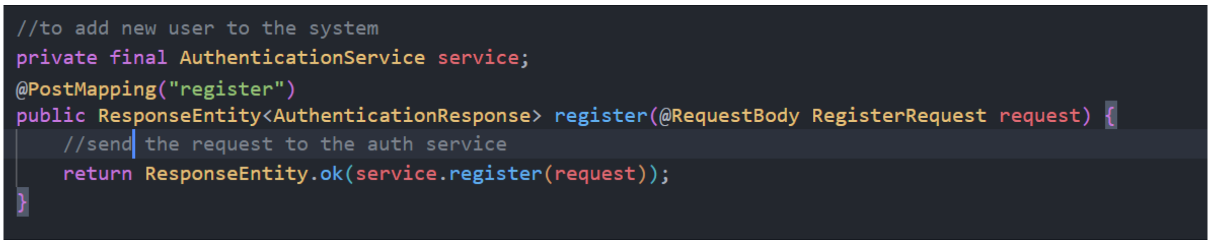
Confirm Password

Confirm Password

Submit

Figure 3.8.1: Admin Add User Form

Processing the form details would be done by the ***register()*** as shown in Figure 3.8.2.



```
//to add new user to the system
private final AuthenticationService service;
@PostMapping("register")
public ResponseEntity<AuthenticationResponse> register(@RequestBody RegisterRequest request) {
    //send the request to the auth service
    return ResponseEntity.ok(service.register(request));
}
```

Figure 3.8.2: register()

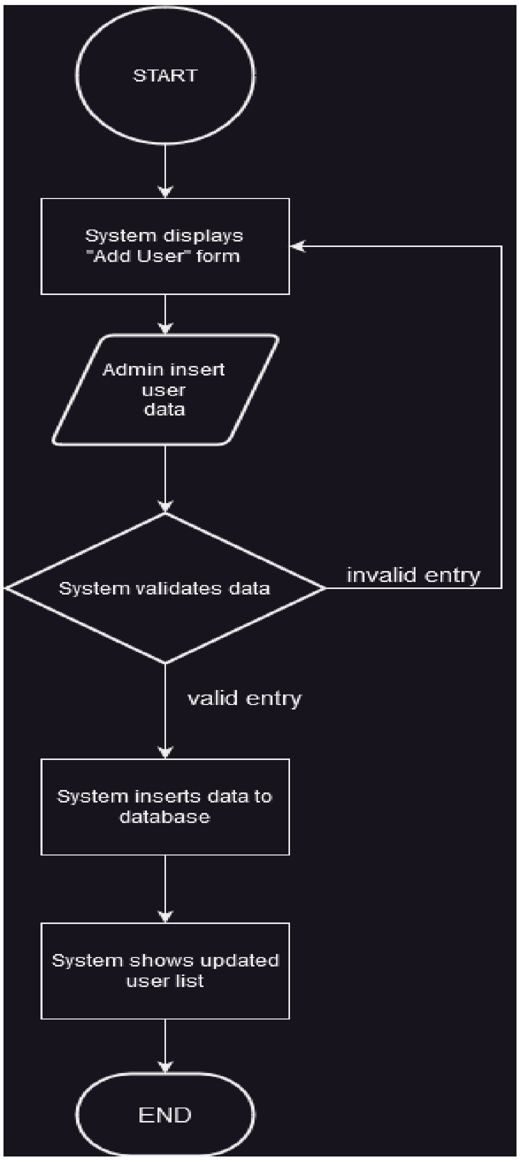

START
System displays "Add User" form
Admin insert user data
System validates data
invalid entry
valid entry
System inserts data to database
System shows updated user list
END

Figure 3.8.3: Add User Flow Chart

3.9: Admin Add Movie Form (Admin Controller)

Admin can use this form to add new movies to the system. The form is as shown in Figure 3.9.3.

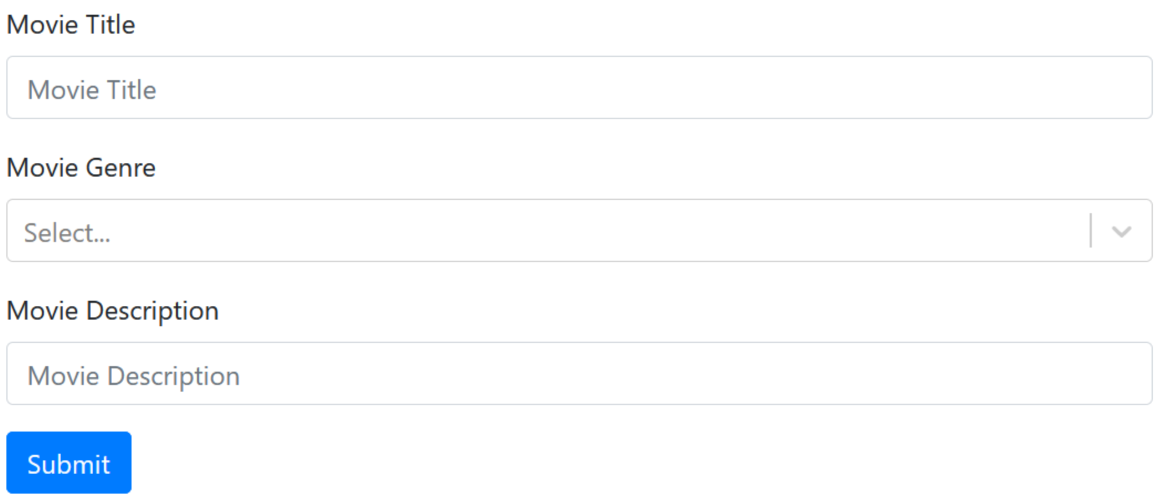
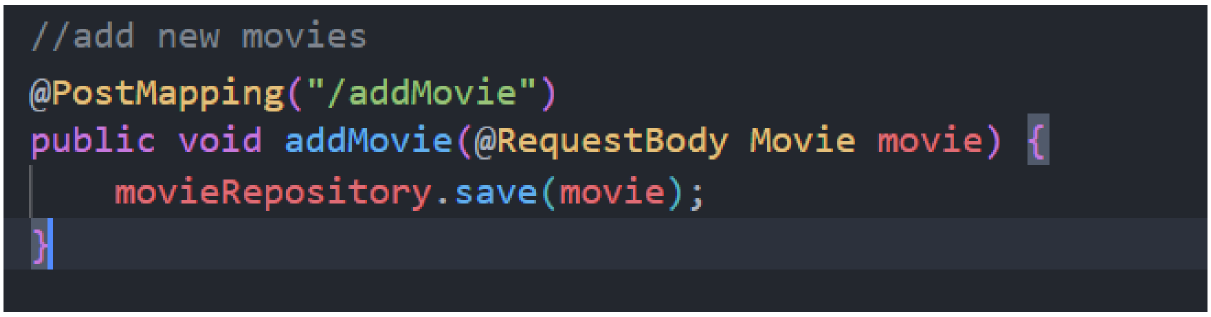


Figure 3.9.1: Admin Add Movie Form

The source code responsible for the processing the form is as shown in Figure 3.8.2.



```
//add new movies
@PostMapping("/addMovie")
public void addMovie(@RequestBody Movie movie) {
    movieRepository.save(movie);
}
```

Figure 3.9.2: addMovie()

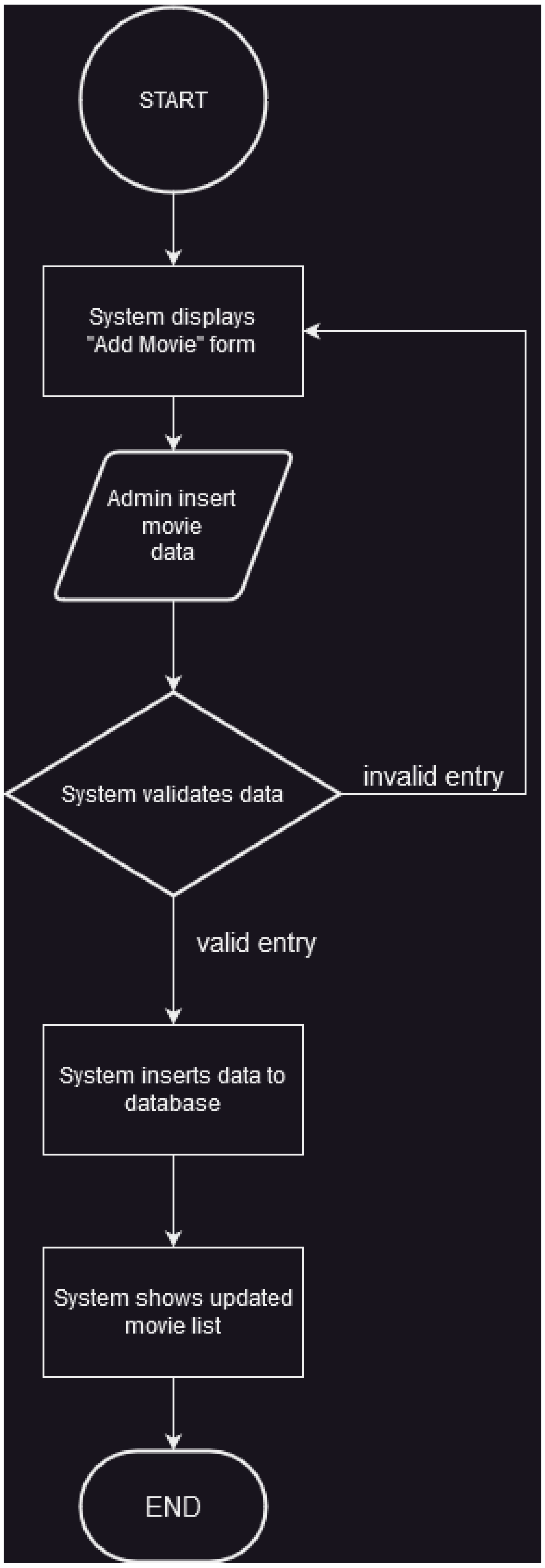

START
System displays "Add Movie" form
Admin insert movie data
System validates data
invalid entry
valid entry
System inserts data to database
System shows updated movie list
END

Figure 3.9.3: Add Movie Flow Chart

4. Entity-Relation Diagram (ERD)

Figure 4.1 shows the ERD which reflects the relationship of all the tables in the movie recommender database.

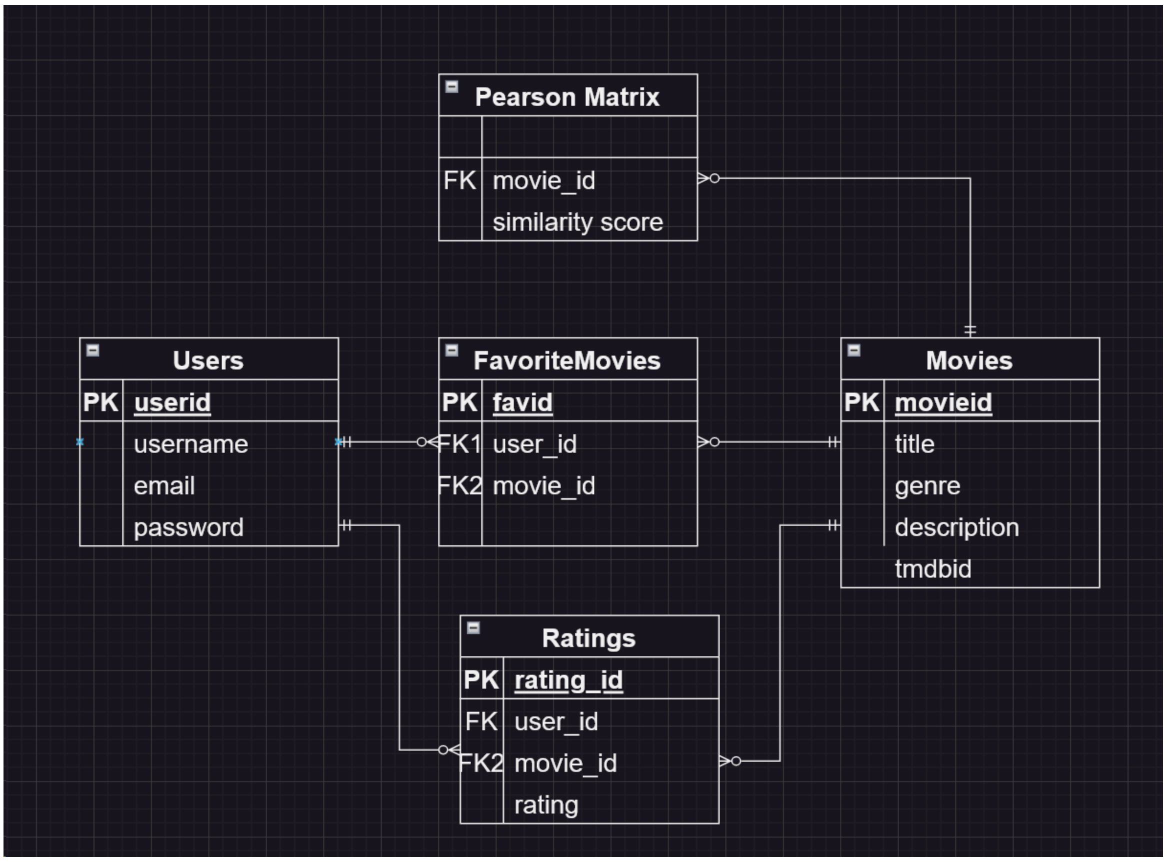


Figure 4.1: ERD of Movie Recommender

5. Pearson Correlation

Pearson correlation is a statistical method used to measure the linear relationship between two continuous variables.

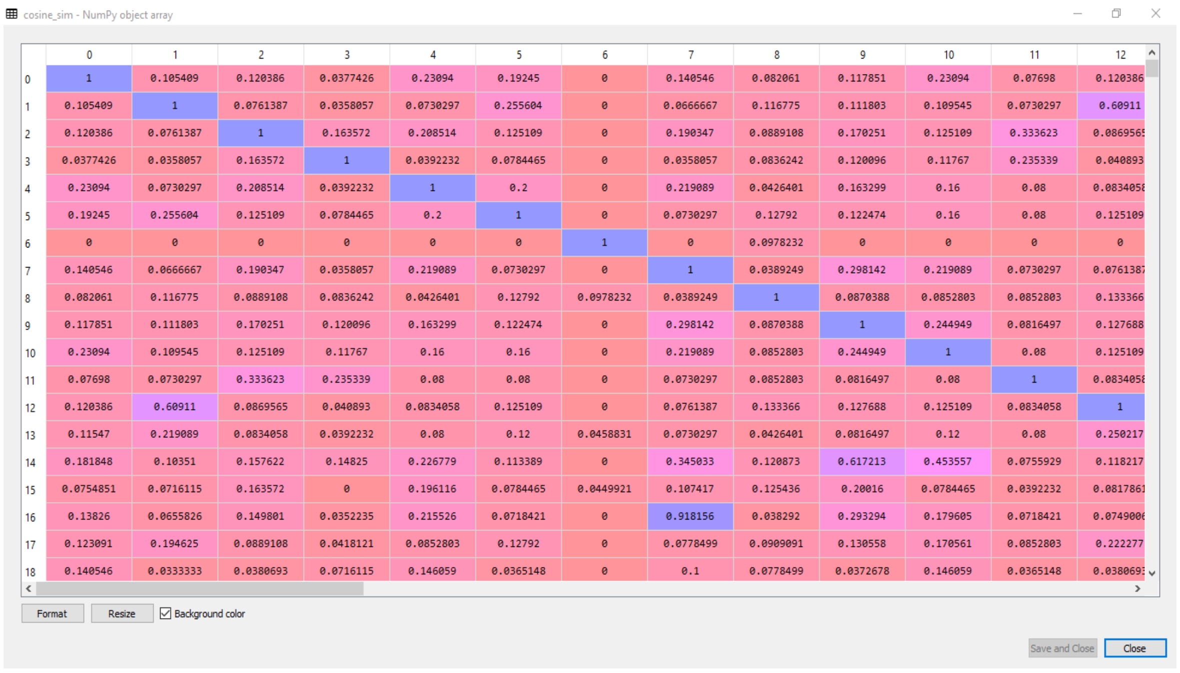
In a movie recommendation system, Pearson correlation can also be used to compare the ratings of different movies. The goal is to identify movies that are similar in terms of the ratings they receive from users. The Pearson correlation between two movies' ratings is calculated by comparing the ratings of each movie by all users. The result is a value between -1 and 1 that represents the strength and direction of the relationship between the two movies' ratings. A positive Pearson correlation indicates that the two movies are similar in terms of their ratings, while a negative Pearson correlation indicates that the movies have dissimilar ratings. If the Pearson correlation is close to 1, it means that the two movies are highly similar and can be recommended to users who liked one of the movies. Conversely, if the Pearson correlation is close to 0, it means that the two movies are not similar and may not be recommended to the same users. The Pearson correlation is a useful tool for movie recommendation systems, as it allows the system to make recommendations based on the similarities between movies, rather than just the popularity of the movies.

To create a movie similarity matrix using Pearson correlation, the following steps can be followed:

1. Gather movie ratings from many users for a set of movies. This data can be stored in a matrix where each row represents a movie, and each column represents a user. This data can be obtained by surveying users or by scraping data from websites like IMDb or Rotten Tomatoes.
2. Create a ratings matrix where each rows represent the user, and the columns represent the movies. The data cell contains the rating of each movie given by a particular user.
3. Calculate the Pearson correlation between each pair of movies. This can be done by comparing the movie ratings for each user. The result is a matrix where each cell contains the Pearson correlation between two movies. The formula for calculating Pearson's correlation is as follows:

$$r = \sum (x - \bar{x})(y - \bar{y}) / (n-1) \sqrt{(\sum (x - \bar{x})^2 (\sum (y - \bar{y})^2))}$$

4. Store the similarity matrix in a data structure such as a table or a matrix. This matrix can be used to make movie recommendations by finding movies that are like a user's favourite movies. Figure 5.1 shows how a similarity matrix looks like.



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1	0.105409	0.120386	0.0377426	0.23094	0.19245	0	0.140546	0.082061	0.117851	0.23094	0.07698	0.120386
1	0.105409	1	0.0761387	0.0358057	0.0730297	0.255604	0	0.0666667	0.116775	0.111803	0.109545	0.0730297	0.60911
2	0.120386	0.0761387	1	0.163572	0.208514	0.125109	0	0.190347	0.0889108	0.170251	0.125109	0.333623	0.086956
3	0.0377426	0.0358057	0.163572	1	0.0392232	0.0784465	0	0.0358057	0.0836242	0.120096	0.11767	0.235339	0.040893
4	0.23094	0.0730297	0.208514	0.0392232	1	0.2	0	0.219089	0.0426401	0.163299	0.16	0.08	0.083405
5	0.19245	0.255604	0.125109	0.0784465	0.2	1	0	0.0730297	0.12792	0.122474	0.16	0.08	0.125109
6	0	0	0	0	0	0	1	0	0.0978232	0	0	0	0
7	0.140546	0.0666667	0.190347	0.0358057	0.219089	0.0730297	0	1	0.0389249	0.298142	0.219089	0.0730297	0.076138
8	0.082061	0.116775	0.0889108	0.0836242	0.0426401	0.12792	0.0978232	0.0389249	1	0.0870388	0.0852803	0.0852803	0.133366
9	0.117851	0.111803	0.170251	0.120096	0.163299	0.122474	0	0.298142	0.0870388	1	0.244949	0.0816497	0.127688
10	0.23094	0.109545	0.125109	0.11767	0.16	0.16	0	0.219089	0.0852803	0.244949	1	0.08	0.125109
11	0.07698	0.0730297	0.333623	0.235339	0.08	0.08	0	0.0730297	0.0852803	0.0816497	0.08	1	0.083405
12	0.120386	0.60911	0.0869565	0.040893	0.0834058	0.125109	0	0.0761387	0.133366	0.127688	0.125109	0.0834058	1
13	0.11547	0.219089	0.0834058	0.0392232	0.08	0.12	0.0458831	0.0730297	0.0426401	0.0816497	0.12	0.08	0.250217
14	0.181848	0.10351	0.157622	0.14825	0.226779	0.113389	0	0.345033	0.120873	0.617213	0.453557	0.0755929	0.118217
15	0.0754851	0.0716115	0.163572	0	0.196116	0.0784465	0.0449921	0.107417	0.125436	0.20016	0.0784465	0.0392232	0.081786
16	0.13826	0.0655826	0.149801	0.0352235	0.215526	0.0718421	0	0.918156	0.038292	0.293294	0.179605	0.0718421	0.074900
17	0.123091	0.194625	0.0889108	0.0418121	0.0852803	0.12792	0	0.0778499	0.0909091	0.130558	0.170561	0.0852803	0.222277
18	0.140546	0.0333333	0.0380693	0.0716115	0.146059	0.0365148	0	0.1	0.0778499	0.0372678	0.146059	0.0365148	0.038069

Figure 5.1: Similarity Matrix of Movie's Ratings