

Software Requirements Specifications<SRS> For Movie Recommendation System

Prepared by Ronan Irvine Gondipon

Contents

1.0 Introduction.....	3
1.1 Purpose.....	3
1.2 Intended Audience and Reading suggestion.	3
1.3 Project Scope	3
2.0 Overall Description.....	3
2.1 Product Perspective.....	3
3.0 System features	4
3.1 Login Page	4
3.1.1 Login Flowchart.....	5
3.2 Registration Page	6
3.2.1 Registration Flowchart.....	8
3.3 Movie Management:	9
3.3.1 List Movies Page.....	9
3.3.2 Add Movie Page	10
3.3.3 Update Movies Page	13
3.3.4 Delete Movie Function	13
3.3.5 View Movie Page.....	15
3.3.6 Movie Recommendations Component.....	16
3.3.7 Rating Movie	18
3.3.8 Add Favourite Movies Function	19
3.3.9 List Favourite movies Pages	21
3.3.10 Remove Favourite movie function.....	22
3.4 User management.....	24
3.4.1 List Users Page	24
3.4.2 Add User Page	26
3.4.3 Update User Page.....	27
3.4.4 Delete User Function	28
3.4.6 Promote User Function	30
4.0 Entity Relationship Diagram (ERD).....	31
5.0 Pearson's correlation.....	31

1.0 Introduction

1.1 Purpose

A recommendation system is a tool that makes suggestions to users about resources like books, movies, songs, etc., based on data. Movie recommendation systems, for example, predict which movies a user will enjoy by analyzing the attributes of movies they've liked in the past. These systems are valuable for organizations that collect data from many customers and want to provide the best recommendations. There are several factors that can influence the design of a movie recommendation system, such as the genre of the movie, as well as how well received the movies are. The system can recommend movies based on one or a combination of these attributes. In this study, the recommendation system is based on the genres of movies that the user is likely to enjoy, using a content-based filtering method that considers genre and rating correlation.

1.2 Intended Audience and Reading suggestion.

The intended audience for a movie recommendation system is typically anyone who is interested in finding new and personalized movie suggestions based on their viewing history and preferences. This can include casual movie-goers, film enthusiasts, and anyone who regularly watches movies and wants to discover new titles that they may enjoy.

1.3 Project Scope

The scope of the movie recommendation system is to provide personalized movie recommendations to users based on their movie preferences and watching history while utilizing different type of calculations for similarity matrixes and see how they will impact the recommendations. The movie recommendation system will use data from a variety of sources, including movie ratings and reviews, user preferences, and other relevant data sources to get the best. The movie recommendation system will allow users to rate and review movies, view recommendations based on their preferences, and provide feedback on the accuracy of the recommendations.

2.0 Overall Description

2.1 Product Perspective

A movie recommendation system from a product perspective would aim to provide personalized movie recommendations to users based on their individual preferences, viewing

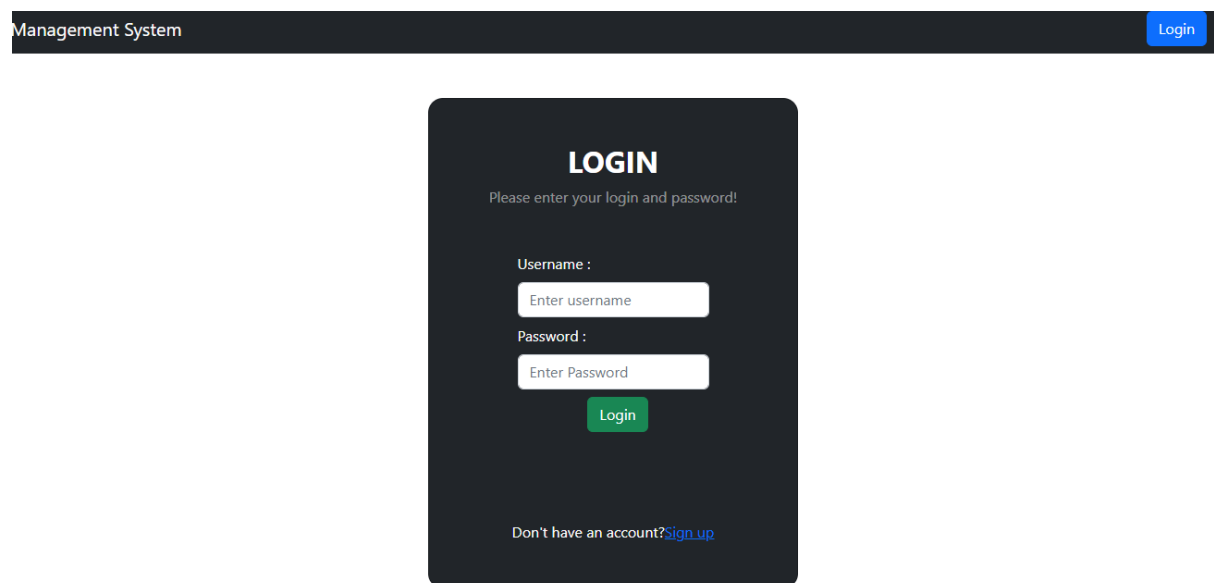
history, and ratings. The goal is to improve the user experience by helping them discover new movies they are likely to enjoy and save time in their search for something to watch.

The main features of the movie recommendation system include:

1. Customized Recommendations: The system should employ sophisticated Pearson Correlation algorithms to analyze the user's tastes and suggest films based on their viewing history and evaluations.
2. Intuitive Design: The system should feature a straightforward, user-friendly interface that enables users to quickly search for movies.
3. Up-to-date movie database: The system should have a complete and up-to-date database of movies and TV shows, including recent releases and well-known titles.
4. Favorites list: Users can add movies to their favorites

3.0 System features

3.1 Login Page



The screenshot displays the login interface of a 'Management System'. At the top, a dark header bar contains the text 'Management System' on the left and a blue 'Login' button on the right. The main content area features a dark, rounded rectangular login card. Inside the card, the word 'LOGIN' is prominently displayed in white. Below it, a prompt reads 'Please enter your login and password!'. The card contains two input fields: 'Username :' with a placeholder 'Enter username' and 'Password :' with a placeholder 'Enter Password'. A green 'Login' button is positioned below the password field. At the bottom of the card, a link states 'Don't have an account? [Sign up](#)'.

Figure 1 Login Page

This login validation function is utilized to ensure that the entered values. Once validated, the user information will be passed to the controller as an object and passed to the service.

```
@PostMapping("/userLogin")
public ResponseEntity<?> loginUser(@RequestBody User user) { //getting user object from body
    return userService.loginUser(user); //passing user object for login function in service
}
```

are validated to be an existing user's and if they are correct.

```
public ResponseEntity<?> loginUser (User user)
{
    User userdata = repo.findByUsername (user.getUsername()); //VERIFY IS USER EXISTS
    if (userdata != null)
    {
        if (user.getUserpassword ().equals(userdata.getUserpassword ()))
        {
            return new ResponseEntity<User> (userdata, HttpStatus.OK);
        }
        else
        {
            return new ResponseEntity<> ( body: "Wrong Password", HttpStatus.BAD_REQUEST);
        }
    }
    else
    {
        return new ResponseEntity<> ( body: "User does not exist!", HttpStatus.BAD_REQUEST);
    }
}
```

3.1.1 Login Flowchart

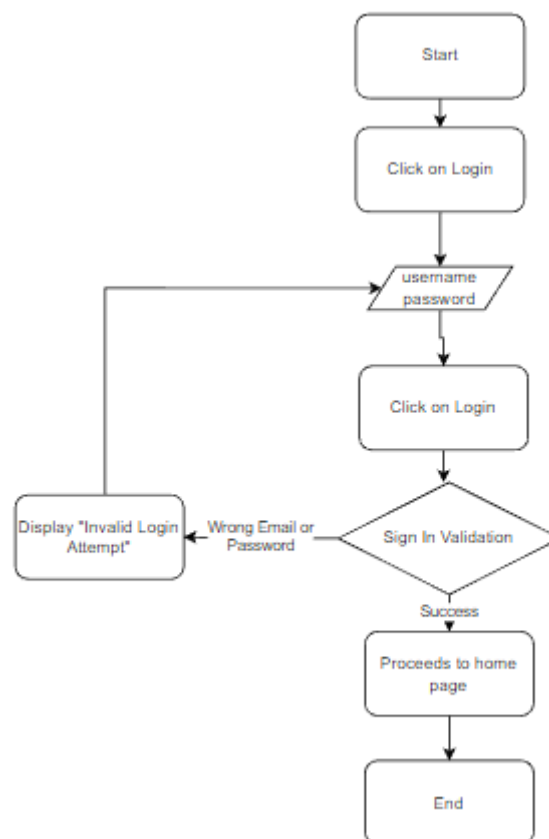
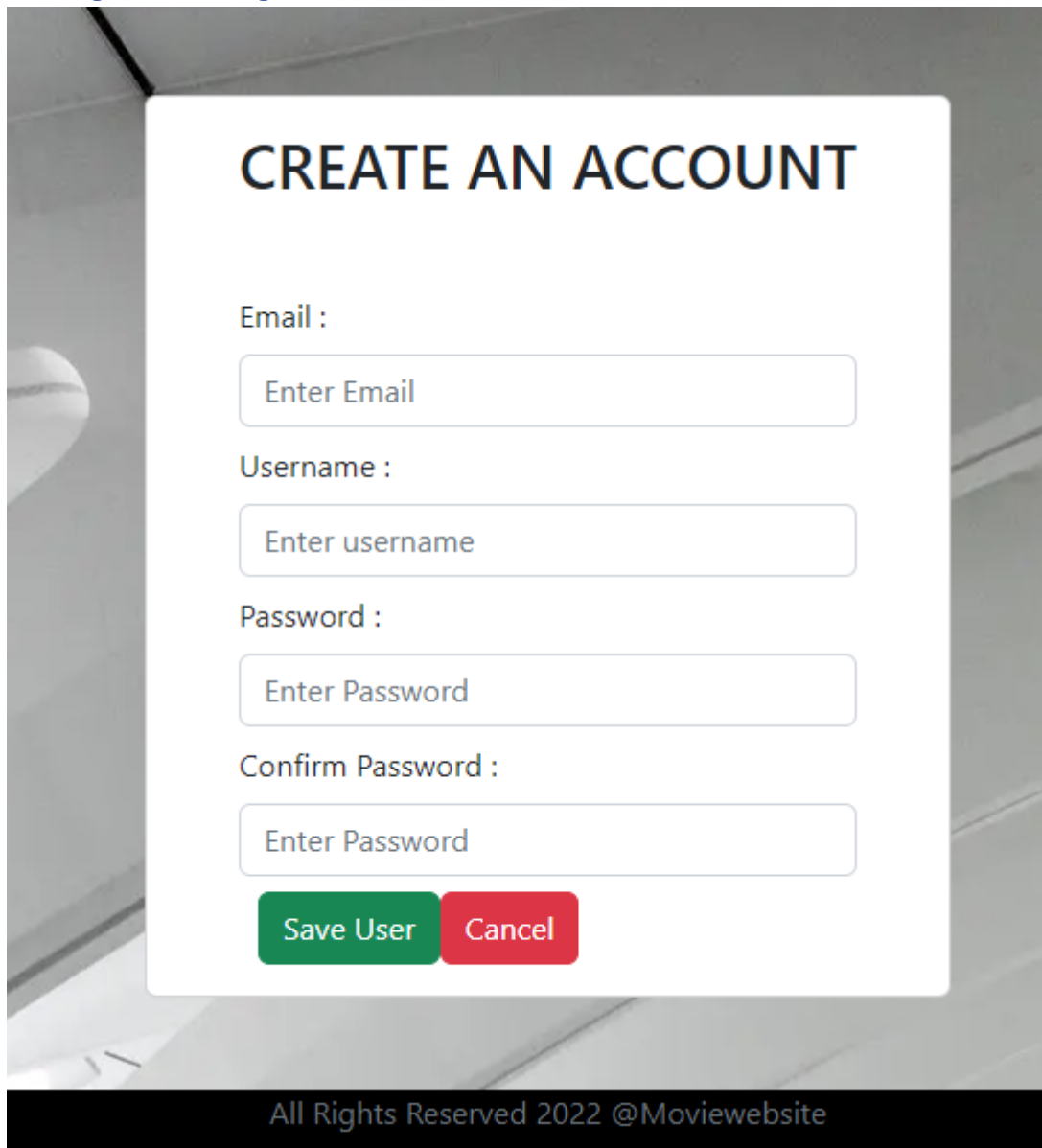


Figure 2 Flowchart for User Login

3.2 Registration Page



The image shows a registration form titled "CREATE AN ACCOUNT" in a white box with rounded corners, centered on a dark background. The form contains four input fields, each with a label above it: "Email :", "Username :", "Password :", and "Confirm Password :". Each input field has a placeholder text "Enter Email", "Enter username", "Enter Password", and "Enter Password" respectively. Below the input fields are two buttons: a green "Save User" button and a red "Cancel" button. At the bottom of the dark background, there is a black bar with the text "All Rights Reserved 2022 @Moviewebsite" in white.

Figure 3 Registration Page

New Users have to enter their, username, email, password and confirmation password, Validations are made to ensure that they have entered the necessary values have matching passwords. Below is the frontend code to ensure that the validation is done correctly.

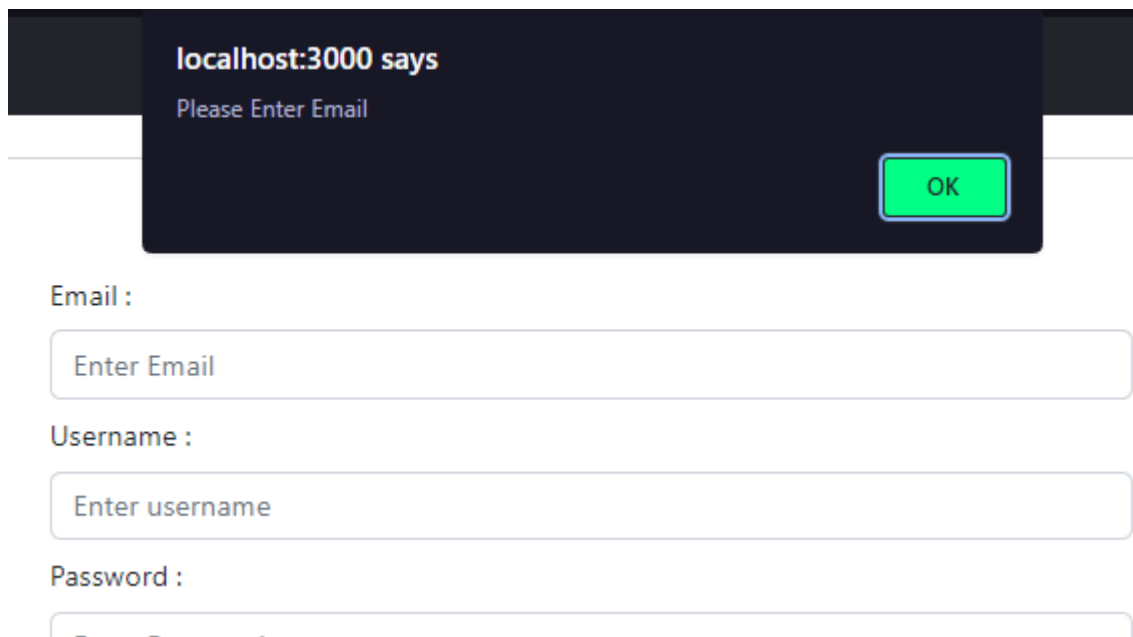
```

const checkTextInput = (e) => {
  //Check for the Email TextInput
  if (!email.trim()) {
    alert('Please Enter Email');
    return;
  }
  //Check for the Username TextInput
  if (!username.trim()) {
    alert('Please Enter Username');
    return;
  }
  //Check for the Password TextInput
  if (!userpassword.trim()) {
    alert('Please Enter Password');
    return;
  }
  if (confirmuserpassword !== userpassword) { //validate if passwords are the same
    alert('Password & Confirm Password are not Equal');
    return;
  }
  //Checked Successfully
  saveUser(e) //call save user function
}

```

Figure 4 Registration Frontend Validation

Alerts will then be used to notify users in fields that they have to fill.



localhost:3000 says

Please Enter Email

OK

Email :

Enter Email

Username :

Enter username

Password :

Figure 5 Alert for empty field

Once values have been validated, the data will be passed as an object to the corresponding API path below which will call a service to add the information.

```

@PostMapping("/newuser")
public boolean newUser(@RequestBody User user) throws Exception {
    return userService.saveuser(user);
}

```

Figure 6 Registration Controller

Below is the code to insert the user registration information to the database as a new user, new users' user type is set to 'User' by default.

```

private String InsertQuery="INSERT INTO Ronan_User (email,username,userpassword,usertype) VALUES (?,?,'User')";
public boolean saveUser(User user){ //function for inserting new user
    return jdbcTemplate.update(InsertQuery, user.getEmail(),user.getUsername(), user.getUserpassword()) > 0;
}

```

Figure 7 Code for Inserting new user into database.

3.2.1 Registration Flowchart

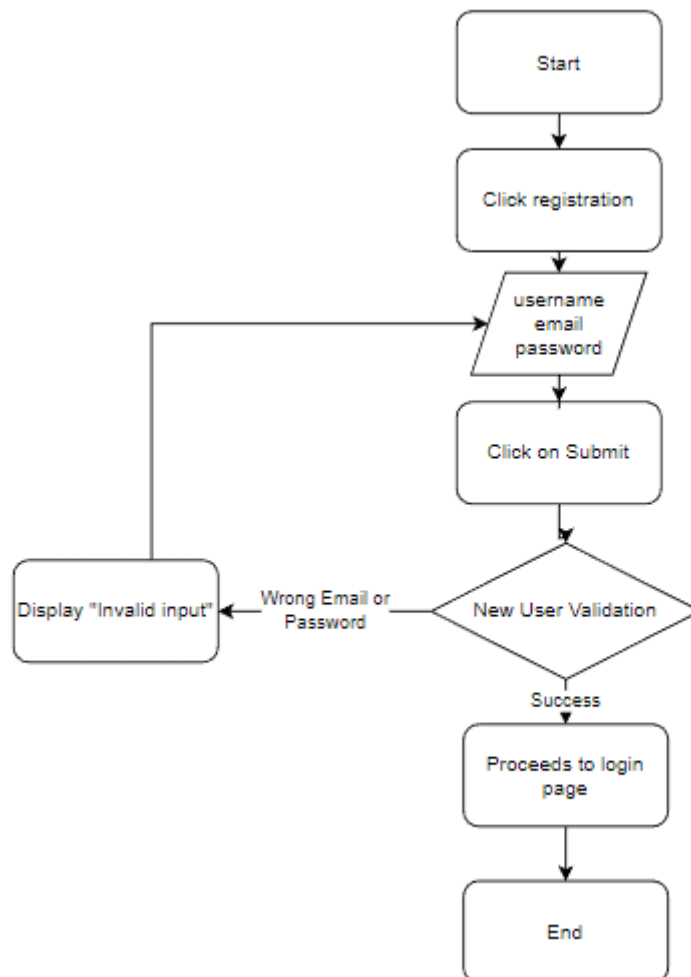


Figure 8 Registration Flowchart

3.3 Movie Management:

3.3.1 List Movies Page









List Movies				
<div>Add New MovieView UsersFavourites</div>				
Movie Id	Posters	Title	Genres	Function
1		Toy Story (1995)	Adventure Animation Children Comedy Fantasy	<div>ViewUpdateDelete</div>
2		Jumanji (1995)	Adventure Children Fantasy	<div>ViewUpdateDelete</div>
3		Grumpier Old Men (1995)	Comedy Romance	<div>ViewUpdateDelete</div>
6		Heat (1995)	Action Crime Thriller	<div>ViewUpdateDelete</div>

Figure 9 List Movies Page (top screen)

10		GoldenEye (1995)	Action Adventure Thriller	<div>ViewUpdateDelete</div>
11		American President, The (1995)	Comedy Drama Romance	<div>ViewUpdateDelete</div>
16		Casino (1995)	Crime Drama	<div>ViewUpdateDelete</div>
17		Sense and Sensibility (1995)	Drama Romance	<div>ViewUpdateDelete</div>
19		Ace Ventura: When Nature Calls (1995)	Comedy	<div>ViewUpdateDelete</div>

1

2

3

4

5

6

7

8

9

10

Figure 10 List Movies Page (bottom screen)

After logging in, the controller is called for all the movies to be listed, and the user click on the view button them to look for more details. Below is the controller code calling the get all movies function from the service which calls the function from the repository.

```
@GetMapping("/")
public List<Movies> getAllmovies() throws Exception {
    return movieservice.getallmovies();
}
```

Figure 11 Controller to get all movies.

The code below is used to extract all the movies from the database

```
public List<Movies> findAll(){
    return jdbcTemplate.query("SELECT * FROM FILTERED_Movies ORDER BY Movieid ASC FETCH NEXT 100 ROWS ONLY",rowMapperfiltered);
}
```

Figure 12 Movie Repository Code to get all movies from Database

3.3.2 Add Movie Page

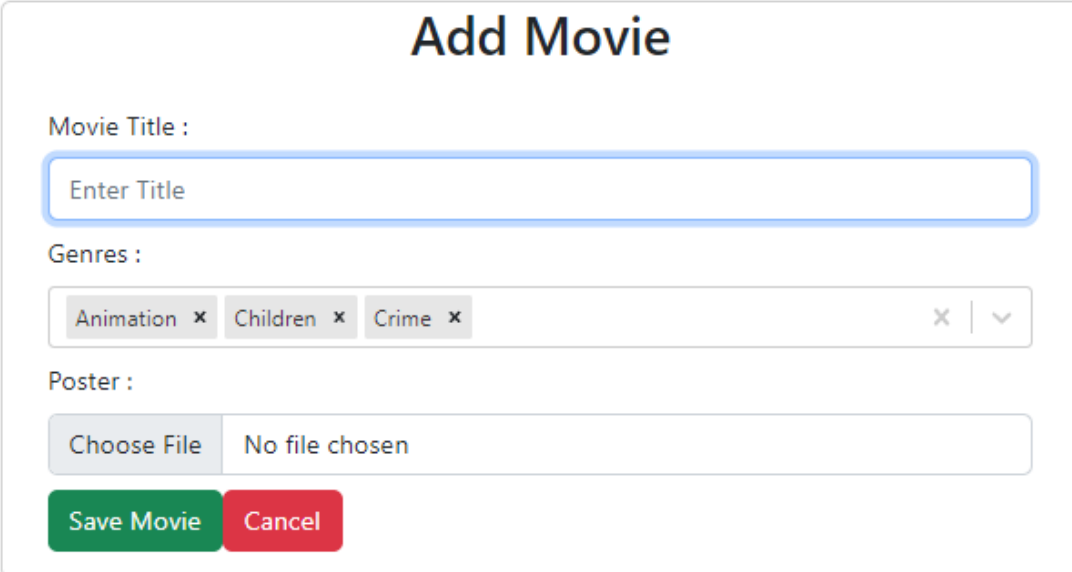


Figure 13 Add Movie Page

If the admin wishes to add movies to the data base, they can simply click on the add new movie button in the list movies page which will take them to the add movie page. This will allow admins to add the title and genres for the movie. Multiple genres can be selected with the multi select component.

The code below is used for the genres selection where multiple can be chosen. Each selection chosen will be concatenated with the next allowing multiple genres to be stored in one column later when submitted.

```

const handleChange=(selectedOption)=>{//concatenate selected genre
  setSelectedoption(Array.isArray(selectedOption)? selectedOption.map(x=>x.value):[])
  for (let i = 0; i < selectedOption.length; i++) {
    result = result.concat(selectedOption[i].value);
    if (i < selectedOption.length - 1) {
      result = result.concat("|");
    }
  }
}
setGenres([result])
console.log( genres);
};

```

Figure 14 Front end code for Genre selection concatenation

The code below is the controller which will pass the movie object received to a movie service function that will pass the data to be inserted to the database.

```

@PostMapping("/newmovie")
public boolean newmovie(@RequestBody Movies movie) throws Exception {
    return movieservice.savemovie(movie);
}

```

Figure 15 Add New Movie Controller

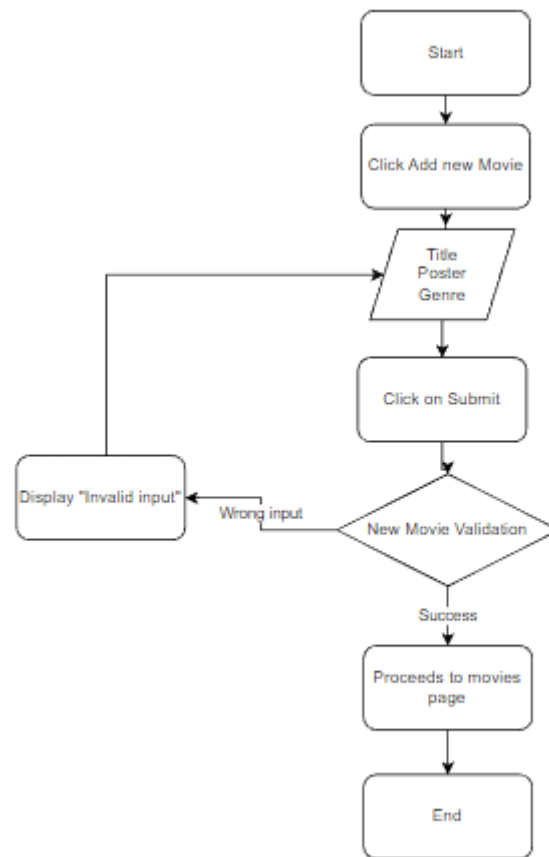
```

private String InsertQuery="INSERT INTO Movies (title,genres) VALUES (?,?)";
public boolean saveMovie(Movies movie){ //function for inserting new movie
    return jdbcTemplate.update(InsertQuery, movie.getTitle(), movie.getGenres()) > 0;
}

```

Figure 16 Movie Repository Insertion Code

3.3.2.1 Add Movie Page Flowchart



3.3.3 Update Movies Page

Should the admin want to modify the movies, they can click on the update button which will let them go to the update page and allow them to change the title along with the genres

Update Movie

Movie Title :

Genres :

Poster :

Choose File

No file chosen

Save Movie

Cancel

Figure 17 Update Movie Page

The update page is very similar to the add page but with the information already filled out. And able to be changed. Pressing the 'Save Movie' button will call the update movie function which will pass the ID of the corresponding movie and a movie object data.

The code below is used to set the new title and genres entered for the selected movie to be updated.

```
private String UpdateQuery="UPDATE  Movies SET title=?,genres=? WHERE movieid=?";
public boolean updateMovie(int id,Movies movie){ //update movie title and genres based on ID given
    return jdbcTemplate.update(UpdateQuery, movie.getTitle(), movie.getGenres(),id) > 0;
}
```

Figure 18 Movie Repository code for updating movie.

3.3.4 Delete Movie Function

Genres	Function
Adventure Animation Children Comedy Fantasy	<div>ViewUpdateDelete</div>

Should the admin want to delete the movie from the database, the admin can press the delete button which will then call the function to remove from the data base with the use of the movie's ID.

```
private String DeleteQuery="DELETE  Movies WHERE movieid=?";
public boolean deleteMovie(int id){//function to movie user data based on movieID entered
    return jdbcTemplate.update(DeleteQuery, id) > 0;
}
```

Figure 19 Movie Repository code for delete movie based on ID given

Once the movie is deleted from the database, the page is refreshed and displays remaining number of movies.

3.3.4.1 Delete Movie Flowchart

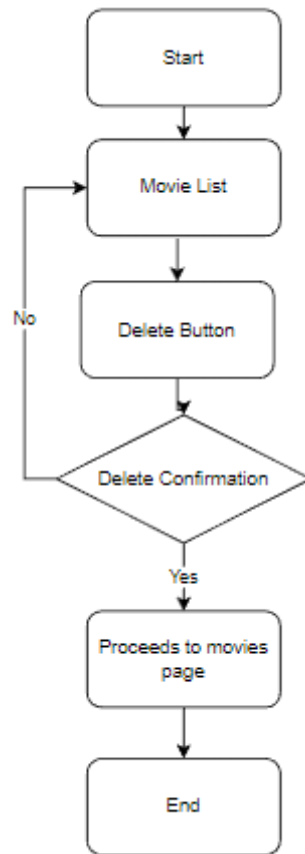



Figure 20 Delete Movie Flowchart

3.3.5 View Movie Page

Movie Info



Movie Title: Toy Story (1995)

Genres: Adventure|Animation|Children|Comedy|Fantasy

Description: Led by Woody, Andy's toys live happily in his room until Andy's birthday brings Buzz Lightyear onto the scene. Afraid of losing his place in Andy's heart, Woody plots against Buzz. But when circumstances separate Buzz and Woody from their owner, the duo eventually learns to put aside their differences.

Rating Score :

Rate MovieAdd to FavouriteBack

Figure 21 View Movie page

By clicking the view button in the list movies page. In this page, Information such as the overview of the film can be seen, and users can also add this to their favorites as well as give it a rating.

This is done by passing the id of the movie in the view button from the previous page which will pass the ID to an API that will retrieve the information regarding that movie. The information will then be set and displayed accordingly.

```
const [postData, setPostData] = useState({}); //setting maximum number of item in page
useEffect(() => { //enable side process to be done
  MovieService.getMoviebyID(id) //call function to pass id to API n retrieve the movie information
    .then((response) => {
      setmovieid(id) //set movie information accordingly based on ID passed
      setTitle(response.data.title)
      setfavTitle(response.data.title)
      setGenres(response.data.genres)
      setfavGenres(response.data.genres)
      getmovies();
    }).catch(error => {
      console.log(error)
    }); setLoading(true);
  const timer = setTimeout(() => {
    getmovies(); //Calling functions to get movies
    setLoading(false); //to disable loading screen after time period
  }, 8000); return () => clearTimeout(timer);
}, []);
const handleChange = event => {
```

Figure 22 Front End View Page code for setting data

```

public Movies findfilteredMoviebyid(int id){
    try {
        return jdbcTemplate.queryForObject( sql: "SELECT * FROM FILTERED_Movies WHERE movieid = ?", new Object[]{id}, rowMapperfiltered);
    } catch (EmptyResultDataAccessException e) {
        return null;
    }
}

```

Figure 23 Movie Repository Movie Data Retrieval

3.3.6 Movie Recommendations Component




By being part of the view movie page, bottom of the page will consist a list of movies that are found to be similar by the system, pressing view on one of them will lead to another view. Page that will display the respective information as well as display other recommendations based on that movie.

Rate Movie

Add to Favourite

Back

Recommendation

Posters	Title	Genres	Function
	Toy Story (1995)	Adventure Animation Children Comedy Fantasy	<div>View</div>
	Jumanji (1995)	Adventure Children Fantasy	<div>View</div>
	Grumpier Old Men (1995)	Comedy Romance	<div>View</div>

1

Figure 24 Movie Recommendation Component

The recommended movies are fetched by using the function below. When viewing a movie, the ID of that movie is automatically passed to fetch movies like it and later populate the recommended movies table.


```

public List<Movies> getsimilarmovies(int id) { //function to get similar movies based on correlation matrix
    try {
        List<String> similarIDs = new ArrayList<>(); //SIMILAR LIST ID
        List<Movies> similarMovies = new ArrayList<>();
        String query = "Select movie_id from pearsons_correlations where ID_" + id + "> 0.5"; //QUERY TO GET SIMILAR IDs
        similarIDs = RunQuery(query, myColumn: "movie_id");
        for (int j = 0; j < similarIDs.size(); j++) {
            Movies movie=findfilteredMoviebyid(j); //find movie based on ID entered
            if (movie == null) {
                continue;
            } else {
                similarMovies.add(movie);
            }
        }
        return similarMovies;
    } catch (EmptyResultDataAccessException e) { //in case if empty
        e.printStackTrace();
        return null;
    }
}

```

Figure 25 Movie Repository Code to retrieve similar movies

The query for getting the movie_ids is getting the ids from the Pearons Correlation matrix where only if similarities of other movies are above 0.5 are selected and added to the list.

3.3.6.1 View Movie and Recommendation Component Flowchart

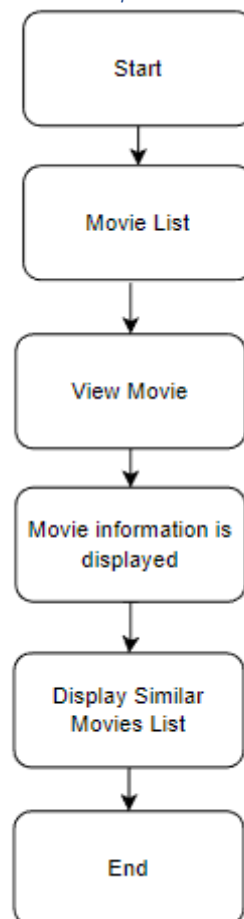


Figure 26 View Movie page and movie recommendation

3.3.7 Rating Movie

By entering a rating and pressing the “Rate Movie” button in the view movie page, users can send their ratings into the database.

```
@PostMapping("/newrating")
public boolean newmovie(@RequestBody Ratings ratings) throws Exception {
    return ratingservice.saverating(ratings);
}
```

Figure 19 Ratings Controller for new movie ratings

When creating a new correlation matrix, the ratings can affect the output of the similar movies.

3.3.6.1 Rating Movie Flowchart

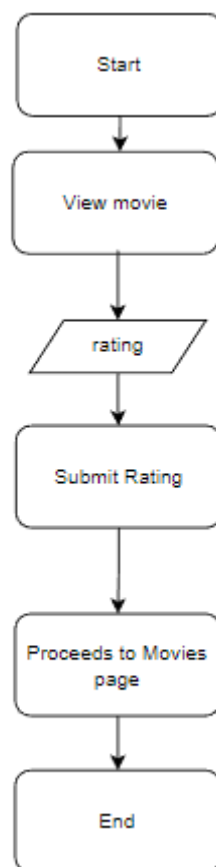


Figure 20 Rating Movie Flowchart

After rating the movie, they will then be brought back to the movies page for them to explore other movies.

3.3.8 Add Favourite Movies Function



Movie Title: Toy Story (1995)

Genres: Adventure|Animation|Children|Comedy|Fantasy

Description: Led by Woody, Andy's toys live happily in his room until Andy's birthday brings Buzz Lightyear onto the scene. Afraid of losing his place in Andy's heart, Woody plots against Buzz. But when circumstances separate Buzz and Woody from their owner, the duo eventually learns to put aside their differences.

Rating Score :

Rate this movie?(1-5)

Rate Movie Add to Favourite Back

Figure 27 Movie information and buttons from view movie page

Users can simply add their favorite movies by viewing a movie and clicking on the “Add to Favorite” button. On the front end , below is the function that will pass a favmovie object to an API that will communicate with the controller to add it to the database.

```
UserFavService.addmovietofav(favmovie) .then((response) => { //call API to pass fav movie data
  if (response.status === 200) { //if success
    console.log(response.data);
    navigate("/movies"); //navigate page
  } else { //else
    console.log(response);
    alert(response.data); // display the response message
  }
}).catch(error=>{ //if movie already added
  console.log(error===400)
  alert("Movie already in favourites") //alerts user that the movie is already in favourites
})
```

Figure 28 Front end function used to call API to pass movie data.

```
@PostMapping("/addmovietofav")
public ResponseEntity<?> addmovietofav(@RequestBody UserFav movie) throws Exception {
    return movieservice.savemovie(movie);
}
```

Figure 29 Mapping to add favourite movie object.

The savemovie() function in the userfavService calls users the Userfavourite repository twice, first is to validate if the use has already added the movie into the database, second when inserting the data.

```

public ResponseEntity<?> savemovie(UserFav UserFavmovie) {
    UserFav movie = repo.findByuseridmovieid(UserFavmovie);
    if (movie == null) {
        repo.saveFavMovie(UserFavmovie);
        return new ResponseEntity<UserFav>(UserFavmovie, HttpStatus.OK);
    } else {
        return new ResponseEntity<>("Movie Already in Favourites", HttpStatus.BAD_REQUEST);
    }
}
};

```

Figure 30 savemovie function to validate if records exists

This is done by checking if the Users's ID and the movieID are present in the same row.

```

private String findQuery = "SELECT * FROM Ronan_User_Fav WHERE movieid=? AND userid=?";
public UserFav findByuseridmovieid (UserFav rating)
{
    try {
        return jdbcTemplate.queryForObject(findQuery, new Object[]{rating.getMovieid(), rating.getUserid()}, rowMapper);
    } catch (EmptyResultDataAccessException e) {
        return null;
    }
}
}

```

Figure 31 Repository function used to validate using userid and movieid

Should that be the case. It Should the movie already be added, will return an error to the API call. An alert will then popup notifying the user regarding it.

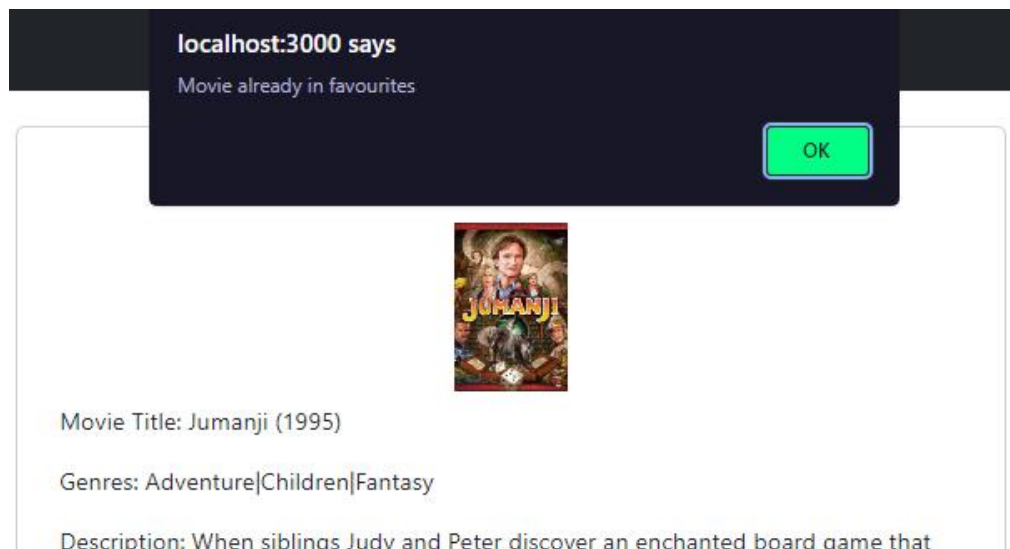


Figure 32 Alert Informing user movie is already in favourites

If not, then the userID and movieID are inserted into the table. After adding movies to their favorites from the view movie page, If Users favorites want to view their favorite movies, they can go to the list movies page and click on the 'Favorites' button that will bring them to a list of movies that have been marked as favorites.

3.3.8.1 Add favourite movies flowchart

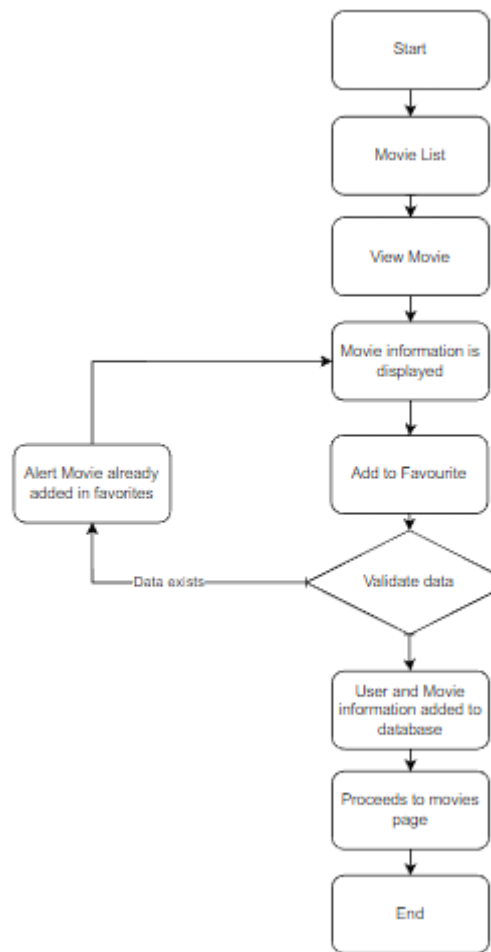


Figure 33 Add movie to favourites flowchart.

3.3.9 List Favourite movies Pages

Favourites




		View Users		View Movies			
Fav Id	Movie Id	Posters	Title	Genres	Function		
153	193620		Shin Ultraman	Sci-Fi Drama Fantasy	View Remove		
154	193626		No Name(2023)	Sci-fi Mystery	View Remove		
157	193583		No Game No Life: Zero (2017)	Animation Comedy Fantasy	View Remove		
156	193609		Andrew Dice Clay: Dice Rules (1991)	Comedy	View Remove		

Figure 34 List Favourite Movie

A list of the users' favorite movies is displayed by taking the user's ID and retrieving all the movies from the Userfav table which stores the IDs of the of the user and movies that the user added.

```
useEffect(() => { //enable side process to be done
  setLoading(true); //to initiate loading screen
  const timer = setTimeout(() => {
    getmovies(userid); //Calling functions to get movies
    setLoading(false); //to disable loading screen after time period
  }, 8000); return () => clearTimeout(timer);
}, []);
const getmovies=(userid)=> {
  UserFavService.getMoviesByUserID(userid).then((response)=> { //call API to get all movies form favourites list
    setMovie(response.data) //set movie to retrieved movies
    console.log(response.data);
  }).catch(error=>{
    console.log(error);
  })
}
```

Figure 35 Front End function to call API to get user's favourite movies

```
public List<UserFav> findAllfromuser(int id){
  return jdbcTemplate.query( sql: "SELECT * FROM Ronan_User_Fav WHERE userid="+id+" ",rowMapper);
}
```

Figure 36 Back-end favourite movie extraction

The view button works just like mentioned before will bring the user to the respective view movie page. The code for the remove button works like the delete button but only removes the movie from favorites. The movie will still exist in the movie database.

```
private String DeleteQuery="DELETE Ronan_User_Fav WHERE favid=?";
public boolean removeUserFavMovie(int id) { return jdbcTemplate.update(DeleteQuery, id) > 0; }
```

3.3.10 Remove Favourite movie function.

Much like the previous delete functions, this will retrieve id from the movie and will call a function to call an API along with a confirmation alert .

```
const handleDelete = (movieid) => { //function to call alert confirmation
  if (window.confirm("Are you sure you want to remove this from favourites?")) {
    {deletemovie(movieid)} //call function to remove movie based on ID passed
  }else{
  }
};
const deletemovie=(favid)=> { //call api to delete movie
  UserFavService.removeMoviefromFav(favid).then((response)=> {
    alert(response.data)
    getmovies(); //use get movies function
  }).catch(error=>{
    console.log(error);
  })
}
```

Figure 37 Front end remove function to remove movie from database.

3.3.10.1 Remove Favourite movie flowchart.

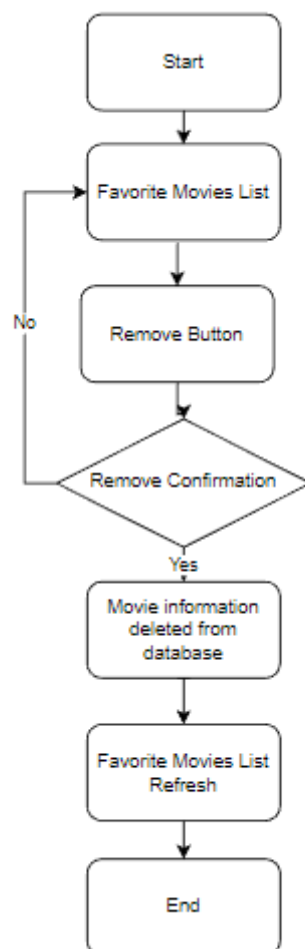


Figure 38 Remove favourite movie flowchart

3.4 User management

3.4.1 List Users Page

List Users						
<div>Add New UserView MoviesFavourites</div>						
User Id	Email	UserName	User Type	Function		
19	user@gmail.com	user	Admin	Update	Delete	Promote
26	555	555	Admin	Update	Delete	Promote
1	ronan@gmail.com	ronan	Admin	Update	Delete	Promote
2	qwe@gmail.com	qwe	Admin	Update	Delete	Promote
3	bob@gmail.com	bob	Admin	Update	Delete	Promote
4	test@gmail.com	test	Admin	Update	Delete	Promote
36	913	913	User	Update	Delete	Promote
41	kent@gmail.com	kent	User	Update	Delete	Promote
34	qwer	qwer	Admin	Update	Delete	Promote
35	ha	ha	User	Update	Delete	Promote

1

Figure 39 List Users Page

By pressing on the view users' button in the movie screen, the list of users will be displayed on the screen with their respective username, userid, email and user type.

All this data is retrieved in the front end of making and API call to the UserController get all the users' data.


```

useEffect(() => { //enable side process to be done
  setLoading(true); //to initiate loading screen
  const timer = setTimeout(() =>{
    getUsers(); //Calling functions to get users
    setLoading(false); //to disable loading screen after time period
  }, 8000); return () => clearTimeout(timer);
}, []);
const getUsers=()=> {
  UserService.getAllUsers().then((response)=> { //calls an API to get all users
    setUser(response.data) //set user objects to be based on return data
    console.log(response.data);
  }).catch(error=>{
    console.log(error);
  })
}

```

Figure 40 Function to get users and setting the objects to populate the list

Figure 41 Front End Function to calling API to get User information.

```

@GetMapping("/")
public List<User> getAllUsers() throws Exception {
    return userService.getAllUsers();
}

```

Figure 42 UserController mapping to get all user information

3.4.2 Add User Page

Add User

Email :

Username :

Password :

Confirm Password :

Save UserCancel

Figure 43 Add User Page

This page works the same as the registration allowing admins to add new users themselves should the situation such as the need to testing arises.

3.4.2.1 Add User Flowchart

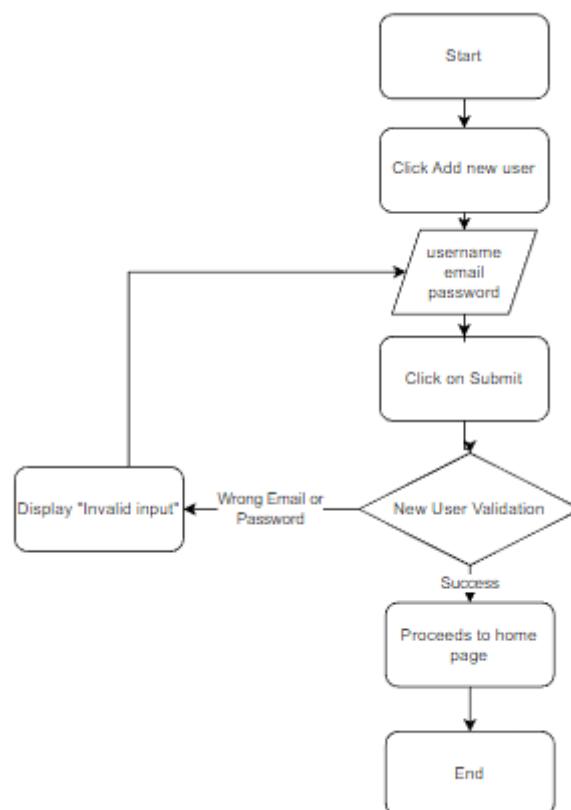
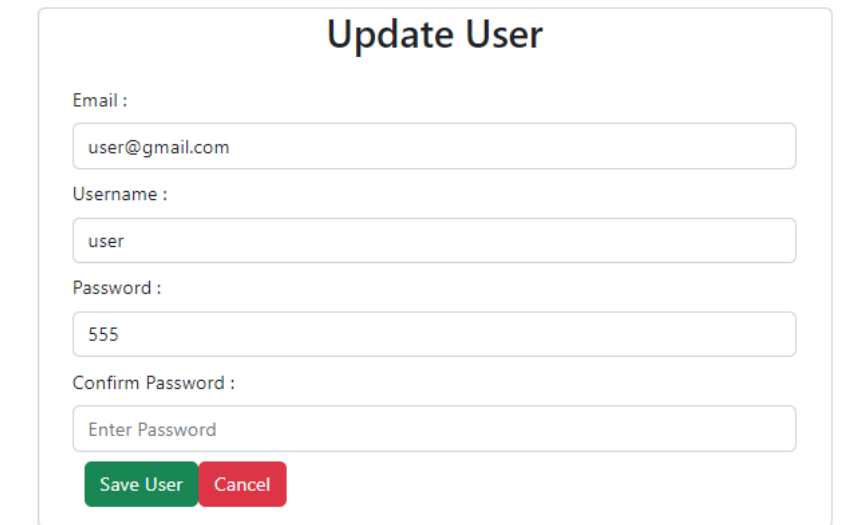


Figure 44 Add new user flowchart.

3.4.3 Update User Page



Update User

Email :
user@gmail.com

Username :
user

Password :
555

Confirm Password :
Enter Password

Save User **Cancel**

Figure 45 Update user Page

By clicking on the update button, it will bring the admin to the update user page where they can edit their information such as their email, username, and password. Like the registration system, the information filled will also be validated using the same method.

Below is the code for updating the user in the database based on the ID given.

```
private String UpdateQuery="UPDATE Ronan_User SET email=?,username=?,userpassword=? WHERE userid=?";  
public boolean updateUser(int id,User user){//function to update user based on new user object and id entered  
    return jdbcTemplate.update(UpdateQuery, user.getEmail(),user.getUsername(), user.getUserpassword(),id) > 0;  
}
```

3.4.3.1 Update User Flowchart

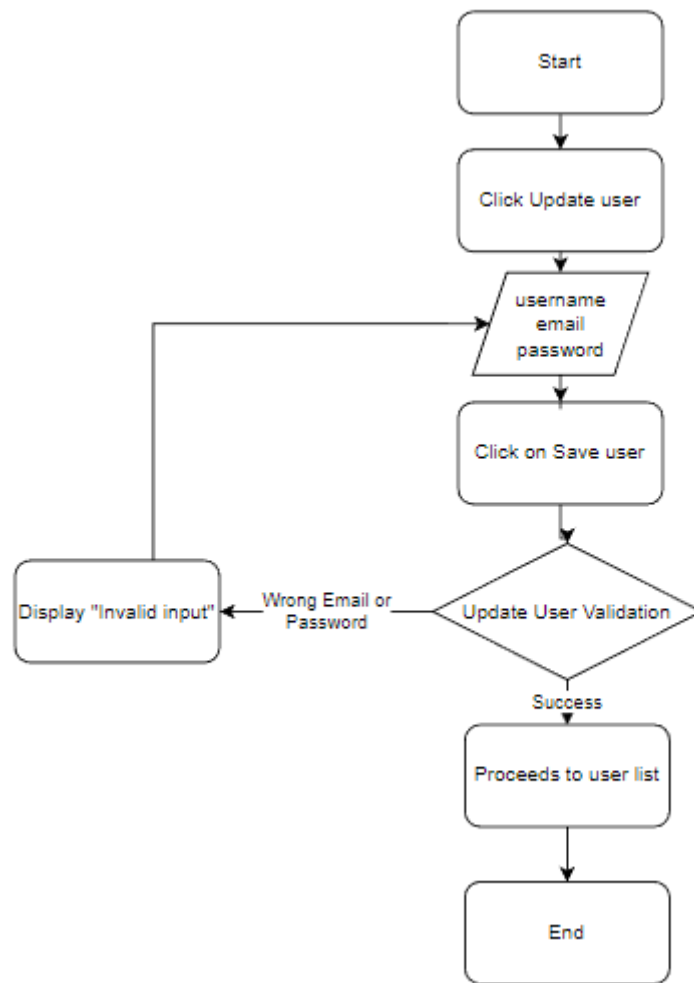


Figure 46 Update User Flowchart

3.4.4 Delete User Function

This function is done by utilizing the ID of the user can passing it through the delete button which will call a function. Prior to deleting the user data, an alert will first be displayed to confirm if the user wants to be deleted. Should the answer be yes, then the deleteUser function is called taking the userid to pass to an API.

```

const handleDelete = (userid) => { //function to call alert confirmation
  if (window.confirm("Are you sure you want to delete this movie?")) {
    {deleteUser(userid)} //call function to delete user based on ID passed
  }else{

  }
};
const deleteUser=(userid)=> { //takes userir and pass it to an API to delete user with the id
  UserService.deleteuser(userid).then((response)=> {
    getusers(); //call function to get users
  }).catch(error=>{
    console.log(error);
  })
}
}

```

Figure 47 Front end alert and delete user function

3.4.4 .1 Delete User Flowchart

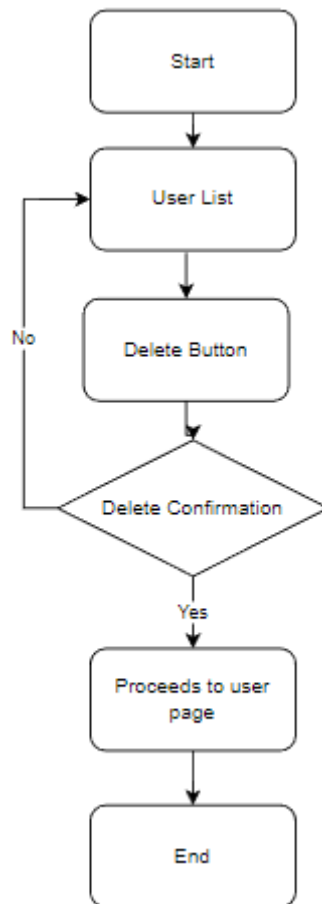


Figure 48 Delete User Flow Chart

3.4.6 Promote User Function

A simple function that will retrieve the ID of the user to the promote button is used to easily call a function to promote the user to an admin. Mainly used for testing purposes.

```
const PromoteUser=(userid)=> {  
  //takes userid to be passed to an api which will change usertype to Admin  
  UserService.promoteuser(userid).then((response)=> {  
    getUsers(); //call function to get users  
  }).catch(error=>{  
    console.log(error);  
  })  
}
```

Figure 49 Front End function to pass ID to API to promote user

3.4.6.1 Promote User Flowchart

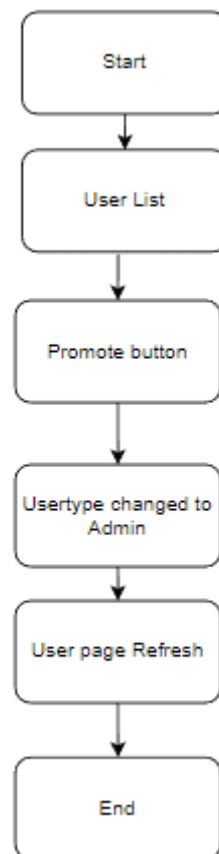


Figure 50 Promote User Flowchart

4.0 Entity Relationship Diagram (ERD)

The diagram below is the ERD that reflects the relationship of all the tables in movie recommendation database.

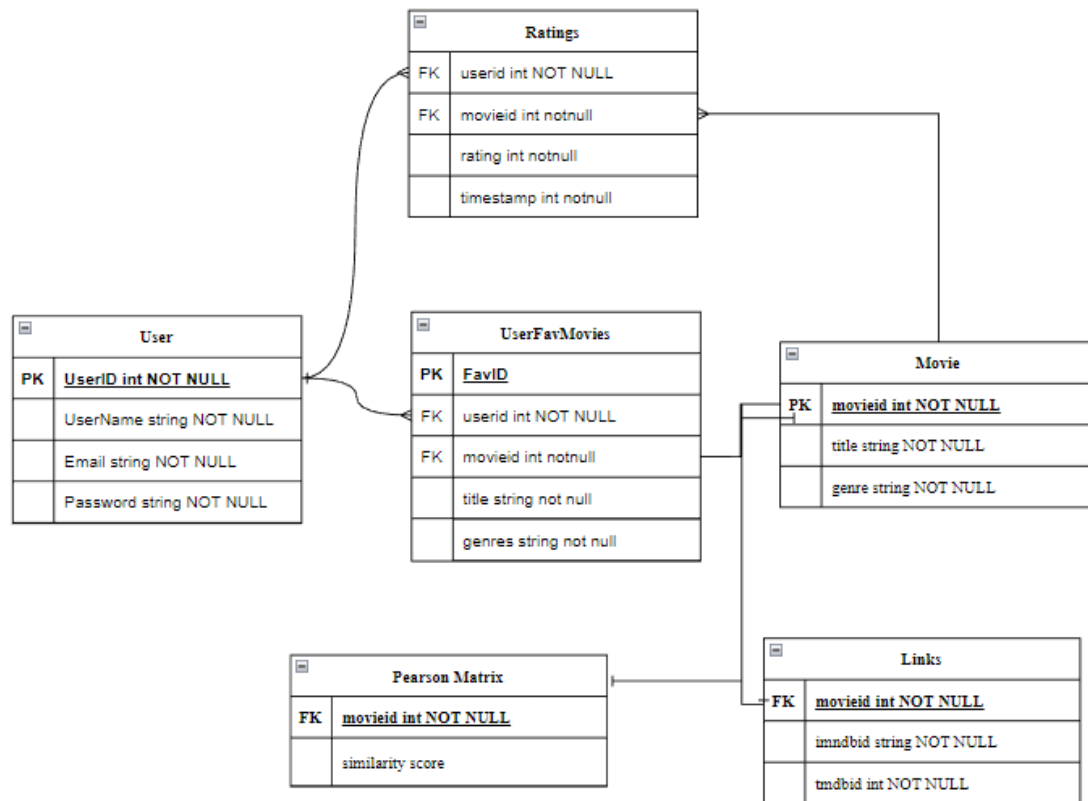


Figure 51 ERD of Movie Recommendation system

5.0 Pearson's correlation

Pearson correlation is a statistical technique used to assess the linear relationship between two continuous variables. In movie recommendation systems, it can be utilized to compare the ratings of various films. The objective is to find movies that have similar ratings based on the evaluations from users. This correlation is calculated by comparing the ratings of each movie from all users. The result is a value ranging from -1 to 1 that reflects the strength and direction of the relationship between the ratings of the two movies. If the Pearson correlation is positive, it means the two movies have similar ratings, while a negative correlation implies dissimilar ratings. A Pearson correlation close to 1 indicates high similarity between the movies, which can be recommended to users who liked one of the movies. On the other hand, a Pearson correlation close to 0 implies low similarity and the movies may not be recommended to the same users. The Pearson correlation is valuable in movie recommendation systems, allowing the system to make recommendations based on movie similarities rather than just their popularity.

To construct a movie similarity matrix using Pearson correlation, follow these steps:

1. Obtain movie ratings from multiple users for a collection of films. Store this information in a matrix format where each row signifies a movie, and each column signifies a user. This data can either be obtained by conducting surveys or by extracting information from websites such as IMDb or Rotten Tomatoes.
2. Build a ratings matrix where each row signifies a user, and each column signifies a movie. The data in the cells represents the ratings of each movie given by a particular user.
3. Determine the Pearson correlation between each pair of movies by comparing the ratings of each user. The outcome is a matrix where each cell holds the Pearson correlation between two films. The formula to compute Pearson's correlation is:

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

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

4. The calculation results are then in the Database table which can be used to make movie recommendations by finding movies that are like a user's favourite movies.

