

# **INFO 6250 - Web Development Tools & Methods – Final Report**

**Project Title:**  
MoviesCorner – Find your next favorite

**Author:**  
Bhavya Patel  
(001351292)

## **PROJECT SUMMARY**

MoviesCorner is an open movie community buff that allows users to read about their favorite movies, to add ratings about movies, to add movie into watchlist, to let review on movies. Also User can follow other users so that other user can see about each other watchlist, ratings, reviews, follow list. There are two kind of users such as User and Critic. Difference between User and Critic is that Critic has additional right such as he can add reviews to movies.

Movie details is fetched from the open source API of themoviedb.org. Popular movies, Now playing movies, movie basic details such as poster path, backdrop path, movie title, overview, runtime, genre, release date, cast details etc.

REST APIs for user watchlist, ratings, reviews, follow list are made in **Spring MVC** Rest Controller, **Hibernate** ORM, **MySQL** database. Each of the call to the end point returns JSON response along with HTTP status code.

Front-end is developed using **React** and **Redux**. Each endpoint of Spring MVC rest APIs are called using AJAX asynchronous call to display the JSON data. To make any changes in back-end, the respected APIs is called.

## FUNCTIONALITY

### **1. User Register**

- Allows user to register using username, password, first name, last name and two kind of user role such as User and critics
- Returns JWT token after successful registration
- Form validator is used to validates the fields.

### **2. User Login**

- Returns JWT token after successful login
- Form validator is used to validates the fields.

### **3. Get User details**

- JWT token which is returned after successful login and registration is passed as a header to this end-point.
- Returns user details after validating JWT token

### **4. Search Users**

- JWT token which is returned after successful login and registration is passed as a header to this end-point.
- Returns user details after validating JWT token

### **5. Bcrypt password store**

- Password is stored using bcrypt encryption in database so that other unauthorized person can no see the password.

### **6. JSON Web Tokens(JWT) token generation:**

- To allow user to access routes in secure manner JSON Web Token is generated for each user and is sent as a x-auth-token header in request calling.
- Filter is used to validate tokens and redirect the reponse accordingly to the controllers and user.

### **7. User follow, unfollow, following list, follower list:**

- JWT token is passed as a header while performing follow, unfollow in order to authorized user to access such functionality.
- Following list and follower list is return for particular user in term of requesting user point of view whether that user is following other user or not.

## **8. Watchlist**

- Allows user add movie to watchlist and remove movie from watchlist
- User can see other users watchlist

## **9. Ratings**

- Users can give ratings to movie out of 10.
- Average ratings for each movie is calculated at a time of requesting movie details.
- User can also update, delete ratings.
- Other user can see the ratings for the user.

## **10. Reviews**

- It lets Critic users to give reviews on movies.
- User can delete reviews and other User can view the ratings for movie.

## **11. Error controller**

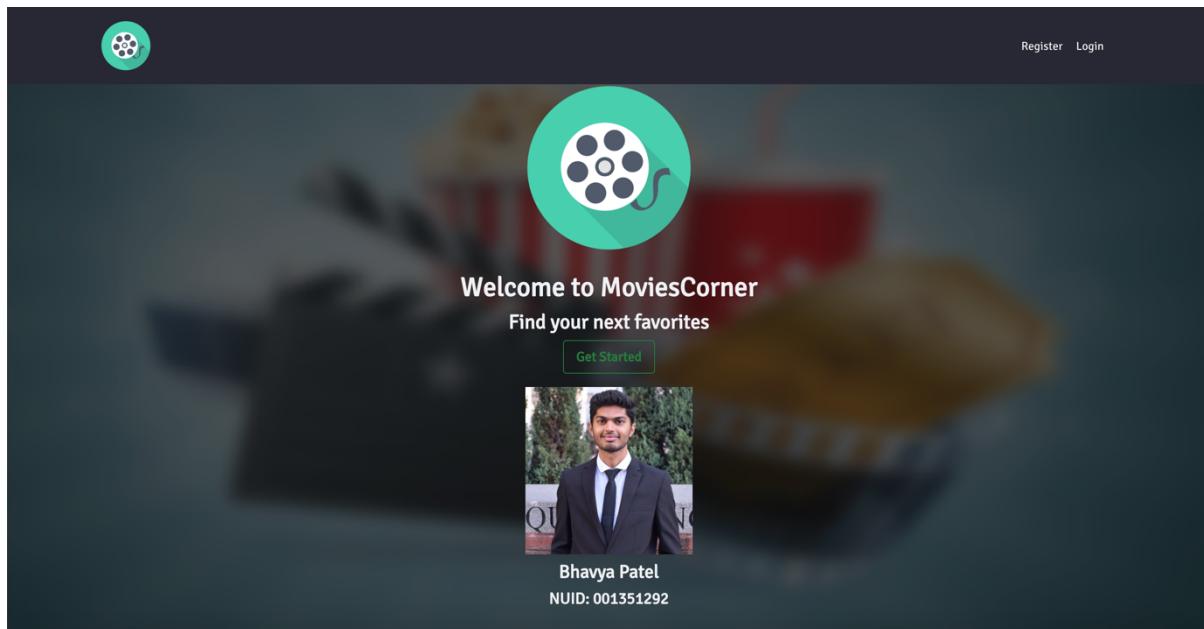
- In case of page not found or Method not allowed, JSON response is returned along with HTTP status code 404 and 405.

## **12. Token not valid**

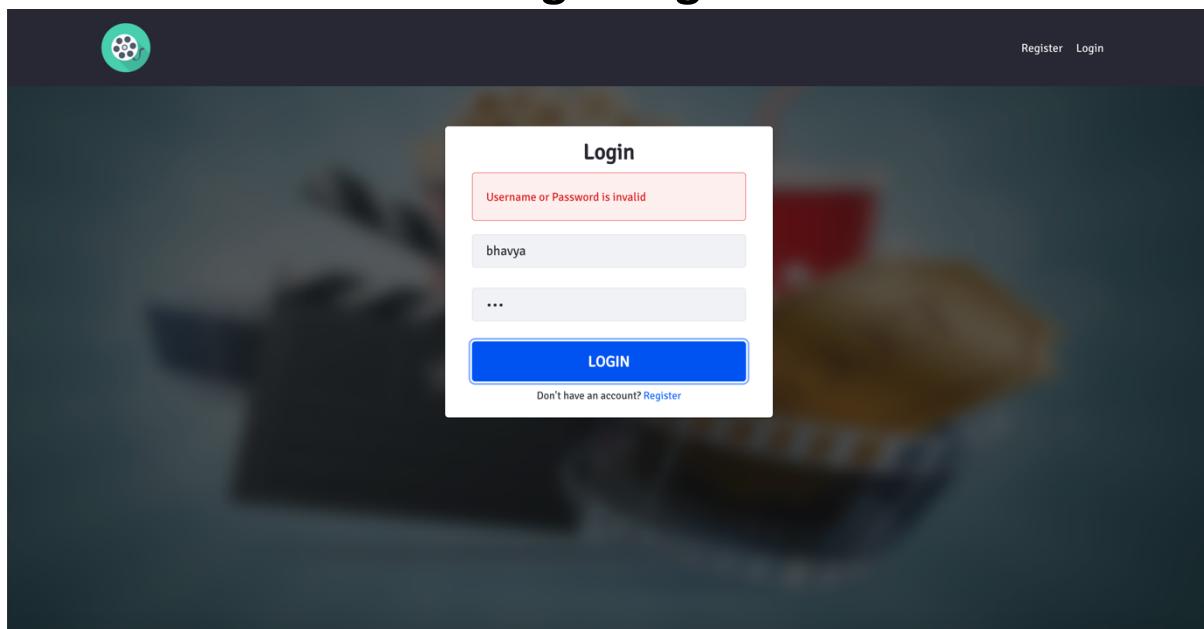
- If there is some unauthorized access to APIs then “Token is not valid” response is returned along HTTP status code 400 and prevents user to perform such action.

## SCREENSHOTS

### Home Screen



### Login Page



## Register Page with User already exists

The screenshot shows a registration form titled "Register". The "username" field contains "bhavaya". The "password" field contains three dots (...). The "firstName" field contains "Bhavya". The "lastName" field contains "Patel". A dropdown menu below the fields shows "User". At the bottom is a blue "REGISTER" button. Above the "REGISTER" button, a small note says "Already have an account? [Sign in](#)". In the top right corner of the page header, there are links for "Register" and "Login".

User Already Exists

bhavaya

...

...

Bhavya

Patel

User

REGISTER

Already have an account? [Sign in](#)

Register Login

## Register Page with validating fields

The screenshot shows a registration form titled "Register". All four input fields ("username", "password", "firstName", and "lastName") are highlighted in red with the error message "Required". Below the fields are four empty input fields: "Username", "Password", "Confirm Password", and "First Name". In the top right corner of the page header, there are links for "Register" and "Login".

userNmae Required

password Required

firstName Required

lastName Required

Username

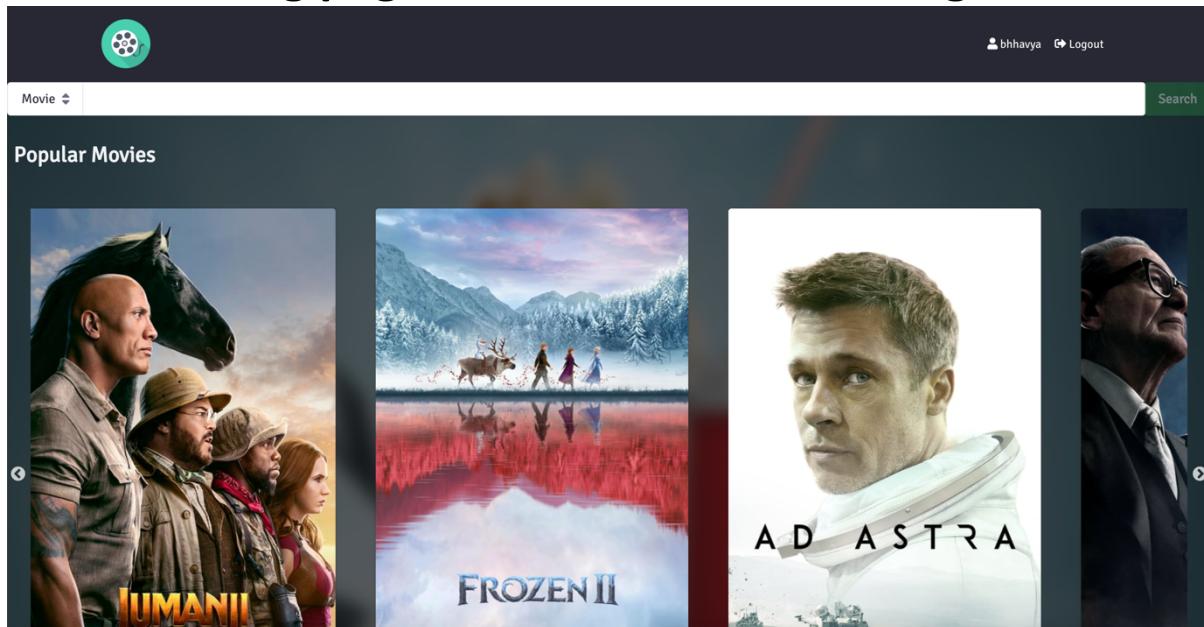
Password

Confirm Password

First Name

Register Login

## Landing page with user detail in navigation



## Movie search result

A screenshot of a movie search results page. The header is identical to the landing page, featuring the circular logo, the search bar with 'Movie' and a dropdown arrow, and the user 'bhavya'. The main content area displays search results for 'spiderman':

- Spider-Man**  
After being bitten by a genetically altered spider, nerdy high school student Peter Parker is endowed with amazing powers.
- Spider-Man: Homecoming**  
Following the events of Captain America: Civil War, Peter Parker, with the help of his mentor Tony Stark, tries to balance his life as an ordinary high school student in Queens, New York City, with fighting crime as his superhero alter ego Spider-Man as a new threat, the Vulture, emerges.
- Spider-Man: Far from Home**  
Peter Parker and his friends go on a summer trip to Europe. However, they will hardly be able to rest - Peter will have to agree to help Nick Fury uncover the mystery of creatures that cause natural disasters and destruction throughout the continent.
- Spider-Man 3**  
The seemingly invincible Spider-Man goes up against an all-new crop of villains—including the shape-shifting Sandman. While Spider-Man's superpowers are altered by an alien organism, his alter ego, Peter Parker, deals with nemesis Eddie

## Movie page without adding to watchlist

This screenshot shows the movie page for "Spider-Man: Far from Home". At the top right, there are user icons for 'bhavya' and a 'Logout' button. The main content area features the movie's poster on the left, showing Spider-Man and his friends in a cityscape. To the right of the poster, the movie title "Spider-Man: Far from Home" is displayed along with its runtime (2 hrs 9 mins), genres (Action, Adventure, Science Fiction), and release date (28 June, 2019). A rating of 9.3/10 is shown with a star icon. Below the title, a brief plot summary states: "Peter Parker and his friends go on a summer trip to Europe. However, they will hardly be able to rest - Peter will have to agree to help Nick Fury uncover the mystery of creatures that cause natural disasters and destruction throughout the continent." To the right of the plot summary is a "Watch Trailer" section with a thumbnail image of a trailer frame and a "TEASER TRAILER" button. At the bottom left, there is a green "ADD TO WATCHLIST" button. Below the poster, there is a "Cast" section with four small profile pictures.

## Movie page with remove from watchlist

This screenshot shows the same movie page for "Spider-Man: Far from Home" as the previous one, but with a red "ADDED TO WATCHLIST" button at the bottom instead of the green "ADD TO WATCHLIST" button. The rest of the page content is identical to the first screenshot, including the movie poster, title, plot summary, trailer section, and cast list.

## User with User right can not give review on movie

This screenshot shows a movie page for "Spider-Man: Far From Home". At the top, there is a red button with a checkmark and the text "ADDED TO WATCHLIST". Below it, the word "Cast" is displayed. Four cast members are shown in separate boxes: Tom Holland (Peter Parker / Spider-Man), Jake Gyllenhaal (Quentin Beck / Mysterio), Zendaya (Michelle "MJ" Jones), and Jon Favreau (Harold "Happy" Hogan). Each box contains a portrait photo and the actor's name along with their character's name.

**Reviews**  
You need to be a certified Critic

## Movie reviews

This screenshot shows the same movie page as above, but with a different user role. A purple circular icon with the letter "C1" is visible on the left. In the reviews section, there is a text input field with the placeholder "Add reviews here". A green button labeled "Add Review" with a video camera icon is located to the right of the input field. Below this, a review from a user named "Critic 1" is displayed, reading: "Peter parker is back in action...this time it is far from home" and "a few seconds ago".

## Review list for user

The screenshot shows a dark-themed web application interface. On the left, there is a purple circular profile picture with the letters 'C1' in white. To its right, the user's name 'Critic 1' and handle 'critic1' are displayed, followed by the title 'Critic'. At the top, a navigation bar includes links for 'WatchList', 'Ratings', **Reviews**, 'Followings', and 'Followers'. Below the navigation, three movie reviews are listed in a grid:

Movie Title	Description	Action
Spider-Man: Far from Home	Peter parker is back in action...this time it is far from home	Trash bin icon
Frozen II	It seems good	Trash bin icon
Joker	Movie worth to watch	Trash bin icon

Each review row includes a small movie poster thumbnail and a timestamp '12 Dec 2019'.

## Review list for user after removing review

This screenshot shows the same application interface after the review for 'Frozen II' has been deleted. The purple circular profile picture with 'C1' is still on the left, and the user information 'Critic 1', 'critic1', and 'Critic' is present. The navigation bar at the top remains the same. The review grid now only contains two items:

Movie Title	Description	Action
Spider-Man: Far from Home	Peter parker is back in action...this time it is far from home	Trash bin icon
Frozen II	It seems good	Trash bin icon

The timestamp '12 Dec 2019' is visible below the second review row.

## Watchlist list for user

The screenshot shows a user profile for 'Bhavya Patel' (bhavya) with a red circular profile picture containing the letters 'BP'. The main content area is titled 'WatchList' and contains five movie entries:

WatchList	Ratings	Followings	Followers
	Jumanji: The Next Level		
	Joker		
	Spider-Man: Far from Home		
	The Irishman		
	Once Upon a Time... in Hollywood		

## Ratings list for user

The screenshot shows a user profile for 'Bhavya Patel' (bhavya) with a red circular profile picture containing the letters 'BP'. The main content area is titled 'Ratings' and contains six movie entries with their respective ratings:

	Ratings
	9/10
	10/10
	9/10
	7/10
	8/10
	9/10

## User followers list for user

A screenshot of a web application interface showing the 'Followers' section for a user profile. The profile picture is a red circle with 'BP' in white. The profile name is 'Bhavya Patel' and the handle is 'bhhavya'. The title 'User' is below the handle. The top navigation bar includes 'WatchList', 'Ratings', 'Followings', and 'Followers'. The 'Followers' tab is active. The follower list contains four entries:

Profile Picture	Handle	Name	Action
Critic 3	critic3	Critic 3 Critic	<button>Unfollow</button>
Critic 2	critic2	Critic 2 Critic	<button>Follow</button>
Critic 1	critic1	Critic 1 Critic	<button>Unfollow</button>
User 1	user1	User 1 User	<button>Unfollow</button>

The URL 'localhost:3000/profile/13' is visible at the bottom left.

## User followings list for user

A screenshot of a web application interface showing the 'Followings' section for a user profile. The profile picture is a pink circle with 'C3' in white. The profile name is 'Critic 3' and the handle is 'critic3'. The title 'Critic' is below the handle. The top navigation bar includes 'WatchList', 'Ratings', 'Reviews', 'Followings', and 'Followers'. The 'Followings' tab is active. The following list contains one entry:

Profile Picture	Handle	Name	Action
BP	bhhavya	Bhavya Patel User	<button>Follow</button>

## User after unfollowing user

C3

Critic 3  
critic3  
Critic

Follow

WatchList Ratings Reviews Followings Followers

Jumanji: The Next Level

Joker

Frozen II

## Followings list after unfollowing user

BP

Bhavya Patel  
bhhavya  
User

WatchList Ratings Followings Followers

C1 critic1 Unfollow

U1 user1 Unfollow

## Search users

A screenshot of a web application interface titled "Search users". The search bar at the top contains the text "Search results for 'cri'...". Below the search bar, there is a list of five user profiles:

- CU**: Critic User Critic. Follow button.
- critic**: Critic. Follow button.
- C1**: Critic 1 Critic. Follow button.
- C2**: Critic 2 Critic. Follow button.
- C3**: Critic 3 Critic. Follow button.
- C4**: Critic 4 Critic. Follow button.

The URL "localhost:3000/profile/21" is visible at the bottom left of the screenshot.

## Unauthorized access response

```
{  
  "error": [  
    {  
      "message": "Token is not valid"  
    }  
  ]  
}
```

## **CONTROLLERS**

### **1. Auth Controller**

```
package com.me.controller;

import com.me.config.JwtTokenUtil;
import com.me.dao.AuthDAO;
import com.me.exception.UserException;
import com.me.pojo.User;
import com.me.response.Errors;
import com.me.response.JwtResponse;
import com.me.response.Message;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

/**
 *
 * @author bhaVYa
 */
@RestController
@RequestMapping("/api/auth")
public class AuthController {

    @Autowired
    @Qualifier("authDao")
    AuthDAO authDao;

    @Autowired
    @Qualifier("jwtTokenUtil")
```

```
private JwtTokenUtil jwtTokenUtil;

// User Login and return token
// access: private
@RequestMapping(value = "/authenticate", method =
RequestMethod.POST, consumes="application/json")
public ResponseEntity<Object>
createAuthenticationToken(@RequestBody User u) throws Exception {
    try {
        User user = authDao.authenticate(u.getUserName(),
u.getPassword());
        final String token = jwtTokenUtil.generateToken(user);
        return new ResponseEntity<>(new JwtResponse(token),
HttpStatus.OK);
    } catch (UserException e) {
        List<Message> errors = new ArrayList<>();
        errors.add(new Message(e.getMessage())));
        return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
    }
}

// Getting user details from token
// access: private
@RequestMapping(value = "/getUserDetails", method =
RequestMethod.GET)
public ResponseEntity<Object> getUserDetails(HttpServletRequest
request) throws Exception {
    try {
        User user = (User) request.getAttribute("user");
        return new ResponseEntity<>(user, HttpStatus.OK);
    } catch (Exception e) {
        List<Message> errors = new ArrayList<>();
        errors.add(new Message(e.getMessage())));
        return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
    }
}
```

## 2. User Controller

```
package com.me.controller;

import com.me.config.JwtTokenUtil;
import com.me.dao.UserDAO;
import com.me.exception.UserException;
import com.me.pojo.User;
import com.me.response.Errors;
import com.me.response.JwtResponse;
import com.me.response.Message;
import com.me.validator.UserValidator;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Set;
import javax.servlet.http.HttpServletRequest;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import org.springframework.web.bind.annotation.InitBinder;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestHeader;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

/**
 *
 * @author bhaVYa
 */
@RestController
@RequestMapping("/api")
```

```
public class UserController {  
  
    @Autowired  
    @Qualifier("userDao")  
    UserDAO userDao;  
  
    @Autowired  
    @Qualifier("userValidator")  
    UserValidator validator;  
  
    @Autowired  
    @Qualifier("jwtTokenUtil")  
    private JwtTokenUtil jwtTokenUtil;  
  
    @InitBinder("user")  
    private void initBinder(WebDataBinder binder) {  
        binder.setValidator(validator);  
    }  
  
    // User registration  
    @RequestMapping(value = "/auth/register",  
        method = RequestMethod.POST,  
        produces = "application/json")  
    public ResponseEntity<Object> register(@RequestBody User user,  
    BindingResult result) throws Exception {  
        // Validting fields in form  
        validator.validate(user, result);  
        if (result.hasErrors()) {  
            List<Message> errors = new ArrayList<>();  
            result.getFieldErrors().stream()  
                .forEach(action -> errors.add(new  
                    Message(action.getDefaultMessage())));  
            return new ResponseEntity<>(new Errors(errors),  
            HttpStatus.BAD_REQUEST);  
        }  
        try {  
            User u = userDao.register(user);  
  
            // Generating token from User object  
            final String token = jwtTokenUtil.generateToken(u);  
        }  
    }  
}
```

```
        return new ResponseEntity<>(new JwtResponse(token),
HttpStatus.OK);
    } catch (UserException e) {
        List<Message> errors = new ArrayList<>();
        errors.add(new Message(e.getMessage())));
        return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
    }
}

// Get user details along with following details in term of requesting
user
@RequestMapping(value = "/users/{userId}",
method = RequestMethod.GET,
produces = "application/json")
public ResponseEntity<Object> getUser(@PathVariable String userId,
@RequestHeader HttpHeaders headers) throws Exception {
    try {
        String requestingUser = "";
        if (headers.get("x-auth-token") == null) {
            requestingUser += "guest";
        } else {
            String token = headers.get("x-auth-token").get(0);
            requestingUser +=
jwtTokenUtil.getUserIdFromToken(token.substring(7));
        }
    }

    User user = userDao.getUser(userId);
    JSONObject userJson = new JSONObject();
    userJson.put("userId", user.getUserId());
    userJson.put("userName", user.getUserName());
    userJson.put("firstName", user.getFirstName());
    userJson.put("lastName", user.getLastName());
    userJson.put("userRole", user.getUserRole());

    if (!requestingUser.equals("guest")) {
        User reqUser = userDao.getUser(requestingUser);
        if (reqUser.getFollowings().contains(user)) {
            userJson.put("isFollowing", true);
        } else {
    
```

```
        userJson.put("isFollowing", false);
    }
} else {
    userJson.put("isFollowing", false);
}
return new ResponseEntity<>(userJson.toMap(), HttpStatus.OK);
} catch (Exception e) {
    List<Message> errors = new ArrayList<>();
    errors.add(new Message(e.getMessage()));
    return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
}

}

// Search Users
@RequestMapping(value = "/users/search/{userName}",
method = RequestMethod.GET,
produces = "application/json")
public ResponseEntity<Object> searchUsers(@PathVariable String
userName) throws Exception {
try {
    List<User> users = userDao.search(userName);
    return new ResponseEntity<>(users, HttpStatus.OK);
} catch(UserException e) {
    List<Message> errors = new ArrayList<>();
    errors.add(new Message(e.getMessage()));
    return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
}
}

// Follow users
// Access: private
@RequestMapping(value = "/users/follow/{followingId}",
method = RequestMethod.POST,
produces = "application/json")
public ResponseEntity<Object> follow(@PathVariable String
followingId, HttpServletRequest request) throws Exception {
try {
```

```
User user = (User) request.getAttribute("user");
userDao.follow(user, followingId);

    return new ResponseEntity<>(new Message("Followed
Successfully"), HttpStatus.OK);
} catch (UserException e) {
    List<Message> errors = new ArrayList<>();
    errors.add(new Message(e.getMessage())));
    return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
}

// Unfollow users
// Access: private
@RequestMapping(value = "/users/unfollow/{unfollowingId}",
method = RequestMethod.POST,
produces = "application/json")
public ResponseEntity<Object> unfollow(@PathVariable String
unfollowingId, HttpServletRequest request) throws Exception {
try {
    User user = (User) request.getAttribute("user");

    userDao.unfollow(user, unfollowingId);

    return new ResponseEntity<>(new Message("Unfollowed
Successfully"), HttpStatus.OK);
} catch (UserException e) {
    System.out.println("UNFOLLOW1"+e.getMessage());
    List<Message> errors = new ArrayList<>();
    errors.add(new Message(e.getMessage())));
    return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
}
}

// Get followings list
@RequestMapping(value = "/users/followings/{id}",
method = RequestMethod.GET,
produces = "application/json")
```

```
public ResponseEntity<Object> getFollowings(@PathVariable String id,
@RequestParam HttpHeaders headers) throws Exception {
    try {
        String requestingUser = "";
        if (headers.get("x-auth-token") == null) {
            requestingUser += "guest";
        } else {
            String token = headers.get("x-auth-token").get(0);
            requestingUser +=
                jwtTokenUtil.getUserIdFromToken(token.substring(7));
        }
    }

    Set<User> followings = userDao.getFollowings(id);

    User reqUser = null;
    if (!requestingUser.equals("guest")) {
        reqUser = userDao.getUser(requestingUser);
    }

    List<Map<String, Object>> list = new ArrayList<>();
    for(User u: followings) {
        JSONObject userJson = new JSONObject();
        userJson.put("userId", u.getUserId());
        userJson.put("userName", u.getUserName());
        userJson.put("firstName", u.getFirstName());
        userJson.put("lastName", u.getLastName());
        userJson.put("userRole", u.getUserRole());

        if(!requestingUser.equals("guest")) {
            if(reqUser.getFollowings().contains(u))
                userJson.put("isFollowing", true);
            else
                userJson.put("isFollowing", false);
        } else {
            userJson.put("isFollowing", false);
        }

        list.add(userJson.toMap());
    }
}
```

```
        return new ResponseEntity<>(list, HttpStatus.OK);
    } catch (Exception e) {
        List<Message> errors = new ArrayList<>();
        errors.add(new Message(e.getMessage())));
        return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
    }
}

// Get followers list
@RequestMapping(value = "/users/followers/{id}",
method = RequestMethod.GET,
produces = "application/json")
public ResponseEntity<Object> getFollowers(@PathVariable String id,
@RequestHeader HttpHeaders headers) throws Exception {
try {
    String requestingUser = "";
    if (headers.get("x-auth-token") == null) {
        requestingUser += "guest";
    } else {
        String token = headers.get("x-auth-token").get(0);
        requestingUser +=
jwtTokenUtil.getUserIdFromToken(token.substring(7));
    }
}

Set<User> followers = userDao.getFollowers(id);

User reqUser = null;
if (!requestingUser.equals("guest")) {
    reqUser = userDao.getUser(requestingUser);
}

List<Map<String, Object>> list = new ArrayList<>();
for(User u: followers) {
    JSONObject userJson = new JSONObject();
    userJson.put("userId", u.getUserId());
    userJson.put("userName", u.getUserName());
    userJson.put("firstName", u.getFirstName());
```

```
userJson.put("lastName", u.getLastName());
userJson.put("userRole", u.getUserRole());

if(!requestingUser.equals("guest")) {
    if(reqUser.getFollowings().contains(u))
        userJson.put("isFollowing", true);
    else
        userJson.put("isFollowing", false);
} else {
    userJson.put("isFollowing", false);
}

list.add(userJson.toMap());
}

return new ResponseEntity<>(list, HttpStatus.OK);
} catch (Exception e) {
    List<Message> errors = new ArrayList<>();
    errors.add(new Message(e.getMessage()));
    return new ResponseEntity<>(new Errors(errors),
    HttpStatus.BAD_REQUEST);
}
}
```

### 3. Movie Controller

```
package com.me.controller;

import com.me.dao.MovieDAO;
import com.me.exception.MovieException;
import com.me.pojo.Movie;
import com.me.pojo.Reviews;
import com.me.pojo.User;
import com.me.pojo UserRole;
import com.me.response.Errors;
import com.me.response.Message;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import javax.servlet.http.HttpServletRequest;
import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

/**
 *
 * @author bhaVYa
 */
@RestController
@RequestMapping("/api")
public class MovieController {

    @Autowired
    @Qualifier("movieDao")
    MovieDAO movieDao;
```

```
@RequestMapping(value = "/movies/watchlist/add",
    method = RequestMethod.POST,
    produces = "application/json")
public ResponseEntity<Object> addToWatchList(@RequestBody
Movie movie, HttpServletRequest request) throws Exception {
    User user = null;
    try {
        user = (User) request.getAttribute("user");
        movieDao.addToWatchList(user, movie);
        return new ResponseEntity<>(new Message("Movie added to
watchlist"), HttpStatus.OK);
    } catch (MovieException e) {
        List<Message> errors = new ArrayList<>();
        errors.add(new Message(e.getMessage()));
        return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
    }
}

@RequestMapping(value = "/movies/watchlist/remove",
    method = RequestMethod.POST,
    produces = "application/json")
public ResponseEntity<Object> removeFromWatchList(@RequestBody
Movie movie, HttpServletRequest request) throws Exception {
    User user = null;
    try {
        user = (User) request.getAttribute("user");
        movieDao.removeFromWatchList(user, movie);
        return new ResponseEntity<>(new Message("Movie removed
from watchlist"), HttpStatus.OK);
    } catch (MovieException e) {
        List<Message> errors = new ArrayList<>();
        errors.add(new Message(e.getMessage()));
        return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
    }
}

@RequestMapping(value = "/movies/watchlist",
    method = RequestMethod.POST,
```

```
    produces = "application/json")
public ResponseEntity<Object> isWatchList(@RequestBody Movie
movie, HttpServletRequest request) throws Exception {
    User user = null;
    try {
        user = (User) request.getAttribute("user");
        boolean isWatchList;
        if (user == null) {
            isWatchList = false;
        } else {
            isWatchList = movieDao.isWatchList(movie, user);
        }
        return new ResponseEntity<>(new
Message(String.valueOf(isWatchList)), HttpStatus.OK);
    } catch (Exception e) {
        List<Message> errors = new ArrayList<>();
        errors.add(new Message(e.getMessage())));
        return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
    }
}

@RequestMapping(value = "/movies/watchlist/{userId}",
method = RequestMethod.GET,
produces = "application/json")
public ResponseEntity<Object> getWatchList(@PathVariable String
userId) {
    try {
        Set<Movie> watchlist = movieDao.getWatchList(userId);
        return new ResponseEntity<>(watchlist, HttpStatus.OK);
    } catch (MovieException e) {
        List<Message> errors = new ArrayList<>();
        errors.add(new Message(e.getMessage())));
        return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
    }
}

@RequestMapping(value = "/movies/ratings/add/{rating}",
method = RequestMethod.POST,
```

```
    produces = "application/json")
    public ResponseEntity<Object> addRatings(@PathVariable String
rating, @RequestBody Movie movie, HttpServletRequest request) {
        try {
            double r = Double.parseDouble(rating);
            if (r < 0 || r > 10) {
                throw new MovieException("Rating should be in range of 0 to
10");
            }
            User user = (User) request.getAttribute("user");
            movieDao.addRatings(movie, user, r);
            return new ResponseEntity<>(new Message("Ratings added"),
HttpStatus.OK);
        } catch (MovieException e) {
            List<Message> errors = new ArrayList<>();
            errors.add(new Message(e.getMessage()));
            return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
        } catch (NumberFormatException e) {
            List<Message> errors = new ArrayList<>();
            errors.add(new Message("Invalid ratings"));
            return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
        }
    }

    @RequestMapping(value = "/movies/ratings/update/{rating}",
method = RequestMethod.POST,
    produces = "application/json")
    public ResponseEntity<Object> updateRatings(@PathVariable String
rating, @RequestBody Movie movie, HttpServletRequest request) {
        try {
            double r = Double.parseDouble(rating);
            if (r < 0 || r > 10) {
                throw new MovieException("Rating should be in range of 0 to
10");
            }
            User user = (User) request.getAttribute("user");
            movieDao.updateRatings(movie, user, r);
        }
    }
}
```

```
        return new ResponseEntity<>(new Message("Ratings updated"),  
HttpStatus.OK);  
    } catch (MovieException e) {  
        List<Message> errors = new ArrayList<>();  
        errors.add(new Message(e.getMessage()));  
        return new ResponseEntity<>(new Errors(errors),  
HttpStatus.BAD_REQUEST);  
    } catch (NumberFormatException e) {  
        List<Message> errors = new ArrayList<>();  
        errors.add(new Message("Invalid ratings"));  
        return new ResponseEntity<>(new Errors(errors),  
HttpStatus.BAD_REQUEST);  
    }  
}  
  
@RequestMapping(value = "/movies/ratings/delete",  
method = RequestMethod.POST,  
produces = "application/json")  
public ResponseEntity<Object> deleteRatings(@RequestBody Movie  
movie, HttpServletRequest request) {  
    try {  
        User user = (User) request.getAttribute("user");  
        movieDao.deleteRatings(movie, user);  
        return new ResponseEntity<>(new Message("Ratings deleted"),  
HttpStatus.OK);  
    } catch (MovieException e) {  
        List<Message> errors = new ArrayList<>();  
        errors.add(new Message(e.getMessage()));  
        return new ResponseEntity<>(new Errors(errors),  
HttpStatus.BAD_REQUEST);  
    }  
}  
  
@RequestMapping(value = "/movies/ratings/get/{movield}",  
method = RequestMethod.GET,  
produces = "application/json")  
public ResponseEntity<Object> getAvgRatings(@PathVariable String  
movield, HttpServletRequest request) {  
    try {  
        double avgRatings = movieDao.getAvgRatings(movield);  
    }
```

```
        return new ResponseEntity<>(new
Message(String.valueOf(avgRatings)), HttpStatus.OK);
    } catch (MovieException e) {
        List<Message> errors = new ArrayList<>();
        errors.add(new Message(e.getMessage())));
        return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
    }
}

@RequestMapping(value = "/movies/ratings/users/get/{movield}",
method = RequestMethod.GET,
produces = "application/json")
public ResponseEntity<Object>
getUserRatingsForMovie(@PathVariable String movield,
HttpServletRequest request) {
    try {
        User user = (User) request.getAttribute("user");
        JSONObject ratings =
movieDao.getUserRatingsForMovie(movield, user);

        return new ResponseEntity<>(ratings.toMap(), HttpStatus.OK);
    } catch (MovieException e) {
        List<Message> errors = new ArrayList<>();
        errors.add(new Message(e.getMessage())));
        return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
    }
}

@RequestMapping(value = "/movies/ratings/users/{userId}",
method = RequestMethod.GET,
produces = "application/json")
public ResponseEntity<Object> getUserRatings(@PathVariable String
userId, HttpServletRequest request) {
    try {
//        User user = (User) request.getAttribute("user");
        List<Object> ratings = movieDao.getUserRatings(userId);

        return new ResponseEntity<>(ratings, HttpStatus.OK);
    }
}
```

```
        } catch (MovieException e) {
            List<Message> errors = new ArrayList<>();
            errors.add(new Message(e.getMessage())));
            return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
        }
    }

// Add Review for Movie
@RequestMapping(value = "/movies/reviews/add",
    method = RequestMethod.POST,
    produces = "application/json")
public ResponseEntity<Object> addReview(@RequestBody String
body, HttpServletRequest request) {
    try {
        User user = (User) request.getAttribute("user");

        if (user.getUserRole() == UserRole.User) {
            throw new MovieException("User is unauthorized");
        }
        JSONObject json = new JSONObject(body);
        if (json.getString("review").equals("")) {
            throw new MovieException("Review is empty text");
        }
        Movie m = new Movie();
        m.setMovield(json.getInt("movield"));
        m.setMovieName(json.getString("movieName"));
        m.setMovieImage(json.getString("movieImage"));
        Reviews r = movieDao.addReview(user, m,
json.getString("review"));

        return new ResponseEntity<>(r, HttpStatus.OK);
    } catch (MovieException e) {
        List<Message> errors = new ArrayList<>();
        errors.add(new Message(e.getMessage())));
        return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
    }
}
```

```
// Update Review for Movie
@RequestMapping(value = "/movies/reviews/update",
    method = RequestMethod.POST,
    produces = "application/json")
public ResponseEntity<Object> updateReview(@RequestBody String
body, HttpServletRequest request) {
    try {
        User user = (User) request.getAttribute("user");
        if (user.getUserRole() == UserRole.User) {
            throw new MovieException("User is unauthorized");
        }
        JSONObject json = new JSONObject(body);
        if (json.getString("review").equals("")) {
            throw new MovieException("Review is empty text");
        }

        movieDao.updateReview(user, json.getInt("reviewId"),
        json.getString("review"));

        return new ResponseEntity<>(new Message("Review updated"),
        HttpStatus.OK);
    } catch (MovieException e) {
        List<Message> errors = new ArrayList<>();
        errors.add(new Message(e.getMessage()));
        return new ResponseEntity<>(new Errors(errors),
        HttpStatus.BAD_REQUEST);
    }
}

// Update Review for Movie
@RequestMapping(value = "/movies/reviews/delete",
    method = RequestMethod.POST,
    produces = "application/json")
public ResponseEntity<Object> deleteReview(@RequestBody String
body, HttpServletRequest request) {
    try {
        User user = (User) request.getAttribute("user");
        if (user.getUserRole() == UserRole.User) {
            throw new MovieException("User is unauthorized");
        }
    }
}
```

```
JSONObject json = new JSONObject(body);

movieDao.deleteReview(user, json.getInt("reviewId"));

return new ResponseEntity<>(new Message("Review deleted"),
HttpStatus.OK);

} catch (MovieException e) {
List<Message> errors = new ArrayList<>();
errors.add(new Message(e.getMessage())));
return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
}

// Get Review for Movie
@RequestMapping(value = "/movies/reviews/{movield}",
method = RequestMethod.GET,
produces = "application/json")
public ResponseEntity<Object> getReviewsForMovie(@PathVariable
String movield, HttpServletRequest request) {
try {

List<Reviews> list = movieDao.getReviewsForMovie(movield);

return new ResponseEntity<>(list, HttpStatus.OK);
} catch (MovieException e) {
List<Message> errors = new ArrayList<>();
errors.add(new Message(e.getMessage())));
return new ResponseEntity<>(new Errors(errors),
HttpStatus.BAD_REQUEST);
}
}

// Get Review for User
@RequestMapping(value = "/movies/reviews/users/{userId}",
method = RequestMethod.GET,
produces = "application/json")
public ResponseEntity<Object> getReviewsForUser(@PathVariable
String userId, HttpServletRequest request) {
```

```
try {  
  
    List<Reviews> list = movieDao.getReviewsForUser(userId);  
  
    return new ResponseEntity<>(list, HttpStatus.OK);  
} catch (MovieException e) {  
    List<Message> errors = new ArrayList<>();  
    errors.add(new Message(e.getMessage()));  
    return new ResponseEntity<>(new Errors(errors),  
    HttpStatus.BAD_REQUEST);  
}  
}  
}
```

## 4. Error Controller

```
package com.me.controller;

import javax.servlet.http.HttpServletRequest;
import org.json.JSONObject;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

/**
 *
 * @author bhaVYa
 */
@RestController
@RequestMapping("/api")
public class ErrorController {
    @RequestMapping(value = "/error/404",
            method = RequestMethod.GET,
            produces = "application/json")
    public ResponseEntity<Object> error404(HttpServletRequest request)
            throws Exception {
        JSONObject json = new JSONObject();
        json.put("error", "Page not Found");
        return new ResponseEntity<>(json.toMap(),
                HttpStatus.NOT_FOUND);
    }

    @RequestMapping(value = "/error/405",
            method = RequestMethod.GET,
            produces = "application/json")
    public ResponseEntity<Object> error405(HttpServletRequest request)
            throws Exception {
        JSONObject json = new JSONObject();
        json.put("error", "Method not allowed");
        return new ResponseEntity<>(json.toMap(),
                HttpStatus.NOT_FOUND);
    }
}
```

## INFO 6250 - Web Development Tools & Methods – Final Report

```
}
```