Movie Ratings

Student: Cristian Casian-Cristi

Team: Sandor Serban Vlad

Tandea Darius Sorin

Cristian Casian-Cristi

Table of contents

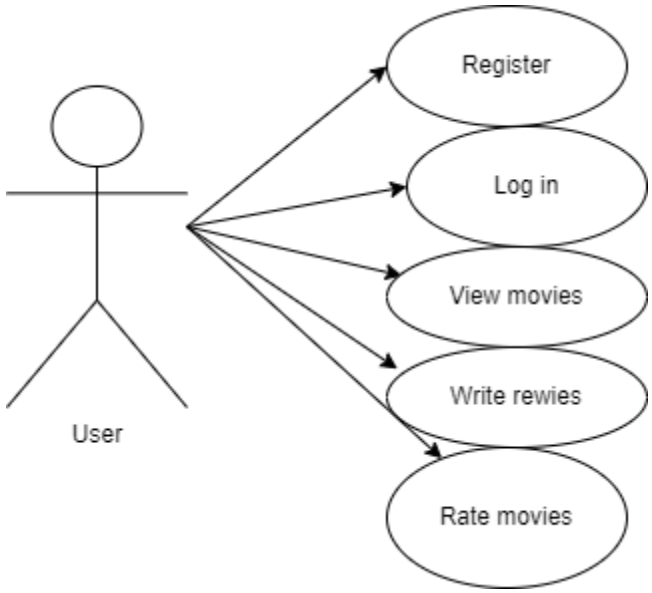
1. Abstract……3
2. Design……..3
   2.1 Use case diagram……..3
   2.2 Class diagram……4
   2.3 Deployment diagram…….5
3. Implementation…….5
4. Appendix: Mini project……10
5. References……14

1. Abstract

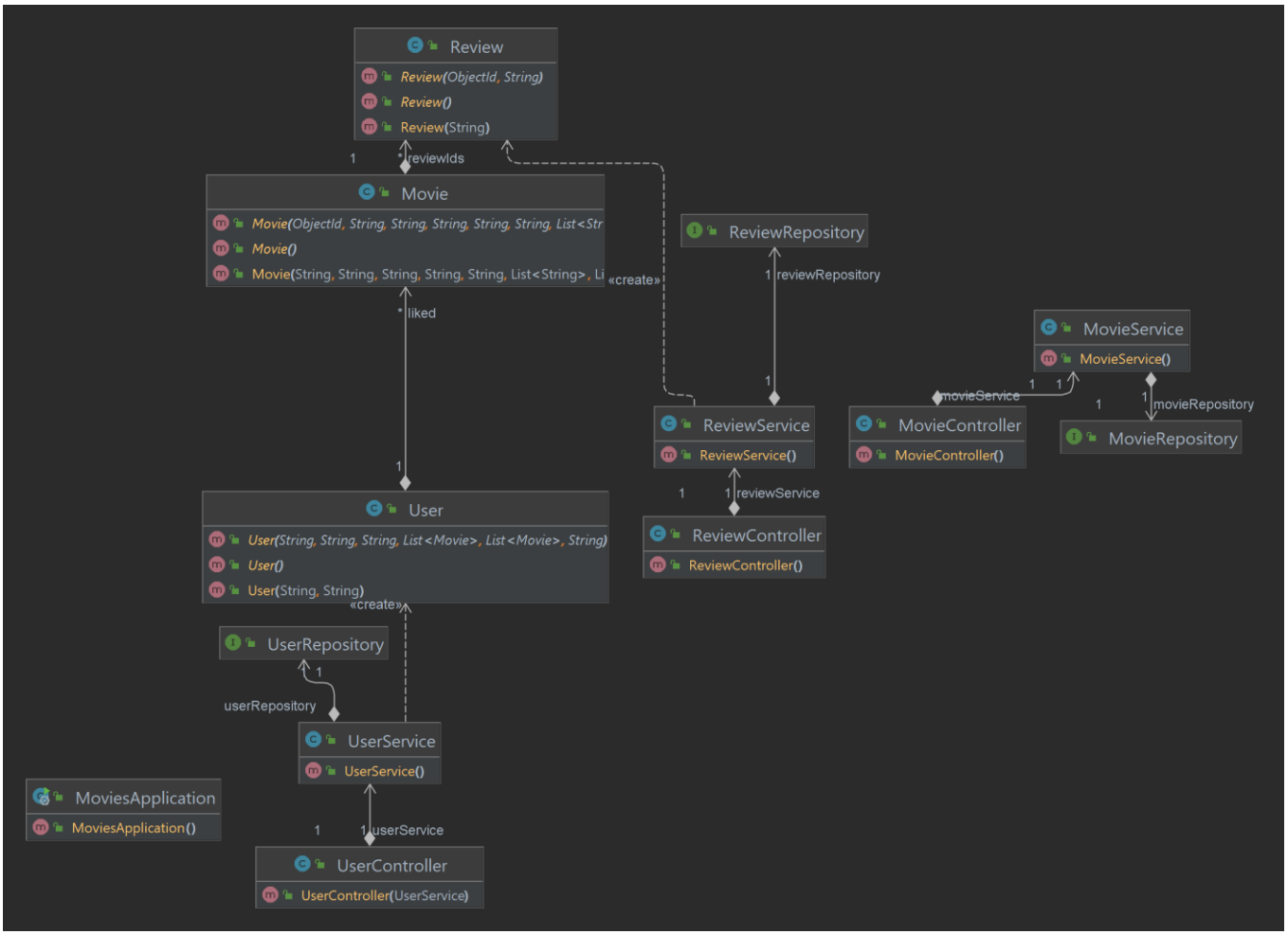
This project is a movie rating application, for the backend we used Java and Spring Boot framework and for the frontend we used React. The application has a list of movies and also recommended movies. The technologies that we used in our project are security, content-based filtering and collaborative filtering. At security part we create register and log in, content-based filtering uses item features to recommend other items similar to what the user likes, based on their previous actions or explicit feedback. Collaborative filtering is a technique that can filter out items that a user might like on the bases of reactions by similar users.

2. Design

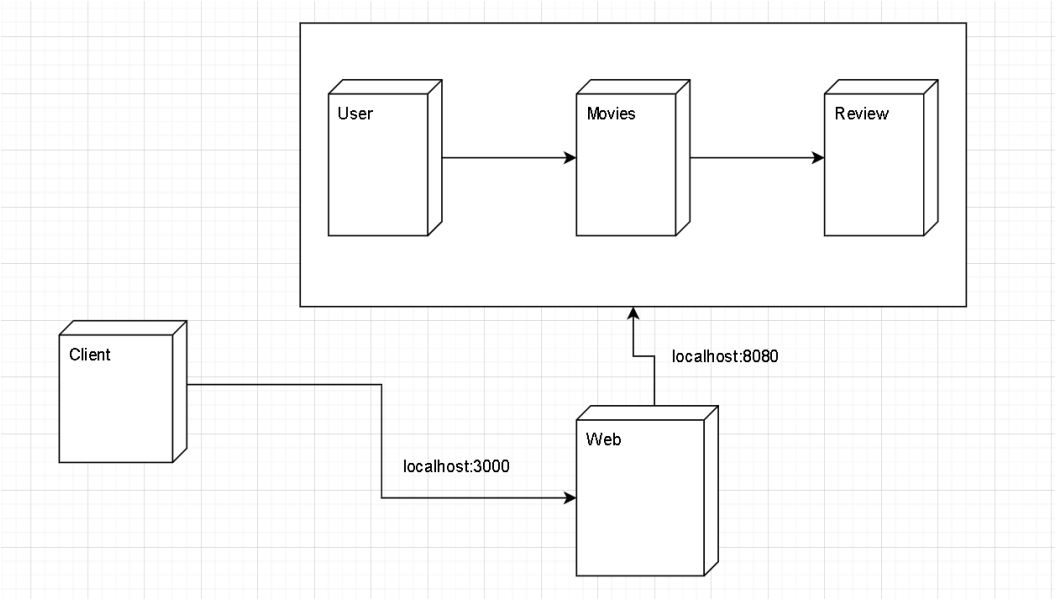
2.1 Use case diagram



2.2 Class diagram



2.3 Deployment Diagram



3. Implementation

In this project, my primary contributions were on implementing the security part, making the log in and register. I create a table in database named “user” where I put the first column is the username, then password and finally the role. I connect the database to java application and then displayed it to the frontend. Also I wrote a PRISM code to describe each state of the application and generate a diagram in order to see how the application works.

***User class**

```java
@Document(collection = "user")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User {
    no usages
    private String id;
    1 usage
    private String username;
    1 usage
    private String password;
    no usages
    @DocumentReference
    private List<Movie> liked;
    no usages
    @DocumentReference
    private List<Movie> disliked;
    1 usage
    private String role;

    1 usage
    public User(String username, String password) {
        this.username = username;
        this.password = password;
        this.role="user";
    }

}
```

***UserRepository interface**

```java
1 usage
@Repository
public interface UserRepository extends MongoRepository<User, String>{
    1 usage
    Optional<User> findUserByUsername(String username);
    1 usage
    Optional<User> findUserByUsernameAndPassword(String username, String password);

}
```

***UserController class**

```java
@RestController
@RequestMapping("/api/v1/users")
@CrossOrigin
public class UserController {
    3 usages
    @Autowired
    private UserService userService;

    no usages
    public UserController(UserService userService) { this.userService = userService; }

    no usages
    @PostMapping("/signup")
    public HttpStatus createUser(@RequestBody String username, String password) {
        if(userService.signUp(username, password)){
            return(HttpStatus.OK);
        }else{
            return(HttpStatus.INTERNAL_SERVER_ERROR);
        }

    }

    no usages
    @PostMapping("/login")
    public HttpStatus loginUser(@RequestBody String username, String password) {
        if(userService.logIn(username, password).equals(null)){
            return(HttpStatus.INTERNAL_SERVER_ERROR);
        }else{
            return(HttpStatus.OK);
        }
```

*UserService class

```java
@Service
public class UserService {
    3 usages
    @Autowired
    private UserRepository userRepository;

    1 usage
    public boolean signUp(String username, String password) {
        if (userRepository.findUserByUsername(username).isPresent()) {
            return false;
        }
        User user = new User(username, password);
        userRepository.save(user);
        return true;
    }

    1 usage
    public User logIn(String username, String password) {
        return userRepository.findUserByUsernameAndPassword(username, password).orElse( other: null);
    }

}
```

*Login.js

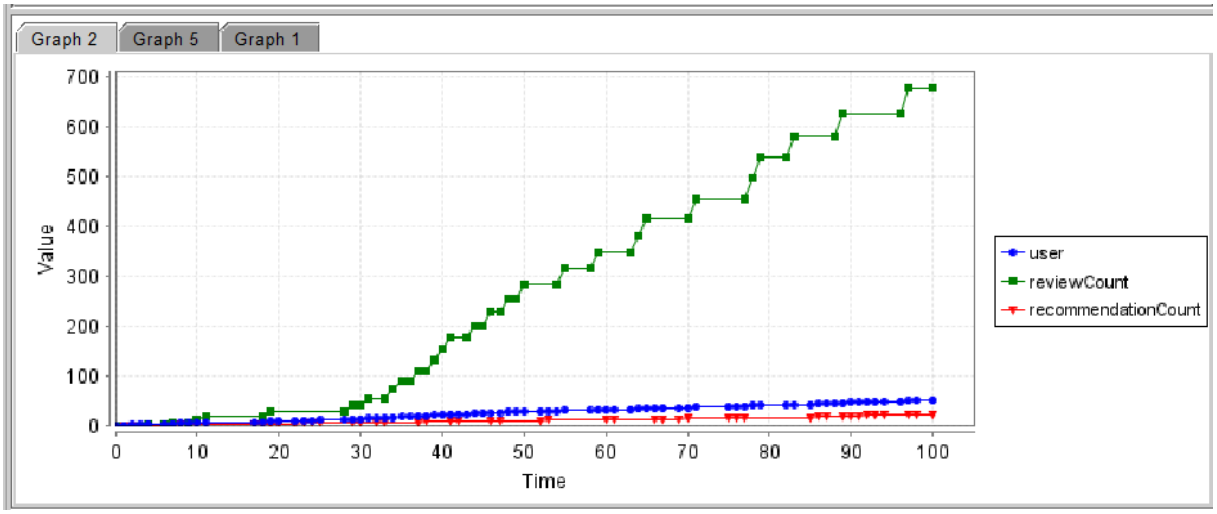
```
const Login = () => {
    const navigate = useNavigate();
    const { setUser } = useUser();
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const [userLikedMovies, setUserLikedMovies] = useState([]);
    const [userDislikedMovies, setUserDislikedMovies] = useState([]);

  const handleSubmit = async (e) => {
    e.preventDefault();

    try {
      const response = await fetch("http://localhost:8080/api/v1/users/login", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          username: username,
          password: password,
          userLikedMovies: userLikedMovies,
          userDislikedMovies: userDislikedMovies,
        }),
      });

      if (response.ok) {
        // Handle successful login (redirect, set state, etc.)
        console.log("Login successful");
        setUser(username);
        navigate("/");
      } else {
        // Handle login error
        console.error("Login failed");
      }
    } catch (error) {
      console.error("Error during login:", error);
    }
```

```jsx
  return (
    <div className="auth-form-container">
      <h2>Login</h2>
      <form className="login-form" onSubmit={handleSubmit}>
        <label htmlFor="username">username</label>
        <input
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          type="username"
          placeholder="username"
          id="username"
          name="username"
        />
        <label htmlFor="password">password</label>
        <input
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          type="password"
          placeholder="********"
          id="password"
          name="password"
        />
        <button type="submit">Log In</button>
      </form>
    </div>
  );
};

export default Login;
```

*TheCodeForPRISM

```
const int MAX_USERS = 1000;
const int MAX_REVIEWS = 1000000;
const int MAX_RECOMMENDATIONS = 10;

module MovieReviewsAndRecommendations
  user: [1..MAX_USERS] init 1;
  reviewCount: [0..MAX_REVIEWS] init 0;
  recommendationCount: [0..MAX_RECOMMENDATIONS] init 0;

[write_review] user < MAX_USERS & reviewCount < MAX_REVIEWS ->
  (user' = user + 1) & (reviewCount' = reviewCount + user);

[view_reviews] reviewCount > 0 ->
  (reviewCount' = reviewCount);

[get_recommendations] user < MAX_USERS ->
  (user' = user + 1) & (recommendationCount' = recommendationCount + 1);

[view_recommendations] recommendationCount > 0 ->
  (recommendationCount' = recommendationCount);

endmodule
```

***TheGraphGeneratedByThePRISM**



This sequence simulates a user writing a review, viewing all reviews, getting recommendations, and viewing the recommendation list for a movie**.** In the ‘write_review’ event, the number of reviews written increases by the current user count.

4.Appendix: Mini project

* User class:

```java
package com.helloworld.helloworld.sql;

import jakarta.persistence.*;

import javax.annotation.processing.Generated;

@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;
    @Column(nullable = false, unique = true, length = 30, name = "string")
    private String string;

    public Integer getId() { return id; }

    public void setId(Integer id) { this.id = id; }

    public String getString() { return string; }

    public void setString(String string) { this.string = string; }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", string='" + string + '\'' +
                '}';
    }
}
```

*UserController class:

```java
package com.helloworld.helloworld.sql;

import ...

@Controller
public class UserController {

    @Autowired private UserService service;

    @GetMapping("/users")
    public String showUserList(Model model){
        List<User> listUsers = service.listAll();
        model.addAttribute( attributeName: "listUsers", listUsers);

        return "users";
    }
}
```

*UserRepository class:

```java
package com.helloworld.helloworld.sql;

import org.springframework.data.repository.CrudRepository;

public interface UserRepository extends CrudRepository<User, Integer> {

}
```

*UserService class:

```java
package com.helloworld.helloworld.sql;

import ...

@Service
public class UserService {
    @Autowired private UserRepository repo;
    public List<User> listAll() { return (List<User>) repo.findAll(); }
}
```

*HelloWorldApplication class:

```java
package com.helloworld.helloworld;

import ...

@SpringBootApplication
public class HelloWorldApplication {

    public static void main(String[] args) { SpringApplication.run(HelloWorldApplication.class, args); }

}
```

*MainController class:

```java
package com.helloworld.helloworld;

import ...

@Controller
public class MainController {


    @GetMapping("")
    public String showHomePage() {

        return "index";
    }


}
```

5.References

https://griddb.net/en/blog/building-a-recommendation-system-in-java/

http://www.cs.ubbcluj.ro/~gabis/DocDiplome/SistemeDeRecomandare/Recommender_systems_handbook.pdf

https://spring.io/projects/spring-security/

https://www.prismmodelchecker.org/manual/RunningPRISM/AllOnOnePage

https://www.upwork.com/resources/what-is-content-based-filtering