

PRODUCTION GUIDE DOCUMENT

Movies Project

Swati Khandare
(swati.khandare@epita.fr)

Table of Contents

Roles:.....	3
User Role	3
Compilation and Installation Steps:	3
Execution:	7
API testing:	10

Roles

User Role

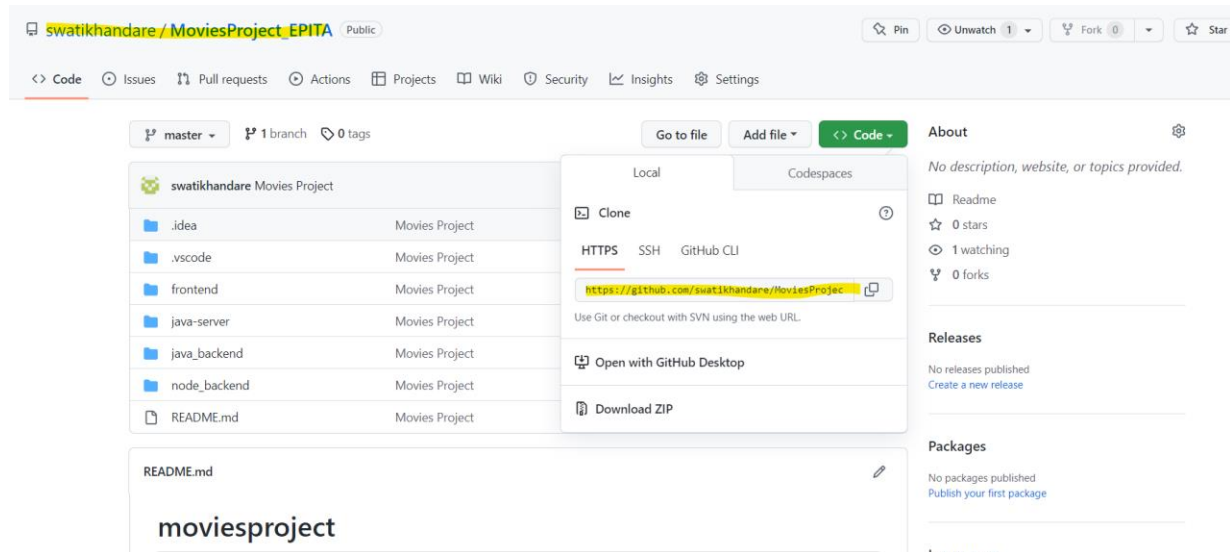
1. Create User Login
2. After login, below options are displayed:
 - HomePage Trailer
 - Continue Watching
 - Latest Movies
 - Recommended Movies
 - Top 10 Most Popular Movies
 - Top 10 Most Rated Movies

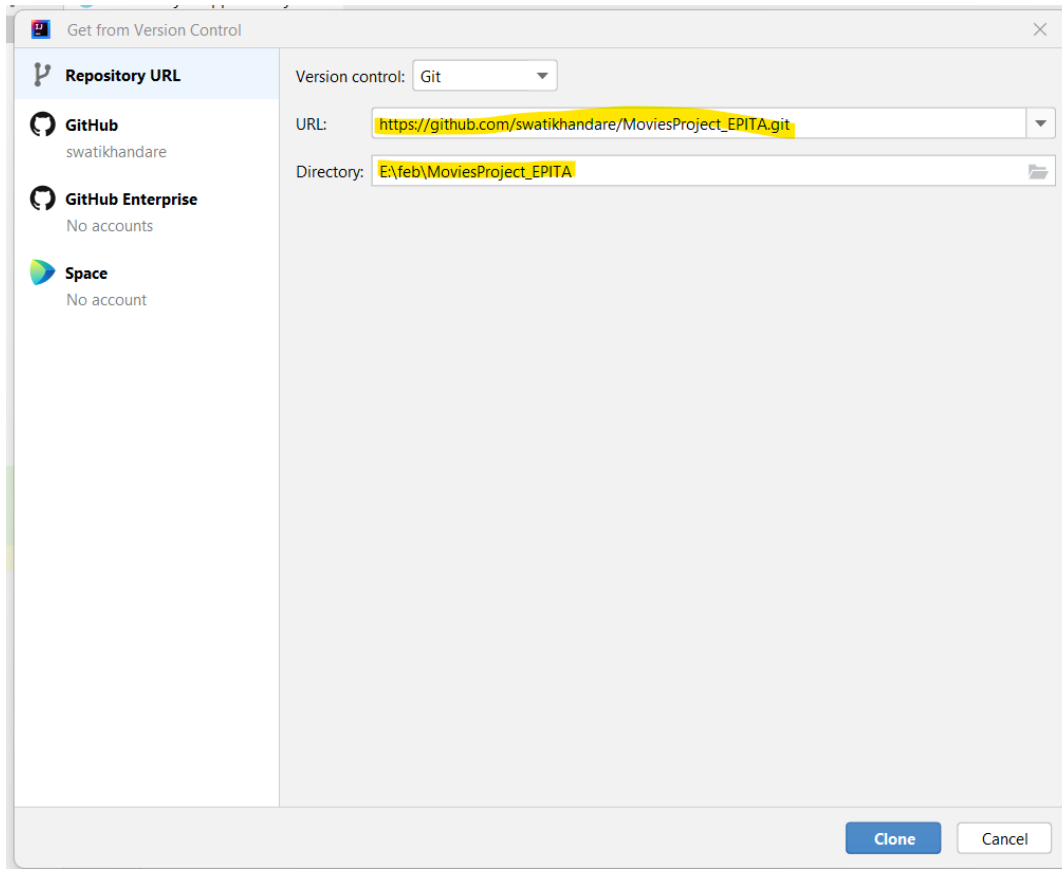
Click on play button to watch the trailer!

Compilation and Installation Steps

1. Copy the URL link and clone the project in IntelliJ.

URL: https://github.com/swatikhandare/MoviesProject_EPITA.git





2. Start the PostgreSQL and create the database using the details from “java_backend/src/main/resources/application.properties” file.
3. Create MongoDB cluster and collection with user and password on MongoDB Atlas
4. Create a file “.env” at path “node_backend” and add the MonogoDB details (for reference you check file “node_backend/.env.example”)
5. Run the SpringBootApplication at path “java_backend/src/main/java/fr/epita/java_backend/MoviesProjectApplication.java” to start the application.

The screenshot shows the VS Code editor with the `MoviesProjectApplication.java` file open. The file is located in the `java_backend` directory. The code is as follows:

```

1 package fr.epita.java_backend;
2 import ...
3
4
5 1 usage
6 @SpringBootApplication
7 public class MoviesProjectApplication {
8     public static void main(String[] args) { SpringApplication.run(MoviesProjectApplication.class, args); }
9 }
10
11
12

```

The Run console at the bottom shows the execution logs for `MoviesProjectApplication`. The logs indicate that the application started successfully on port 8080. Key log messages include:

- Tomcat initialized with port(s): 8080 (http)
- Starting service [Tomcat]
- Starting Servlet engine: [Apache Tomcat/9.0.64]
- Initializing Spring embedded WebApplicationContext
- Root WebApplicationContext: initialization completed in 2062 ms
- Processing PersistenceUnitInfo [name: default]
- Hibernate ORM core version 5.6.14.Final
- Hibernate Commons Annotations {5.1.2.Final}
- HikariPool-1 - Starting...
- HikariPool-1 - Start completed.
- Using dialect: org.hibernate.dialect.PostgreSQLDialect
- Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
- Initialized JPA EntityManagerFactory for persistence unit 'default'
- spring.jpa.open-in-view is enabled by default. Therefore, database queries may be executed during view rendering. This will not be a problem in an actual application.
- Tomcat started on port(s): 8080 (http) with context path ''
- Started MoviesProjectApplication in 6.816 seconds (JVM running for 7.7)

6. Open the project in VSCode.

7. Navigate to folder “node_backend” and execute command “npm install”.

The screenshot shows the VS Code editor with the `README.md` file open. The file content is as follows:

```

1 # MoviesProject
2 Movies Project is developed during SE 3rd semester using React, Node, Express, MongoDB, Java and PostgreSQL
3
4 Technical Specification Document contains the software and hardware requirement for the project.
5
6 Production document contains the installation steps and steps to execute the project.

```

The terminal at the bottom shows the execution of the `npm install` command in the `node_backend` directory. The output is:

```

PS D:\S3\MoviesProject_EPITA> cd .\node_backend\
PS D:\S3\MoviesProject_EPITA\node_backend> npm install
[... output ...]
idealTree:mongoose: [... output ...] Completed in 1312ms

```

8. Execute command “npm run dev” to start the backend server.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

●
● added 206 packages, and audited 207 packages in 18s

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\S3\MoviesProject_EPITA\node_backend> npm run dev

○ > server@1.0.0 dev
> nodemon index.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
MongoDB Connected...
(node:2644) [MONGODB] DeprecationWarning: Mongoose: the `strictQuery` option will be switched back to `false` by default
in Mongoose 7. Use `mongoose.set('strictQuery', false);` if you want to prepare for this change. Or use `mongoose.set('s
trictQuery', true);` to suppress this warning.
(Use `node --trace-deprecation ...` to show where the warning was created)
Server running on port 3001

```

9. Navigate to folder “frontend” and execute command “npm install”.

```

EXPLORER
  MOVIESPROJECT_EPITA
    .idea
    .vscode
    frontend
    public
    src
    .gitignore
    package.json
    java_backend
    node_backend
    testAPI_postman_collection
    README.md

README.md
1 # MoviesProject
2 Movies Project is developed during SE 3rd semester using React, Node, Express, MongoDB, Java and PostgreSQL
3
4 Technical Specification Document contains the software and hardware requirement for the project.
5
6 Production document contains the installation steps and steps to execute the project.

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

● PS D:\S3\MoviesProject_EPITA> cd .\node_backend\
● PS D:\S3\MoviesProject_EPITA\node_backend> npm install

added 206 packages, and audited 207 packages in 18s

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\S3\MoviesProject_EPITA\node_backend>

PS D:\S3\MoviesProject_EPITA> cd .\frontend\
PS D:\S3\MoviesProject_EPITA\frontend> npm install
[.....] \ idealTree:frontend: sill idealTree bui

```

10. Execute command “npm run start” to start the front end.

```

TERMINAL

○ PS D:\S3\MoviesProject_EPITA\frontend> npm run start

```

SOLE TERMINAL

Compiled successfully!

You can now view **client** in the browser.

Local: <http://localhost:3000>

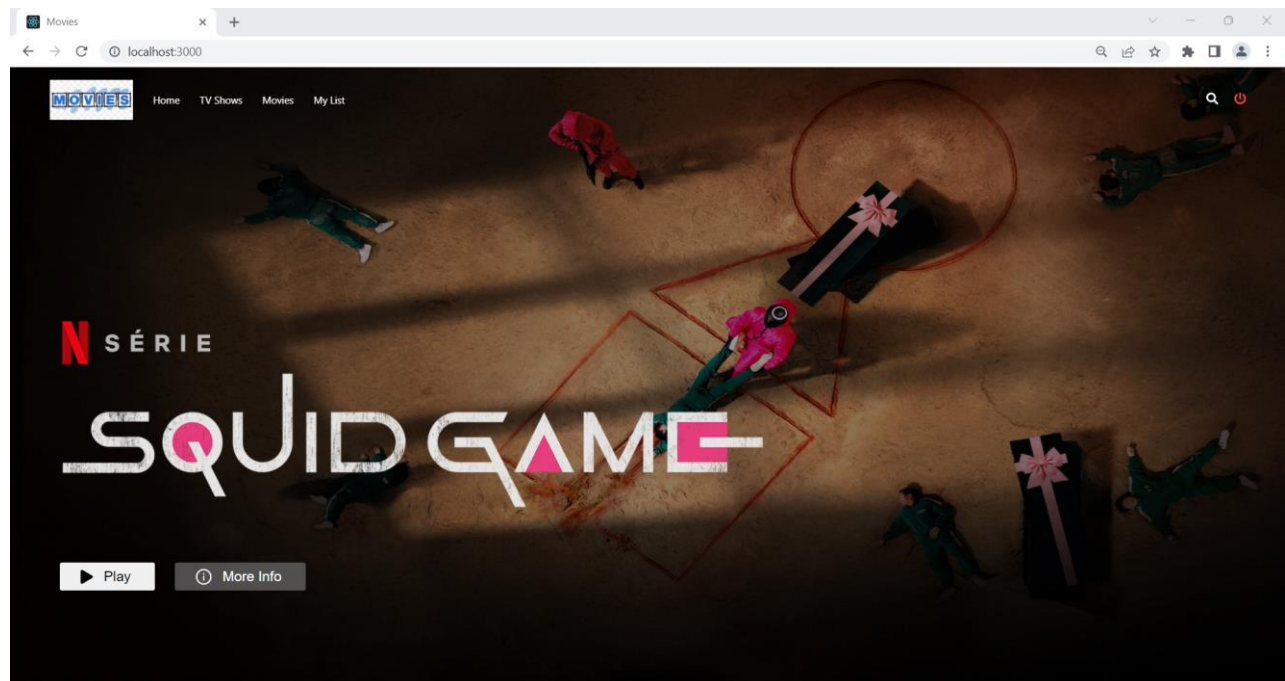
On Your Network: <http://10.188.83.198:3000>

Note that the development build is not optimized.
To create a production build, use `npm run build`.

webpack compiled successfully

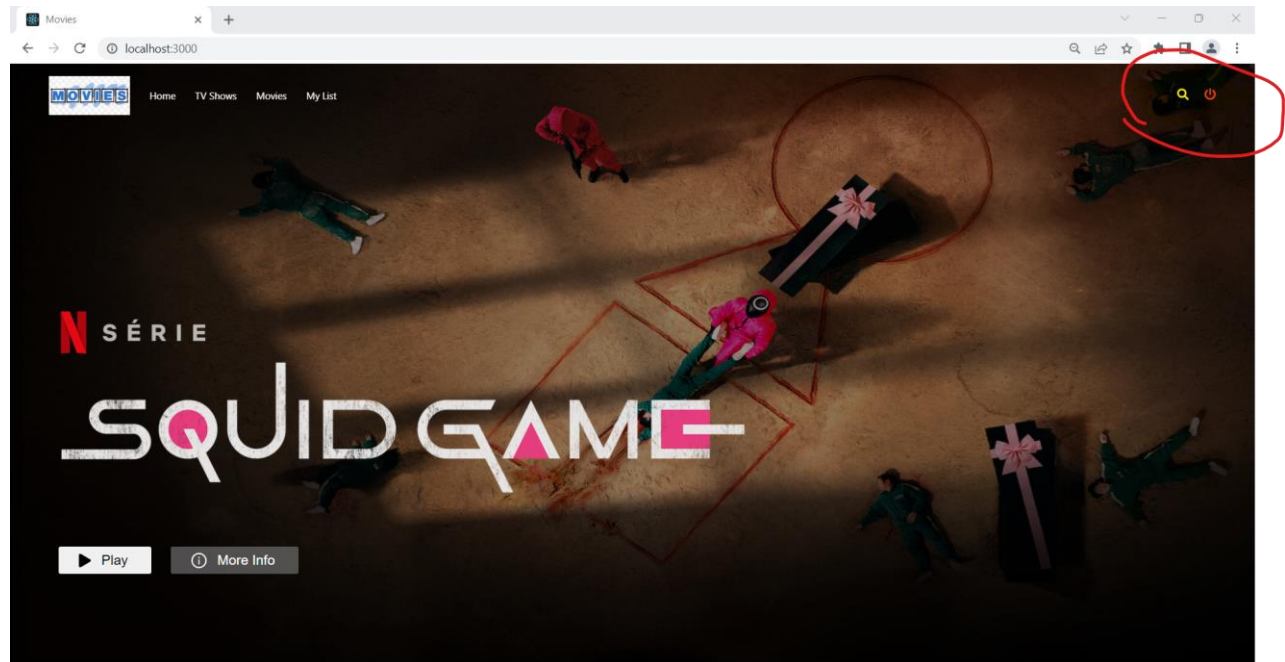
□

11. The application will be launched in your localhost at port 3000!

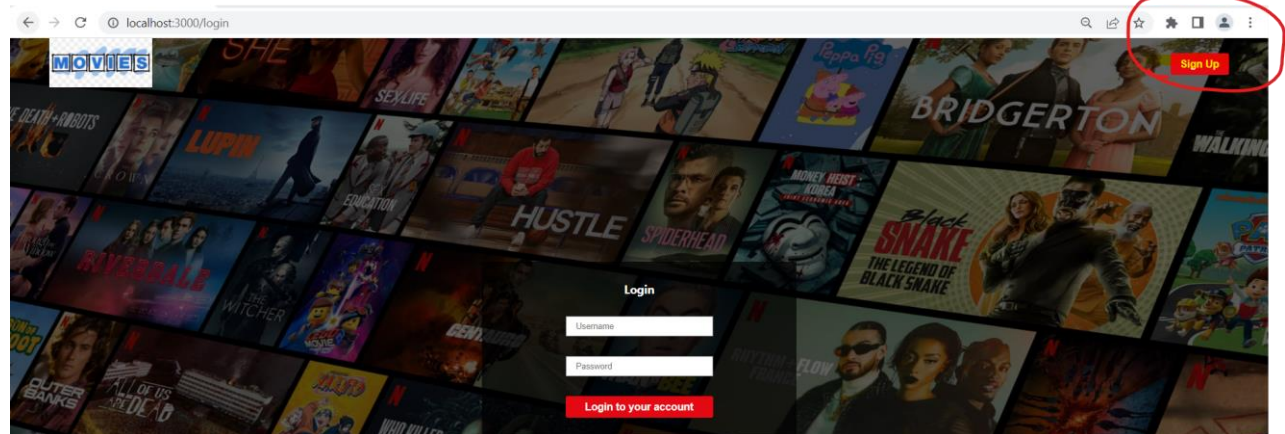


Execution

1. Landing page



2. Sign Up



MOVIES

localhost:3000/signup

Sign In

Username

First Name

Last Name

dd- - - - - yyyy

Male

Sign Up

MOVIES

localhost:3000/signup

Sign In

dummyuser

dummy

user

17 - Jun - 2000

Female

Sign Up

MOVIES

localhost:3000/login

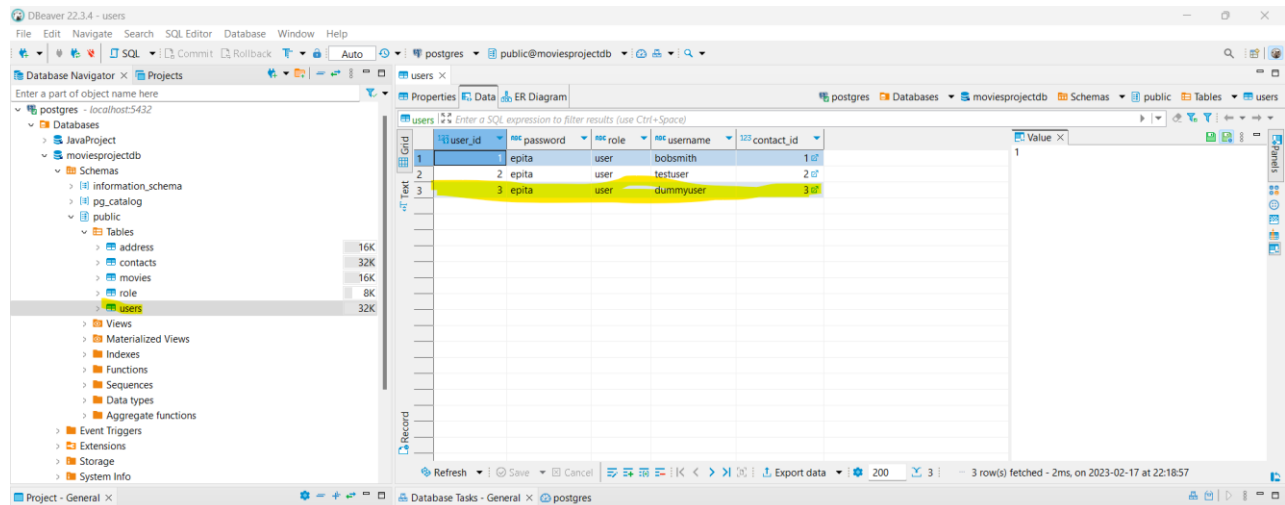
Sign Up

Login

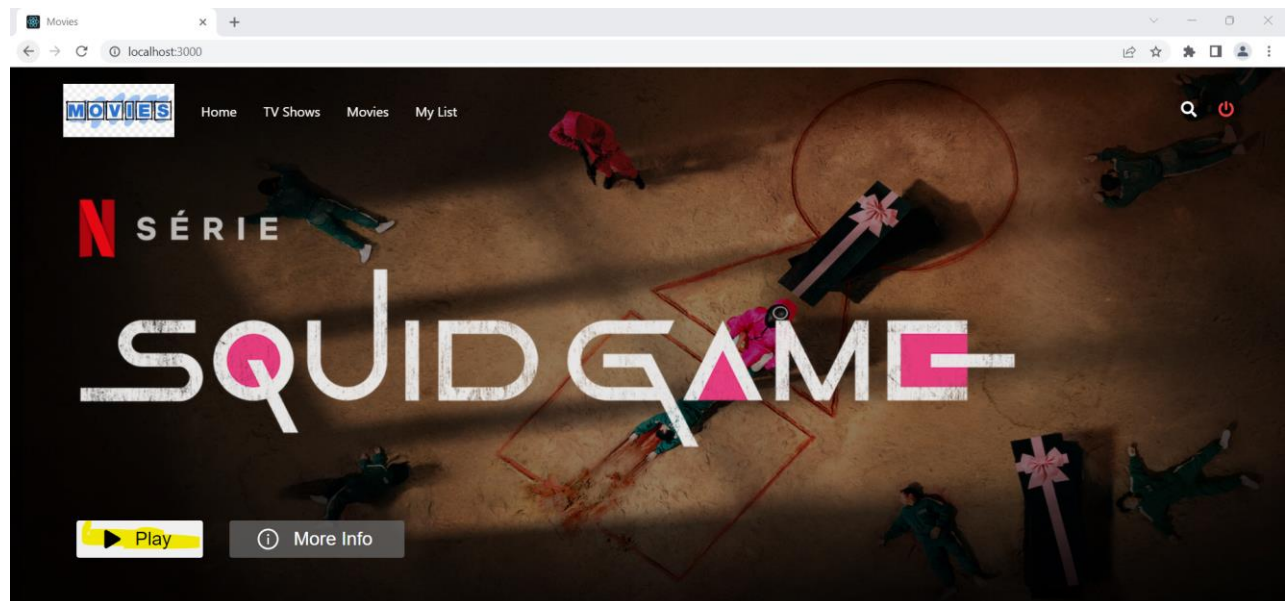
bobsmith

Login to your account

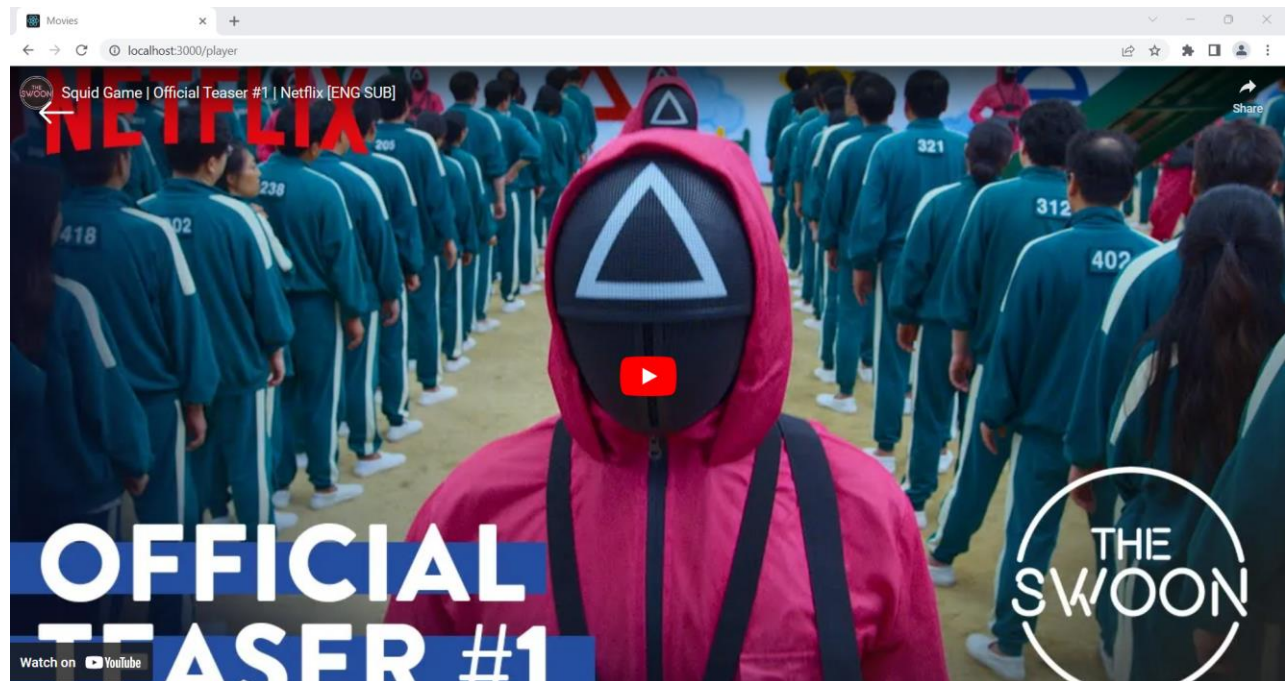
3. User added in users postgresql db



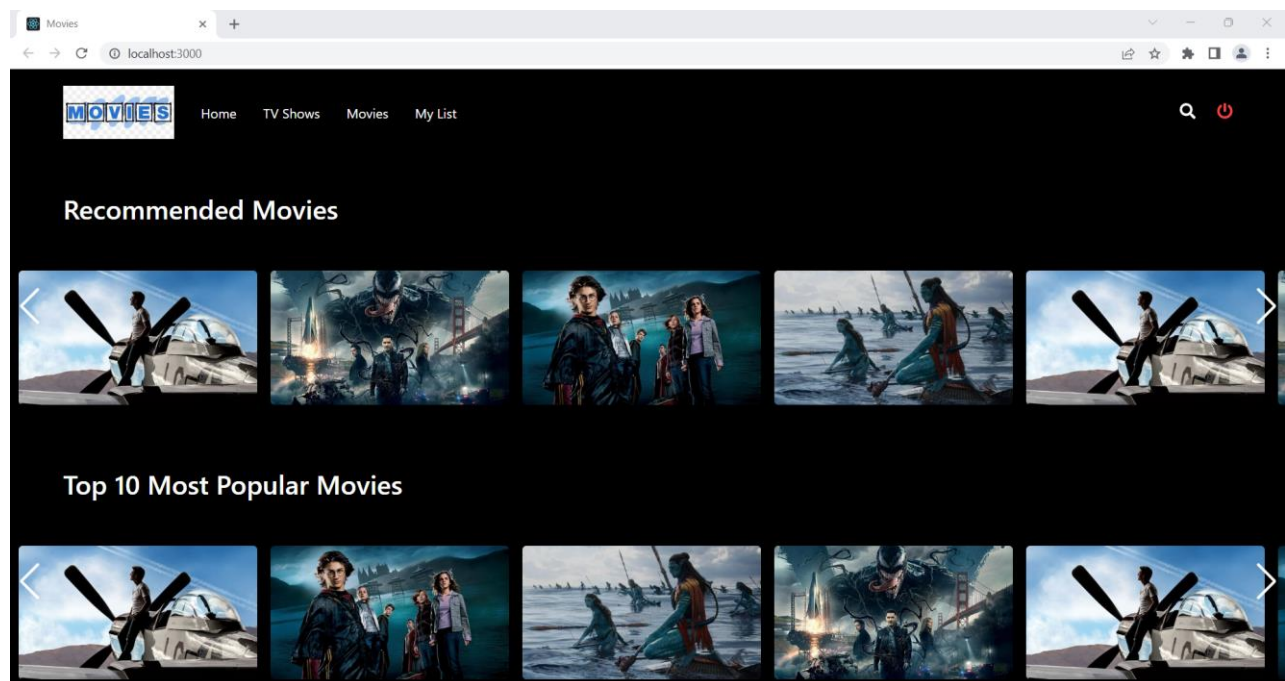
4. Sign in with the user created



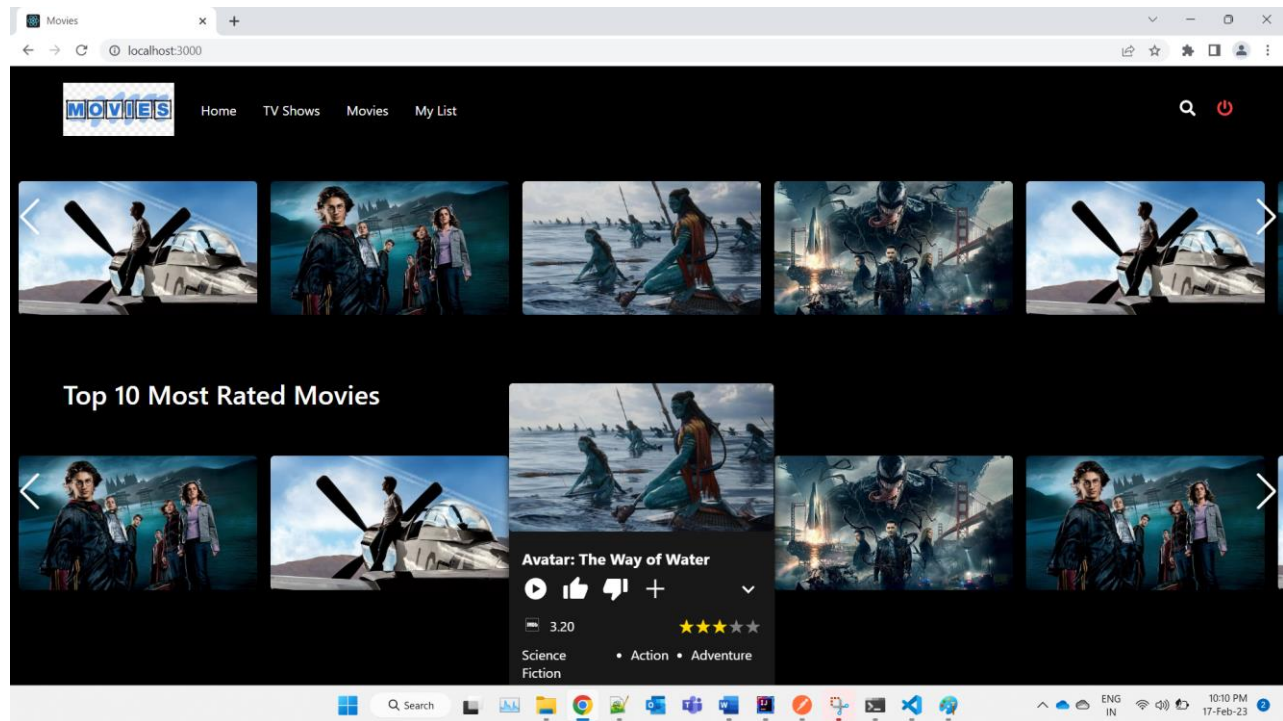
5. Click on play to play the Squid Game trailer.



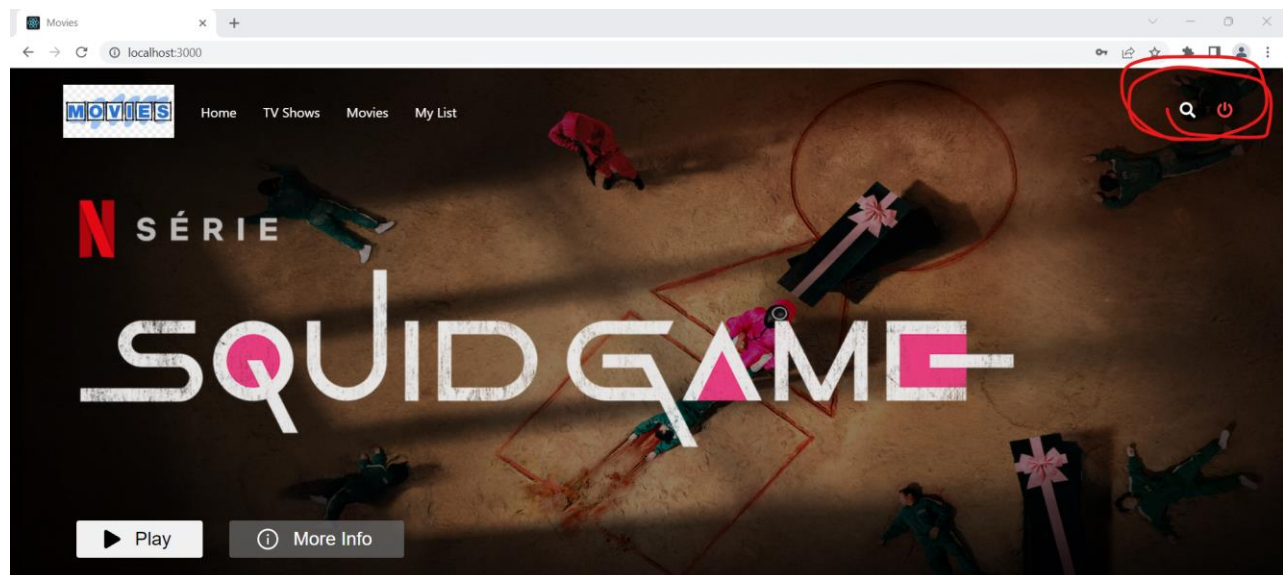
6. Scroll through different sections



7. Check for the ratings by hovering over a particular movie



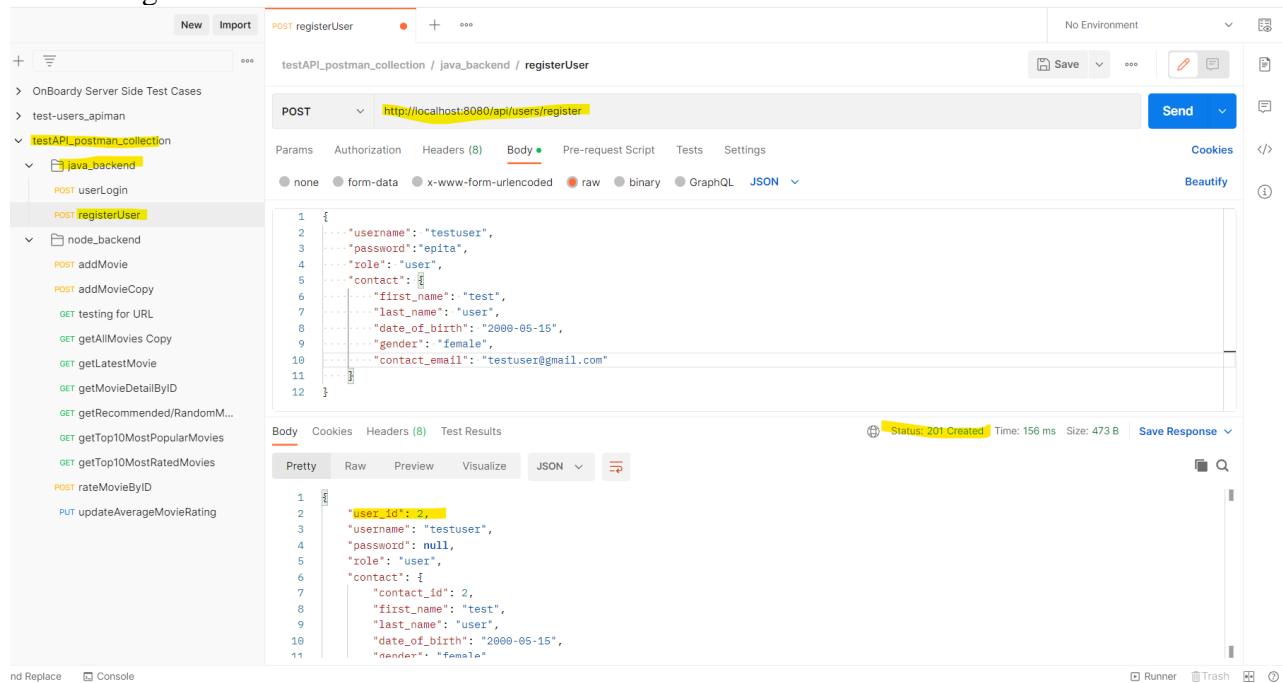
8. Sign out



API testing

Import the postman collection from folder “**testAPI_postman_collection**” to your local postman to test the **java backend** and **node backend** for APIs “**api/users**”, “**/api/movies/**” and “**/api/movies/rating**” .

1. Register user API

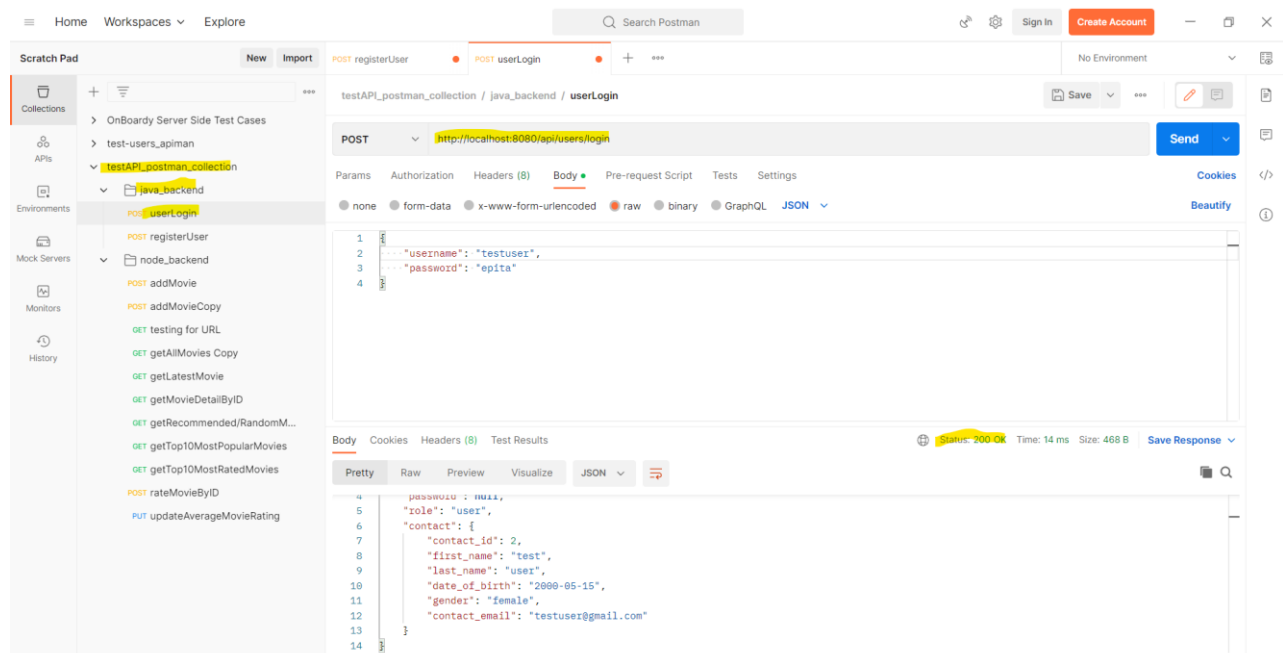


The screenshot shows the Postman interface for the 'registerUser' API endpoint. The request is a POST to 'http://localhost:8080/api/users/register'. The body is a JSON object with the following fields: 'username' (testuser), 'password' (epita), 'role' (user), 'contact' (object with 'contact_id' 2, 'first_name' test, 'last_name' user, 'date_of_birth' 2000-05-15, 'gender' female, 'contact_email' testuser@gmail.com). The response is a 201 status code with a JSON body containing the same user details and a 'user_id' of 2.

```
1 {
2   "username": "testuser",
3   "password": "epita",
4   "role": "user",
5   "contact": {
6     "contact_id": 2,
7     "first_name": "test",
8     "last_name": "user",
9     "date_of_birth": "2000-05-15",
10    "gender": "female",
11    "contact_email": "testuser@gmail.com"
12  }
13 }
```

Body: { "user_id": 2, "username": "testuser", "password": null, "role": "user", "contact": { "contact_id": 2, "first_name": "test", "last_name": "user", "date_of_birth": "2000-05-15", "gender": "female" } }

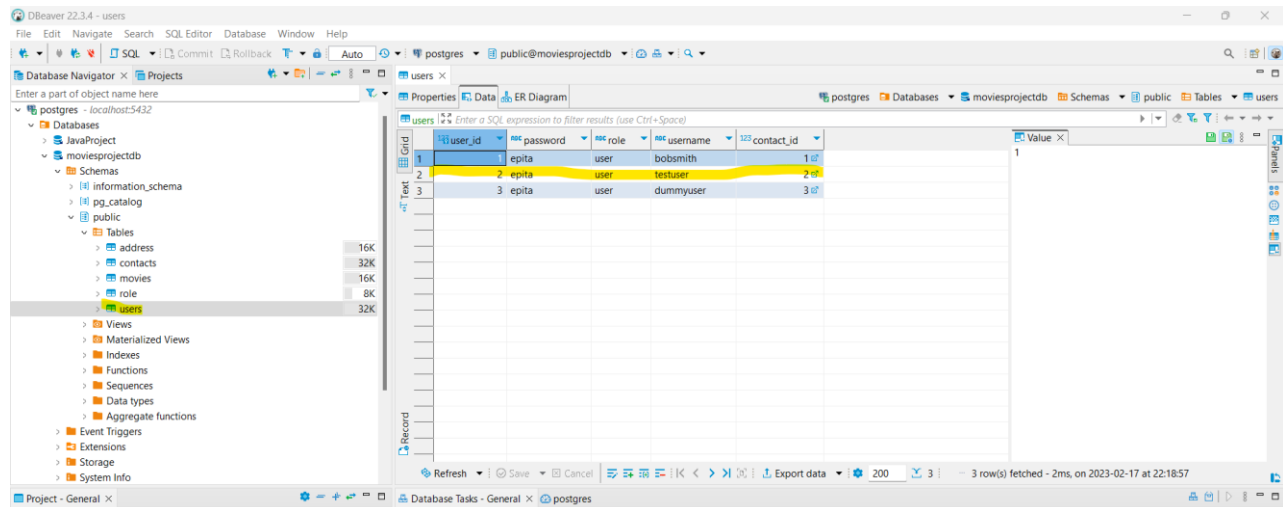
2. User login API



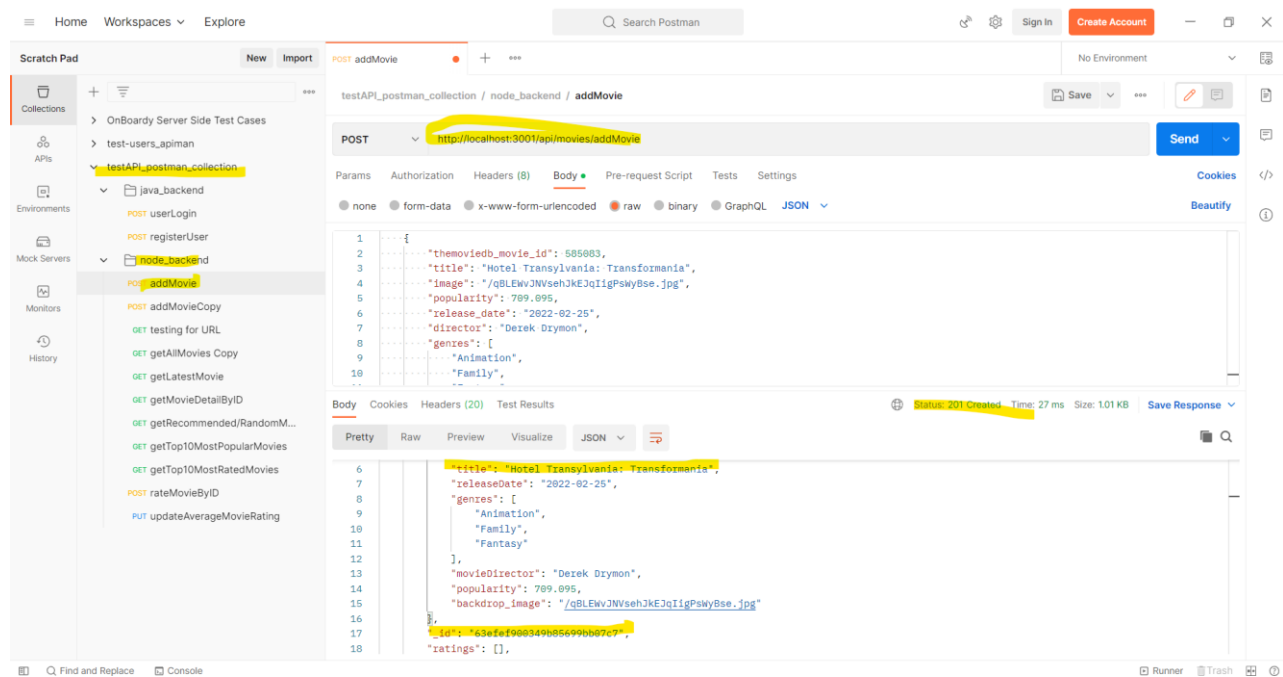
The screenshot shows the Postman interface for the 'userLogin' API endpoint. The request is a POST to 'http://localhost:8080/api/users/login'. The body is a JSON object with the following fields: 'password' (null), 'role' (user), 'contact' (object with 'contact_id' 2, 'first_name' test, 'last_name' user, 'date_of_birth' 2000-05-15, 'gender' female, 'contact_email' testuser@gmail.com). The response is a 200 status code with a JSON body containing the same user details.

```
1 {
2   "password": null,
3   "role": "user",
4   "contact": {
5     "contact_id": 2,
6     "first_name": "test",
7     "last_name": "user",
8     "date_of_birth": "2000-05-15",
9     "gender": "female",
10    "contact_email": "testuser@gmail.com"
11  }
12 }
```

New user is added in the postgresql db table



3. Add movie API



4. Add movie rating by movie id API

The screenshot shows the Postman interface with a POST request selected. The URL is `http://localhost:3001/api/movies/rating/63efef900349b85699bb07c7`. The request body is a JSON object:

```

1 {
2   "rating": "4.1",
3   "comment": "Good",
4   "userId": 3
5 }

```

The response body is also shown in JSON format:

```

11 {
12   "movieDirector": "Wesek Weyson",
13   "popularity": 799.095,
14   "backdrop_image": "/qBLKwJNVseh3kEjIgfPswyBse.jpg",
15   "_id": "63efef900349b85699bb07c7",
16   "ratings": [
17     {
18       "rating": "4.1",
19       "comment": "Good",
20       "userId": "3",
21       "_id": "63efef900349b85699bb07c7"
22     },
23   ]
24 }

```

The status bar indicates a 201 Created response with a time of 38 ms and a size of 1.14 KB.

5. Add average rating by movie id API

The screenshot shows the Postman interface with a PUT request selected. The URL is `http://localhost:3001/api/movies/rating/updateMovieAverageRating/63efef900349b85699bb07c7`. The request body is a JSON object:

```

1 {
2   "rating": "4.1",
3   "comment": "Good",
4   "userId": 3
5 }

```

The response body is also shown in JSON format:

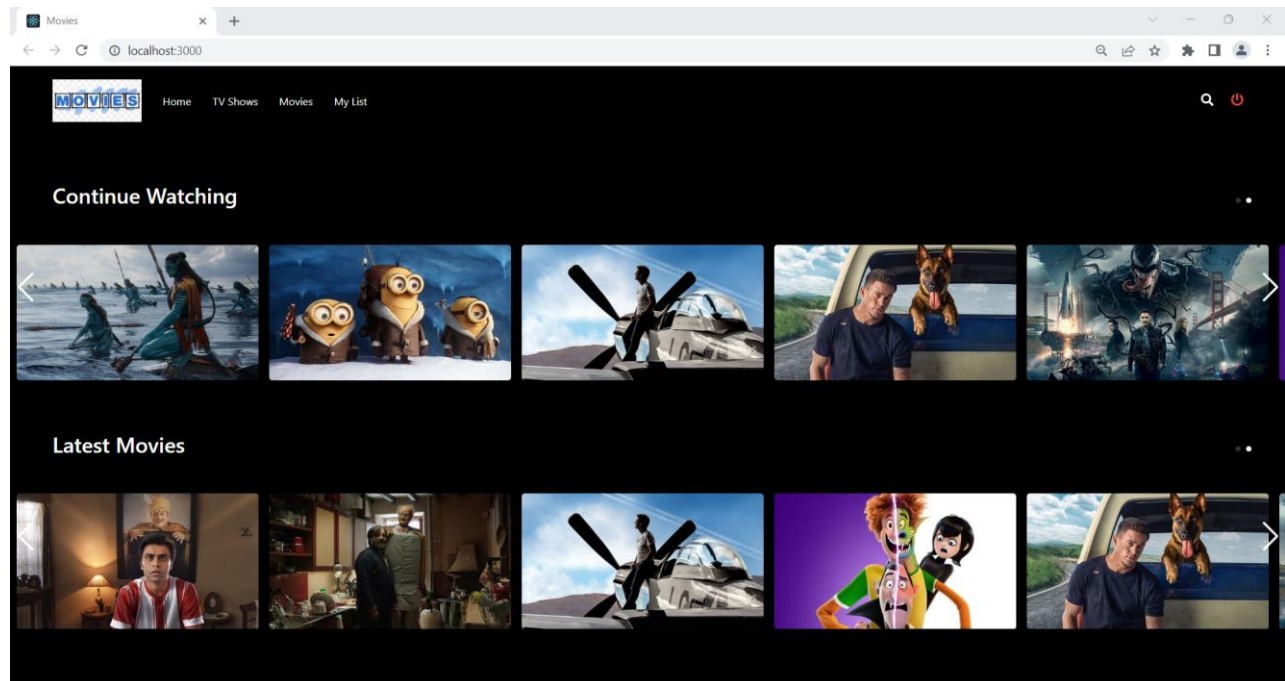
```

20 {
21   "userId": "3",
22   "_id": "63efef900349b85699bb07c7",
23   "rating": 4.1,
24   "comment": "Good",
25   "userId": "3",
26   "_id": "63efef900349b85699bb07c7"
27 },
28 {
29   "v": 0,
30   "averageRating": 4.1
31 }
32 }

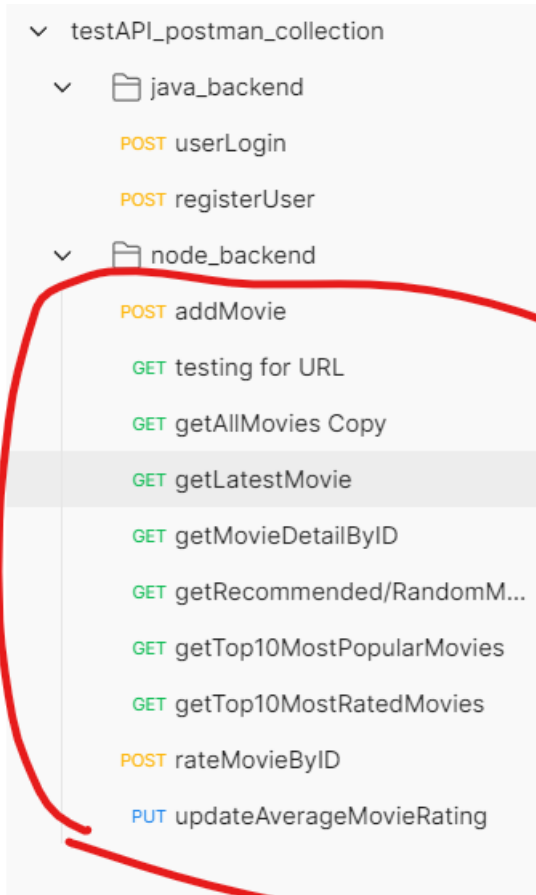
```

The status bar indicates a 201 Created response with a time of 43 ms and a size of 1.16 KB.

Movie is added



6. Other different test APIs for get all movies, get latest movies, get movies details , get top 10 movies etc.



S