Advanced Web and Javascript And API Management Project

Report

By
Elias Afara

Professors
Thomas Broussard
Loïc Dandoy

EPITA Graduate School of Computer Science

MSc – Software Engineering

August 12, 2022

Table of Contents

Goal

This project aim at realizing a real-world application.

Needs to setup the whole infrastructure to develop the project and achieve the goal.

This project includes:

- Front-end development : develop a new application from scratch
- Back-end development : using node and java to provide REST APIs
- Database modeling and harnessing using mongo and postgresql
- Use GitHub for version control: https://github.com/EliasAfara/netflix-clone

Functional Description

This application implements the following:

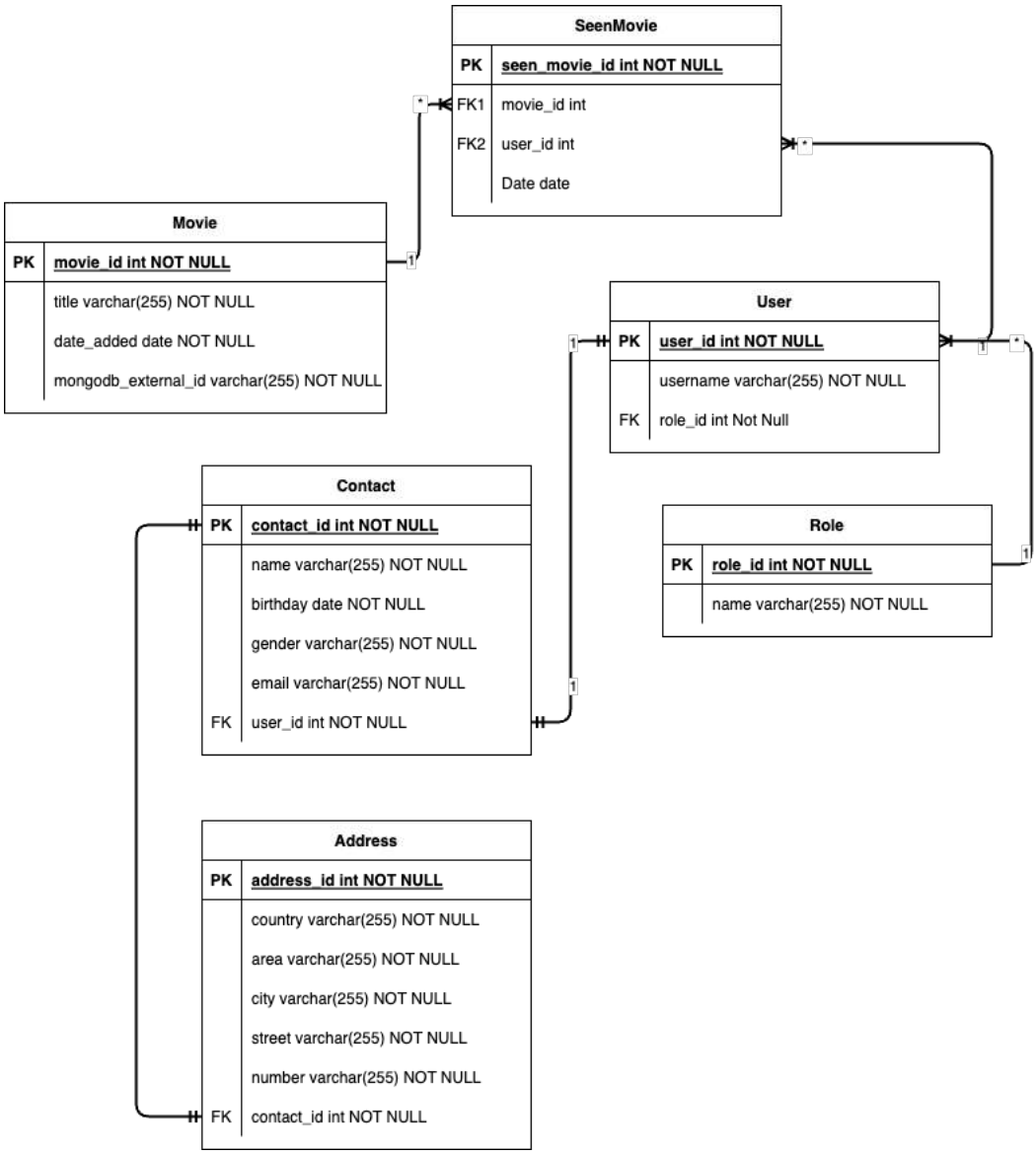
- User authentication
- Login & Registration pages
- Netflix home page with movies lists being displayed
- Movie player (mock)
- Showcase movie rating & rating functionality
- compute stats (10 most popular movies, 10 most viewed movies)

Database

I have used two types of databases as was required:

PostgreSQL

Entity Relational Diagram



Database creation:

```
→  ~ psql postgres
psql (14.4)
Type "help" for help.

postgres=# DROP DATABASE IF EXISTS netflixclonedb;
DROP DATABASE
postgres=# DROP USER IF EXISTS netflixcloneuser;
DROP ROLE
postgres=# CREATE USER netflixcloneuser with password 'password';
CREATE ROLE
postgres=# CREATE DATABASE netflixclonedb with template=template0 owner=netflixcloneuser;
CREATE DATABASE
postgres=# alter default privileges grant all on tables to netflixcloneuser;
ALTER DEFAULT PRIVILEGES
postgres=# \l
                                  List of databases
      Name       |      Owner       | Encoding | Collate | Ctype |     Access privileges
-----------------+------------------+----------+---------+-------+-----------------------
 netflixclonedb  | netflixcloneuser | UTF8     | C       | C     |
 perntodo        | newuserelias     | UTF8     | C       | C     |
 postgres        | eliasafara       | UTF8     | C       | C     |
 ral_tasks_queue | newuserelias     | UTF8     | C       | C     |
 template0       | eliasafara       | UTF8     | C       | C     | =c/eliasafara           +
                 |                  |          |         |       | eliasafara=CTc/eliasafara
 template1       | eliasafara       | UTF8     | C       | C     | =c/eliasafara           +
                 |                  |          |         |       | eliasafara=CTc/eliasafara
(6 rows)

postgres=# \du
                                   List of roles
    Role name     |                         Attributes                         | Member of
------------------+------------------------------------------------------------+-----------
 eliasafara       | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 netflixcloneuser |                                                            | {}
 newuserelias     | Create DB                                                  | {}
```

Queries

```
→  ~ psql postgres -U netflixcloneuser
psql (14.4)
Type "help" for help.

postgres=> CREATE TABLE role(
                    role_name varchar(10) NOT NULL,
                    PRIMARY KEY (role_name)
);

INSERT INTO role (role_name) VALUES ('admin');
INSERT INTO role (role_name) VALUES ('user');
CREATE TABLE
INSERT 0 1
INSERT 0 1
postgres=> CREATE TABLE contacts(
                        contact_id serial PRIMARY KEY,
                        first_name varchar(255),
                        last_name varchar(355),
                        date_of_birth DATE,
                        gender varchar(6),
                        contact_email varchar(150) NOT NULL,
                        UNIQUE(contact_email)
);

CREATE TABLE users(
                     user_id serial PRIMARY KEY,
                     username varchar(45) UNIQUE,
                     password text NOT NULL,
                     "role" varchar(10) not null REFERENCES role (role_name),
                     contact_id int not null REFERENCES contacts (contact_id)
);
CREATE TABLE
CREATE TABLE
postgres=> CREATE TABLE movies(
                       movie_id serial PRIMARY KEY,
                       movie_title varchar(250) NOT NULL,
                       added DATE NOT NULL,
                       mongodb_movie_id varchar(150) NOT NULL
);

CREATE TABLE seenMovie(
                         seenMovie_id serial PRIMARY KEY,
                         user_id int REFERENCES users (user_id) ON UPDATE CASCADE ON DELETE CASCADE,
                         movie_id int REFERENCES movies (movie_id) ON UPDATE CASCADE ON DELETE CASCADE,
                         seenMovie_date DATE NOT NULL
);

CREATE TABLE address(
                       address_id serial PRIMARY KEY,
                       country varchar(45) NOT NULL,
                       area varchar(45) NOT NULL,
                       city varchar(45) NOT NULL,
                       street varchar(150) NOT NULL,
                       address_number varchar(45) NOT NULL,
                       contact_id int REFERENCES contacts (contact_id)
);
CREATE TABLE
CREATE TABLE
CREATE TABLE
postgres=>
```

MongoDB

Mongoose Schema

node-server > models > JS movies.model.js > ...

You, now | 1 author (You)

```
1  const mongoose = require('mongoose');
2
3  const moviesRatingSchema = mongoose.Schema({
4    movie: {
5      themoviedb_movie_id: {
6        type: Number,
7      },
8      title: {
9        type: String,
10       required: true,
11     },
12     releaseDate: {
13       type: String,
14       required: true,
15     },
16     genres: [String],
17     movieDirector: {
18       type: String,
19       required: true,
20     },
21     popularity: {
22       type: Number,
23     },
24     backdrop_image: {
25       type: String,
26       required: true,
27     },
28   },
29   averageRating: {
30     type: Number,
31     min: 0,
32     max: 10,
33   },
34   ratings: [
35     {
36       rating: {
37         type: Number,
38         required: true,
39       },
40       comment: {
41         type: String,
42         required: true,
43       },
44       userId: {
45         type: String,
46         required: true,
47       },
48     },
49   ],
50 });
51
52 module.exports = mongoose.model('Rating', moviesRatingSchema);
```

Single user JSON data


```
{
        "movie": {
            "themoviedb_movie_id": 76600,
            "title": "Avatar: The Way of Water",
            "releaseDate": "2022-12-14",
            "genres": [
                "Science Fiction",
                "Action",
                "Adventure"
            ],
            "movieDirector": "James Cameron",
            "popularity": 173.253,
            "backdrop_image": "/wEQ0Pu2jEyqHKOJRAdAKaeTFCML.jpg"
        },
        "_id": "62dd86b289947f7eedf3b09c",
        "ratings": [
            {
                "rating": 8,
                "comment": "I loved this movie!!",
                "userId": "1",
                "_id": "62ddb6bbd769fd6746379391"
            },
            {
                "rating": 9,
                "comment": "I liked this!",
                "userId": "2",
                "_id": "62ddb720d769fd6746379396"
            },
            {
                "rating": 5,
                "comment": "not bad",
                "userId": "3",
                "_id": "62ddb7d1bd7972df48badcde"
            },
            {
                "rating": 5,
                "comment": "good",
                "userId": "6",
                "_id": "62e5cbc21f3522f546fab795"
            },
            {
                "rating": 4,
                "comment": "nice",
                "userId": "4",
                "_id": "62f56506496cba0b69504a91"
            }
        ],
        "__v": 0,
        "averageRating": 6.2
    }
```

Backend

Created Rest APIs in both Java and Nodejs

Java

Running on port **8080**

Endpoints:

Login: **/api/users/login**

Registration: /**api/users/register**

Nodejs

Running on port **3001**

Movies: **/api/movies/**

- get all movies: **/**
- add new movie: **/addMovie**
- get top 10 movies most popular: **/top10MoviesMostPopular**
- get top 10 movies most rated: **/top10MoviesMostRated**
- get recommended movies: **/randomRecommendation**
- get movie details by id: **/:movieId**

Movies ratings: **/api/movies/ratings**

- update ratings: **/updateRatings**
- update movie average rating: **/updateMovieAverageRating/:movieId**
- add movie rating: **/:movieId**
- get movie ratings: **/:movieId**
- update movie rating: **/:movieId**
- delete movie rating: **/:movieId**

Frontend

Used React for client side

Screenshots

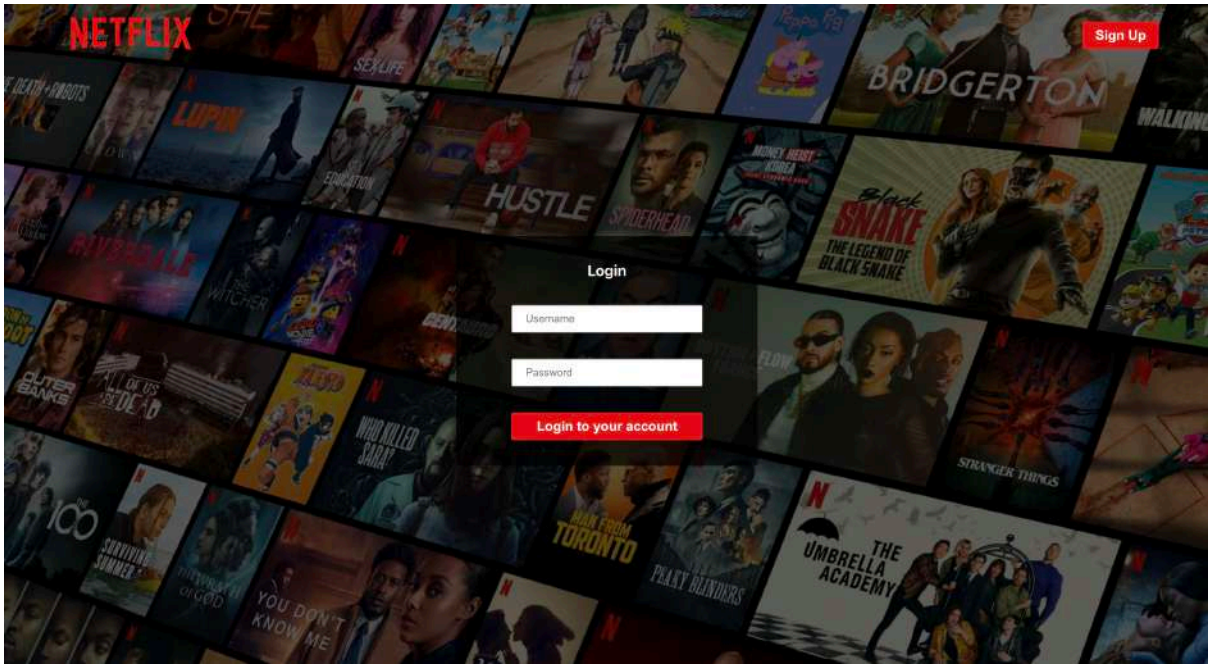


Figure 1 - Login Page

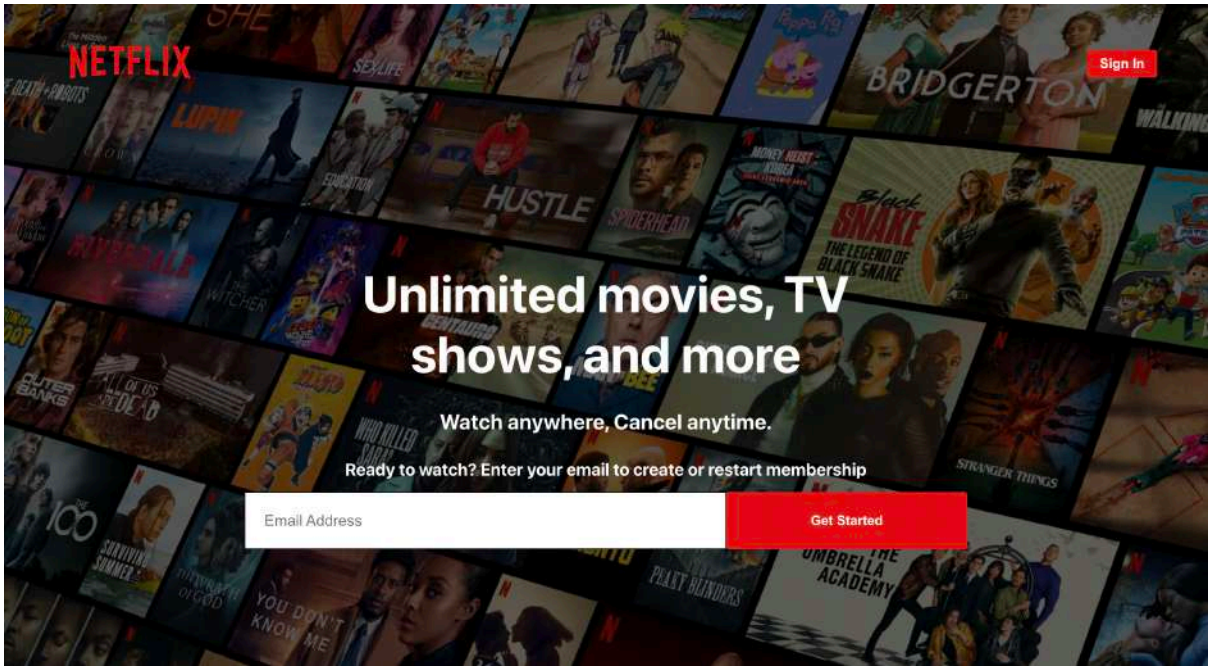


Figure 2 - Registration Page 01

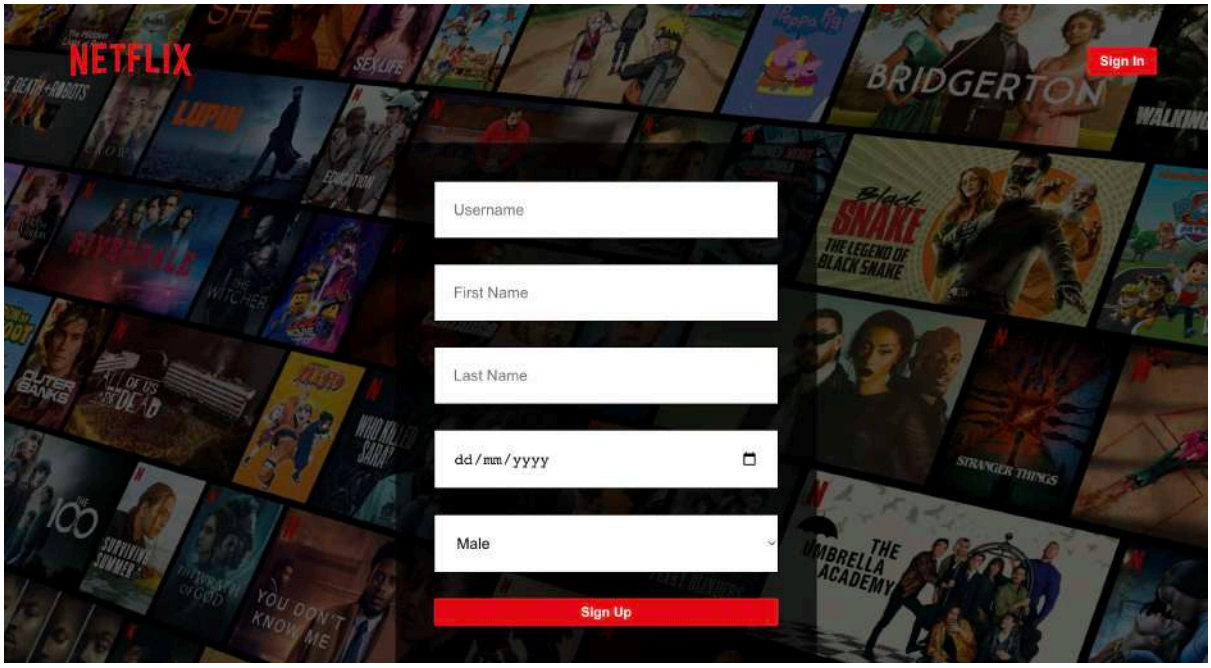


Figure 3 - Registration Page 02

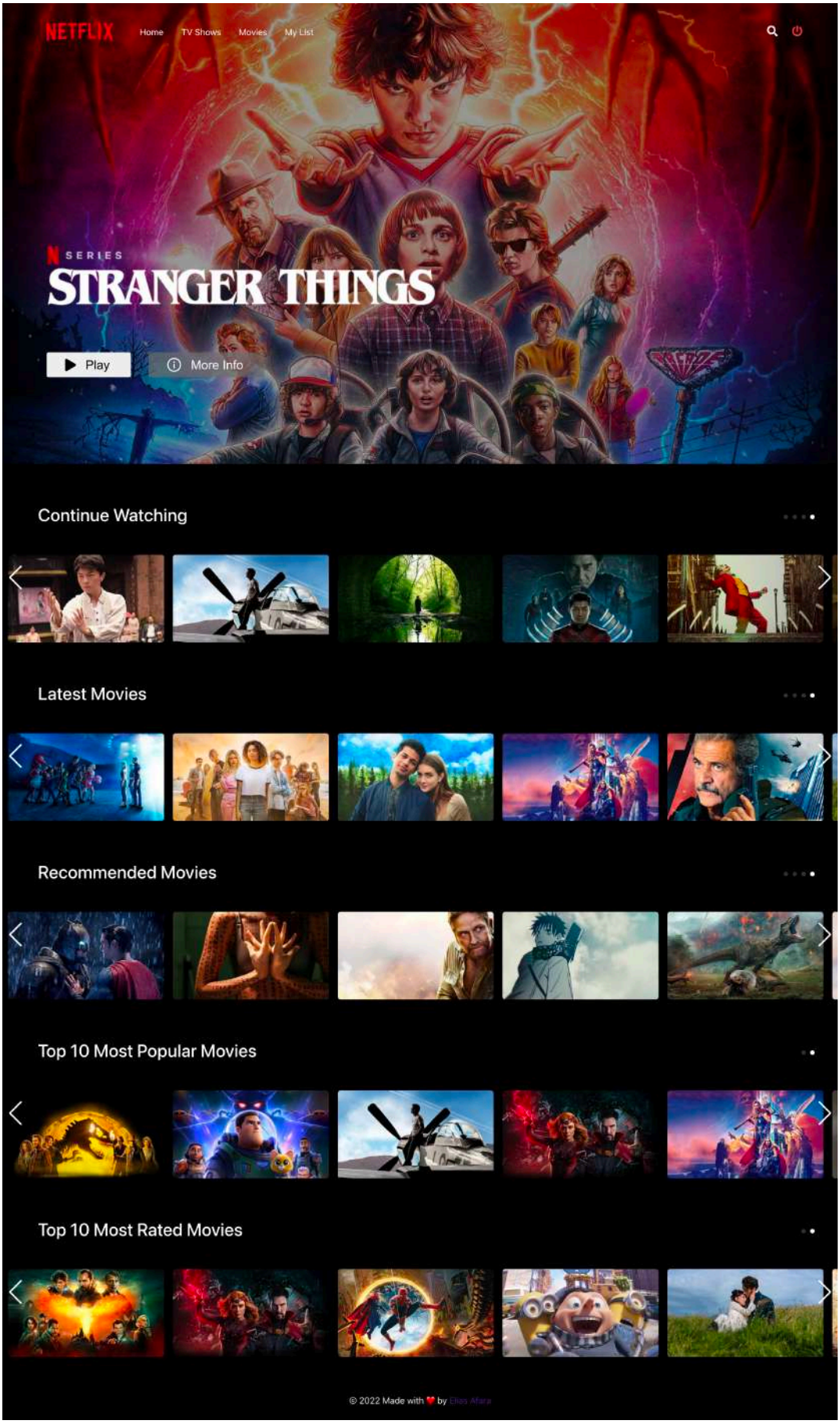


Figure 4 - Netflix Home Page

Testing

I have used postman to test the APIs for both backends.