Table of Contents

[Starting Servers: 2](#_Toc111485260)

[Part: Web-Application Patterns 2](#_Toc111485261)

[Part: Frontend React 2](#_Toc111485262)

[Part: Backend node 2](#_Toc111485263)

[Part: Java backend 2](#_Toc111485264)

[Documentation of API man 3](#_Toc111485265)

[Title 1: WAP API gateway Management 3](#_Toc111485266)

[Title 2: Keycloak generating token 4](#_Toc111485267)

[Title 3: Make endpoint private from public 5](#_Toc111485268)

[Title 4: Get data from endpoint using tokens 5](#_Toc111485269)

[Title 5: Connecting API with CORS 6](#_Toc111485270)

[Documentation of Front End: 9](#_Toc111485271)

[Documentation of Back End Node: 10](#_Toc111485272)

[API created 10](#_Toc111485273)

[Documentation Java Backend: 11](#_Toc111485274)

[MovieController: 11](#_Toc111485275)

[RoleController: 11](#_Toc111485276)

[UserController: 11](#_Toc111485277)

Subject: Report of Netflix

Created by: Anshul Bikram Rana

# Github: https://github.com/anshulbrana/netflix\_complete.git

# Starting Servers:

## Part: Web-Application Patterns

Run APIMAN

URL: localhost:8080/apimanui (For API man)

URL: localhost:8080/auth (For realms)

## Part: Frontend React

Package name: new

Command: yarn start

## Part: Backend node

Package name: netflix-backend

Command: yarn start

## Part: Java backend

Package name: epita\_netflix

Command: src.main.java.parent.application (run)

# Documentation of API man

It is used for tokenization of user. When user is registered, the username and password is saved in API man. The realm is created, and API is made private. The API received can only be accessed using the tokens which is created using login of authorized users.

**Procedure of tokenization and making API private:**

Open API man and login

## Title 1: WAP API gateway Management

* Organizations -> My Organizations -> New Organization
* Enter Organization Name, Description and click create organization
* Click created organization -> Plans -> New Plan
* Enter plan name (default) -> Create Plan
* Click lock plan
* Go to Organizations -> My Organizations -> Click the organization that you created
* Click APIs -> New API
* Give API Name -> Create the API
* Go to Implementation -> Write the endpoint URL -> Select API type and Content type -> Save
* Go to Plans
* Sate whether you want to make plan public or private. If private, then create and choose plans that will make it private
* Default: Choose public
* One endpoint is defined and fact that is public or private click Publish
* Go to Endpoint -> The API gateway has generated URL
* (Copy paste in browser to see if it is working)

## Title 2: Keycloak generating token

* Login into keyclock realm: <http://localhost:8080/auth>
* Create Realm -> Enter realm name and create (Test-authentication)
* Goto Users -> Add user -> add Username (Anshul) -> Save
* Goto Credientials -> set password -> uncheck temporary -> Save
* Goto Test-authentication Realm -> Clients -> Create
* Write client id -> enable access grant
* Goto Realm Setting -> Click OpenID Endpoint Configuration
* Find link of token-endpoint
* (<http://localhost:8080/auth/realms/test-authentication/protocol/openid-connect/token>)
* **Goto postman (POST)**

In Body (x-222-form-urlencodded)

client\_id:test

grant\_type:password

username:anshul

password:anshul

* Copy token
* Put in <https://jwt.io> to decode the token

## Title 3: Make endpoint private from public

* Goto APIman -> Client -> Create new client
* Give name
* Goto contacts -> New Contract -> Select API -> Create Contract -> Register
* Goto Client -> My clients -> your client name -> APIs -> Click endpoint and see the new endpoint with apikey
* Now go to your API which is implemented through APIs -> My API -> Plans -> uncheck public

## Title 4: Get data from endpoint using tokens

* Goto API man welcome page
* In system Administrator -> Manage Plugins -> available plugins
* Select Keycloak OAuth Policy Plugin and install
* Now got Clients -> My clients -> select your client
* Goto Policy -> Add policy
* Select Keycloak OAuth Policy
* Require transport security -> false
* Realm (Goto keycloak -> Choose realm -> Realm Setting -> Click OpenID endpoint configuration -> Copy issuer (<http://localhost:8080/auth/realms/test-authentication>)
* For keycloak Realm Certificate -> Goto keycloak -> Choose realm -> Realm Setting -> Keys -> Click public key certificate of RS256
* Publish
* Reregister
* Check the API key again of endpoint

## Title 5: Connecting API with CORS

* Goto Keycloak
* Choose your relm
* Choose clients -> name (test)
* In Web origins: \*
* In valid direct uri: /index.html
* Goto APIman home/ welcome page
* Goto System Administrator -> Manage Plugins -> Available plugins -> CORS policy plugin -> Install
* Goto APIman
* Clients -> My client -> Policies -> Add policy -> CORS policy
* In Access-Control-Allow-Origin -> +item -> write \*
* Done
* In access control allow header -> + -> authorization
* In Terminate on CORS error: false
* Access control allow method -> + -> POST + -> GET
* Access-Control-Allow-Header -> authorization
* Move CORS policy before KeycloakAuth

**For creating token and getting Data from postman (Checking):**

**POST** http://localhost:8080/auth/realms/test-netflix/protocol/openid-connect/token

Parameters:

client\_id:test

grant\_type:password

username:testing

password:testing

Copy access tokenGraphical user interface, text, application, email

Description automatically generated

**GET** <https://localhost:8443/apiman-gateway/Netflix/movies/1.1?apikey=77fd0277-104c-4192-be6d-8f7a83c598b3>

Put the copied access token

The movie will be available

Graphical user interface, text, application, email

Description automatically generated

# Documentation of Front End:

Front end is done in react using CSS Tailwind and JSX libraries. All the necessary pages were developed (Login, Signup, Home)

TMDB API was used for the initial movies and testing.

The user when registers, their username and password is saved in REALM (Web Application Patterns) using axios POST method. When same user tries to login it is authenticated and token is generated which is again passed with the private API to fetch the movies in our local database using axios GET method.

# Documentation of Back End Node:

Firstly, the connection to mongodb is done (<http://localhost:8070/netflix>)

Movie definition model is defined in such a way that title, release date, category, movie director and externalID is saved.

The movie rating model is created for ratings.

## API created

**POST** <http://127.0.0.1:4500/movie/add-moviedef> : For adding movies in database

**GET** <http://127.0.0.1:4500/movie/get-movie/:category>: For getting movies according to the categories

**(** <http://127.0.0.1:4500/movie/get-movie/Action>

<http://127.0.0.1:4500/movie/get-movie/Adventure>

<http://127.0.0.1:4500/movie/get-movie/Comed>y )

**GET** <http://127.0.0.1:4500/movie/getrating/:userid> : Where all ratings are get according to user ID

**POST** <http://127.0.0.1:4500/movie/rating> : where ratings of the user will be posted in database

**GET** <http://127.0.0.1:4500/movie/get-movie/allrating> : where all ratings will be displayed

# Documentation Java Backend:

Connection was done with postgres SQL (jdbc:postgressql://localhost:5432/epita\_netflix)

All the necessary datamodel for creating and fetching data from SQL was created and configured.

Controllers were created with API’s for posting and getting data

## MovieController:

**POST** <http://localhost:8070/netflix/movies>: Posting movie to database

**GET** <http://localhost:8070/netflix/movies>: Getting movies from database

**GET** [http://localhost:8070/netflix/movies/{{id}}](http://localhost:8070/netflix/movies/%7b%7bid%7d%7d): Getting specific movie of id from database

**DELETE** [http://localhost:8070/netflix/movies/{{id}}](http://localhost:8070/netflix/movies/%7b%7bid%7d%7d): Deleting specific movie of id from database

## RoleController:

**POST** http://localhost:8070/netflix/roles: To store the specific user to determine either user is admin or not

**GET** <http://localhost:8070/netflix/roles>: To get all roles from database

**GET** [http://localhost:8070/netflix/roles/{{id}}](http://localhost:8070/netflix/roles/%7b%7bid%7d%7d): To get roles of specific user of id from database

## UserController:

**POST** <http://localhost:8070/netflix/users>: To store the user in database

**GET** <http://localhost:8070/netflix/users>: To get all the users from the database

**DELETE** [http://localhost:8070/netflix/users/{{name}}](http://localhost:8070/netflix/users/%7b%7bname%7d%7d): To delete specific user from database according to username