# WORKING INSTRUCTIONS

Thu Tran – Bocheng Peng

18 MAY 2025

NHL STENDEN UNIVERSITY OF APPLIED SCIENCE

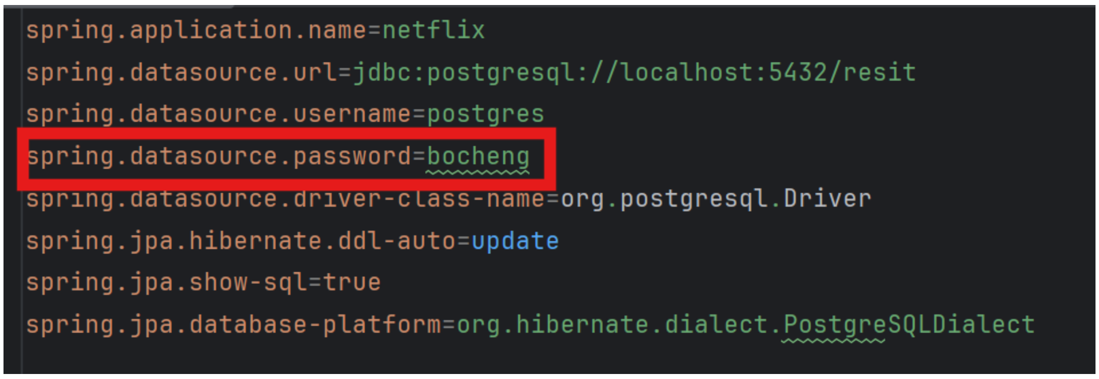
## Database Setup

First, you need to connect to the database; the API uses PostgreSQL

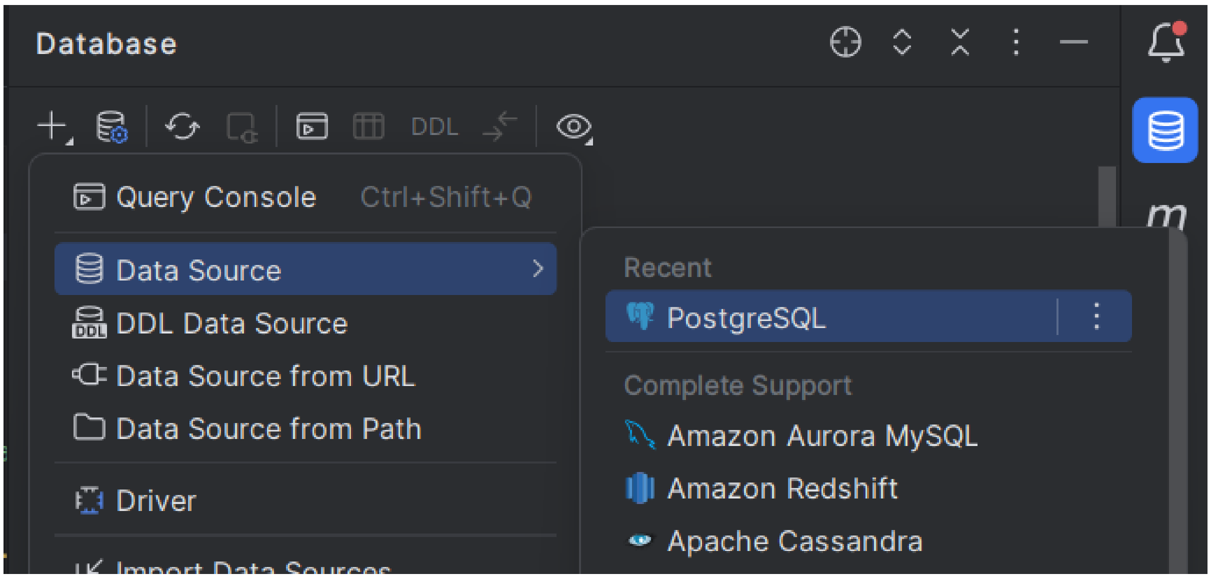
You can use Intelij (prefer version 2023/2024) to manage the database and the code at the same time.

First, go to file “**application.properties**” and change the password to **your own password** for database.

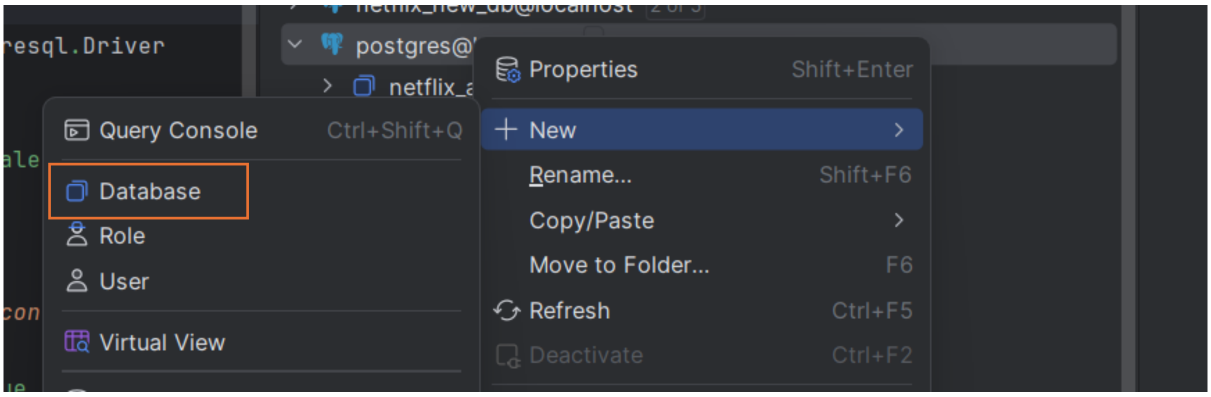
Navigate to the right corner of Inteliij, choose the database symbol, then connect to the PostgreSQL.



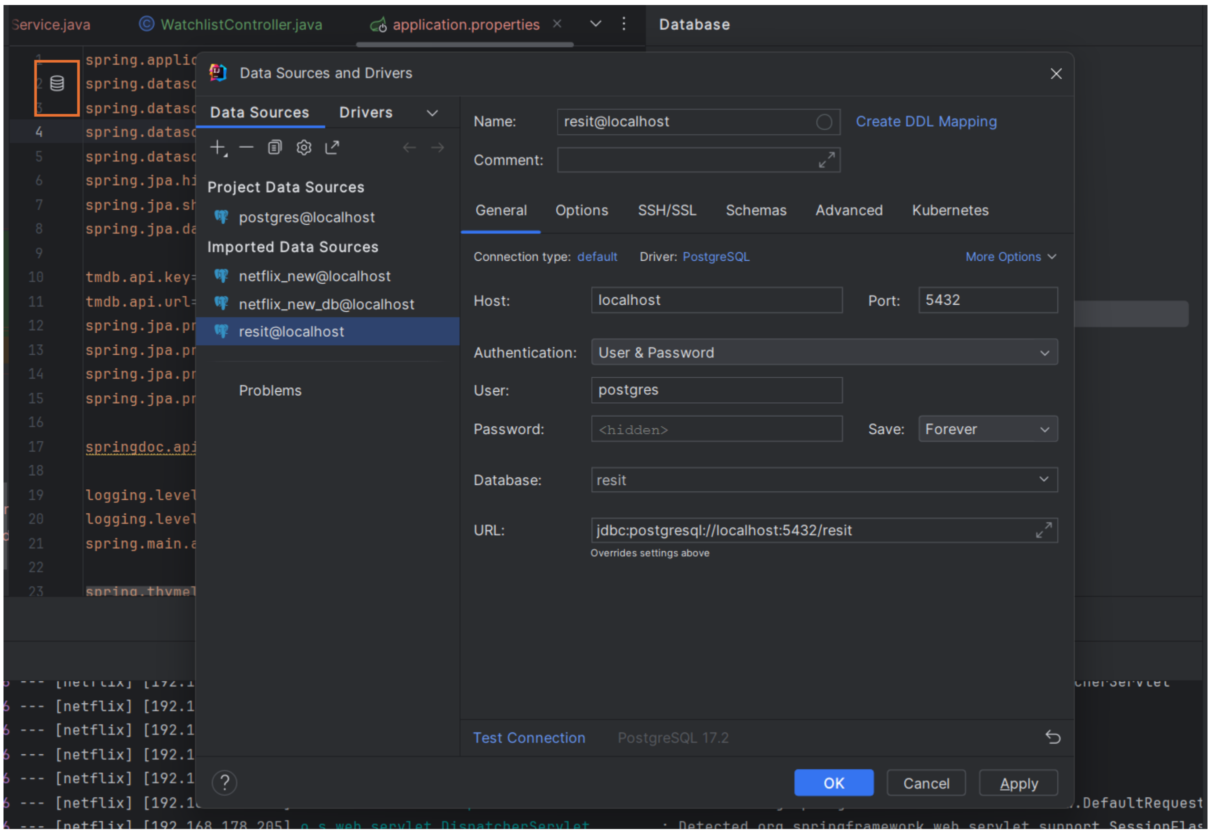
```
spring.application.name=netflix
spring.datasource.url=jdbc:postgresql://localhost:5432/resit
spring.datasource.username=postgres
spring.datasource.password=bocheng
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
```



After the connection, please create a database named “**resit**” by right-clicking at the admin database in your local machine, then click create database



After creation, you can connect the database to the application by opening file "application.properties" and click on the database symbol, then use your admin account to manage the setup



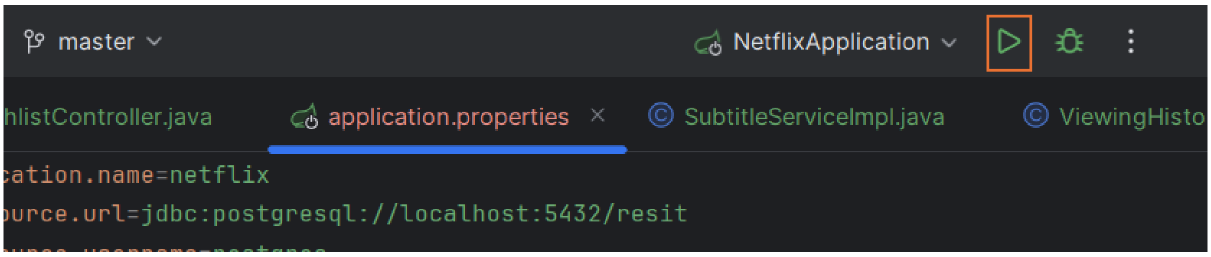
Name: resit@localhost
Comment:
Connection type: default Driver: PostgreSQL
Host: localhost Port: 5432
Authentication: User & Password
User: postgres
Password: <hidden> Save: Forever
Database: resit
URL: jdbc:postgresql://localhost:5432/resit
Overrides settings above

**Database name: resit**

**User: postgres**

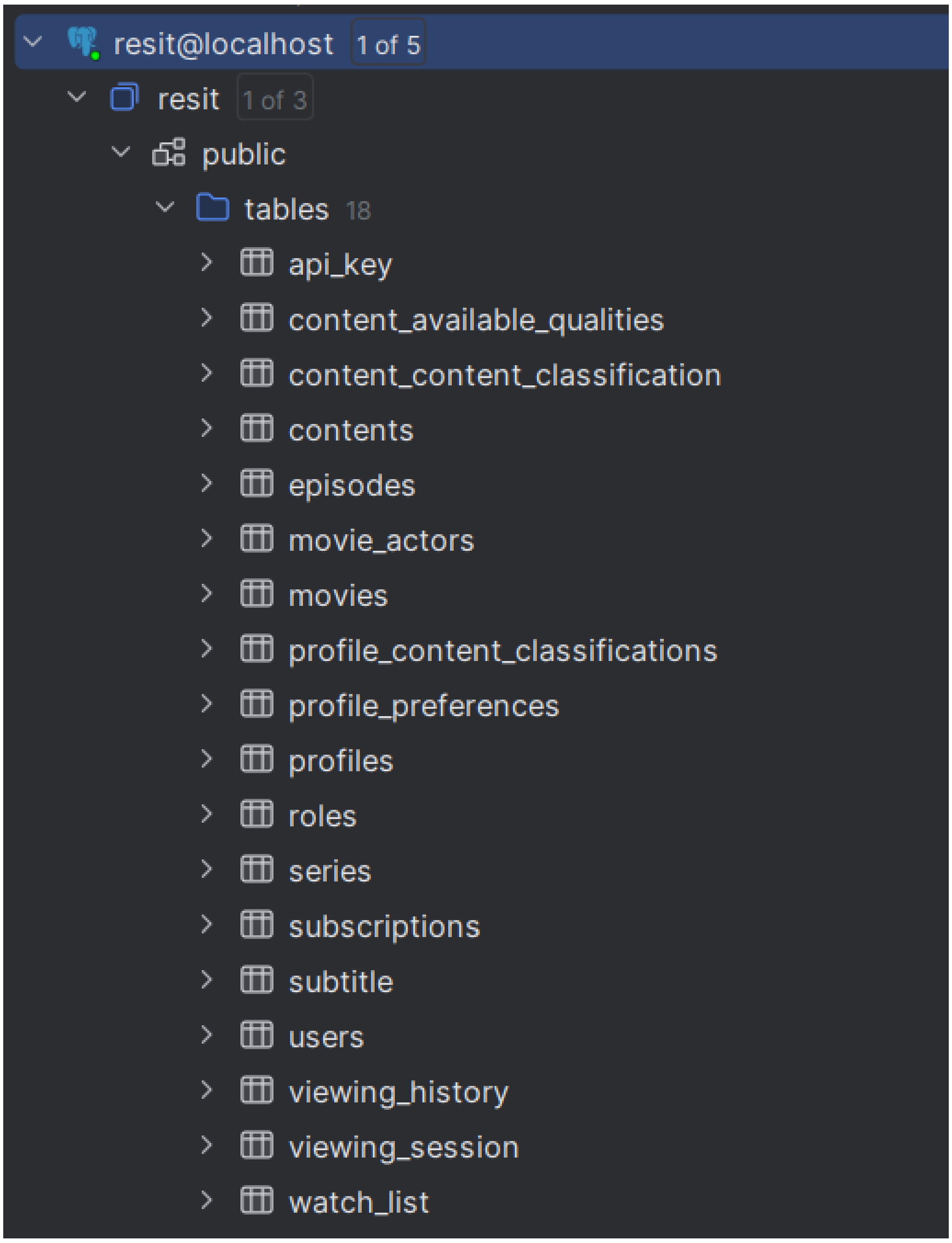
**Password: your log in passoword**

After the connection, **please run the application** by clicking on the "play button" on top of the nav bar.

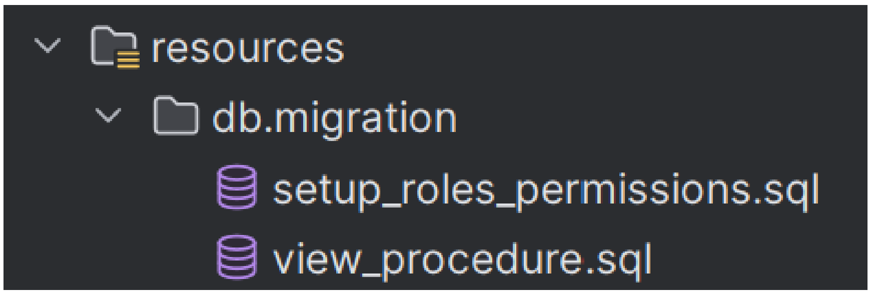


```
cation.name=netflix
ource.url=jdbc:postgresql://localhost:5432/resit
```

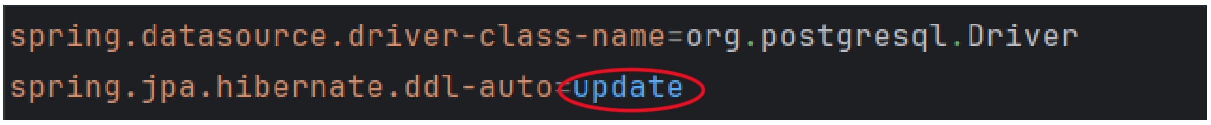
**Once the application runs successfully, the database will be created automatically.**



After the database is created, please navigate to the resources folder and run the **setup_roles_permissions.sql** to create users and the **view.sql** to create views, **stored_procedures.sql** for **stored procedures, and trigger.sql for trigger**.

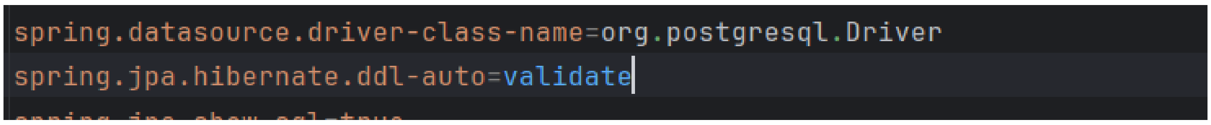


After the setup, for security reasons, you should change the DDL mode in the “application.properties”, because it’s not secure to allow the database architecture to be **updated** everytime the application start running.



```
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=update
```

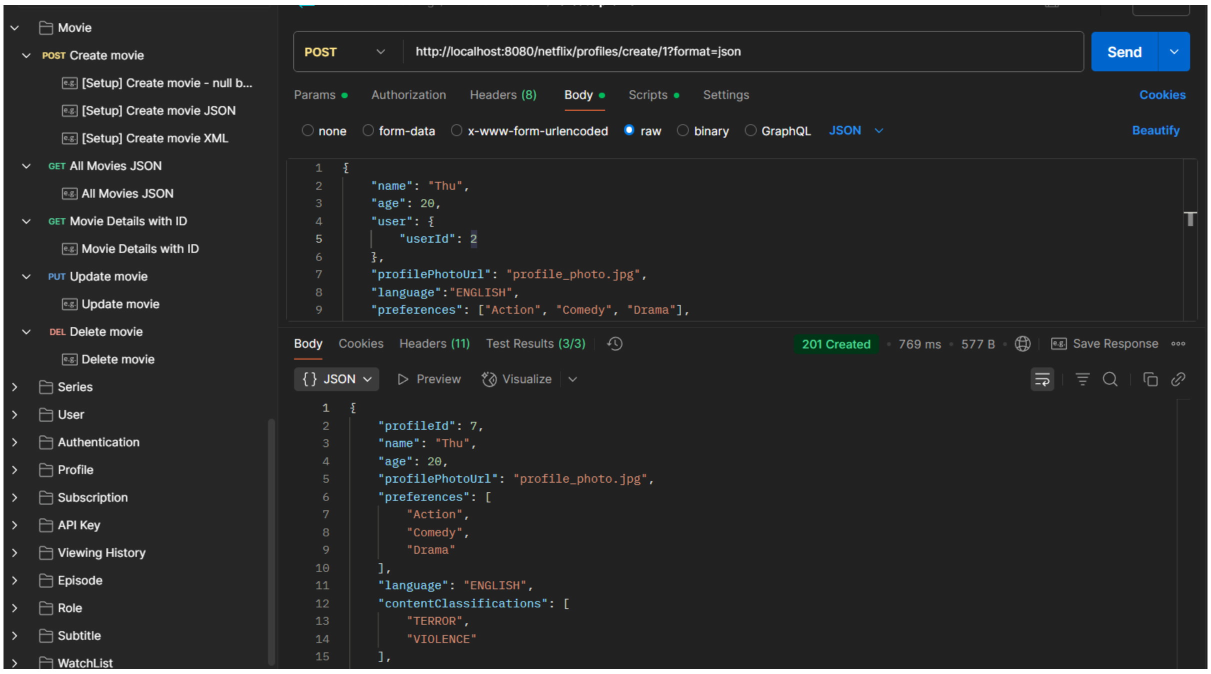
You should change from update to validate, so it will validate the schema and make no changes to the database.



```
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=validate
```

## API Endpoints Testing

Provided in the assessment is the file that you can put into Postman to test the application end-to-end, functional testing, and automated test scripts from Postman. If you change the database user, you will receive an error when you try to query the table that the user does not have access to.



```
POST http://localhost:8080/netflix/profiles/create/1?format=json
```

```
{
    "name": "Thu",
    "age": 20,
    "user": {
        "userId": 2
    },
    "profilePhotoUrl": "profile_photo.jpg",
    "language":"ENGLISH",
    "preferences": ["Action", "Comedy", "Drama"],
```

```
{
    "profileId": 7,
    "name": "Thu",
    "age": 20,
    "profilePhotoUrl": "profile_photo.jpg",
    "preferences": [
        "Action",
        "Comedy",
        "Drama"
    ],
    "language": "ENGLISH",
    "contentClassifications": [
        "TERROR",
        "VIOLENCE"
    ],
```

Every controller has at least 1 type of CRUD operations.

## API Validation

Every function in the service layer and mapping in the controller layer is handled by a try-catch to catch all the possible exceptions. The DTOs are also validated by using annotations such as @NotNull, @Min, etc. to ensure the input and output are validated, so it will be safe for deployment purposes.

Aside from that, all the functions that made changes to the database, such as update, create, delete, have a @Transactional annotation, so all the transactions will be recorded, admin can recover the database in case of rollback or collapse.

## System Design

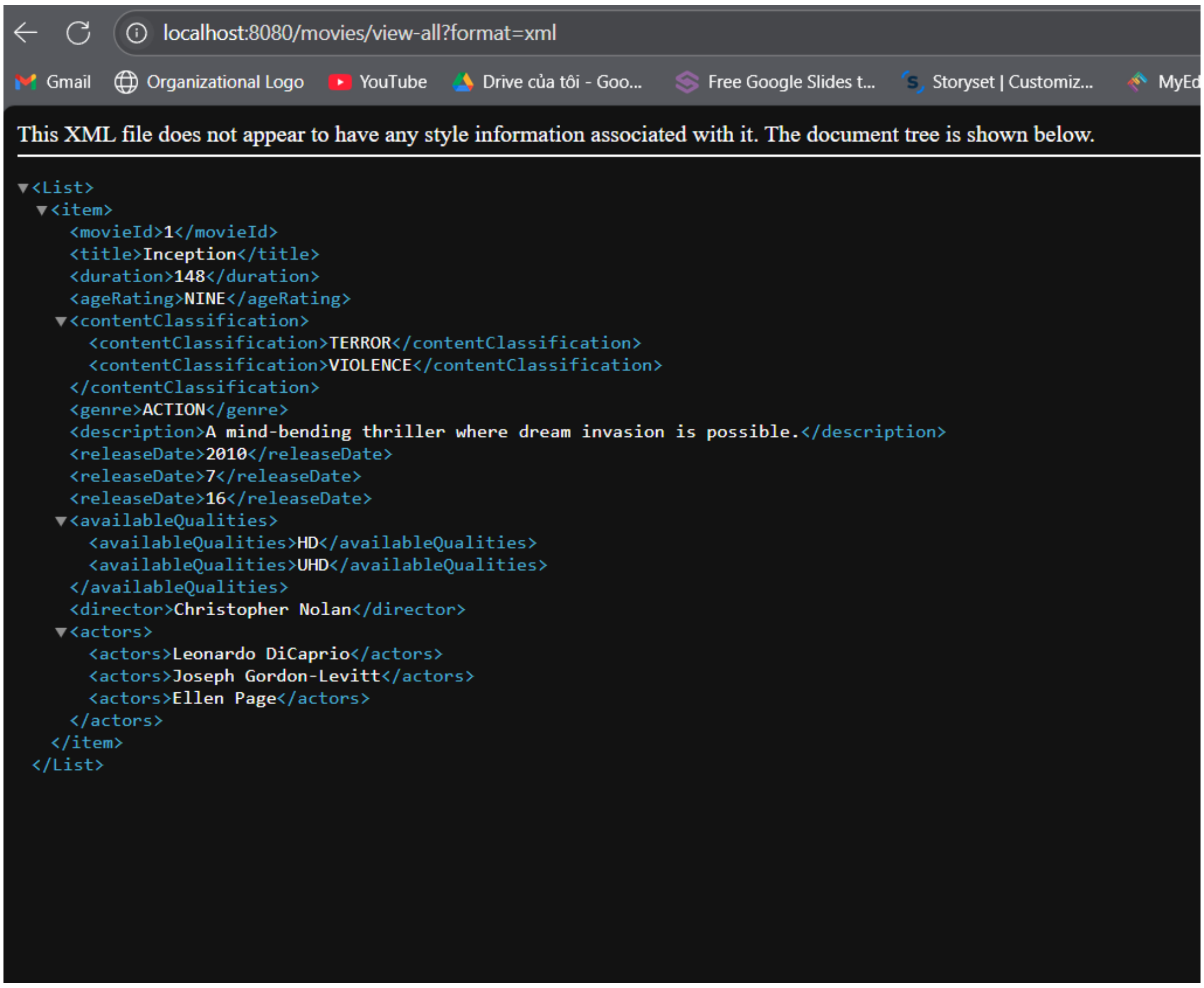
The OOP concept is implemented in the system design to ensure that the design is logical and can support future expansion. For example, “Content” entity is an abstract class, “Movie” and “Series” are inherited from this class. In the future, if we have other content such as “Sitcom”, we can easily extend from the Content class.

For the Service layer, the service interface provides a clear view of which functions to be implemented, so it’s easier to maintain the serviceimpl classes.

We added several enum and classes to cover all the cases from the use case provided.

## Data Communication

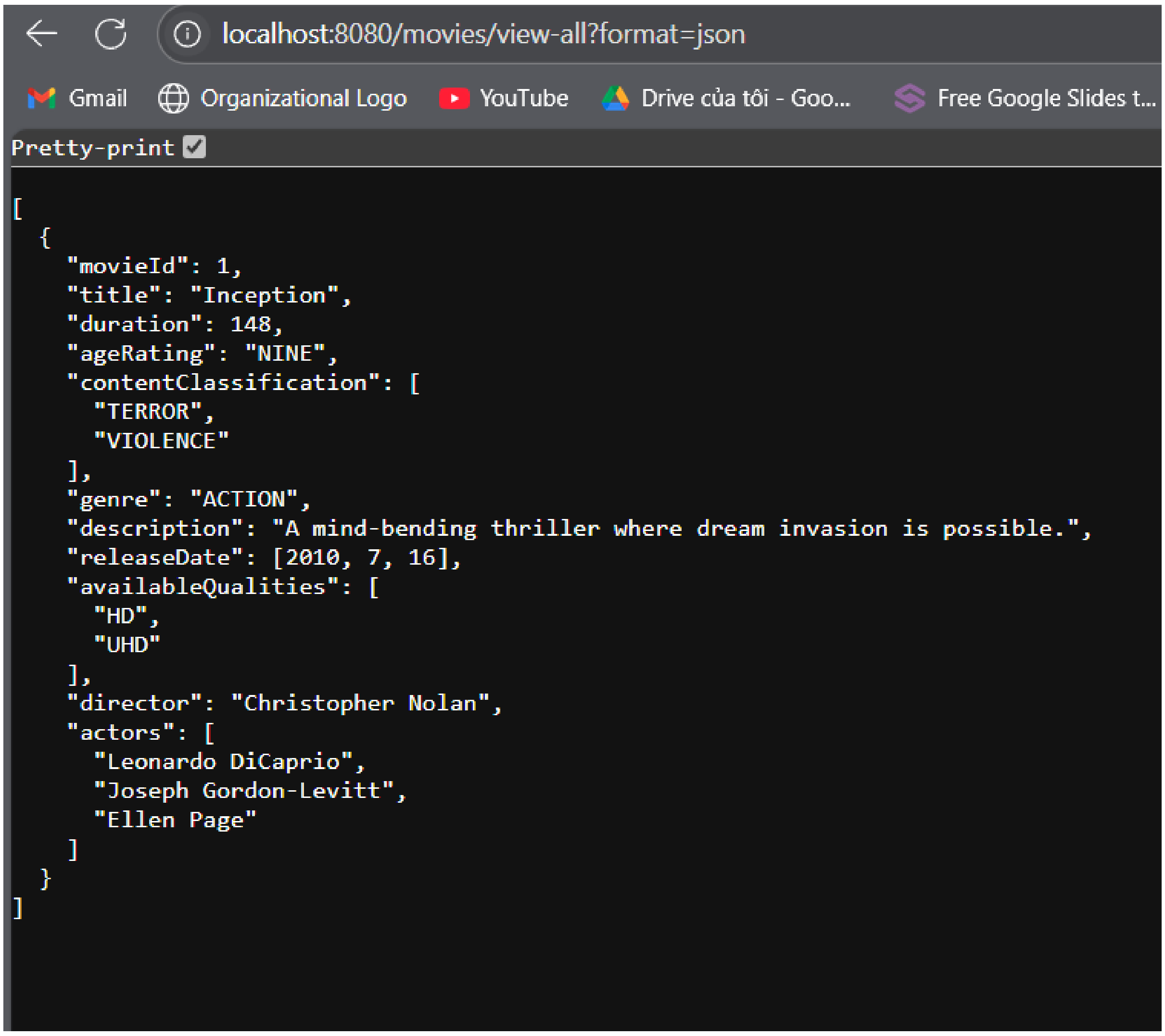
The API can return the response in 2 formats which are JSON and XML. When you make a request, you can add ?format= at the end of the request to indicate which format you would like to see from the response. (This works for all the responses)



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<List>
  ▼<item>
     <movieId>1</movieId>
     <title>Inception</title>
     <duration>148</duration>
     <ageRating>NINE</ageRating>
   ▼<contentClassification>
       <contentClassification>TERROR</contentClassification>
       <contentClassification>VIOLENCE</contentClassification>
     </contentClassification>
     <genre>ACTION</genre>
     <description>A mind-bending thriller where dream invasion is possible.</description>
     <releaseDate>2010</releaseDate>
     <releaseDate>7</releaseDate>
     <releaseDate>16</releaseDate>
   ▼<availableQualities>
       <availableQualities>HD</availableQualities>
       <availableQualities>UHD</availableQualities>
     </availableQualities>
     <director>Christopher Nolan</director>
   ▼<actors>
       <actors>Leonardo DiCaprio</actors>
       <actors>Joseph Gordon-Levitt</actors>
       <actors>Ellen Page</actors>
     </actors>
   </item>
 </List>
```

XML Response



```
[
  {
    "movieId": 1,
    "title": "Inception",
    "duration": 148,
    "ageRating": "NINE",
    "contentClassification": [
      "TERROR",
      "VIOLENCE"
    ],
    "genre": "ACTION",
    "description": "A mind-bending thriller where dream invasion is possible.",
    "releaseDate": [2010, 7, 16],
    "availableQualities": [
      "HD",
      "UHD"
    ],
    "director": "Christopher Nolan",
    "actors": [
      "Leonardo DiCaprio",
      "Joseph Gordon-Levitt",
      "Ellen Page"
    ]
  }
]
```

JSON Response

By default, the response will be in JSON format.

Each controller has all CRUD operations. For some controllers, it will have more than 1 operation for a type, such as Post, Get or Update. This implementation is to cover some cases, such as account activation, password reset, etc.

## Authentication and Authorisation

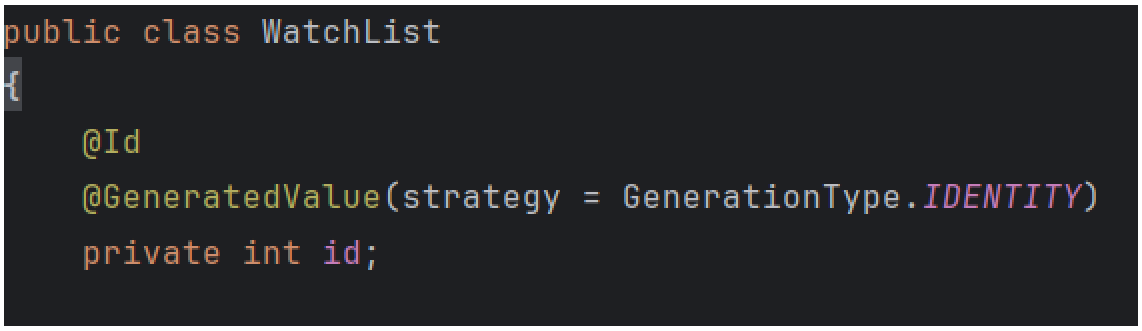
The token, such as an activation token and a reset password token, has an expiration time and will be generated randomly.

The Role Controller allows the creation of a new user for the database.

The reset token is not returned as a response for security purposes. It should only be sent via email in the actual product, so in this case, please go to the database and copy the reset token to reset the password.

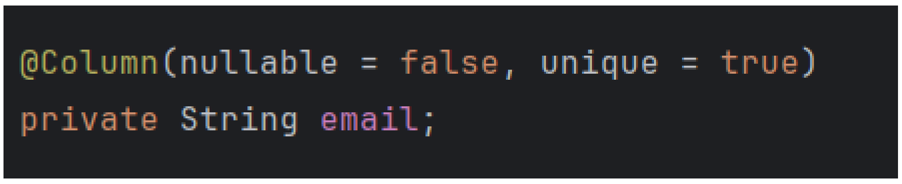
## Database Integrity

Tables in the database are provided with referential integrity. The following constraints are used: Unique, Check, Index, Primary and Foreign Key.



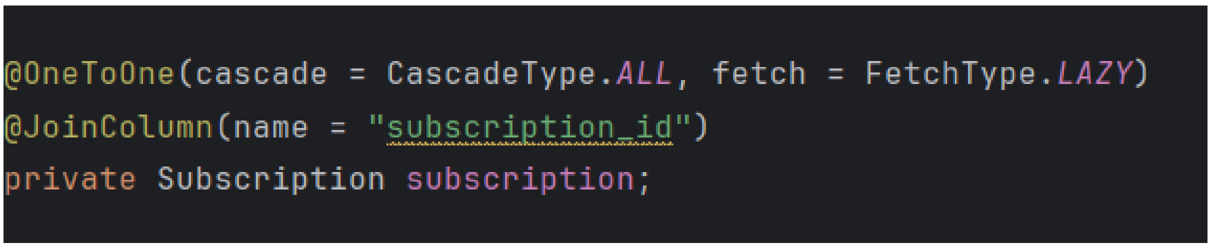
```java
public class WatchList
{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
```

Primary key: each entity has an ID



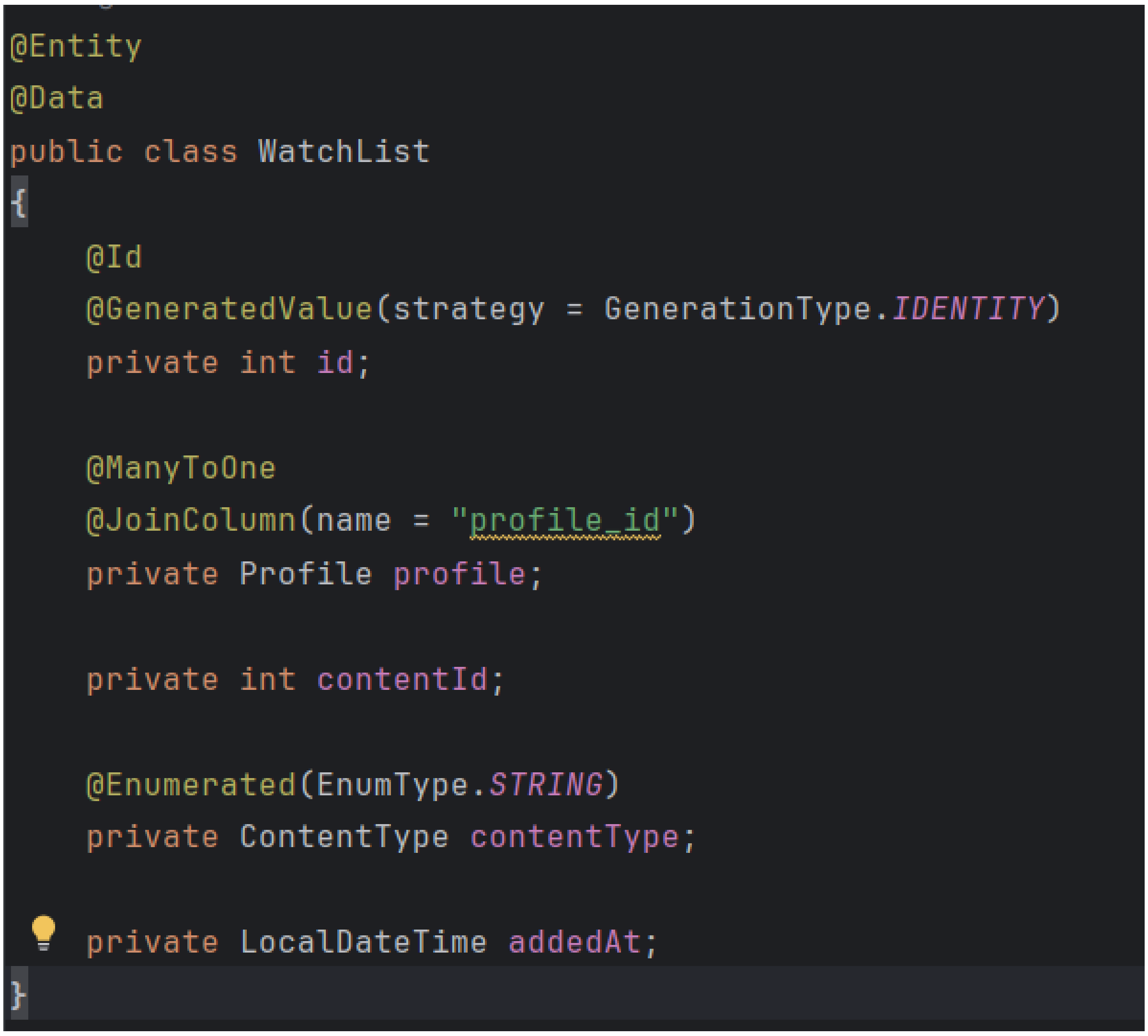
```java
@Column(nullable = false, unique = true)
private String email;
```

Unique check



```java
@OneToOne(cascade = CascadeType.ALL, fetch = FetchType.LAZY)
@JoinColumn(name = "subscription_id")
private Subscription subscription;
```

Foreign Key: for example, fk of User entity is subscriptionId

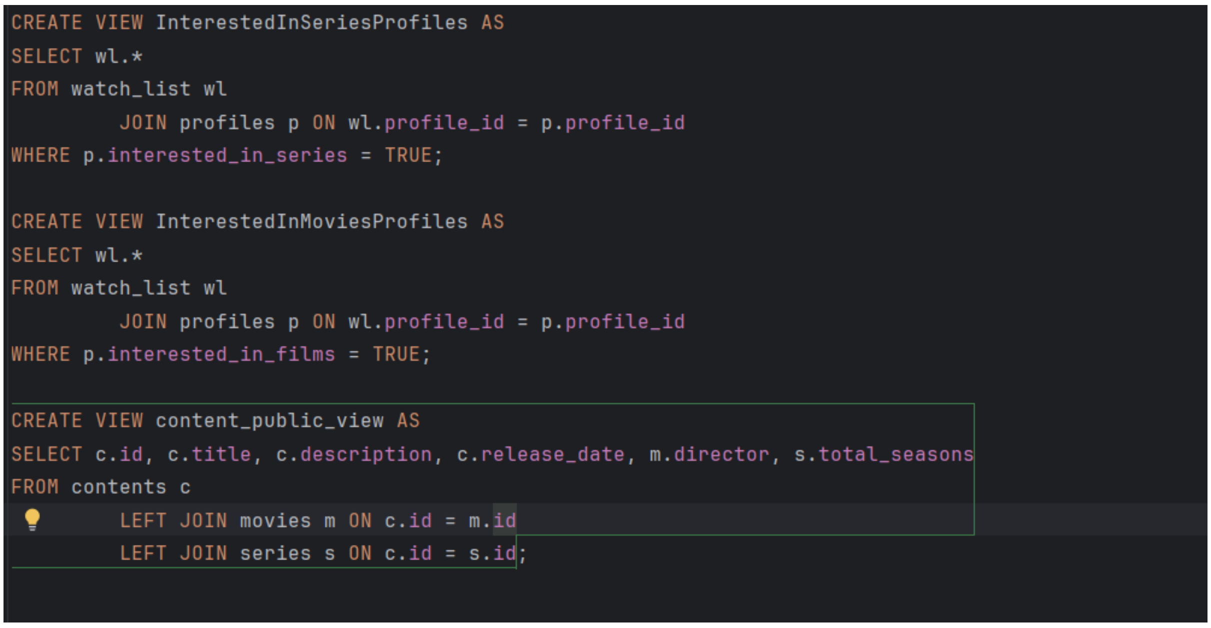


```
@Entity
@Data
public class WatchList
{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne
    @JoinColumn(name = "profile_id")
    private Profile profile;

    private int contentId;

    @Enumerated(EnumType.STRING)
    private ContentType contentType;

    private LocalDateTime addedAt;
}
```

Index: contentId in WatchList to enhance the query performance for frequently searched columns

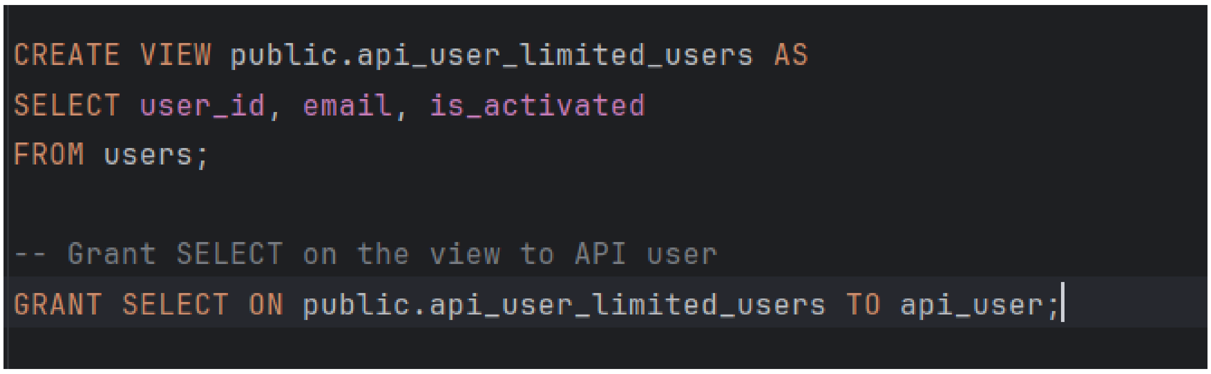
## View

There are some views created to display specific information such as to show only profiles that have interest in films/series, or to show the content publicly



```sql
CREATE VIEW InterestedInSeriesProfiles AS
SELECT wl.*
FROM watch_list wl
         JOIN profiles p ON wl.profile_id = p.profile_id
WHERE p.interested_in_series = TRUE;

CREATE VIEW InterestedInMoviesProfiles AS
SELECT wl.*
FROM watch_list wl
         JOIN profiles p ON wl.profile_id = p.profile_id
WHERE p.interested_in_films = TRUE;

CREATE VIEW content_public_view AS
SELECT c.id, c.title, c.description, c.release_date, m.director, s.total_seasons
FROM contents c
         LEFT JOIN movies m ON c.id = m.id
         LEFT JOIN series s ON c.id = s.id;
```

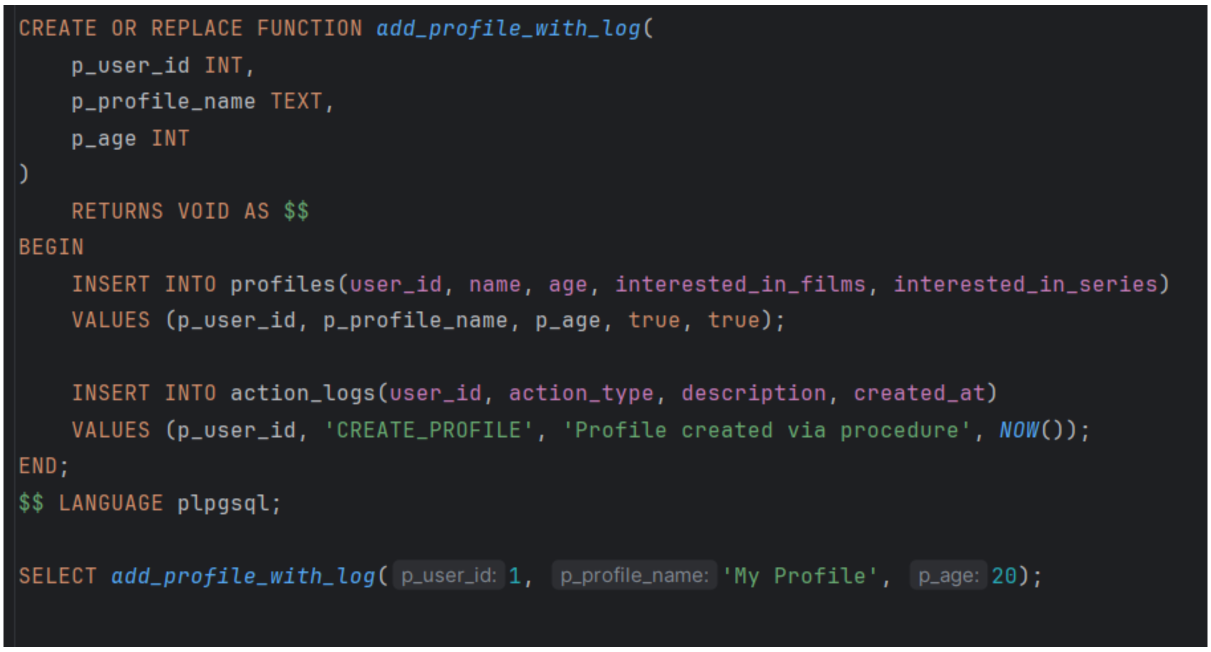
There is a view for API user to get access to the data but in a restricted way



```sql
CREATE VIEW public.api_user_limited_users AS
SELECT user_id, email, is_activated
FROM users;

-- Grant SELECT on the view to API user
GRANT SELECT ON public.api_user_limited_users TO api_user;
```

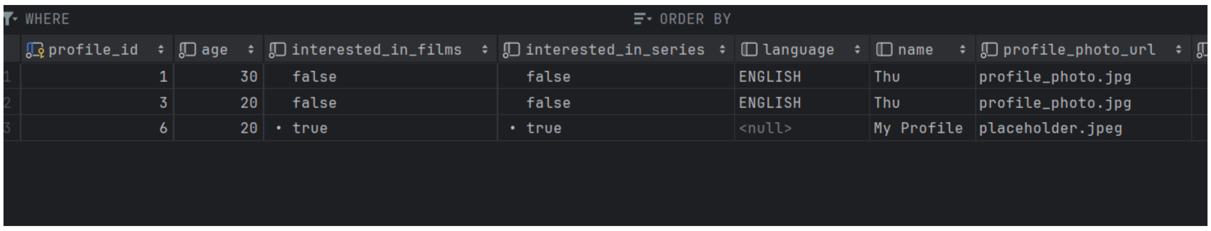
## Stored Procedures

There are several stored procedures that can be called to create users/ profiles or watchlist



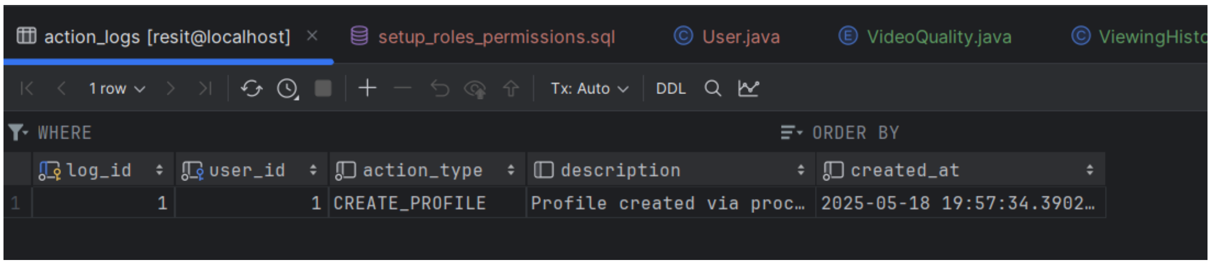
```sql
CREATE OR REPLACE FUNCTION add_profile_with_log(
    p_user_id INT,
    p_profile_name TEXT,
    p_age INT
)
    RETURNS VOID AS $$
BEGIN
    INSERT INTO profiles(user_id, name, age, interested_in_films, interested_in_series)
    VALUES (p_user_id, p_profile_name, p_age, true, true);

    INSERT INTO action_logs(user_id, action_type, description, created_at)
    VALUES (p_user_id, 'CREATE_PROFILE', 'Profile created via procedure', NOW());
END;
$$ LANGUAGE plpgsql;

SELECT add_profile_with_log( p_user_id: 1,  p_profile_name: 'My Profile',  p_age: 20);
```

The SELECT query is an example of how to use the stored procedures. It will insert profile data directly into the profile table (as can be seen, profile name, userId and age are the same as in the query)



```
WHERE                                   ORDER BY
   profile_id  age  interested_in_films  interested_in_series  language  name        profile_photo_url
1           1   30    false                false                 ENGLISH   Thu         profile_photo.jpg
2           3   20    false                false                 ENGLISH   Thu         profile_photo.jpg
3           6   20  • true               • true                  <null>    My Profile  placeholder.jpeg
```

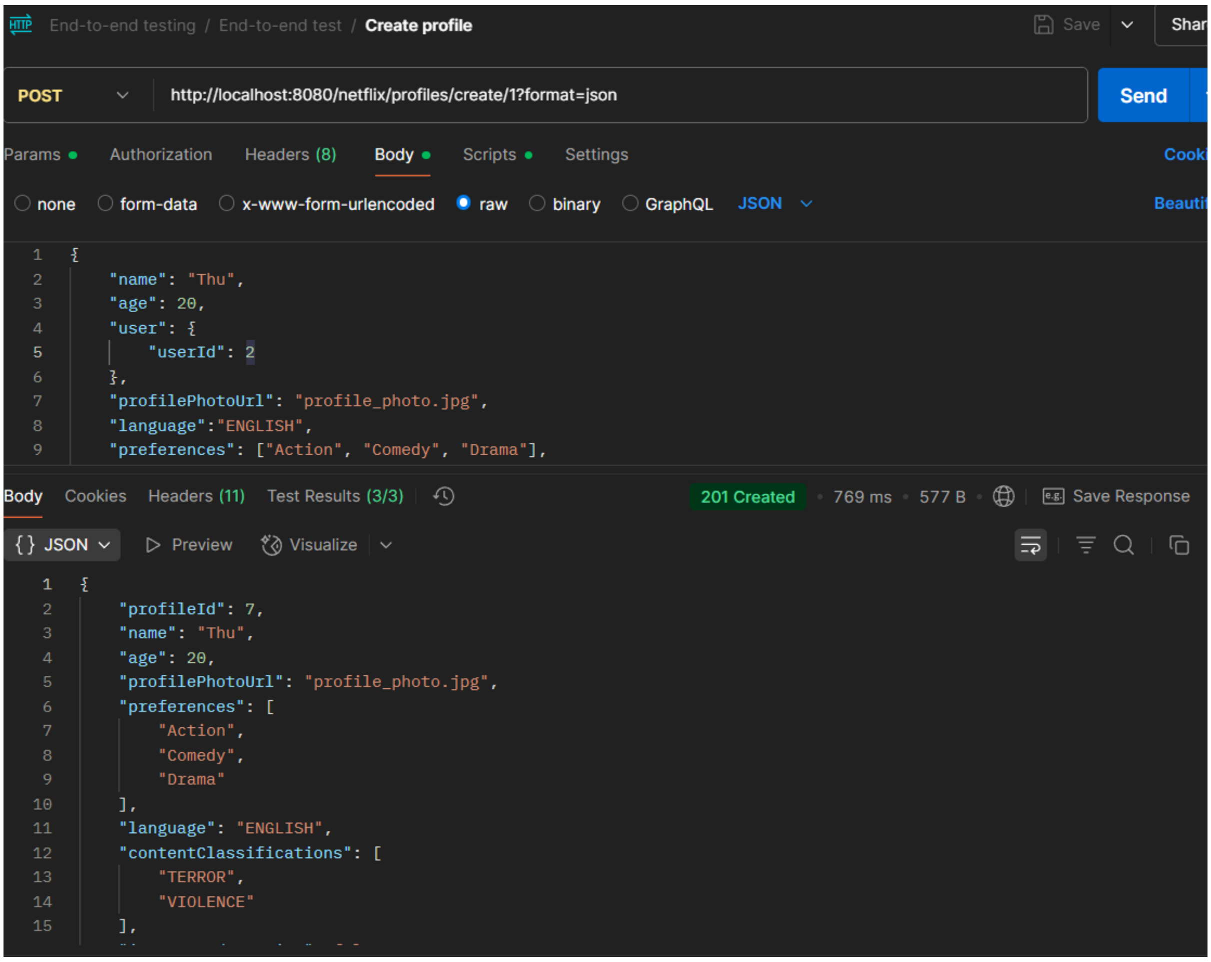
It also insert into the action logs table to keep track of the database transaction



```
WHERE                                        ORDER BY
   log_id  user_id  action_type     description                   created_at
1       1        1  CREATE_PROFILE  Profile created via proc…     2025-05-18 19:57:34.3902…
```

## Trigger

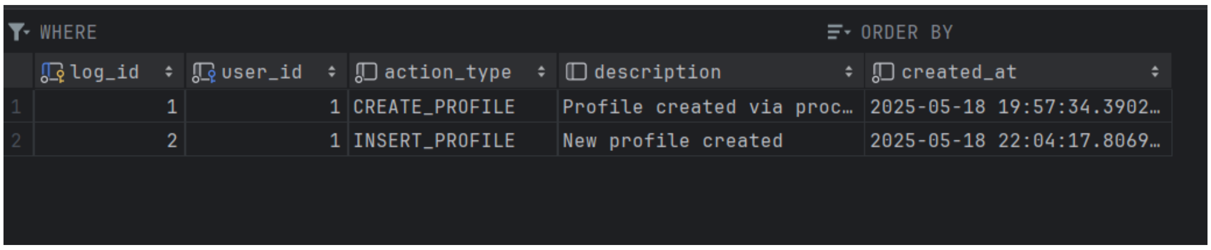
Whenever the profile is added or deleted, it will be recorded in the action logs by the trg_log_profile_activity



POST http://localhost:8080/netflix/profiles/create/1?format=json

```
{
    "name": "Thu",
    "age": 20,
    "user": {
        "userId": 2
    },
    "profilePhotoUrl": "profile_photo.jpg",
    "language":"ENGLISH",
    "preferences": ["Action", "Comedy", "Drama"],
```

```
{
    "profileId": 7,
    "name": "Thu",
    "age": 20,
    "profilePhotoUrl": "profile_photo.jpg",
    "preferences": [
        "Action",
        "Comedy",
        "Drama"
    ],
    "language": "ENGLISH",
    "contentClassifications": [
        "TERROR",
        "VIOLENCE"
    ],
```

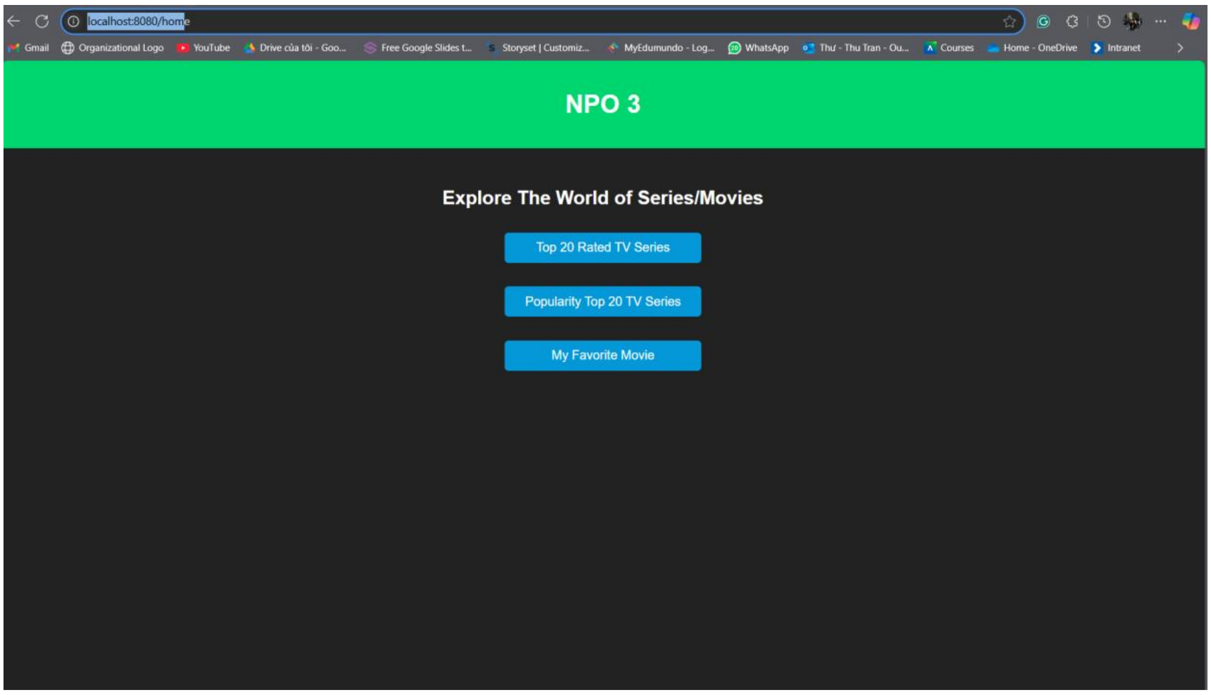


WHERE
ORDER BY

| | log_id | user_id | action_type | description | created_at |
|---|---|---|---|---|---|
| 1 | 1 | 1 | CREATE_PROFILE | Profile created via proc… | 2025-05-18 19:57:34.3902… |
| 2 | 2 | 1 | INSERT_PROFILE | New profile created | 2025-05-18 22:04:17.8069… |

## External API Fetching

When the application is run, you can navigate to http://localhost:8080/home to access the front-end we created to display the movies and series fetched by TMDB API.



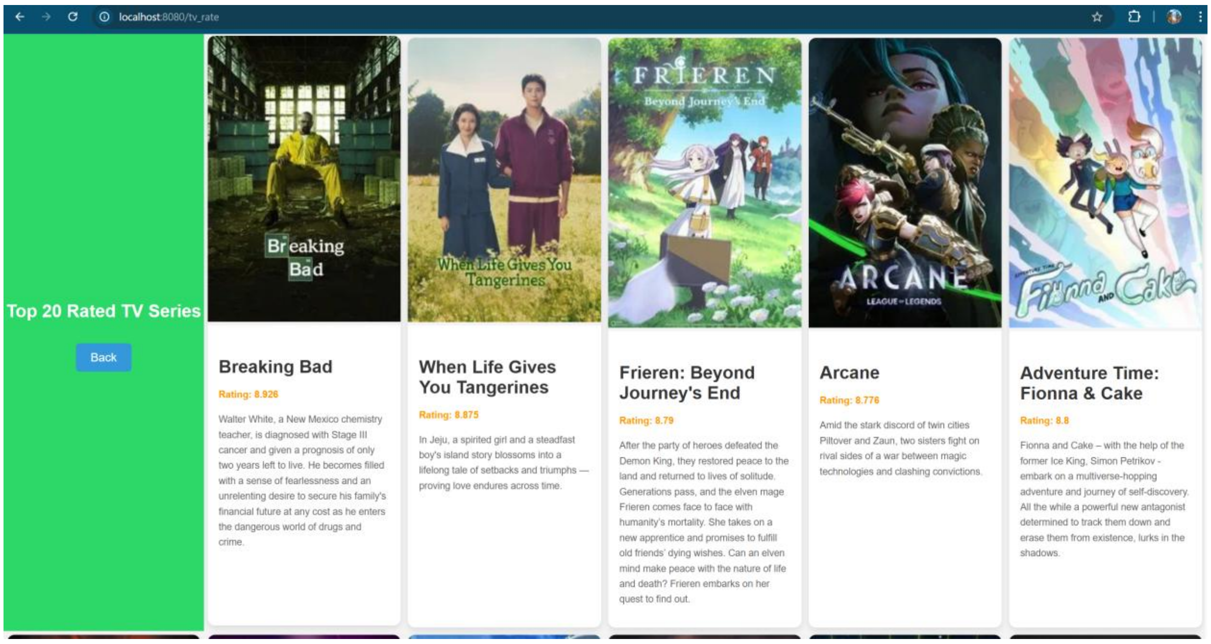
NPO 3

Explore The World of Series/Movies

Top 20 Rated TV Series

Popularity Top 20 TV Series

My Favorite Movie



Top 20 Rated TV Series

Back

Breaking Bad

Rating: 8.926

Walter White, a New Mexico chemistry teacher, is diagnosed with Stage III cancer and given a prognosis of only two years left to live. He becomes filled with a sense of fearlessness and an unrelenting desire to secure his family's financial future at any cost as he enters the dangerous world of drugs and crime.

When Life Gives You Tangerines

Rating: 8.875

In Jeju, a spirited girl and a steadfast boy's island story blossoms into a lifelong tale of setbacks and triumphs — proving love endures across time.

Frieren: Beyond Journey's End

Rating: 8.79

After the party of heroes defeated the Demon King, they restored peace to the land and returned to lives of solitude. Generations pass, and the elven mage Frieren comes face to face with humanity's mortality. She takes on a new apprentice and promises to fulfill old friends' dying wishes. Can an elven mind make peace with the nature of life and death? Frieren embarks on her quest to find out.

Arcane

Rating: 8.776

Amid the stark discord of twin cities Piltover and Zaun, two sisters fight on rival sides of a war between magic technologies and clashing convictions.

Adventure Time: Fionna & Cake

Rating: 8.8

Fionna and Cake – with the help of the former Ice King, Simon Petrikov - embark on a multiverse-hopping adventure and journey of self-discovery. All the while a powerful new antagonist determined to track them down and erase them from existence, lurks in the shadows.