# General information

* Cardiff student number: st20315692
* VUM student number: 2218036
* Github link to source code (includes this document): <https://github.com/IliyanManolov/AdvancedProgramming_Assignment>

# Table of Content

* Overview………………………………………………………………………………………………1
* Prerequisites…………………………………………………………………………………………2
* Bundling….……………………………………………………………………………………………2
* Usage……………………………………………………………………………………………………3
* Security.. ……………………………………………………………………………………………..10
* Observability………………………………………………………………………………………..10
* Testing Coverage………………………………………………………………………………….11
* Database Structure………………………………………………………………………………12

# Overview

The application aims to mimic the functionalities of IMDB – browsing movies/shows by genre, actors, ratings, etc.

The project utilizes different languages and frameworks for its backend and frontend. The entire techstack is as follows:

* Frontend – React using Tailwind CSS for majority of styling
* Backend – ASP.NET (C#) with additional services for observability (details further down)
* Database – Local instance of Microsoft SQL Server
* Local development/deployment – Docker
* Observability – OpenSearch

# Prerequisites

Setting up the project requires having a local instance of Microsoft SQL Server installed on your machine or having access to such. The backend infrastructure is mostly setup and sample-configuration files have been provided with the project as well as the necessary migrations. You will need to replace the server instance, user and password to your actual ones in order to connect the backend container to your SQL server instance. Furthermore, upon first installation it is recommended, although completely optional, to setup your OpenSearch dashboard with the provided imports in the project by following the detailed instructions provided.

Another requirement for running the project is having WSL installed and configured if you are running a Windows machine. Furthermore, if the application does not start from the Visual Studio Solution navigate to the root project folder and run the following command: “docker-compose -f .\Backend\IMDB\docker-compose.yml up”.

The target audience for the project UI are monitors with resolution of 2K but the design is responsive down to 1080p.

When initially creating a first account it is required to manually edit it in the database to give it an administrator role. With the following:

UPDATE imdb.users

SET [role] = 'Administrator'

WHERE username = 'MyAdminAccount'

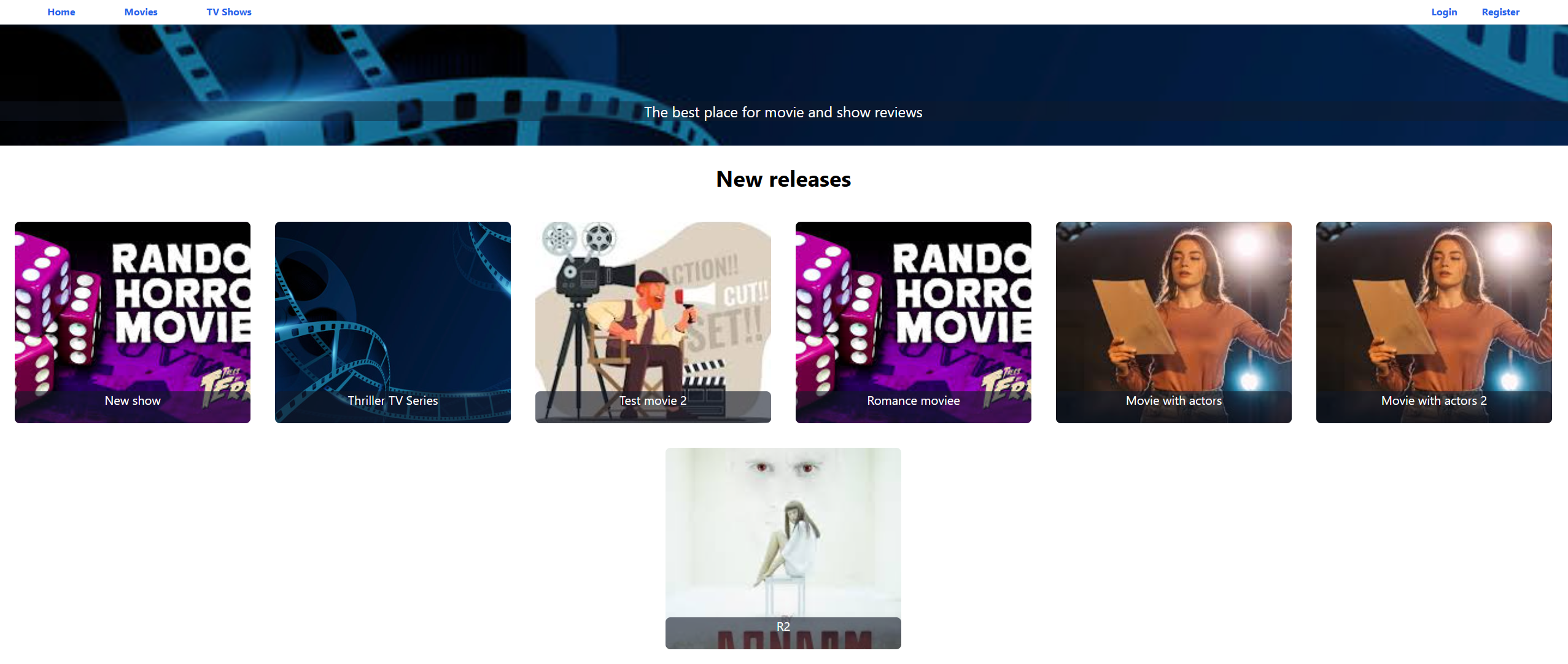
# Bundling

The project is bundled and run using a docker-compose project. It creates 2 major containers – API (.NET application acting as Backend) running on port 8080 and a UI (React application) running on port 3000 (same port as non-container react applications by default). Additionally, the usage of OpenSearch for logs collecting/monitoring requires 3 additional containers – 1 for the dashboard and 2 for the nodes. They are configured and ready to run as-is, however in case local password needs to be changed it must be done in the .env file.

In order to run the application, you need to copy the “Sample-Config” folder and rename it to “Config”. This is done to ensure that no local secrets are spilled from the config and setting up on a new machine can be as fast as possible. The only things that need to be changed are the Database, Data Source (Server), User Id and password for the database.

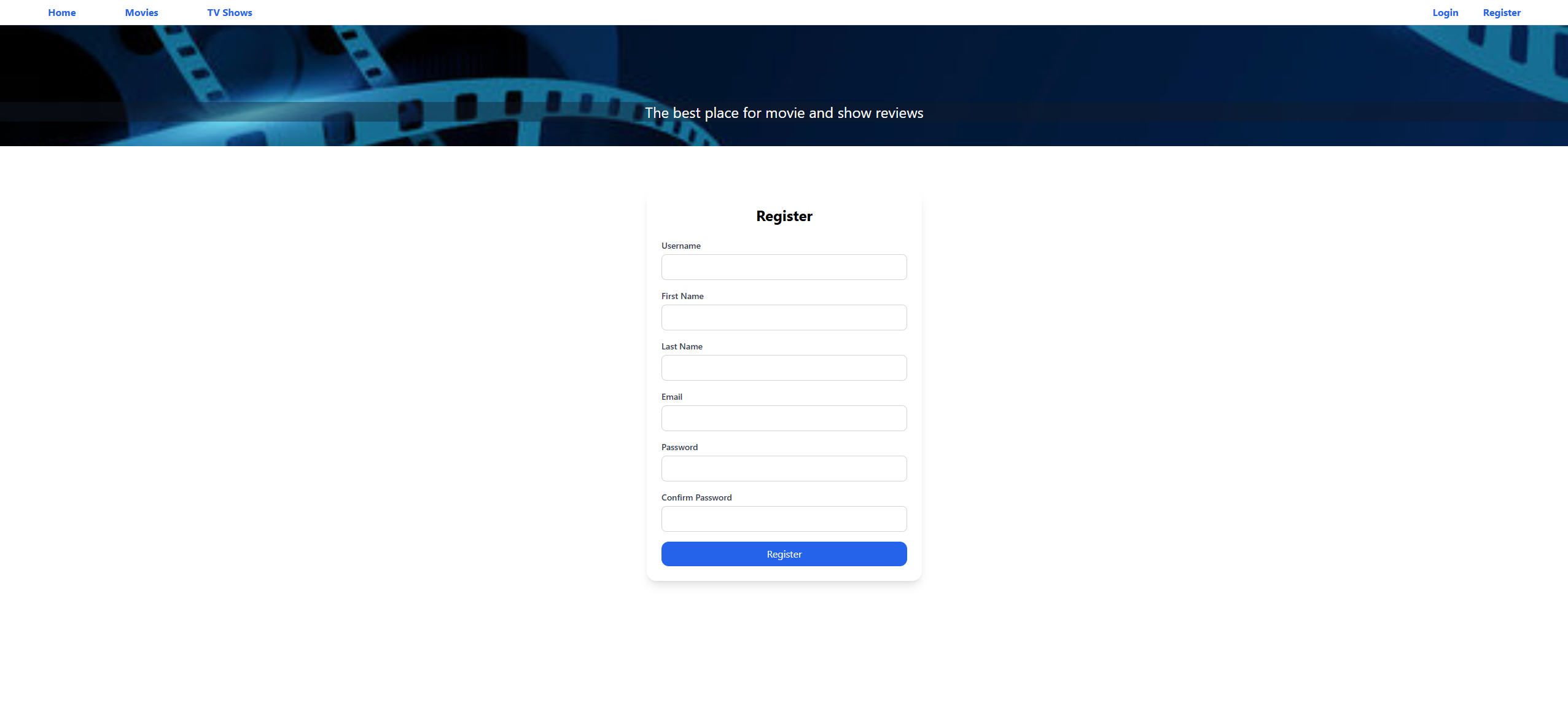
# Usage

Upon opening the home page, the user is met with a Navigation bar, a banner for the site and a list of “New releases” which showcases the latest 10 releases (5 Movies and 5 Shows) using their Poster image and title partially overlapping with it. The titles are truncated to 20 symbols to keep inside the image box. In order to still allow users to see the actual title hovering over the image reveals it as a hint. Each displayed media can be clicked on which redirects to the user to the details page for it.

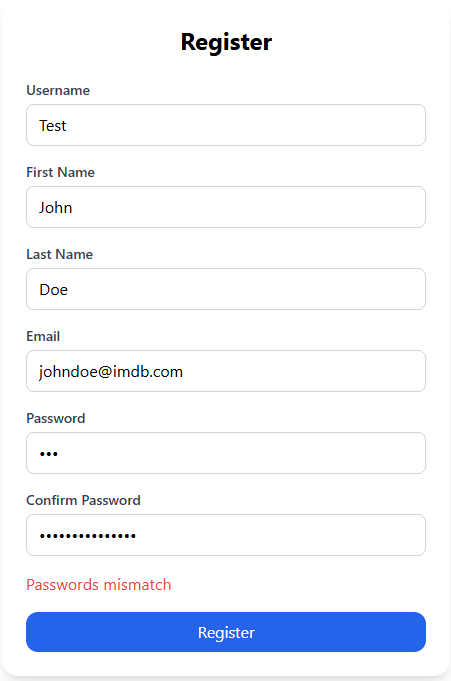


Examining the Navigation Bar closer reveals 5 navigation items for users that have not yet logged into the system. “Home” button which brings them to the initial screen with new releases, “Movies” and “TV shows” which bring the user to the dedicated Movies or Shows display page which we will examine further in the documentation. Last but not least, there are the “Login” and “Register” buttons which should be self-explanatory.

Next up we will examine the “Register” page which contains a singular form with Username, First and Last name, Email, Password and Confirm Password fields.

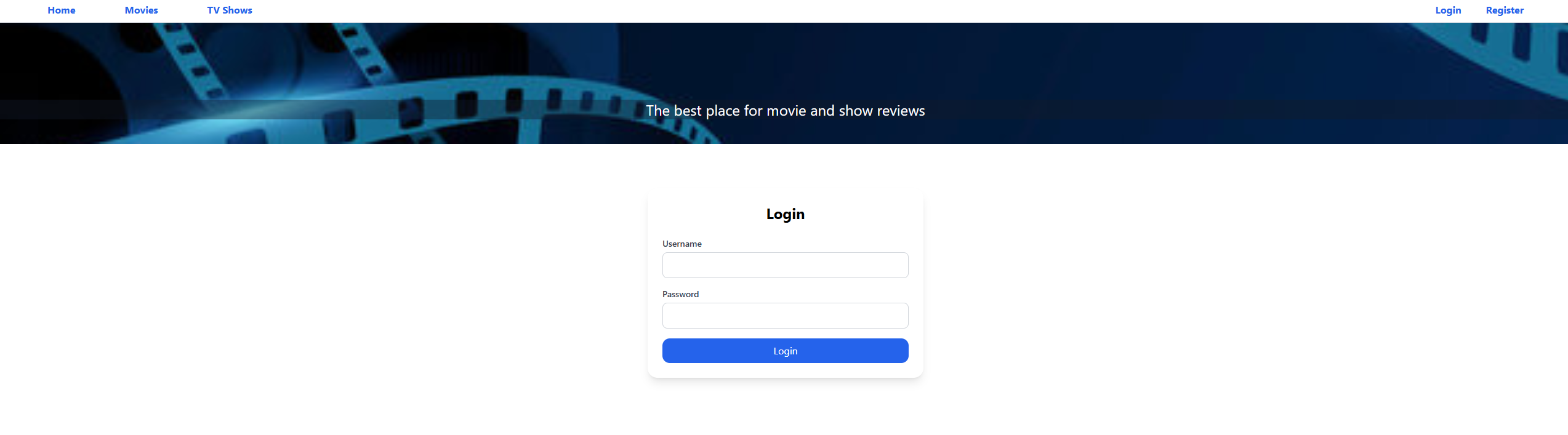


Upon submitting the form if a validation error is returned it is displayed below the form as seen in the picture below.

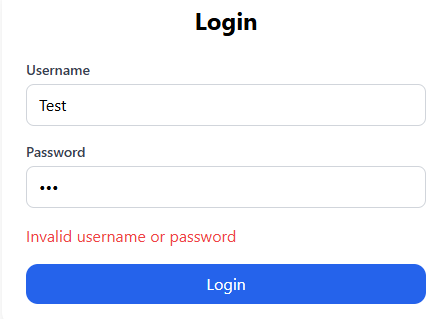


The current validations done are for passwords matching, email being unused and UserName (case sensitive) being unused. Upon a successful registration the user will be automatically redirected to the Login screen after 2 seconds.

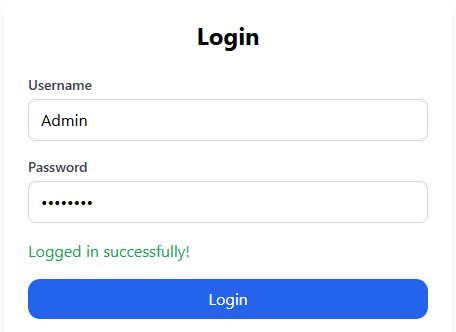
The login screen itself, once again, consists of a simple form with 2 fields – UserName and Password.



Upon an invalid username/password a validation error is returned and displayed. However, due to security concerns only a generic “Invalid username or password” error is showcased to the user. The backend application logs the exact reason (e.g. invalid password, deleted user, non-existing user) to the Observability module in order to not allow brute forcing for finding users.



Upon a successful login the user will be notified and redirected to the Home screen after 2 seconds.

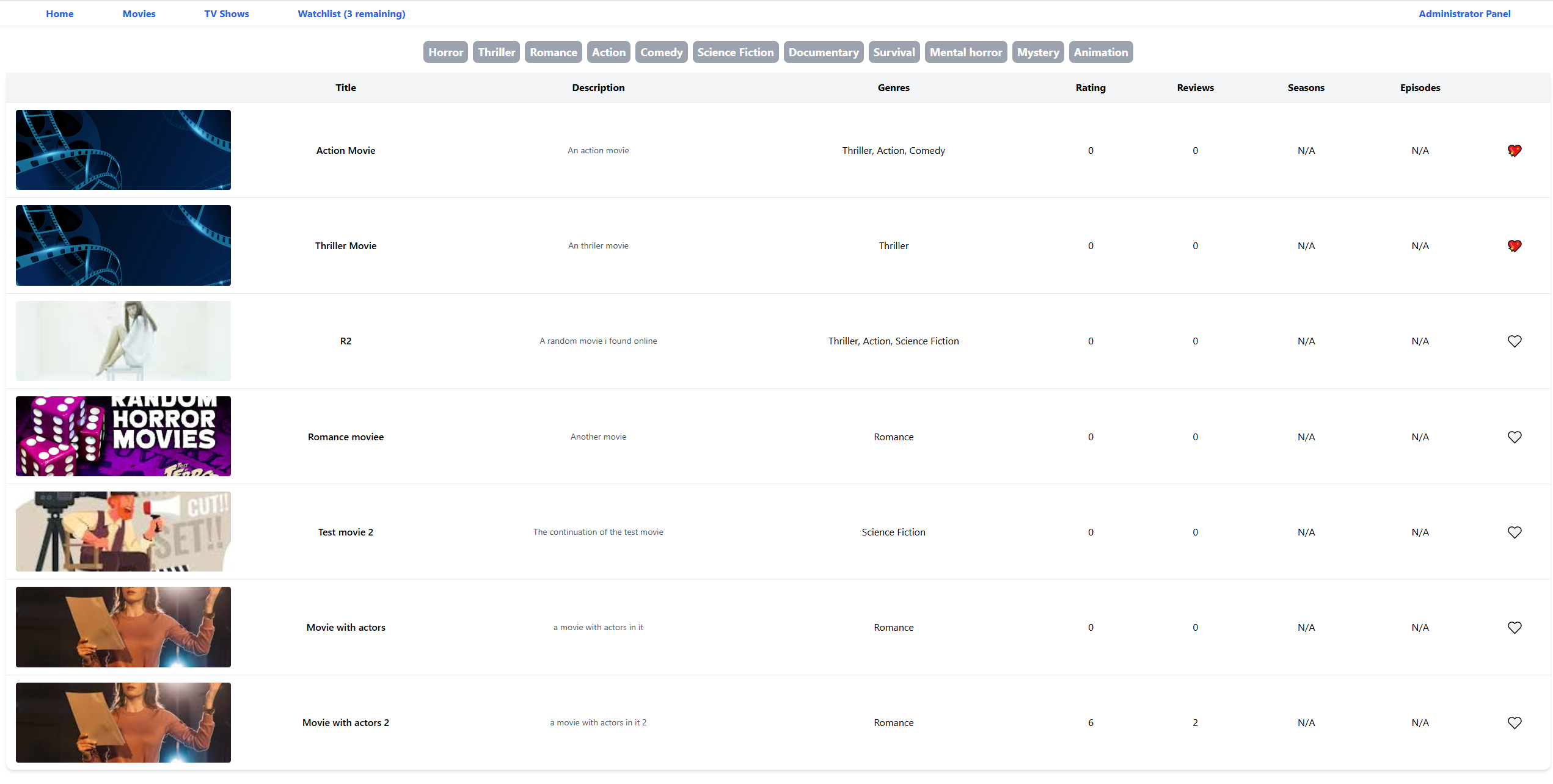


After being redirected the Navigation Bar is updated with 1 new navigation item – “Watchlist” which allows the user to both view their watchlist as well as the number of items left inside it from a glance.

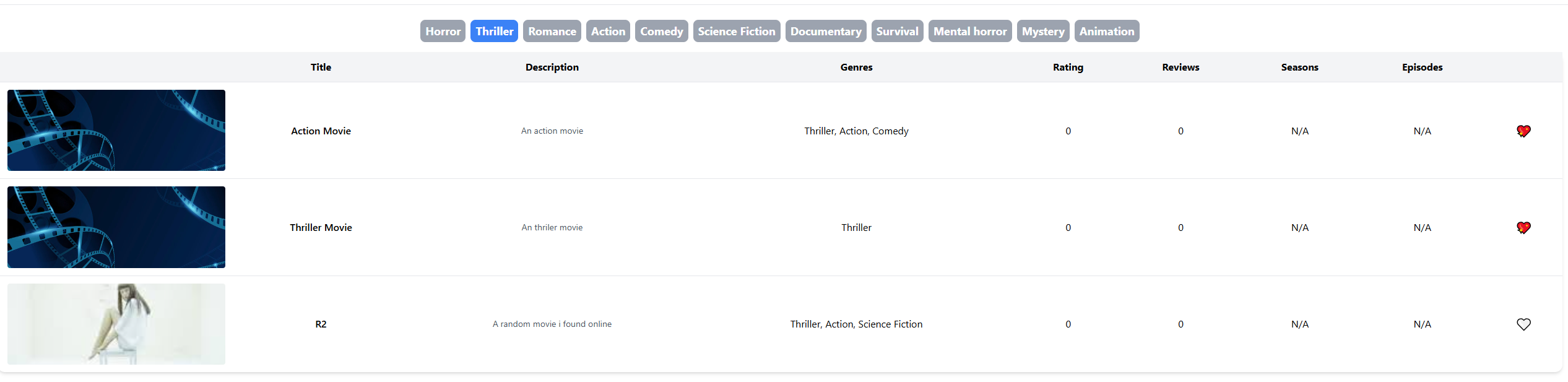


Assuming that the user is logged in as an administrator/moderator they receive access to an Administrator panel, which we will discuss at the end of the user usage guide.

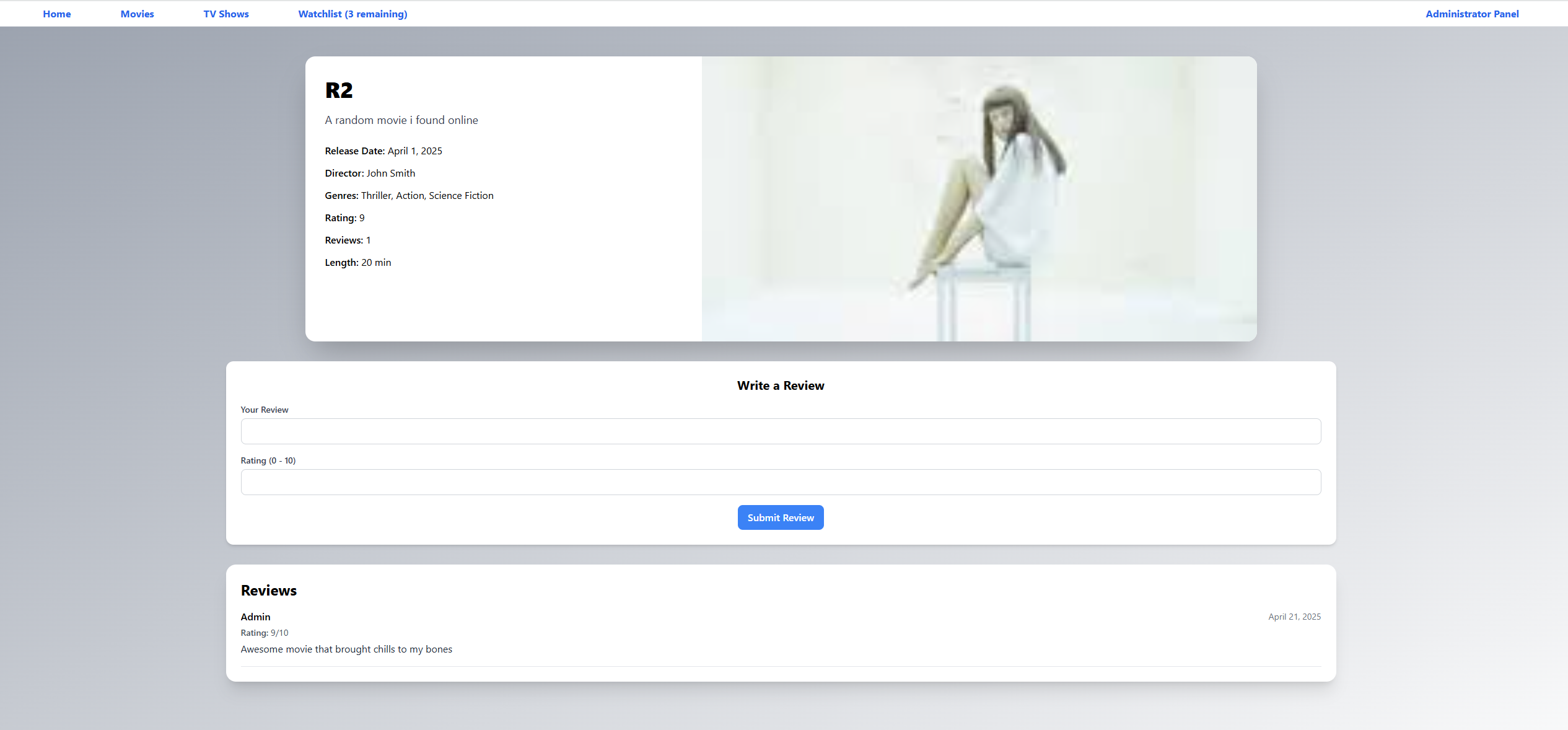
The next major page is the “Movies” page which displays all movies in the database as well as allowing filtering based on Genres which supports multiple selection.



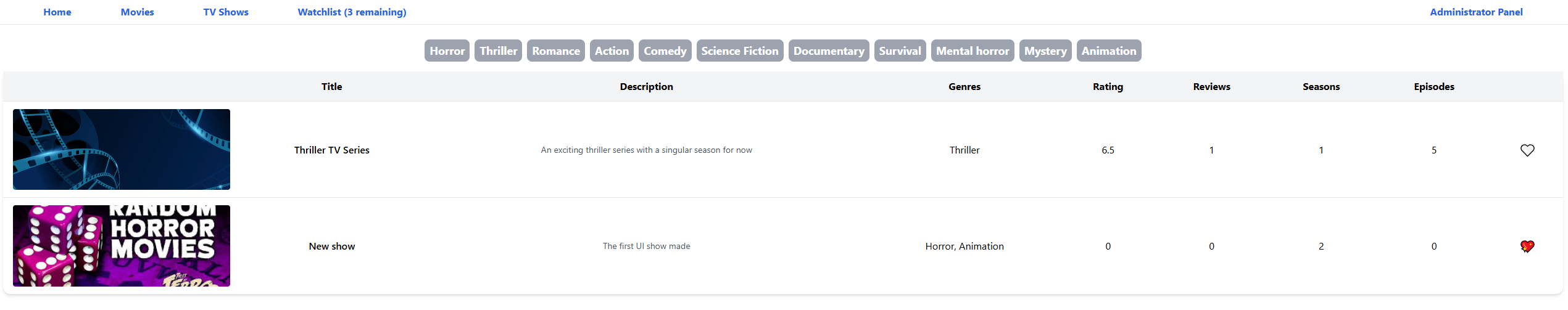
The last column indicated whether or not the user has the media in their watchlist. It is only visible for logged in users and will automatically update the Navigation Bar watchlist counter upon addition/removal of an item. Clicking on a Genre will light the button up in blue to indicate filtration and display only the required media as seen in the image below.

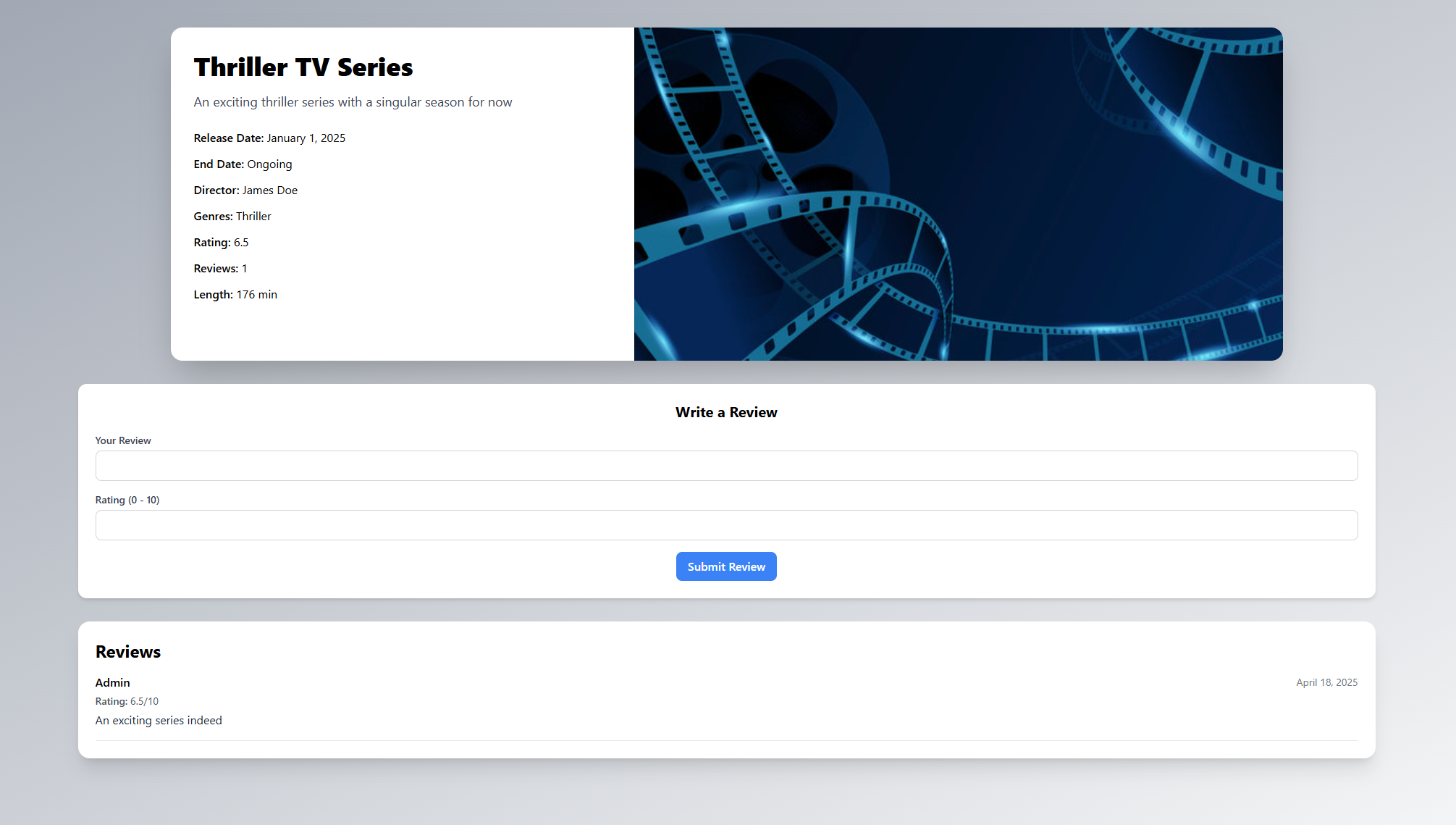


Upon clicking on an image the user is redirected to the details page of the movie. This page showcases the basic information about the movie such as it’s release date, length, rating, reviews, etc. Additionally, a window for submitting a review is visible if the user is logged in.

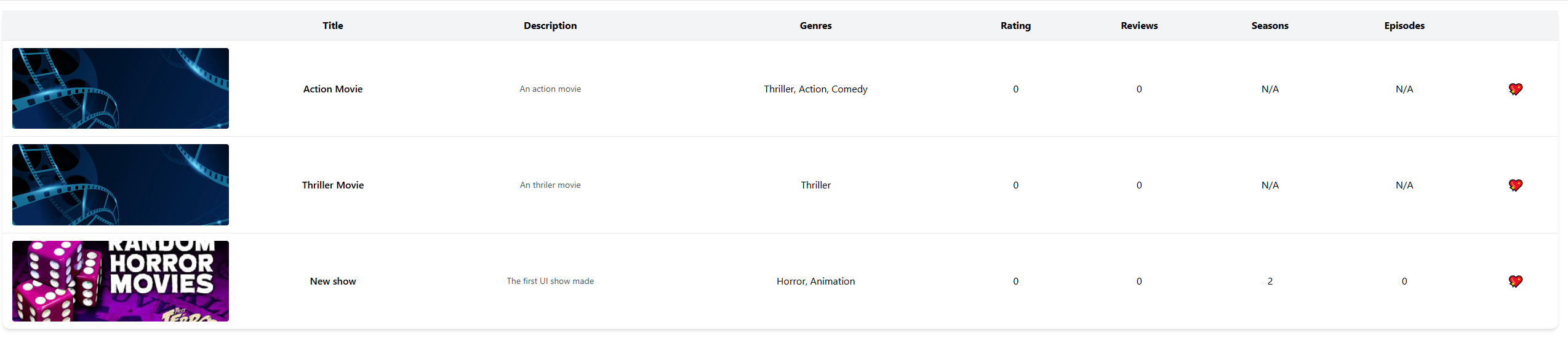


The shows display is identical to the movies – including a genre filtering tool.

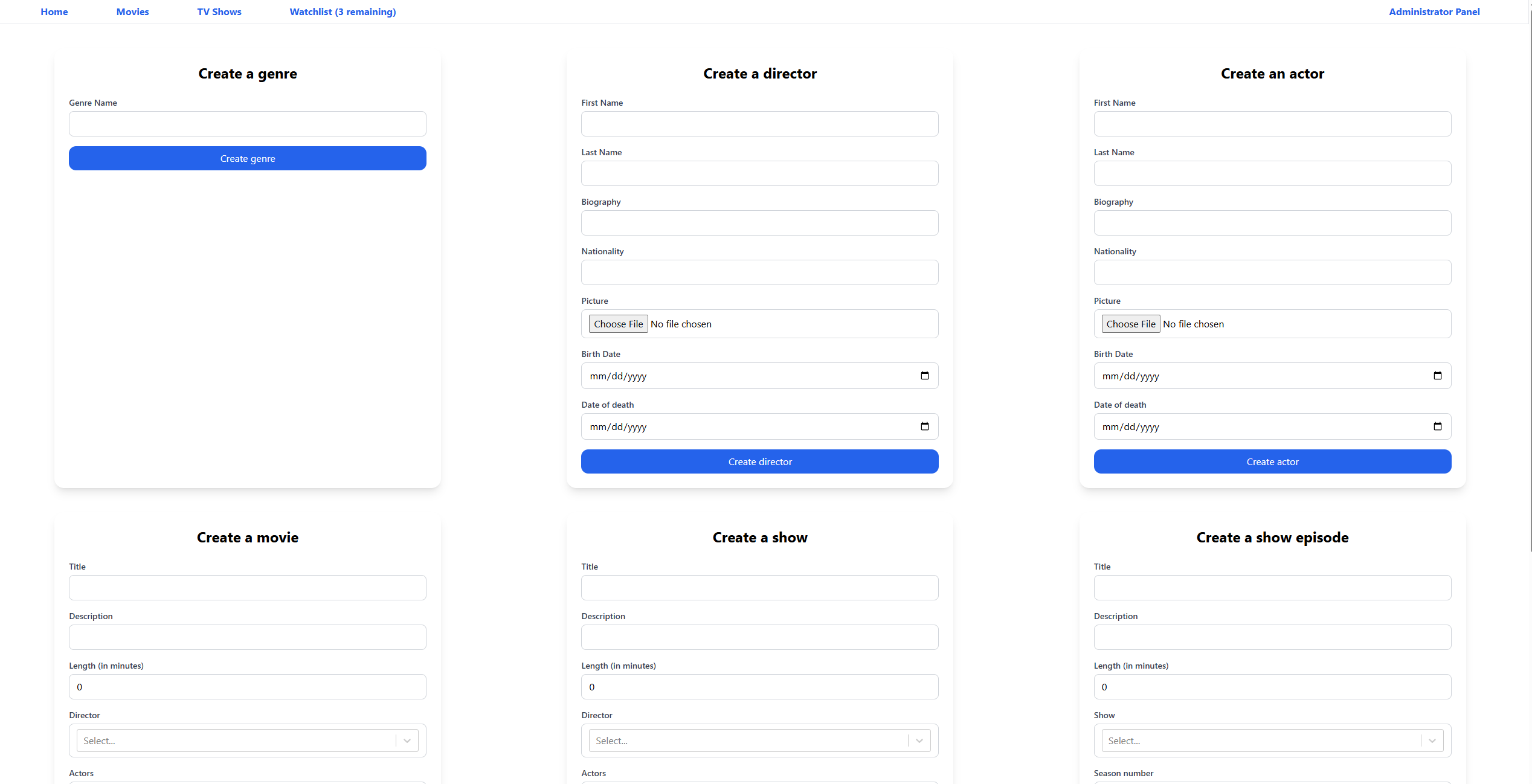


Viewing details for a series returns a screen which is the same as those for movies also showing the end date for a series (or “Ongoing” in case there is none). 

Last but not least for the user is the watchlist page which is identical to the previous 2 but ordered by Time of addition to the watchlist and without the Genre filtering ability.



This was the entire required guide for a normal user. The last remaining part is the Administrator Panel, only visible and available for users with the Administrator or Moderator roles. The flow for creating new Movies/Shows is the following: Genre -> Director -> Actor(s) -> Movie/Show -> Show episodes (only applicable in case of show). This is why the forms are ordered in their current configuration of a grid with 3 columns. The required fields are marked and do not allow a submission without adding values to them. Images currently do not have a limit for their size but this could be changed in the future easily.

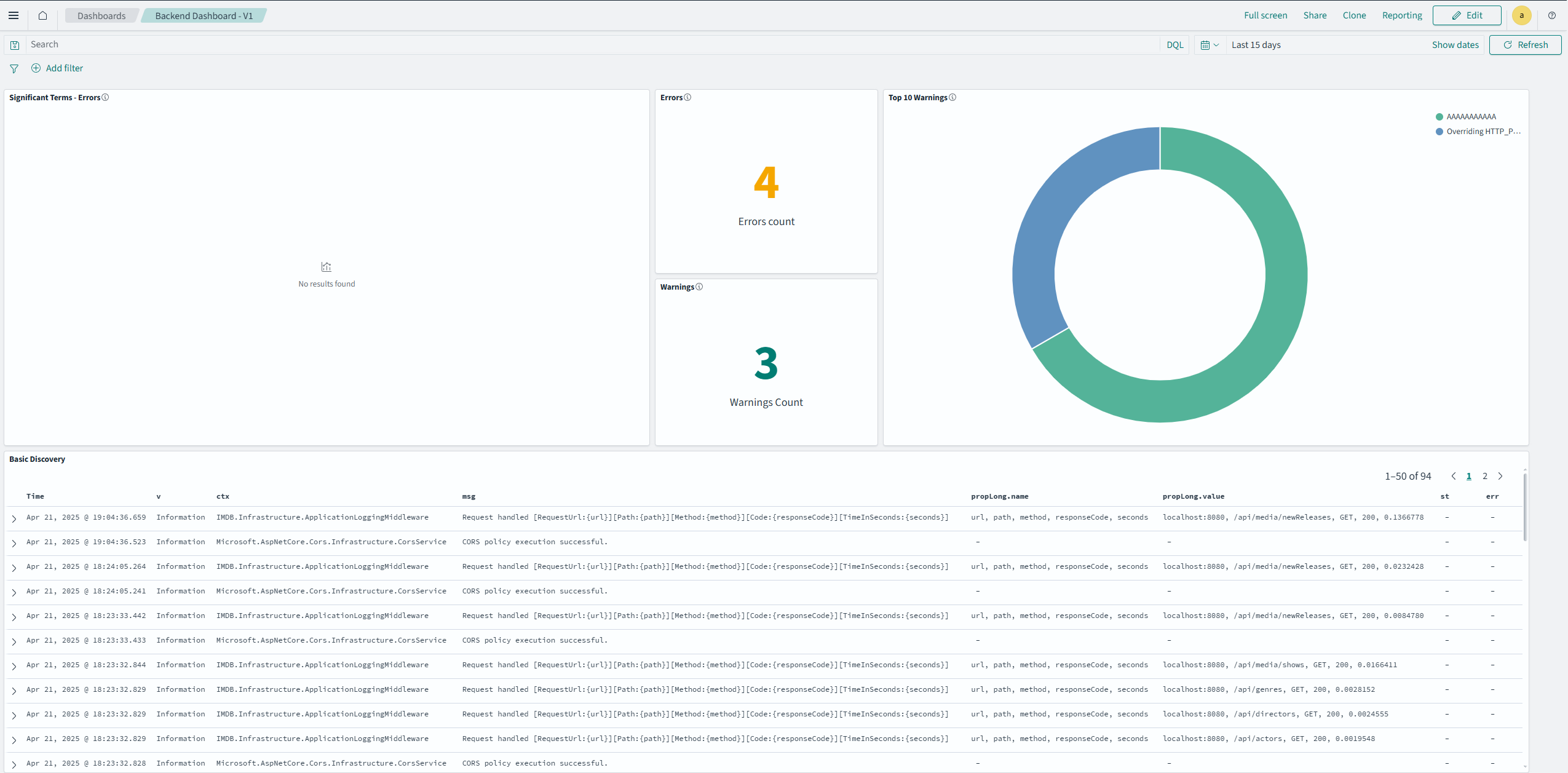


# Security

The security is done through a cookie with multiple validations in the backend for role of the user. Authentication errors are intentionally obscure as not to expose unnecessary information about existence of users. In case a user is determined to not have access to a resource in the backend a 404 Status is returned where possible. Some endpoints return 401 instead which is by design. Additionally, unhandled exceptions in the backend will not bubble up but will instead return a custom InternalServer response in order to not expose sensitive information about the status of the application.

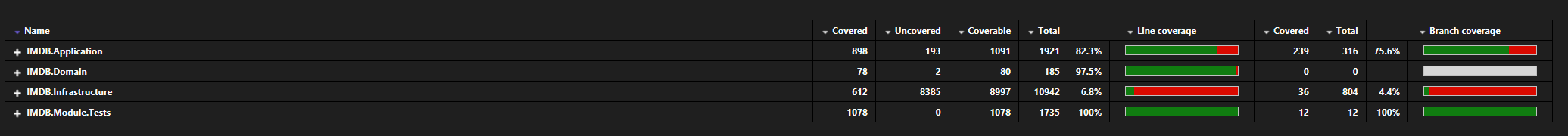
# Observability

Currently the backend application behaviour and logs can be observed through an instance of OpenSearch running locally as part of the docker orchestration. It is necessary to do an initial setup by importing and configuring certain elements caused by OpenSearch limitations. While not necessary, utilizing the Dashboard is recommended when developing locally and almost a must when diagnosing production issues. Below is an illustration of how the abovementioned dashboard looks like. In terms of disabling it, one of the easiest ways is to remove/set one of the configuration properties in the JSON file for the .NET application to an empty string (.e.g. “user”). Some namespaces which emit a log of logs have been overridden to NOT emit if the application is set to INFORMATION level logs or the logs themselves are WARNING level.



# Testing coverage

The backend application is tested using unit (module) tests communicating with an In-memory database seeded with specific data before each test run. The current test coverage (if we exclude migration files) is 82.3% Line coverage and 75.6% Branch coverage as can be seen in the image below. The uncovered parts of the codebase are related to Configurations and setting up the observability which do not require tests. The coverage was gathered using Fine Code Coverage in Visual Studio.



# Database structure

The current database structure does NOT use entities for Show seasons although this could change in the future. For many to many (N:N) Relationships (such as Genre -> Movie) a join-entity is used in order to combine the tables. The current architecture takes a different approach to handling responsibility of who-made-what and uses the Users table as a central table. Majority of tables have a foreign key that is many to one (N:1) towards the user that created each record. This is done in order to know which staff member created what in order of a compromised account and to allow easier creation of statistics. Furthermore, each entity has a “create\_date” and “update\_date” tracking the creation and latest update date of each record. The time is based on UTC. A relational diagram can be found below. Furthermore, such a table can be automatically created if you have access to the database instance using the “Database Diagrams” -> “New Database Diagram” and including some/all tables. All migrations and changes are kept in the “changelog” table.

