Project Report: COMP 4754 Group 1

Movie Streaming Services Application

Prepared by

Jason Wheeler

ID: 202018057 | **Email:** jpwheeler@mun.ca

Md. Zubayer Ahmed

ID: 202160438 | **Email:** mzahmed@mun.ca

Tanjet Tanjet

ID: 202174678 | **Email:** tanjett@mun.ca

Date Submitted: November 30, 2024

Table of Contents

Project Summary

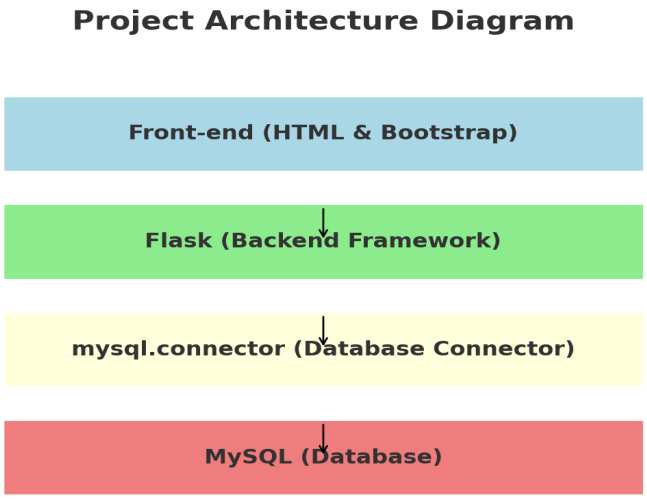
This database and web application are designed to simulate the functionalities and operations of a real-world subscription-based movie streaming platform. Some of the key features of this application are:

1. User Dashboard: Displays aggregated insights like Total users, Monthly revenue from subscriptions, Total active subscriptions, and Most reviewed movie.
2. Entities and Relationship Management: View all users, movies, payments, and subscription records with all the attributes, and manage these.
3. Provide seamless CRUD (Create, Retrieve, Update, Delete) functionality for managing users, movies, genres, ratings, subscriptions, and payments.
4. Reports and Analytics: Integrated visual reports viewing, summarizing user engagement and engagement/payment trends.
5. A dashboard showcasing key performance metrics.

This application is specially designed for the Movie Streaming services Platform Admin (the company), instead of the clients/customers as users in the mind. This application was developed using GitHub VCS, and the source code of this program can be found at https://github.com/tanjet/Movie-streaming-services-CS-4754

Technology stack

The project architecture follows a modular structure, with MySQL(MySQL workbench, and mysql.connector framework) for managing database operations, Python using Flask framework for handling the backend, and HTML and Bootstrap for front-end development. So the technologies layers are following.



Database Design

Database Schema

The database consists of the following main tables:

Entities:

1. **Users**: Stores information about users. The attributes are userID, userName, email, password, and date_of_birth. userID is the primary key of Users entity.
2. **Movies**: Contains details about movies. The attributes are movieid, title, release_date, duration and description. movieid is the primary key of the Movies entity.
3. **Genres (weak entity)**: Lists all movie genres and their metadata. The attributes are movieid, and movie_genre. movieid and movie_genre together is the primary key and movieid is the foreign key of the Genres entity.
4. **Ratings**: Collects user-submitted ratings and reviews for movies. The attributes are userID, movieid, ratingScore, reviews, and ratingDate. movieid and userID together is the primary key of the Ratings entity.
5. **Subscriptions**: Tracks subscription plans and status. The attributes are subscription_id, userID, startdate, end_Date, and subscription_status. subscription_id is the primary key and userID is the foreign key of the Subscriptions entity.
6. **Payments**: Manages payment history and the necessary information regarding payments.The attributes are payment_id, payment_amount, card_no, payment_date, payment_method, and subscription_id. payment_id is the primary key and subscription_id is the foreign key of the Payments entity.

Other Tables:

1. trigger_user: creates the movie_streaming database with the existing trigger_user data table, to set up relationships, index, and insert dummy user data created by Faker Class. It ensures referential integrity and optimizes bulk inserts for our initial setup.
2. Routines table: sets up the whole movie_streaming database, ensuring proper configurations and creating reusable routines like stored procedures and functions. It includes procedures for adding users, movies, and subscriptions, as well as a function to count user ratings for specific movies. We intended to create all the stored procedures and functions for all entities in these tables, but couldn't wrap up everything.

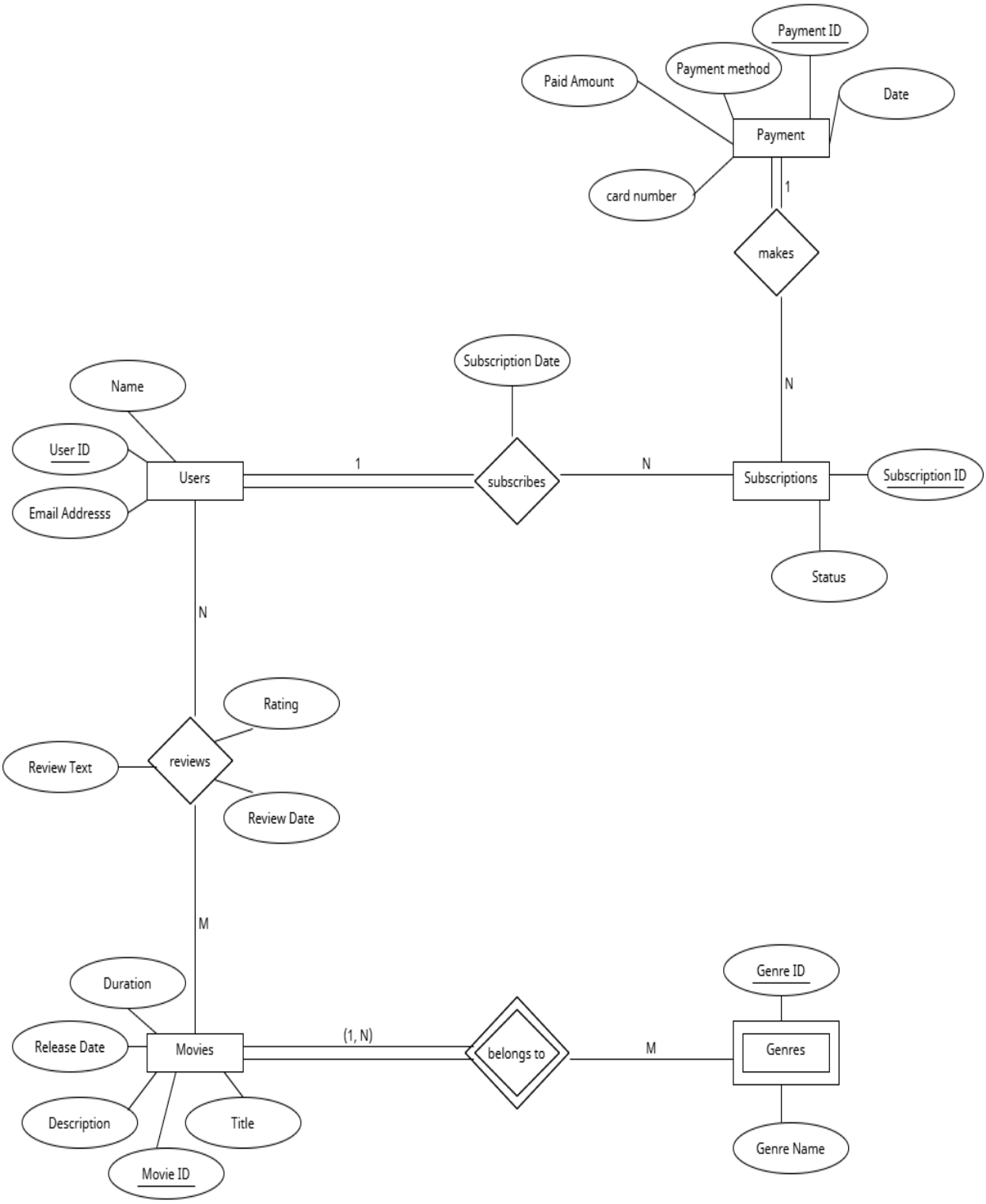
Relationships:

.

- Users have Subscriptions. Users can subscribe to multiple Subscriptions, but a Subscription belongs to one User.
- Payments are associated with a Subscription. Each subscription can have many payments, but each payment belongs to a single subscription.
- Users provide Ratings of Movies. Users can give multiple Reviews, and each movies can have multiple reviews by multiple users, but each Review is linked to one User and one Movie only.
- Movies belong to Genres. Movies can belong to multiple Genres, and a Genre can include multiple Movies.

Updated Entity Relationship Diagram

according to Phase 2 feedback



Database concepts applied in the project:

1. **View:** A view is a virtual table based on a SQL query.

   Not Used: Views were not implemented due to the focus on CRUD operations and the direct use of optimized queries in application logic. We didn't see any usage of this in our project.

2. **Trigger**: A trigger is a procedural code automatically executed in response to certain events on a table.

   Used: Triggers table was used for new users to add a new message, user name and id

3. **Stored Procedure:** A stored procedure is a precompiled SQL code that can be reused.

   Used: Procedures like addmovies, AddUser, and addSubscription were used to simplify and standardize data insertion and retrieval while ensuring consistency.

4. **SQL Function**: A function is a reusable SQL block that returns a single value.

   Used: The get_usercount function was used to calculate the number of users who rated a specific movie, providing efficient encapsulation of this logic.

5. **Indexing**: Indexing speeds up query execution by creating a reference to the database rows.

   Used: Indexes were applied to frequently queried columns like user_id and movie_id to optimize search and retrieval operations.

6. **Transaction**: A transaction ensures a series of SQL operations either all succeed or all fail to maintain data integrity.

   Not Used: Transactions were not implemented due to the short amount of time we had in our hands, but we understand the concept and can implement it if time permits.

7. **Backup**: A backup is a saved copy of the database at a specific point in time.

   Used: We have used the backup to get the SQL backup to run the database on another device. A database creation script was included with the project, but no populated backup was provided, as the focus was on demonstrating functionality with dynamically generated data.

GitHub Repository

Repository Structure

```
Movie-Streaming-Application/
├── app/
│   ├── __init__.py                 # Initializes the Flask app
│   ├── routes.py                   # Defines API endpoints and routing
│   ├── queries.py                  # Stores and executes the SQL Queries (as per demo feedback)
│   ├── db.py                       # Manages database connections
│   ├── templates/
│   │   ├── base.html               # Base layout used across all templates
│   │   ├── dashboard.html          # Admin dashboard page
│   │   ├── reports.html            # Page to display and generate reports
│   │   ├── add_genre.html          # Form to add a new genre
│   │   ├── edit_genre.html         # Form to edit an existing genre
│   │   ├── genres.html             # Page to list and manage genres
│   │   ├── add_movie.html          # Form to add a new movie
│   │   ├── edit_movie.html         # Form to edit an existing movie
│   │   ├── movies.html             # Page to list and manage movies
│   │   ├── add_user.html           # Form to add a new user
│   │   ├── edit_user.html          # Form to edit an existing user
│   │   ├── users.html              # Page to list and manage users
│   │   ├── add_subscription.html   # Form to add a new subscription
│   │   ├── edit_subscription.html  # Form to edit an existing subscription
│   │   ├── subscriptions.html      # Page to list and manage subscriptions
│   │   ├── add_payment.html        # Form to add a new payment
│   │   ├── edit_payment.html       # Form to edit an existing payment
│   │   ├── payments.html           # Page to list and manage payments
│   │   ├── add_rating.html         # Form to add a new rating
│   │   ├── edit_rating.html        # Form to edit an existing rating
│   │   ├── ratings.html            # Page to list and manage ratings
├── database/
│   ├── movie_streaming_users.sql         # SQL script for users tables
|   ├── movie_streaming_movies.sql        # SQL script for movies tables
|   ├── movie_streaming_genre.sql         # SQL script for genres tables
|   ├── movie_streaming_ratings.sql       # SQL script for ratings tables
|   ├── movie_streaming_payments.sql      # SQL script for payments tables
|   ├── movie_streaming_subscriptions.sql # SQL script for subscriptions tables
├── Documents/
|   ├── Group1-Phase1.pdf                 # Phase-1 submission of the project (Project Overview)
|   ├── Group1-Phase2.pdf                 # Phase-2 submission (ERD, Relational Schema, and Normalization)
|   ├── project-Report.pdf                # Final Project Report
├── run.py                          # Main application entry point
├── readme.md                       # The Readme file of the project
```

Installation and Running the Project

Prerequisites:

- Python 3.8
- Flask framework 3.1
- MySQL (mysql.connector, MySQL 8.0)
- MySQL Workbench 6.0

Steps:

Please Clone the repository:

git clone <https://github.com/tanjet/Movie-streaming-services-CS-4754>
cd Movie-streaming-services-CS-4754/

Install dependencies:

pip install flask mysql.connector

1. Set up the database:
    - Open MySQL Workbench
    - CREATE DATABASE movie_streaming;
    - Configure the app files with your database credentials.
    - Populate the database using the provided SQL files.

Run the application:

1. python run.py

2. The application will be live at local server port 5000 aka: https://127.0.0.1:5000.

Key Functionalities and Features

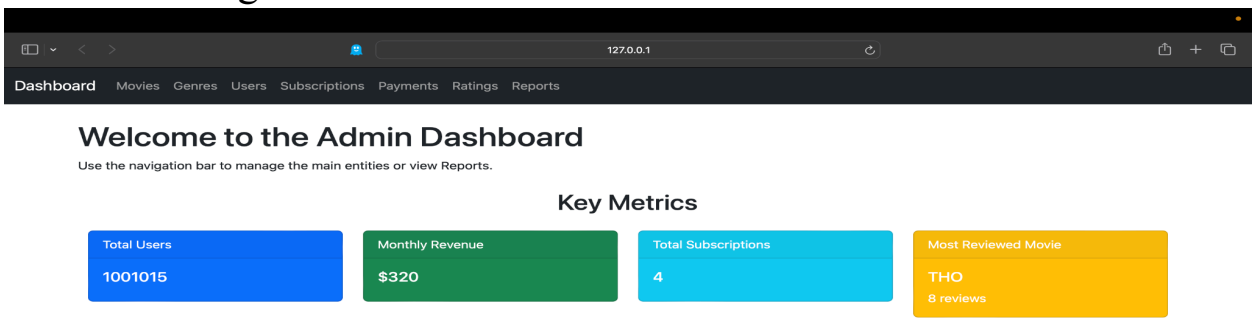
- **Dashboard:** The main page of the server. It displays a welcome message, and provides data for the admin dashboard, summarizing the application's key statistics.
- **User Management:** View, add, edit, and delete user records.
- **Movies and Genres:** Manage a comprehensive movie database and their associated genres.
- **Ratings and Reviews:** Enable users to add, edit, and view movie ratings.
- **Subscriptions and Payments:** Track and manage user subscriptions and payment history.
- **Reports:** Provides a summary of key application statistics, including total users, monthly revenue, total subscriptions, and the most reviewed movie etc.

Front-end Pages and Their Functionalities

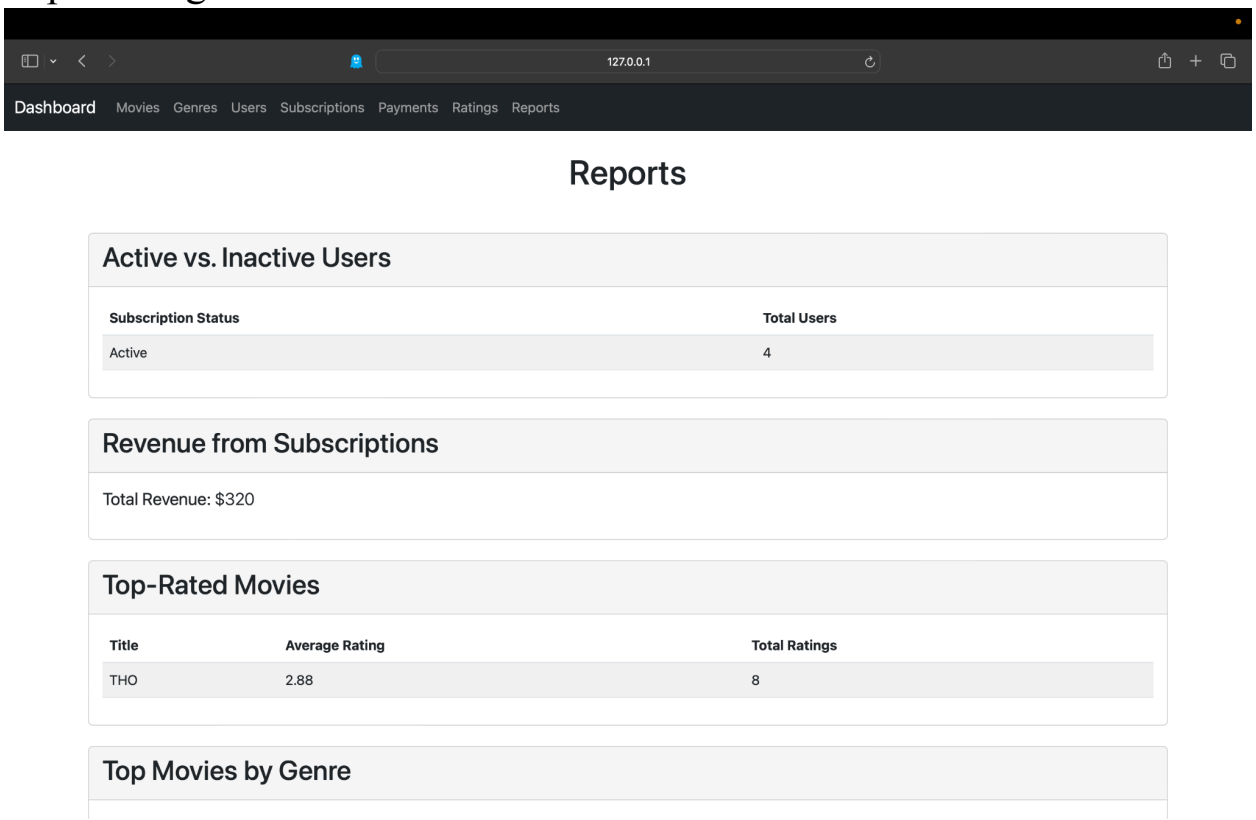
- http://127.0.0.1:5000: Dashboard (under development for key metrics, data, and operations visualizations).
- /movies, /movies/add, /movies/edit/<movieID>, /movies/delete/<movieID>: For viewing movies list with attributes, adding, editing, and deleting movies.
- /genres, /genres/add, /genres/edit/<genreID>, /genres/delete: For viewing genres list with movie titles, adding, editing, and deleting genres.
- /users, /users/add, /users/edit/<userID>, /users/delete/<userID>: For viewing the users list with attributes, adding, editing, and deleting users.
- /subscriptions, /subscriptions/add, /subscriptions/edit/<subscriptionID>, /subscriptions/delete/<subscriptionID>: For viewing the subscriptions list, status, adding, editing, and deleting subscriptions.
- /payments, /payments/add, /payments/edit/<paymentID>, /payments/delete/<paymentID>: For viewing, adding, editing, and deleting payments.
- /ratings, /ratings/add, /ratings/edit/<movieID><userID>, /ratings/delete/<movieID><userID>: For viewing, adding, editing, and deleting ratings.
- /reports: For showing a comprehensive database report (currently under development)

Screenshots

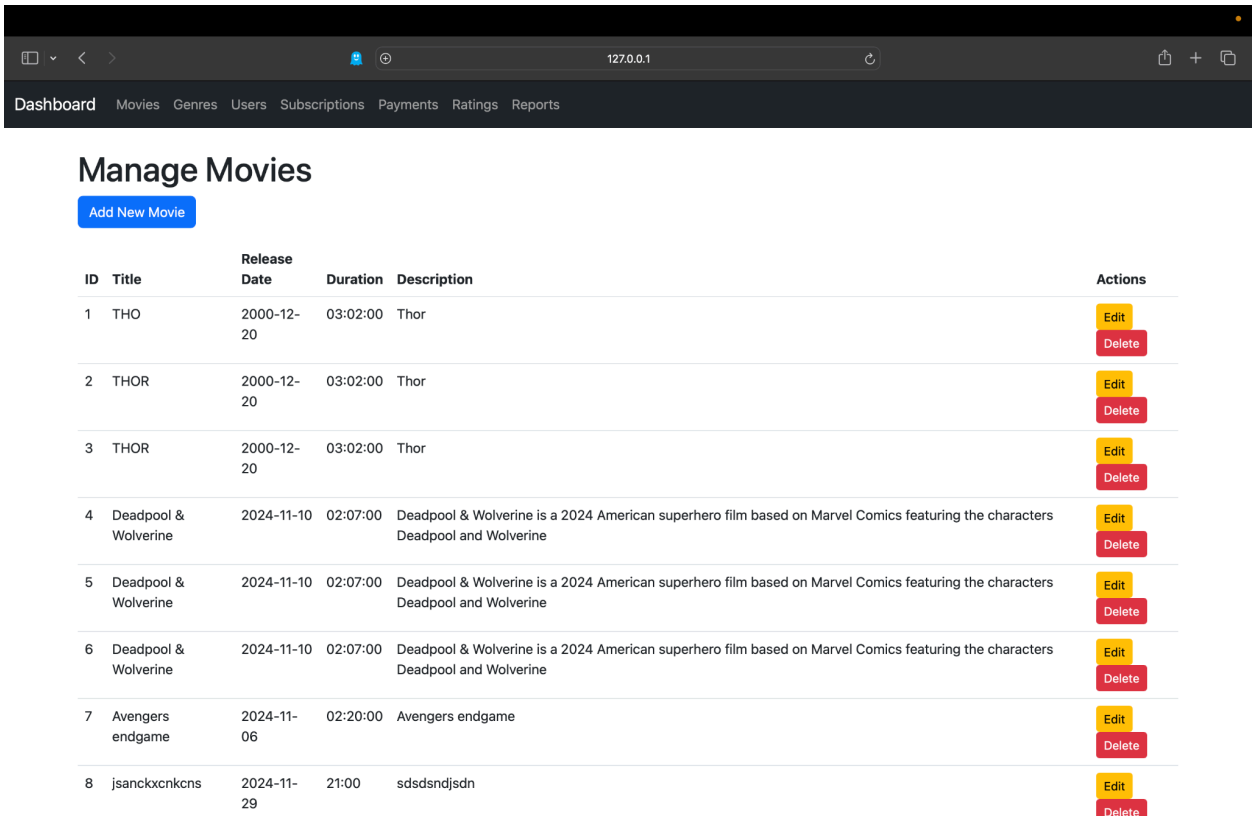
Dashboard Page:



Reports Page:



Entity (Movies) Page:



Data Collection:

We have created mass data using Python's Faker library, and individual data points using SQL queries in MySQL WorkBench.

Discussion

We learned a lot from this project. We got hands-on experience in designing and implementing a movie streaming service application. We learned to integrate multiple database concepts such as stored procedures, SQL functions, indexing, and foreign key relationships to maintain data integrity and optimize performance. Tools like Flask and MySQL was proved very effective for backend development and database management, while Faker facilitated generating realistic test data. Our key takeaways include understanding the importance of efficient database design, query optimization, and the interplay

between frontend and backend components. The project demonstrated how to handle complex relationships between entities and generate actionable insights through reports and dashboards. Moving forward, adding more automation through triggers and enhancing scalability with advanced transaction handling would be beneficial. But we think the time was a big constraint for this project, maybe this is something we should have started in the beginning of the semester.

Future Development Thoughts

- Role-based access control with user authentication.
- Expansion of database attributes, integrity, triggers, procedures, and transactions.