Group 3 Project Report

Movie Recommendation System (ScreenSort)

Contents of Report

Introduction and Background

The struggle with current recommendation services is that they are ad driven and feature newer films that they're trying to promote rather than what the user would actually be interested in. There is so much media that we can consume that many of us get overwhelmed with choice paralysis and one study found that for those of us who watch TV daily, we can lose 30 minutes a day searching for something to watch. This translates to almost 183 hours a year simply searching for some TV.

To combat this we have built a movie recommendation system for users called ScreenSort. The system primarily pulls movie data from TMDb and uses Rotten

Tomatoes to help give better insights into what the user would enjoy. Users will need to make an account and will then have the option to set their preferences based on genre, user rating and age rating. The recommendations will start by using the currently trending films from TMDb and will learn over time what the user likes by tracking user preferences and likes/dislikes as stored data via their profiles. The idea is the algorithm will improve over time and give better recommendations as the user base grows.

We have decided to run ScreenSort through the python console and use html templates with a style.css file. Users will be sent to the home page where they will either need to log in or create a new account. Once they do this, they will be shown a home page of the current trending films and have the option to either thumbs up or down the recommendation. Recommendations that are disliked will be removed from the list of films. The user has the option to see their stored preferences on their profile tab. We have included a list of key links on the top of the screen to make for a better user experience. Users can change their preferences at any time but they are currently limited to only one genre being stored which is something we would like to expand on in the future. There is also a ‘categories’ tab at the top of the page where users can select between seeing their recommendations based on minimum rating, genre or age rating.

1. Objectives

The primary objectives of this project are:

- **Develop a Recommendation System:** Create a movie recommendation system that tailors movie suggestions based on user preferences for genre, user rating, and age rating.
- **API Integration**: Integrate the TMDb and Rotten Tomatoes APIs to fetch real-time movie data, including genres, ratings, and age classifications.
- **User-Friendly Interface:** Provide a simple and intuitive interface where users can input their preferences and receive recommendations.
- **Database Management:** Utilise SQL to manage and query movie data efficiently, enabling rapid retrieval and filtering based on user input.

- **Testing and Evaluation**: Ensure the system's accuracy and efficiency by conducting thorough testing and evaluating its performance.

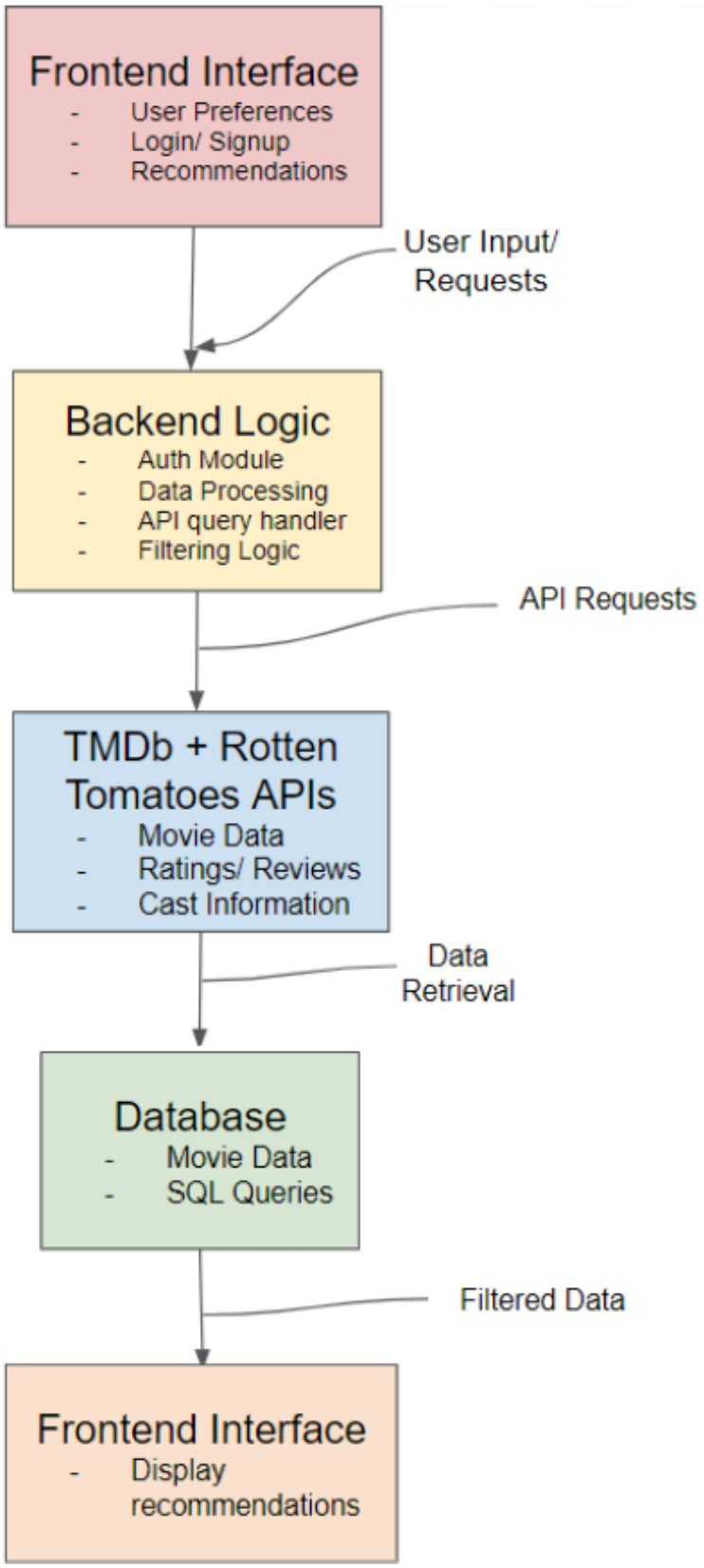
2. Specification and Design

2.1: ScreenSort is designed as a web application, consisting of the following components:

- **Frontend Interface:** A user interface developed using a Python framework (Flask) where users can input their preferences, log in, sign up and store user data.
- **Backend Logic**: Python scripts that process user inputs, query the TMDb and Rotten Tomatoes APIs, and filter results based on the database.
- **Database:** A SQL database that stores movie data, including genre, user rating, age rating, and other metadata, fetched from the APIs.
- **API Integration:** Interaction with the TMDb and Rotten Tomatoes APIs to retrieve up-to-date movie information and ratings.

2.2: **Key Features**

- **User Input:** Users can specify their preferred genres, a minimum user rating (e.g., IMDb score), and an acceptable age rating (e.g., PG-13, R).
- **Movie Retrieval:** The system fetches movie data from TMDb and Rotten Tomatoes based on the user's preferences.
- **Filtering Mechanism:** Movies are filtered according to user-specified criteria, with SQL queries executed to narrow down the list of movies in the database.
- **Recommendation Display:** The final list of movies is displayed to the user, providing titles, descriptions, user ratings, and age ratings.


Frontend Interface
- User Preferences
- Login/ Signup
- Recommendations
User Input/ Requests
Backend Logic
- Auth Module
- Data Processing
- API query handler
- Filtering Logic
API Requests
TMDb + Rotten Tomatoes APIs
- Movie Data
- Ratings/ Reviews
- Cast Information
Data Retrieval
Database
- Movie Data
- SQL Queries
Filtered Data
Frontend Interface
- Display recommendations

3. Implementation and Execution

3.1: Development Approach and Team Member Roles

The development of the movie recommendation system was executed by a team of four members, each contributing to specific areas of the project:

- **Frontend:** Handled the design and implementation of the user interface.
- **Backend**: Focused on API integration, database management, and developing the core logic for querying and filtering movie data.
- **Machine Learning:** Implemented an algorithm to allow recommendations to become smarter as the user interacts with the application more.
- **Full Stack:** Handled the integration of frontend and backend codes to create a fully rounded application and tested endpoints.

Each person was responsible for QA on their individual code/section with overview from other team members. The team followed an iterative development approach via slack catch ups, GitHub branch updates, Google Docs and Huddles.

3.2: Tools and Libraries

The following tools and libraries were used in the project:

- **Python:** The primary programming language for backend development and API integration.
- **Flask (including Flask_Login, Flask-WTF and FlaskSQLAlchemy):** Used for developing the application's frontend interface with Python.
- **SQL:** For database management and querying.
- **PyMySQL:** A library for connecting to and interacting with MySQL databases in Python.
- **SQLAlchemy:** An Object-Relational Mapping (ORM) tool to manage database operations in a more Pythonic way.
- **Requests:** A Python library used to interact with the TMDb and Rotten Tomatoes APIs.

- **Git/GitHub:** Version control and collaboration platform used to manage code and track progress.
- **Postman:** For testing API endpoints and ensuring correct data retrieval.
- **WTForms:** To create and validate web forms.
- **Werkzeug:** A utility library to provide tools for routing, request handling, and other web server tasks.
- **MarkUpSafe**: Ensures that strings are safely rendered in HTML, preventing security vulnerabilities.
- **Urllib3:** User-friendly HTTP client for Python.
- **Idna:** Internationalised Domain Names in Applications (IDNA) support.
- **ItsDangerous:** Provides various helpers to pass trusted data to untrusted environments.
- **Jinja2:** A templating engine for Python, used to generate HTML, XML, or other markup formats.
- **Greenlet:** A lightweight coroutine (micro-thread) library for Python.
- **Email_validator:** Validates email addresses.
- **Dnspython:** A DNS toolkit for Python.
- **Blinker:** A signal/event dispatching library for Python.
- **Certifi:** Provides a curated collection of trusted root certificates.
- **Charset-normaliser**: A library to help detect the character encoding of a given text.
- **Typing_extensions:** Provides backports of new type hints that are added to the standard library's typing module.
- **Cryptography:** A robust cryptographic library for Python.
- **Pandas**
- **Re (for recommendations)**
- **SKlearn**
- **Surprise**
- **OS**
- **Matplotlib**
- **Seaborn**

- **flask_restful from flask_cors import CORS**
- **flask_marshmallow**
- **numpy**

3.3: Implementation Process

Achievements:

- Successfully integrated TMDb and Rotten Tomatoes APIs to fetch real-time movie data.
- Implemented a responsive user interface allowing users to input their preferences and view personalised recommendations.
- Developed a robust backend capable of querying and filtering large datasets efficiently.

Challenges:

- **Database Performance:** Optimising SQL queries to handle large datasets efficiently, which involved indexing and query optimisation techniques.
- **Data Consistency**: Ensuring that data fetched from multiple APIs is consistent and synchronised was difficult. Differences in the API's data structure and how it is managed lead to minor inconsistencies.
- **Scalability**: Due to the usage of this programme being unknown, managing its scalability and optimisation was difficult.

4. Testing and Evaluation

4.1: Functional Testing

- **API Functionality:** Ensured that API calls returned the correct data and that the system handled errors gracefully (e.g., handling missing data or API downtime) by creating a test file and dummy accounts to see what data was saved and how it was presented to different users.
- **Database Operations:** Tested SQL queries to confirm they returned the correct movie lists based on user preferences.

- **User Interface Testing**: Verified that the frontend displayed movie recommendations correctly and that user inputs were processed accurately by integrating lines of code which display error messages on the Python console to solve potential errors.

4.1: System Limitations

- **Data Freshness:** Since the system relies on external APIs, any downtime or delay in API data updates could affect the timeliness and accuracy of recommendations.
- **API Rate Limits:** frequent user requests could still hit API rate limits, leading to potential delays in fetching fresh data.
- **Scope of Recommendations:** The system is limited by the data provided by TMDb and Rotten Tomatoes. Movies not well-represented in these databases may not be recommended, reducing the diversity of suggestions.

5. Conclusion

This project successfully implemented a movie recommendation system that provides personalised movie suggestions based on user-defined preferences. By leveraging Python, SQL, and the TMDb and Rotten Tomatoes APIs, the system delivers accurate and timely recommendations.

Going forward, it would be great to see this expand to have the option to incorporate preferences together so that the user can get the best recommendation. An extension to the project would be to have the option for users to reset their password if needed.

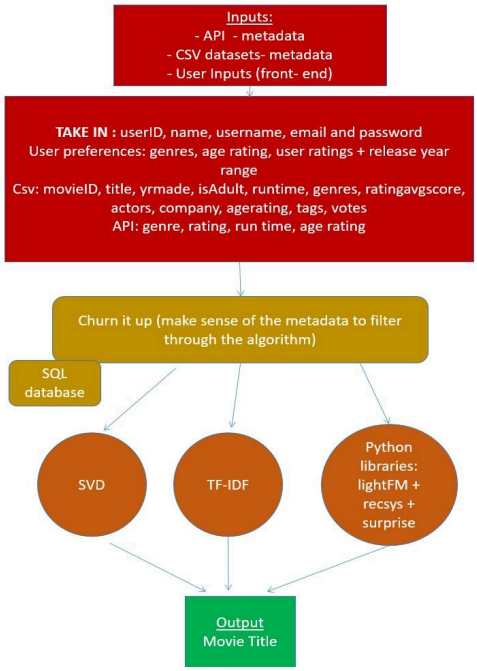
Appendix:

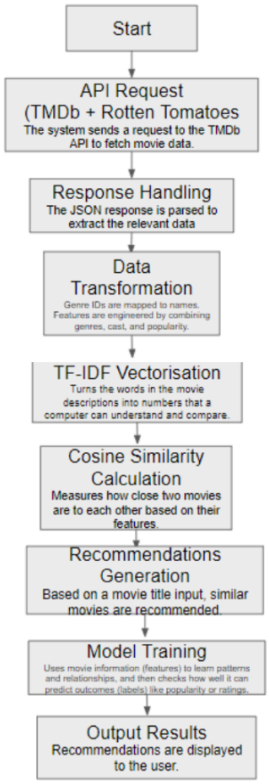
How to use ScreenSort

You will need to ensure that you have all the requirements file programmes installed and change your details in the config file on line 5 to your own username and password from MySQL. This is run through a virtual environment so if you chose to do the same, you will need to export your TMDb API key. There is a ‘Secret Key’ in this file as a protection against clients altering the content. You will need to ensure that there is a connection between PyCharm and MySQL Workbench (or your own SQL programme) to use the models file; alternatively, you can make the database and corresponding tables in SQL.

Users will need to make an account to use the service as we need to have a way to store the users data to give more appropriate recommendations. We will store the user data in one table where the password will be protected from us seeing it.The other table will store the user preferences which include the minimum user rating of films, their preferred genre, preferred age rating and which films they liked/ disliked from their recommended list. This will feed into machine learning. When logging in, the system has security measures in place to make sure that the name/ email is unique and that the email address is an actual address. We require users to enter their chosen password twice and it is hashed out.

Machine Learning Algorithm and Data Science Diagrams:





Formula for cosine similarity:

.

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}^{\mathsf{T}}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|} = \frac{\sum_{i=1}^{n} \mathbf{x}_i \cdot \mathbf{y}^{\mathsf{T}}{}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{x}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{y}_i)^2}}$$

Formula for the TF-IDF algorithm:

$$TF - IDFscore(w_{ij}) = TF_{ij} * IDF_i$$

Group Members:

Hana - Data science elements and coding
Hannah - backend, frontend and full stack integration, project manager
Bella - machine learning algorithms and implementing ML code

Jemma - frontend, wireframes, (documentation)
Jade - N/A
Jasmine - N/A
