CineMood Web- Application

Introduction:

CineMood is a movie recommendation platform designed to simplify the process of finding movies by offering suggestions based on the user's mood. Users can register as new members, select their current mood, and receive curated movie recommendations tailored to their emotional state.

This mood-based approach eliminates the need for extensive browsing, providing a personalized and seamless movie discovery experience. CineMood aims to make movie selection intuitive, enjoyable, and perfectly aligned with how users feel.

This document outlines the creation of CineMood, explaining its purpose, how it works, and the process behind its development. It also reflects on key challenges, system performance, and opportunities for future improvements.

Background

With so many movies available today, it can be difficult to decide what to watch. CineMood solves this problem by recommending movies based on the user's current mood. Instead of spending time searching through countless options, users can simply select how they're feeling, and the app will suggest movies that fit that mood, making the movie selection process faster and easier.

Motivation:

CineMood was created to make choosing movies more enjoyable. The goal is to help users find a movie quickly without having to browse through long lists, based on how they feel.

Target Audience:

CineMood is designed for:

- Movie lovers who want new movie suggestions.
- People who prefer personalized movie recommendations that fit their mood.
- Groups of friends looking for a movie that everyone will enjoy based on a shared mood.

Specification and design:

Functional requirements:

1. User Registration and Authentication
    - Users should be able to create accounts, log in, and manage their profiles securely.
    - Authentication should be handled using secure methods, such as email/password
2. Mood Selection Interface
    - A user-friendly interface that allows users to select their current mood from a list of predefined options or through an interactive mood scale.
    - The mood should be mapped to movie recommendations accordingly.
3. Movie Recommendation Algorithm

- A recommendation engine that matches movies to the user's selected mood based on a predefined mapping system that learns from user preferences.

4. Database
   - A database to store user profiles, selected moods, search history, and movie preferences.
   - The database should allow for easy retrieval and updating of information.
5. Movie Data Integration
   - Integration with an external movie database (e.g TMDb) to fetch movie details such as titles, genres, descriptions, and ratings.
6. Responsive User Interface (UI)
   - A responsive and intuitive UI that works across various devices (desktop, mobile, tablet).
   - The design should focus on ease of navigation and a pleasant user experience.

Non-functional requirements:

1. User-Friendly Design
   - The app should have a simple and clean interface that is easy to navigate, even for users who may not be familiar with technology.
2. Security and Privacy
   - Ensuring user data is protected by encryption and secure protocols.
3. Scalability
   - The system should be able to handle growing numbers of users and movie data over time, without performance degradation.
4. Performance
   - The app should load quickly and provide movie recommendations with minimal delay.
   - It should be able to handle simultaneous users without crashing.
5. User Support
   - Include clear instructions for registration, mood selection, and using the recommendation features.
6. Testing and Quality Assurance
   - Thorough testing to ensure all features work as expected, including the mood-based recommendations, user registration, and movie data integration.
   - Include bug fixes and continuous improvements to the app based on user feedback.

Design and Architecture

The high-level architecture designed to handle user authentication, interaction, and personalized

content delivery.

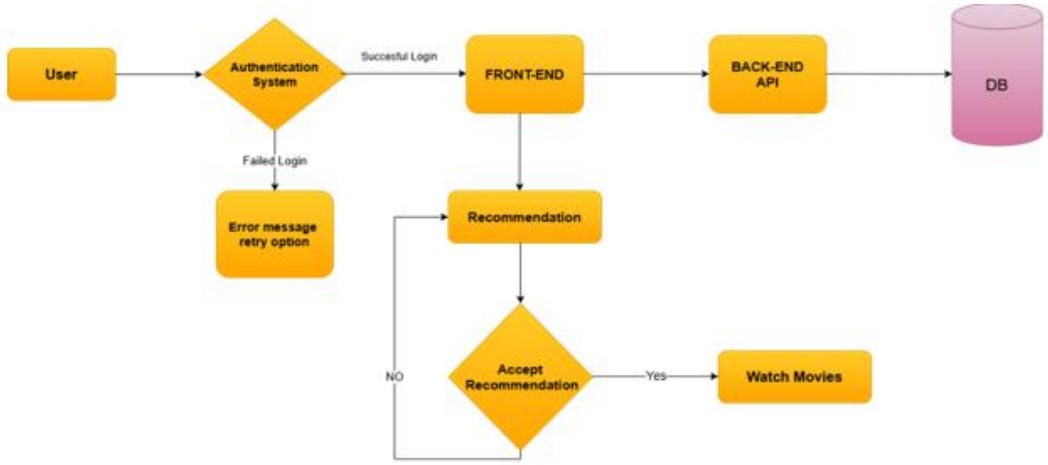

Fig1. CineMood application high-level architecture design

It integrates the following core components:

Front-End: Manages user input (e.g., login credentials, preferences).

Back-End: Serves as the logic layer, validating user credentials and interacting with the database. It also Implements robust mechanisms for error handling, ensuring secure and efficient operations.

Database: Stores critical information like user credentials, login attempts, and content data (e.g., movie recommendations). Acts as the backbone for both authentication and personalized content generation.

Functional Output: After successful authentication, users gain access to features like movie recommendations or dashboard views. Uses data-driven logic to personalize experiences.

Core features of the application

Login and Registration System: Users can create an account, log in, and log out. Credentials are stored securely (hashed passwords).

User Interface: Simple text-based interface (CLI) allowing interaction with the app upon login.

Mood-to-Genre Mapping: Create a mapping that links moods to suitable movie genres. Which in our case we only chose 4 genres that are Action, Comedy, Adventure and Drama based on the moods selected as excited, happy and sad.

Data Retrieval from API: Used TMDB API to retrieve movie lists based on genres and store those movie lists in out database.

Recommendation Algorithm: Implement an algorithm that returns a list of movies based on user input and history.

Database: implemented a database (MySQL) to store User information (login, password) and list of movies which are retrieved from the API.

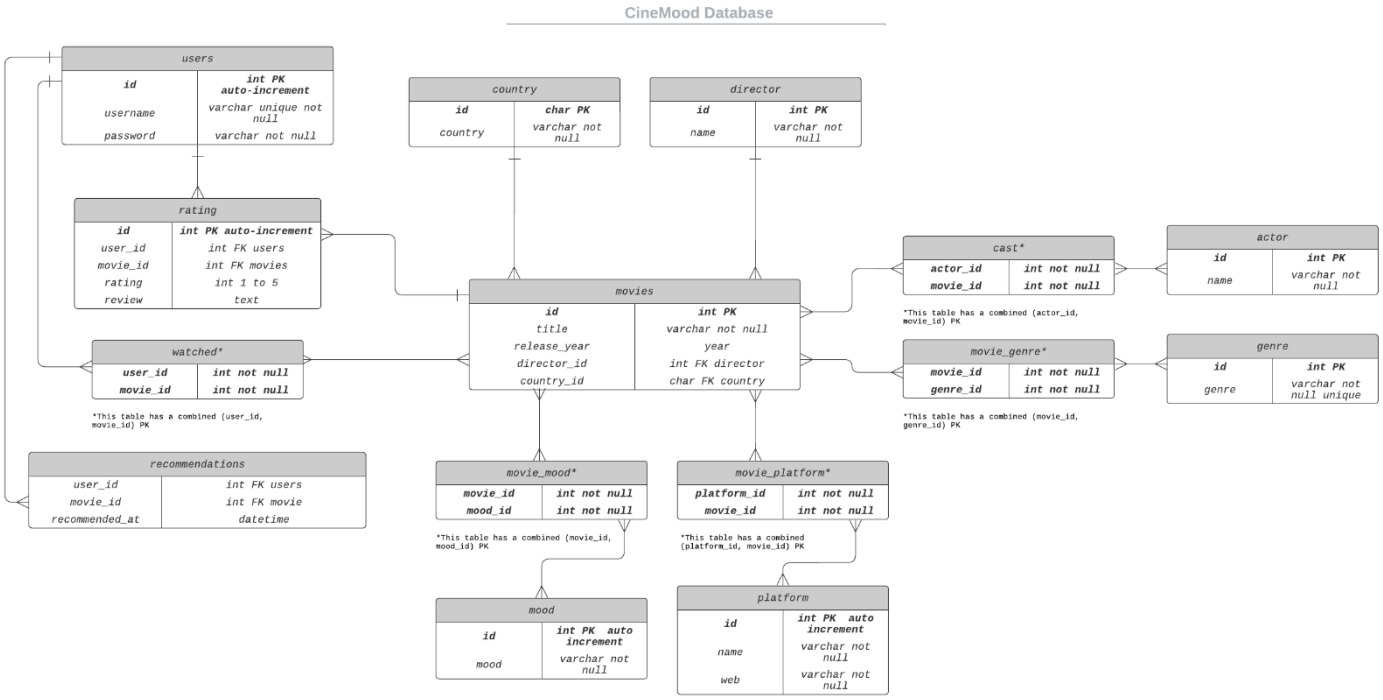

fig 2. Database Schema for CineMood application

Project Structure: Modular code structure with logically separated modules. Main script with a run() function.

Testing: Used unit tests for functions and classes.

Implementation and execution

The development of the application followed a structured and modular approach, aligning with the project's objectives of delivering personalized movie recommendations based on user-selected moods

1. Data Handling and API Integration

- API Retrieval: Integrated with a third-party movie API to fetch a limited set of movies (1 page with 6 genres).
- Database Storage: Movies fetched from the API were stored in a relational database (MySQL) during an initial setup phase. Once stored, the API was not queried again, ensuring minimal server overhead.

2. Mood-Based Recommendations**:** Designed a backend logic to map user-selected moods to the appropriate genres. Queried the database to fetch movie titles associated with the mood/genre

selected by the user. Provided a simple, efficient response mechanism to deliver movie recommendations.

3. User Authentication:

- Implemented user registration and login functionality to provide personalized access.
- Passwords were securely hashed using algorithms like bcrypt to ensure user security.
- Used token-based authentication (e.g., JWT) for session management and secure communication.

4. Testing with Unittest

- Unit Testing: Used Python's unittest framework to test individual modules and ensure functional correctness. Covered the following key components:
    - API data retrieval and transformation.
    - Database insertion and querying logic.
    - User authentication (registration and login).
    - Mood-based recommendation responses

The responsibilities of our team members are clearly defined and divided according to their respective skill sets, as detailed in the table below

	Project Definition	Kanban management	Aunthentication Implementation	Aunthentication Testing	DB Implementation	DB Handler	API Handler	Recommendation System Implementation	Front-end Design & Configuration	Front-end Implementation	Front-end Testing	System Testing	Compilation of .MD file	Compilation of documentation	GitHub management	Installation process	Contribution to Sprint Review
Justyna Gadjek	X		X	X						X	X	X	X			X	X
Hermella Kebede						X	X	X						X			X
Magdalena Kurek									X	X	X	X				X	X
Aleksandra Dragan								X									X
Pamela Smardz							X	X				X	X		X	X	X
Noelia Ruiz Barrales		X			X	X								X			X

Fig 3. Task division table

Tools and libraries

- Python 3.x

Standard Libraries**:**

- itertools
- collections

- unittest
- hashlib or bcrypt (for password hashing)

External Libraries**:**

- requests (for API communication)
- SQLAlchemy (optional, for more efficient database management)
- pytest (for test automation)

API:

- The Movie Database □TMDb) API

Database

- MySQL□ A relational database suitable for medium to large-scale projects, providing robust data handling capabilities.

Version Control:

- Git: For version control and team collaboration

Implementation process:

Our group collaborated to build a user-friendly app for mood-based movie recommendations while overcoming challenges. We designed an intuitive interface and used Git for version control, resolving branch management issues with clear strategies and regular updates. After reviewing the TMDB API, we optimized performance by storing only essential movie data in the database.

Balancing work, classes, and meetings was challenging, but we used Slack for updates and virtual meetings for urgent discussions. Integrating the API with the database required debugging and teamwork, but limiting the data simplified this process. Using an Agile approach, we tracked tasks, adapted quickly, and successfully delivered a functional app despite obstacles.

Testing and evaluation:

Testing was a critical part of ensuring our application's functionality and reliability. We used Python's unittest framework to test key features, including user registration (successful and error scenarios), login (successful login, incorrect password handling, and account lockout/unlock), password hashing, and database operations. These tests ensured that data was securely stored and correctly retrieved for recommendations.

Additionally, during development, we often used the print() function as a quick debugging tool to verify smaller code snippets before implementing larger features. This approach helped identify errors early, complementing the structured unit testing and ensuring a smooth development process.

Conclusion:

Our application successfully provides personalized movie recommendations based on user moods with a simple and user-friendly design. We achieved key goals like API integration, database handling, and user authentication through teamwork and effective planning.

However, the system has some limitations. We originally planned to add fun features like cinequizzes, but this was not implemented. The movie data is limited, stored once in the database, and requires each user to configure their own database to run the app. The app is also not deployed on a server, which could improve accessibility. These are areas for future improvements, including expanding features, centralizing the database, and server deployment.

Below is the final detailed design of the app, showing how it works.

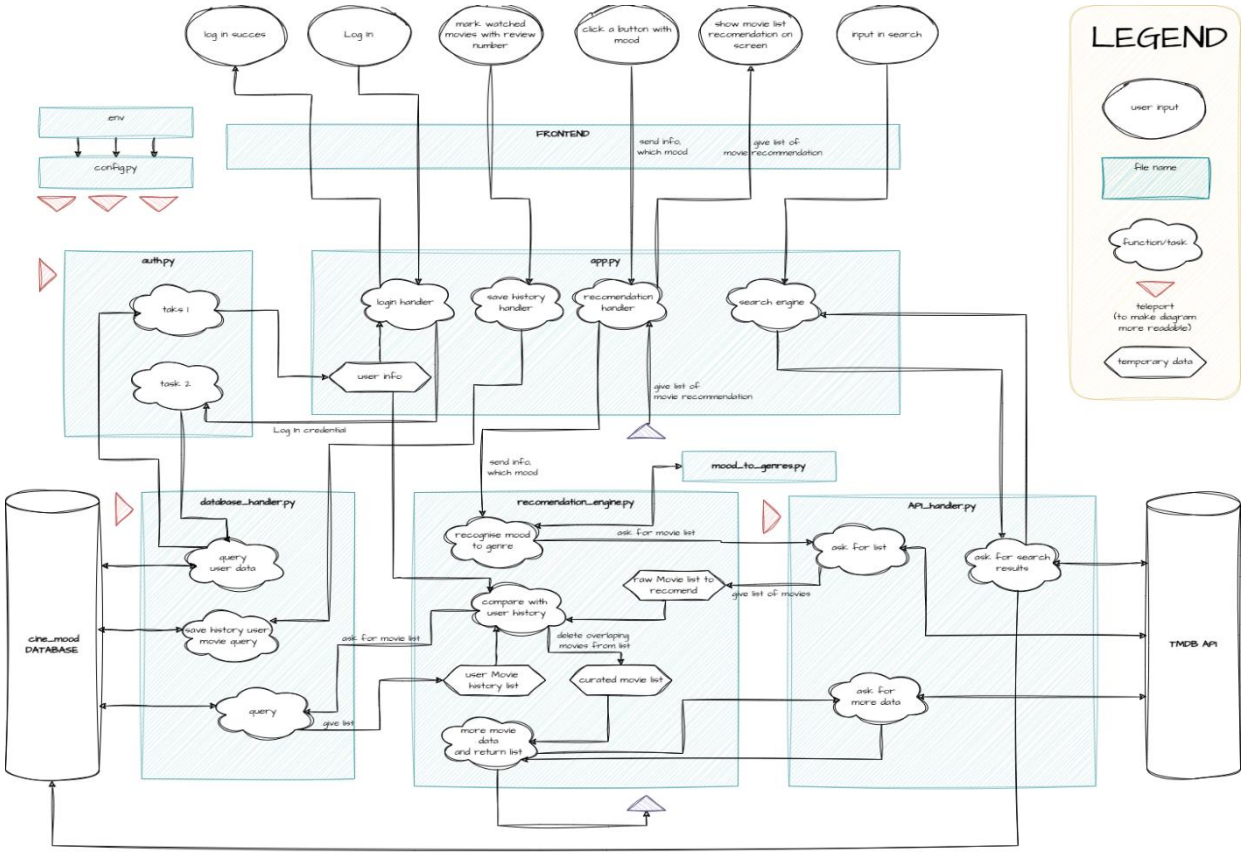

Fig 4. End to End application design