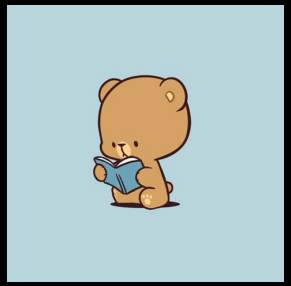
Recommender Companion

- Ashi Sinha, Chiranjeewee Koirala, Yukti Sharma, Vrinda Goel



Objective:

Our web application provides personalized book and movie recommendations. The app will allow users to input ratings for at least three books and movies they've experienced, and based on these preferences, generate tailored recommendations. The aim is to simplify content discovery by suggesting books and movies that match the user's distinct taste.

Data Sources:

- We use 2 datasets for our source information, both of which have been sourced from Kaggle:
  - Movies - The Ultimate 1 Million Movie Dataset (Link)
  - Books - Book Recommendation Dataset (Link)
- The movies dataset emulates “The Movies Database” (TMDB) repository, while the books dataset emulates the Goodreads repository.
- Both the dataset required significant clean up due to null/missing values, data corruption, duplicate values etc.

Data Pre-processing:

- Movies dataset originally had **1,023,264** entries, while the Books dataset originally had **271,360** entries.
- Cleanup:
  - Selected only the necessary columns from each of the datasets and dropped the rest (Eg: tagline, popularity, IMDB ID)
  - Missing/Null value removal: Removed several thousand rows in both the datasets
  - Data Corruption: Found columns like “Year of Publication” in the books dataset containing information of the publishing house instead; removed such rows in both datasets.
  - Data Truncation: Due to limitation of resources, we could not work with files over 100+ MB, so we truncated the movies file by selecting random entries to an acceptable size.

Infra Setup

Basic Setup:

- **Docker**: Containerised the application to ensure consistency across different environments.
  - We have defined services like the frontend, databases, and other dependencies in the docker compose file.
  - The Docker Compose file manages and simplifies the setup process by allowing all services to be launched with a single command.
- **MySQL Database**: Utilized as the primary database system.
  - Configured within Docker for development and testing purposes.
- **Flyway**: Database migration tool integrated for version control and schema migration.
  - Ensures database schema is up-to-date and consistent across deployments.
  - Configured to run migrations automatically at service startup.
- **Microsoft Authentication**: Integrates Microsoft OAuth for secure user authentication.
  - Utilizes Microsoft Graph API to fetch user details like email, display name, and roles.

Basic Setup:

- **Backend Technology**:
  - Python with Flask framework for the API server.
  - SQLAlchemy ORM for database interactions.
  - Secured API routes with user and admin middleware for authentication and authorization.
- **Frontend Technology**:
  - Angular for the client-side framework.
  - Communicates with the backend through RESTful APIs.
  - Implements reactive forms and client-side routing.
- **Security Features**:
  - Utilizes HTTPS for secure communication between client and server.
  - JWT tokens used for maintaining user sessions and securing API endpoints.
- **OpenAI GPT 3.5 Turbo API**: Used for Recommendation generation

Data Model

We have 5 tables in our database:

- Movies table which acts as the repository of the TMBD repo.

ID	Title	Release Date	Language	Genres	Cast	Director	Poster Path

- Books table which acts as the repository of the GoodReads repo.

ISBN	Title	Author	Year of Publication	Image URL

- User table which stores information of all active users.

ID	Display Name	Email	Age	Created At	Onboarding Completed

- User Watched Movies table which stores the information about movies watched by every user.

UUID	Email	Movie ID	User Rating

- User Read Books table which stores the information about books read by every user.

UUID	Email	Book ISBN	User Rating

Features

Authentication

- **User Interaction** - Completed
  - Both admins and regular users are presented with a "Login with Microsoft" button.
  - When clicked, the user is redirected to Microsoft's login page
- **Microsoft Authentication** - Completed
  - User enters their Microsoft credentials
  - Microsoft verifies the user's identity
- **Redirection to Our Application** - Completed
  - Microsoft redirects the user back to our application with an authorization code

Authentication

- **Backend Processing** - Completed
  - Our frontend sends the authorization code to our backend
  - Backend exchanges the code for an ID token from Microsoft
- **Token Verification** - Completed
  - Backend verifies the ID token's signature using Microsoft's public key
  - Backend checks that the token is intended for our application
- **Session Establishment** - Completed
  - Backend sets the ID token as a cookie to maintain the user's session

Authentication

- **Role Determination** - Completed
  - Backend checks the ID token for an "admin" role
  - Sends role information back to the frontend
- **User Interface** - Completed
  - Frontend displays either the admin or user interface based on the role

User Home Page (Read):

- The user can see all the books and movies they have added to their profile - Completed
- The content is displayed in a **paginated, tabbed** format, allowing for easy navigation and browsing of their collection - Completed
- The user sees the following buttons:
    - Add movies - Completed
    - Add books - Completed
    - Generate recommendations - Completed
    - Delete profile - Completed

Add Movie / Book (Create):

- During signup, users are required to provide ratings for at least 3 books and movies they've watched/read - Completed

- After initial sign up, users can continue adding more movies and books to their profile - Completed

- Our app will include an **autocomplete** feature that enables users to find titles quickly as they type and the results are **ranked, and paginated** - Completed

- When they add a movie/book to the list, they can also add their rating for the same to indicate how much they like/dislike it - Completed

Delete Book / Movie (Delete):

- Users have the flexibility to remove any books or movies from their profile. This helps them maintain an up-to-date list of rated content, ensuring that the recommendations remain aligned with the current preferences of the user - Completed

Generate Recommendations:

- The user can click a button to generate recommendations for books and movies similar to the ones they have already watched, and liked/disliked - Completed
- The app will use GPT to generate a customised and personalised list of recommendations for the user while factoring their likes and dislikes - Completed
- The prompt will be engineered and experimented with to generate the best possible results for the user - Completed
- The user's age is also factored into the recommendations to ensure age-appropriate recommendations - Completed

Filter Recommendations:

- The users can also filter the type of recommendations before they are generated.
- Some of these filters include:
    - Generate recommendations for only books, only movies, or for both.
        - If the user requests recommendations for both movies and books, the app organizes the results into **3 separate tabs: Books, Movies, and All (a combined view of both movies and books)** - Completed

Filter Recommendations:

- Set how many recommendations they want to generate - Set to 10 due since we decided to use GPT API paid credits (instead of mock server) and were tied up by limited resources.
- Choose if they want to generate recommendations based on their entire history, or only based on the history of either books or movies - Completed

Admin Profile / Panel:

- There will be an admin role to help maintain and update the users database, movie and book repositories - Completed
- Functionalities:
  - Add Movies - Completed
  - Add Books - Completed
  - Delete User Account - Gave authority to User (Completed)

User Profile:

- When the user signs up, their personal information including name and age is taken/entered - Completed

- The user has an option to delete their profile - Completed

Testing

Frontend Testing:

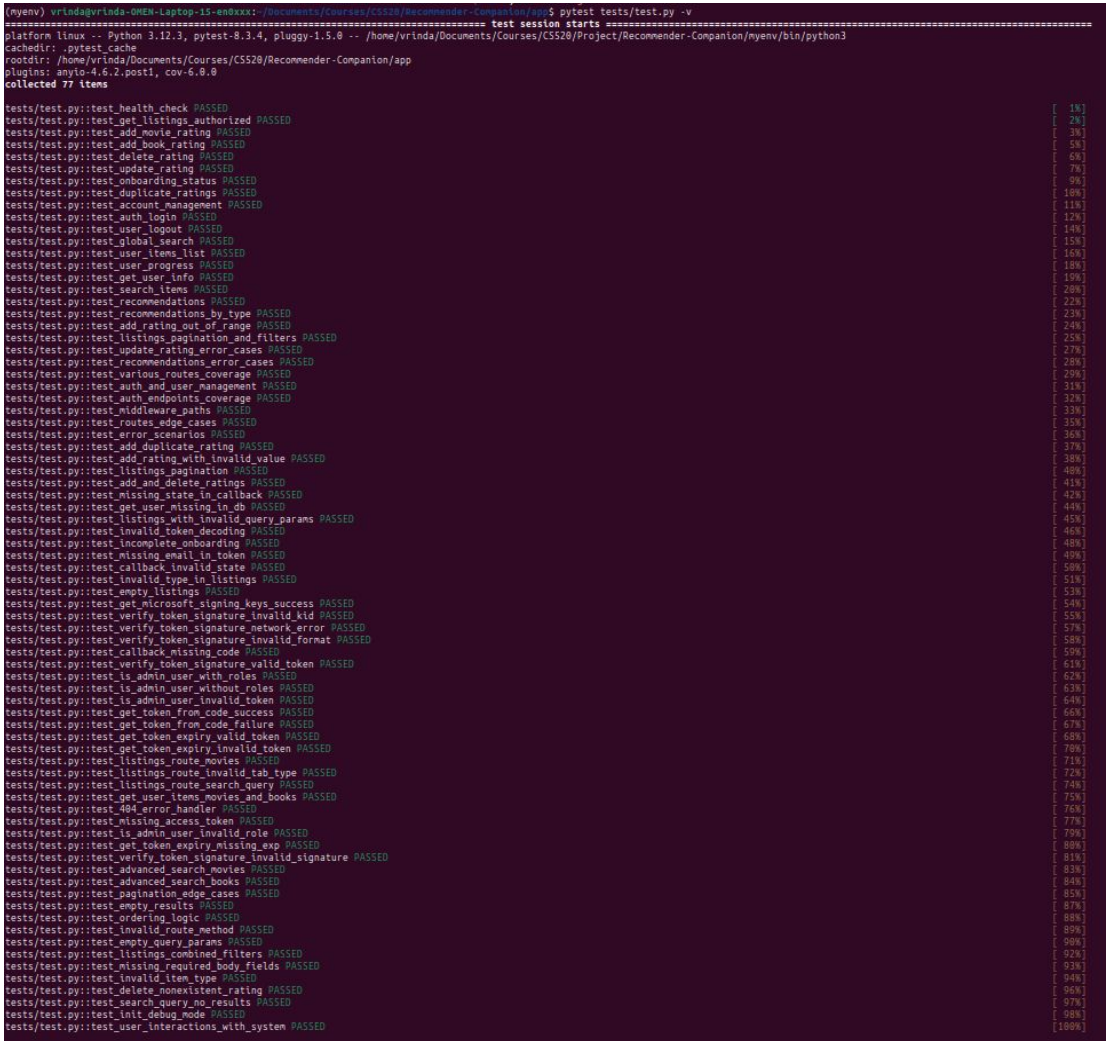
90% coverage achieved

```
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
ook', title: 'Test Book', description: 'A test book', confidence: 0.75, author: 'Test Author', genre: 'Fiction'}]
Chrome 132.0.0.0 (Linux x86_64): Executed 157 of 170 SUCCESS (0 secs / 1.007 secs)
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
ook', title: 'Test Book', description: 'A test book', confidence: 0.75, author: 'Test Author', genre: 'Fiction'}]
Chrome 132.0.0.0 (Linux x86_64): Executed 157 of 170 SUCCESS (0 secs / 1.007 secs)
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
ook', title: 'Test Book', description: 'A test book', confidence: 0.75, author: 'Test Author', genre: 'Fiction'}]
Chrome 132.0.0.0 (Linux x86_64): Executed 158 of 170 SUCCESS (0 secs / 1.011 secs)
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
ook', title: 'Test Book', description: 'A test book', confidence: 0.75, author: 'Test Author', genre: 'Fiction'}]
Chrome 132.0.0.0 (Linux x86_64): Executed 159 of 170 SUCCESS (0 secs / 1.015 secs)
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
ook', title: 'Test Book', description: 'A test book', confidence: 0.75, author: 'Test Author', genre: 'Fiction'}]
Chrome 132.0.0.0 (Linux x86_64): Executed 160 of 170 SUCCESS (0 secs / 1.019 secs)
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
ook', title: 'Test Book', description: 'A test book', confidence: 0.75, author: 'Test Author', genre: 'Fiction'}]
Chrome 132.0.0.0 (Linux x86_64): Executed 161 of 170 SUCCESS (0 secs / 1.023 secs)
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
ook', title: 'Test Book', description: 'A test book', confidence: 0.75, author: 'Test Author', genre: 'Fiction'}]
Chrome 132.0.0.0 (Linux x86_64): Executed 163 of 170 SUCCESS (0 secs / 1.028 secs)
LOG: 'Recommendations received:', [Object{id: 1, type: 'movie', title: 'Test Movie', description: 'A test movie', confidence: 0.85, cast: [..., ...], genre: 'Action'}, Object{id: 2, type: 'b
Chrome 132.0.0.0 (Linux x86_64): Executed 170 of 170 SUCCESS (1.342 secs / 1.089 secs)
TOTAL: 170 SUCCESS

============================== Coverage summary ==============================
Statements   : 90.55% ( 441/487 )
Branches     : 76.55% ( 111/145 )
Functions    : 91.32% ( 158/173 )
Lines        : 90.88% ( 429/472 )
==============================================================================
vrinda@vrinda-OMEN-Laptop-15-en0xxx:~/Documents/Courses/CS520/Recommender-Companion/frontend$
```

Backend Testing:

All test cases pass successfully!

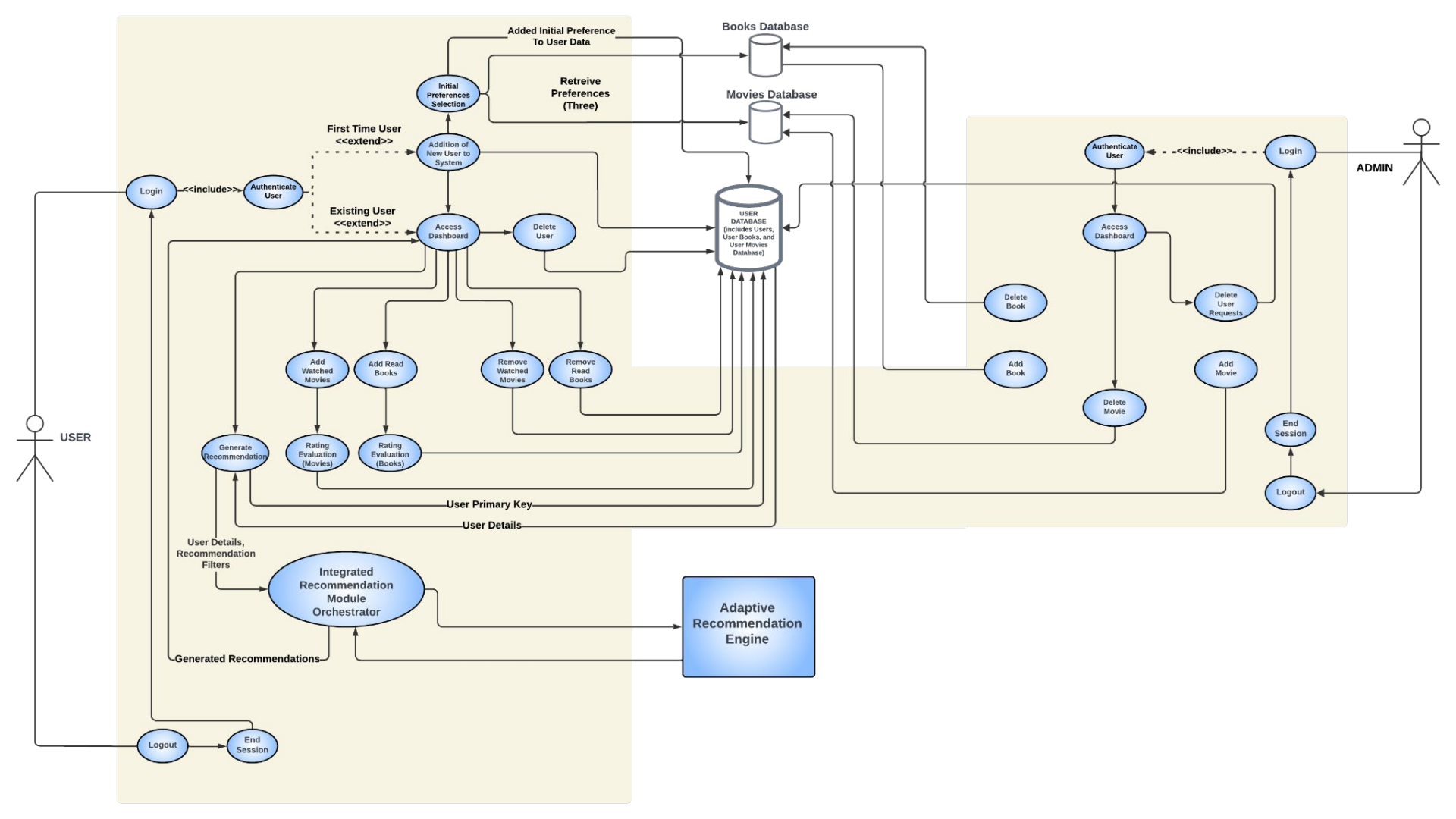


Backend Testing:

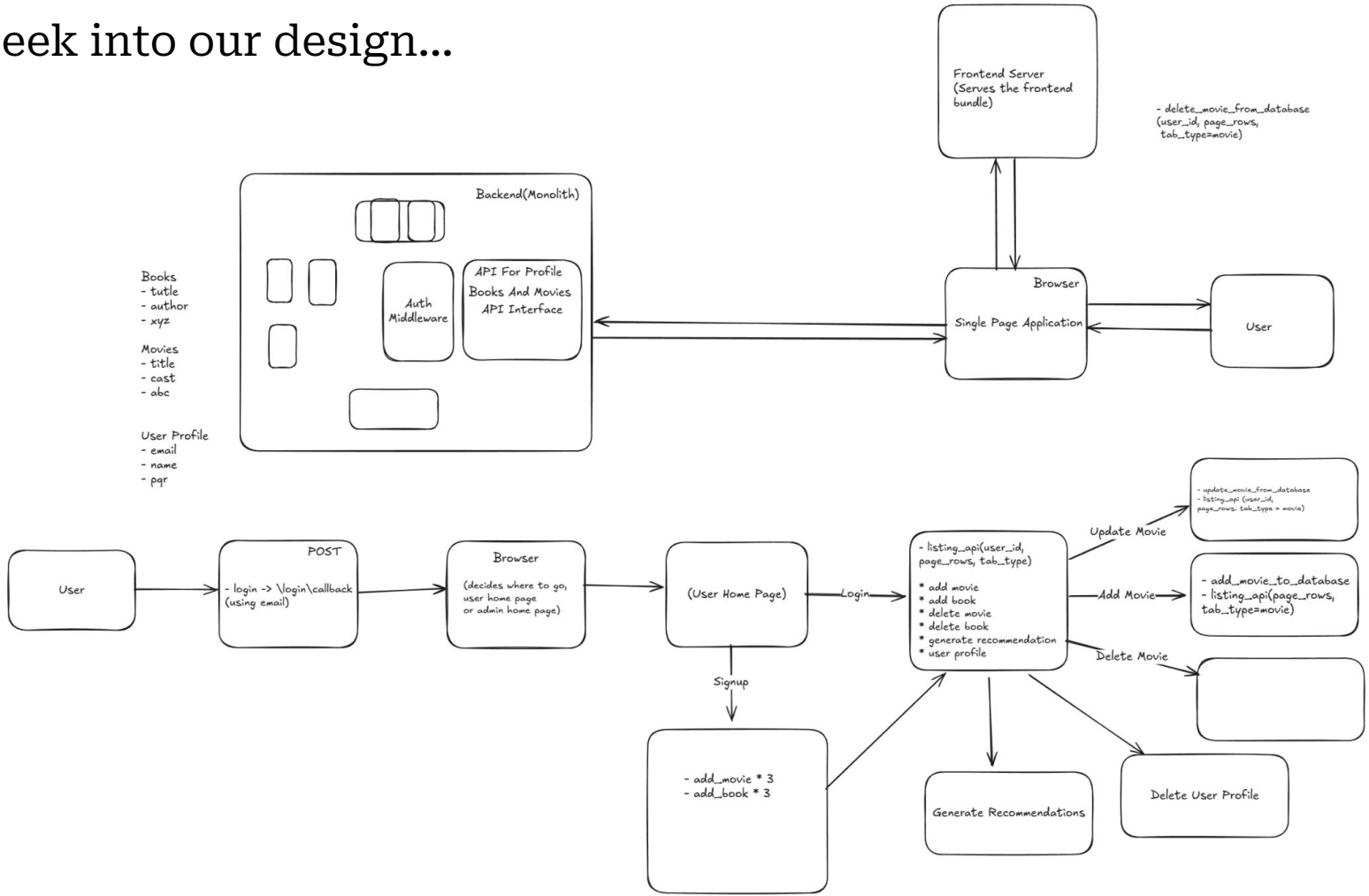
The test cases provide **over 90%** coverage. Some files were harder to cover, since their functionalities involved knowing information about GPT's internal tokenisation structure for mocking, which is not well documented.

```
---------- coverage: platform linux, python 3.12.3-final-0 -----------
Name                  Stmts   Miss  Cover
---------------------------------------
__init__.py              14      0   100%
auth.py                 169     36    79%
config.py                20      0   100%
extensions.py             2      0   100%
middleware.py            43      8    81%
models.py                47      2    96%
routes.py               253     79    69%
tests/__init__.py         0      0   100%
tests/test.py           740      3    99%
---------------------------------------
TOTAL                  1288    128    90%
```

Diagrams


Added Initial Preference To User Data
Books Database
Movies Database
Initial Preferences Selection
Retreive Preferences (Three)
First Time User <<extend>>
Addition of New User to System
Existing User <<extend>>
Login
<<include>>
Authenticate User
Access Dashboard
Delete User
USER DATABASE (includes Users, User Books, and User Movies Database)
Add Watched Movies
Add Read Books
Remove Watched Movies
Remove Read Books
Generate Recommendation
Rating Evaluation (Movies)
Rating Evaluation (Books)
User Primary Key
User Details
User Details, Recommendation Filters
Integrated Recommendation Module Orchestrator
Adaptive Recommendation Engine
Generated Recommendations
Logout
End Session
USER
Delete Book
Add Book
Authenticate User
<<include>>
Login
ADMIN
Access Dashboard
Delete User Requests
Add Movie
Delete Movie
End Session
Logout

A sneak peek into our design...



Links:

User Manual

Github Link