A Project Report

On

BOOK RECOMMENDATION SYSTEM USING MACHINE LEARNING

Submitted By

Mr. Samsuddin Khan (66012)

Mr. Vishal Kanojiya (66011)

In partial fulfilment for the award of the degree
Of

BACHELOR OF SCIENCE

In

COMPUTER SCIENCE

Under the guidance of

Mrs. Rashmi Waykole

DEPARTMENT OF COMPUTER SCIENCE

SONOPANT DANDEKAR ARTS, V.S. APTE COMMERCE & M.H.MEHTA SCIENCE COLLEGE

(Sem V)

(2024-2025)

SONOPANT DANDEKAR ARTS, V.S. APTE COMMERCE & M.H.MEHTA SCIENCE COLLEGE,

PALGHAR 401404

MAHARASHTRA

Department of Computer Science

CERTIFICATE

This is to certify that **Mr. Samsuddin Khan and Mr.Vishal Kanojiya** of T.Y.B.Sc. (SemV/VI) class has satisfactorily completed the Project **BOOK RECOMMENDATION SYSTEM USING MACHINE LEARNING** to be submitted in the partial fulfillment for the award of **Bachelor of Science** in **Computer Science** during the academic year 2024-2025.

Date of Submission:

Project Guide

Head/Incharge,

Department Computer Science

DECLARATION

I, **Mr.Samsuddin Khan and Mr.Vishal Kanojiya** hereby declare that the project entitled **“BOOK RECOMMENDATION SYSTEM USING MACHINE LEARNING”** submitted in the partial fulfilment for the award of **Bachelor of Science** in **Computer Science** during the academic year **2024-2025** is my original work and the project has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles.

Signature of the Student:

Place:

Date:

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who supported and guided me throughout the development of this project. Their valuable insights, expertise, and encouragement have been instrumental in the successful completion of this Book Recommender System.

First and foremost, I would like to thank my **Mes. Reshmi Waykole**, whose constant support and constructive feedback helped shape this project. Their advice and mentorship have been crucial throughout the entire process.

I am deeply grateful to **Sonopant Dandekar College** and the **Department of Computer Science** for providing me with the resources and learning environment necessary for developing my skills. Special thanks to my professors for their guidance and encouragement during my academic journey.

I would also like to thank **Kaggle** for providing access to the Book Recommendation Dataset, which was key to building and testing the machine learning algorithms used in this project.

Lastly, I am thankful to the various online resources and communities, including **YouTube tutorials** and **Stack Overflow**, for their valuable knowledge and technical assistance.

This project would not have been possible without the contributions of all these individuals and platforms. I am truly grateful for their support.

Mr. Samsuddin Khan

Mr. Vishal Kanojiya

Table of Contents

Introduction

This project focuses on developing a **Book Recommender System** that provides personalized book recommendations based on user preferences and historical data. Leveraging machine learning algorithms like **collaborative filtering** and **content-based filtering**, the system suggests books tailored to individual users. The project utilizes a dataset containing user ratings, book information, and interaction history to train the model.

1.1.Problem Statement

In the modern digital age, online platforms provide users with a massive amount of information and content, from books and movies to music and products. With the increasing amount of data, finding relevant content for users becomes a complex task. Recommender systems have emerged as powerful tools to solve this challenge by providing personalized suggestions based on user preferences and behavior.

This project focuses on developing a **Book Recommender System** using machine learning techniques. The recommender system is designed to suggest books to users based on their past preferences and the preferences of similar users. The project implements two major types of recommendation approaches: **Collaborative Filtering** and **Content-Based Filtering**. It is built using Python, Flask for the web interface, and deployed on Render, making it accessible to users through a web-based interface.

The system uses three datasets: **Books.csv**, **Users.csv**, and **Ratings.csv**, which store information about books, users, and the ratings provided by users for specific books, respectively. These datasets are critical in training machine learning models to generate relevant recommendations.

In this project documentation, we will explore the system's architecture, methodology, datasets, machine learning models, web application development, deployment process, and results.

1.2 Objectives

The primary objectives of this project are:

- To develop a system that can recommend books to users based on their preferences and interests.
- To implement two key recommendation algorithms: **Collaborative Filtering** and **Content-Based Filtering**.
- To build a web application using Flask that provides a user-friendly interface for interacting with the recommendation system.
- To deploy the system on a platform like **Render** to make it accessible online.

1.3 Methodology

The project is divided into several phases, each focused on different aspects of system development and deployment. Below is an overview of the methodology:

1.3.1 Data Collection and Preparation

The datasets used in this project are:

- **Books.csv**: Contains metadata about books such as title, author, and genre.
- **Users.csv**: Contains user information such as user IDs and demographic data.
- **Ratings.csv**: Contains the ratings provided by users for various books.

The datasets are preprocessed to ensure that they are in a suitable format for training machine learning models. Missing or irrelevant data is removed, and the datasets are cleaned for consistency.

1.3.2 Recommender System Algorithms

Two major types of recommendation algorithms are implemented in this system:

- **Collaborative Filtering**: This method recommends items (books) based on the preferences of similar users. It looks at patterns in the ratings provided by users and suggests items that users with similar preferences have liked.
- **Content-Based Filtering**: This method recommends items based on the content of the items and a profile of the user's preferences. For example, if a user has liked books in a specific genre, the system will recommend books in that genre.

1.3.3 Model Training and Evaluation

Machine learning models are trained using the preprocessed datasets to generate recommendations. The models are evaluated based on metrics such as accuracy and user satisfaction.

1.3.4 Web Application Development

The system's front-end is built using **Flask**, a lightweight web framework. The user interface consists of templates like **index.html** and **recommend.html**, where users can interact with the system, input their preferences, and view recommendations.

1.3.5 Deployment

The system is deployed on **Render**, a cloud platform that allows for the deployment of web applications. This makes the recommender system accessible online, enabling users to access it from anywhere and receive book recommendations.

1.4 System Architecture

The system consists of three major components:

- **Data Layer**: This includes the datasets (Books.csv, Users.csv, Ratings.csv) and the machine learning models that process the data.
- **Logic Layer**: The core algorithms (collaborative filtering and content-based filtering) are implemented here. This layer processes user input and generates recommendations.
- **Presentation Layer**: The web interface developed using Flask provides the front-end for users to interact with the system.

1.5 Results and Discussion

The recommender system provides users with relevant book recommendations based on their input or past behavior. Both collaborative filtering and content-based filtering approaches have been successfully implemented and integrated into the system.

Key features of the system include:

- The ability to recommend books to users based on their preferences and the preferences of similar users.
- A simple and intuitive web interface that allows users to easily interact with the system and receive recommendations.
- The flexibility of the system, allowing for the addition of new books and user preferences dynamically.

Challenges faced during the project included:

- Handling large datasets and optimizing the algorithms for faster performance.
- Balancing accuracy with the complexity of the models to ensure that recommendations are both relevant and computationally efficient.

2. Features

2.1 User-Friendly Web Interface

- **Simple Navigation**: The system is built with a clear, intuitive web interface using Flask, making it easy for users to navigate through the site.
- **Responsive Design**: The front-end design adapts to various devices, offering a seamless experience on both desktop and mobile platforms.
- **Input Flexibility**: Users can provide input based on either preferences, to receive book recommendations.

2.2 Personalized Book Recommendations

- **Collaborative Filtering**: Users are provided with personalized recommendations based on the preferences and behaviors of other users with similar tastes. This feature ensures that users discover books that they are more likely to enjoy.
- **Content-Based Filtering**: This method recommends books based on the attributes of the books themselves, such as genre, author, and description. Users will receive suggestions that match their specific preferences.
- **Hybrid Approach**: The system combines both collaborative and content-based filtering approaches, providing even more accurate and personalized recommendations.

2.3 Dataset Integration

- **Extensive Book Metadata**: The system utilizes the **Books.csv** dataset, which contains detailed information about various books, such as titles, authors, publishers, and genres.
- **User Demographics**: The **Users.csv** dataset provides insights into the demographics of users, allowing the system to customize recommendations based on age, location, or other user-specific details.
- **Ratings Dataset**: The **Ratings.csv** dataset captures user ratings of books, providing the core data needed for collaborative filtering algorithms.

2.4 Machine Learning Models

- **Collaborative Filtering Model**: This model uses the matrix factorization technique to predict user preferences based on previous ratings. It focuses on recommending books that other users with similar tastes have liked.

- **Content-Based Model**: The content-based model uses feature extraction techniques to analyze book descriptions, genres, and authors, making recommendations based on similar books.
- **Scalability**: The models are designed to handle increasing amounts of data efficiently, enabling the system to scale and accommodate more books and users over time.

2.5 Search and Filter Options

- **Advanced Search**: Users can search for books based on specific criteria such as title, author, or genre, providing a personalized experience even before interacting with recommendations.
- **Filter by Genre**: Users can filter their recommended books based on genres like fiction, mystery, sci-fi, and more, giving them the flexibility to explore recommendations within their preferred category.

2.6 Flask Integration for Web Deployment

- **Interactive User Experience**: Flask's lightweight framework allows for the creation of a dynamic web interface where users can receive book suggestions, interact with the system, and explore recommendations.
- **Secure User Input**: The system ensures that all user interactions are securely handled, with input validations and protections against common web vulnerabilities.

2.7 Extensibility and Flexibility

- **Easily Expandable Dataset**: New books and users can be added to the dataset without disrupting the existing system, ensuring that the recommender stays up-to-date with the latest book releases.
- **Modular Design**: The system is designed in a modular fashion, making it easy to upgrade individual components such as adding more advanced machine learning algorithms or integrating additional data sources.

2.8 Performance Optimization

- **Efficient Data Processing**: The system employs optimized machine learning models that process data quickly, allowing for fast recommendation generation even with large datasets.
- **Caching Mechanisms**: Key recommendations are cached to improve response times and reduce the load on the system for frequently accessed book suggestions.

3. Datasets

The Book Recommender System utilizes three primary datasets to provide personalized book recommendations to users. These datasets are crucial for training the machine learning models and ensuring accurate predictions. Below is a detailed description of each dataset used in the project:

3.1 Books Dataset (Books.csv)

- **Description**: The Books.csv dataset contains information about a wide range of books available for recommendation. It serves as the foundational dataset for content-based filtering.
- **Key Attributes**:
  - **Book ID (ISBN)**: A unique identifier for each book.
  - **Title (Book-Title)**: The title of the book.
  - **Author (Book-Author)**: The name(s) of the author(s) of the book.
  - **Publisher**: The publishing company for the book.
  - **Genre**: The genre/category of the book (e.g., Fiction, Non-Fiction, Mystery, Science Fiction).
  - **Description**: A brief summary or description of the book's content.
  - **Publication Year (Year-Of-Publication)**: The year the book was published.
  - **Language**: The language in which the book is written.
  - **Image URLs**: Links to images of the book cover in different sizes (Image-URL-S, Image-URL-M, Image-URL-L).
- **Usage**: This dataset is used for content-based filtering, allowing the system to recommend books similar to those a user has already liked based on attributes like genre, author, and description.
- **Attributes Overview**:

```
Index(['ISBN', 'Book-Title', 'Book-Author', 'Year-Of-Publication', 'Publisher',
       'Image-URL-S', 'Image-URL-M', 'Image-URL-L'], dtype='object')
```

3.2 Users Dataset (Users.csv)

- **Description**: The Users.csv dataset contains information about the users of the recommender system. This data is essential for understanding user demographics and preferences.
- **Key Attributes**:
  - **User ID (User-ID)**: A unique identifier for each user.
  - **Username**: The name chosen by the user for the application (not explicitly listed but typically included).
  - **Age**: The age of the user, which may help in understanding reading preferences.

    - **Gender**: The gender of the user, which can be used to tailor recommendations (not explicitly listed but typically included).
    - **Location**: The geographical location of the user.
    - **Registration Date**: The date when the user registered on the platform (not explicitly listed but typically included).
- **Usage**: This dataset is used to personalize recommendations by considering the demographic information of users and tailoring suggestions based on similar users' reading habits.
- **Attributes Overview**:

  Index(['User-ID', 'Location', 'Age'], dtype='object')

3.3 Ratings Dataset (Ratings.csv)

- **Description**: The Ratings.csv dataset captures the ratings provided by users for various books. This dataset is vital for collaborative filtering, which relies on user behavior to generate recommendations.
- **Key Attributes**:
    - **User ID (User-ID)**: A unique identifier for the user who provided the rating.
    - **Book ID (ISBN)**: A unique identifier for the book that was rated.
    - **Rating (Book-Rating)**: The rating given by the user, typically on a scale (e.g., 1 to 5).
    - **Timestamp**: The time when the rating was provided, which can be used to analyze trends over time (not explicitly listed but typically included).
- **Usage**: This dataset is primarily used for collaborative filtering algorithms, allowing the system to recommend books based on the ratings given by similar users. It plays a crucial role in understanding user preferences and behaviors.
- **Attributes Overview**:

  Index(['User-ID', 'ISBN', 'Book-Rating'], dtype='object')

.

4. Requirements

The Book Recommender System requires specific software, hardware, and dependency configurations to function effectively. Below is a detailed overview of the necessary requirements to set up and run the project successfully.

4.1 Software Requirements

1. **Programming Language**:
   - **Python** (Version 3.6 or higher)
     - Python serves as the primary programming language for developing the recommender system.
2. **Web Framework**:
   - **Flask** (Version 2.0 or higher)
     - A lightweight web framework used for building the web interface of the application.
3. **Data Science Libraries**:
   - **Pandas** (Version 1.2 or higher)
     - Utilized for data manipulation and analysis.
   - **NumPy** (Version 1.19 or higher)
     - Used for numerical computations and array handling.
   - **Scikit-learn** (Version 0.24 or higher)
     - Provides tools for implementing machine learning algorithms for recommendations.
   - **Seaborn** (Version 0.11 or higher)
     - For advanced data visualization and graphing capabilities.
4. **Database**:
   - **SQLite** (or any other relational database)
     - Used to store user information, book details, and ratings.
5. **Front-End Technologies**:
   - **HTML/CSS**:
     - For designing and styling the user interface of the web application.
6. **Development Environment**:
   - **Anaconda** or **Virtual Environment** (recommended)
     - To manage dependencies and create isolated environments for the project.
   - **Jupyter Notebook** (optional)
     - Ideal for prototyping and data exploration tasks.

4.2 Hardware Requirements

1. **Minimum System Specifications**:

  - **Processor**: Intel i3 or equivalent (or higher)
  - **RAM**: At least 4 GB
  - **Storage**: At least 500 MB of free disk space for the application and datasets.
2. **Recommended System Specifications**:
  - **Processor**: Intel i5 or equivalent (or higher)
  - **RAM**: 8 GB or more
  - **Storage**: 1 GB or more for handling larger datasets.

4.3 Installation Requirements

1. **Environment Setup**: To set up the environment for your Book Recommender System project, follow these steps

Create a virtual environment (recommended) using the following commands:

Run this command to create the environment:
```
python -m venv recommender_env
```

Activate the environment:
```
source recommender_env/bin/activate  # For Linux/Mac

recommender_env\Scripts\activate  # For Windows
```

4.4 Install Required Packages:

Install the necessary libraries using pip:

```
pip install Flask
pip install pandas
pip install numpy
pip install pickle
```

Database Setup:
  - Set up the SQLite database by creating a new database file and initializing it with the necessary tables (users, books, ratings).

4.5 Deployment Requirements

1. **Deployment Platform**:

- **Render** or any other cloud platform (such as Heroku, AWS, or Google Cloud) to deploy the web application.

2. **Version Control**:
    - **Git**: For version control and managing project changes.
3. **Configuration Files**:
    - **requirements.txt**: List of required packages for easy installation.
    - **Procfile**: Configuration file for deployment specifying the web server to run.

5. Installation and Setup

Setting up the **Book Recommender System** involves several key steps, including installing required software, configuring the environment, and preparing the project for execution. This section provides a comprehensive guide to getting the application up and running.

5.1 Prerequisites

Before proceeding with the installation, ensure that the following prerequisites are met:

1. **Python** (Version 3.6 or higher) is installed on your system.
2. A package manager like **pip** should be available for installing Python packages.
3. Optional: **Git** should be installed if you plan to clone the repository or track your project changes.

You can check your Python and pip installations by running the following commands in your terminal or command prompt:

```
python --version
pip --version
```

If Python is not installed, you can download it from the official Python website.

5.2 Setting Up a Virtual Environment

To keep dependencies organized and avoid conflicts with other projects, it's recommended to use a virtual environment. Follow these steps to create and activate a virtual environment:

1. **Create a Virtual Environment**:

Open your terminal and navigate to the directory where you want to create your project. Run the following command:

```
python -m venv recommender_env
```

1. **Activate the Virtual Environment**:

For **Windows**:

```
recommender_env\Scripts\activate
```

For **macOS/Linux**:

```
source recommender_env/bin/activate
```

3. Once activated, your terminal prompt should change to indicate that you are now working inside the virtual environment.

5.3 Installing Required Packages

With the virtual environment activated, install the necessary Python packages for the project using **pip**. Create a requirements.txt file (if not already provided) in your project directory and include the following packages:

Shell:

```
Flask>=2.0
pandas>=1.2
numpy>=1.19
```

You can install all required packages at once using the following command:

```
pip install -r requirements.txt
```

Alternatively, if you prefer to install packages one by one, you can use:

```
pip install Flask pandas numpy
```

5.4 Project Structure

Ensure your project directory has the following structure:

```
book_recommender/
│
├── recommender_env/        # Virtual environment folder
│
├── templates/              # HTML templates for the web application
│   ├── index.html          # Homepage
│   ├── recommend.html      # Recommendations page
│
├── app.py                  # Main application script
├── requirements.txt        # List of dependencies
├── Procfile                # Deployment configuration
└── README.md               # Project documentation
```

6. How to Run the Project

Running the Book Recommender System is a straightforward process that involves ensuring your environment is correctly set up and executing the application script. This section outlines the steps necessary to launch the project locally and access its features.

6.1 Prerequisites

Before running the project, ensure the following:

1. Python (Version 3.6 or higher) is installed and configured on your system.
2. The virtual environment is activated, and all required packages have been installed as detailed in the previous sections.
3. The SQLite database is created and initialized with necessary tables, as mentioned in the installation setup.

6.2 Activating the Virtual Environment

To run the application, you must first activate your virtual environment. Open your terminal or command prompt and navigate to the directory where your project is located. Use the following command based on your operating system:

- **For Windows**:

```
recommender_env\Scripts\activate
```

- **For macOS/Linux**:

```
source recommender_env/bin/activate
```

You should see the environment name (e.g., (recommender_env)) prefixed in your terminal prompt, indicating that the virtual environment is active.

6.3 Running the Application

- **Navigate to the Project Directory**:
  Make sure you are in the root directory of your project where the app.py file is located. Use the cd command to navigate if needed:

```
cd path_to_your_project_directory/book_recommender
```

1. **Run the Application**:
   With the virtual environment activated and in the project directory, run the following command:

```
python app.py
```

If everything is set up correctly, you should see output similar to the following:

```
Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

This indicates that the Flask development server is running and the application is ready to be accessed.

6.4 Accessing the Application

1. **Open a Web Browser**:
   Open your preferred web browser (e.g., Chrome, Firefox, Safari).
2. **Navigate to the Application URL**:
   In the address bar, type the following URL and press Enter:

```
http://127.0.0.1:5000
```

You should see the homepage of the Book Recommender System, where you can start interacting with the application.

6.5 Interacting with the Application

Once the application is running, you can perform various actions:

- **View Books**: Access the book catalog to browse available titles.
- **Get Recommendations**: Based on user ratings, the system will provide personalized book recommendations.

7. Explanation of the Code

This section provides an overview of the key components and code structure of the Book Recommender System. Understanding how the code is organized and its functionality will help in maintaining, updating, and extending the application in the future.

7.1 Jupyter Notebook Code

```
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
import pickle

# Load the datasets
books = pd.read_csv('Books.csv', sep=";", on_bad_lines='skip', encoding='latin-1')
users = pd.read_csv('Users.csv', sep=";", on_bad_lines='skip', encoding='latin-1')
ratings = pd.read_csv('Ratings.csv', sep=";", on_bad_lines='skip', encoding='latin-1')

# Checking for missing values and duplicates
print(books.isnull().sum())
print(users.isnull().sum())
print(ratings.isnull().sum())

print(books.duplicated().sum())
print(ratings.duplicated().sum())
print(users.duplicated().sum())

# Merge ratings with book titles
ratings_with_name = ratings.merge(books, on='ISBN')

# Convert 'Book-Rating' to numeric, coercing non-numeric values to NaN, and remove NaN
values
ratings_with_name['Book-Rating'] = pd.to_numeric(ratings_with_name['Book-Rating'],
errors='coerce')
ratings_with_name.dropna(subset=['Book-Rating'], inplace=True)
ratings_with_name['Book-Rating'] = ratings_with_name['Book-Rating'].astype(float)

# Number of ratings per book
num_rating_df = ratings_with_name.groupby('Book-Title').count()['Book-
Rating'].reset_index()
num_rating_df.rename(columns={'Book-Rating': 'num_ratings'}, inplace=True)

# Average rating per book
```

```
avg_rating_df = ratings_with_name.groupby('Book-Title')['Book-
Rating'].mean().reset_index()
avg_rating_df.rename(columns={'Book-Rating': 'avg_rating'}, inplace=True)

# Merge the number of ratings and average rating
popular_df = num_rating_df.merge(avg_rating_df, on='Book-Title')

# Filter for books with more than 250 ratings and sort by rating
popular_df = popular_df[popular_df['num_ratings'] >= 250].sort_values('avg_rating',
ascending=False).head(50)

# Merge with the books dataframe to get author and image information
popular_df = popular_df.merge(books, on='Book-Title').drop_duplicates('Book-Title')
popular_df = popular_df[['Book-Title', 'Book-Author', 'Image-URL-M', 'num_ratings',
'avg_rating']]

# Round the average rating to 1 decimal place
popular_df['avg_rating'] = popular_df['avg_rating'].round(1)

# Collaborative Filtering: Identify users with more than 200 ratings
x = ratings_with_name.groupby('User-ID').count()['Book-Rating'] > 200
padhe_likhe_users = x[x].index

# Filter ratings for these users and for books with more than 50 ratings
filtered_rating = ratings_with_name[ratings_with_name['User-ID'].isin(padhe_likhe_users)]
y = filtered_rating.groupby('Book-Title').count()['Book-Rating'] >= 50
famous_books = y[y].index

# Create a pivot table for collaborative filtering
final_ratings = filtered_rating[filtered_rating['Book-Title'].isin(famous_books)]
pt = final_ratings.pivot_table(index='Book-Title', columns='User-ID', values='Book-Rating')
pt.fillna(0, inplace=True)

# Calculate similarity scores using cosine similarity
similarity_scores = cosine_similarity(pt)

# Recommendation function with exception handling
def recommend(book_name):
    # Check if the book exists in the pivot table
    if book_name in pt.index:
        index = np.where(pt.index == book_name)[0][0]
        similar_items = sorted(list(enumerate(similarity_scores[index])), key=lambda x: x[1],
reverse=True)[1:5]
```

```
        data = []
        for i in similar_items:
            temp_df = books[books['Book-Title'] == pt.index[i[0]]]
            if not temp_df.empty:
                item = []
                item.extend(list(temp_df.drop_duplicates('Book-Title')['Book-Title'].values))
                item.extend(list(temp_df.drop_duplicates('Book-Title')['Book-Author'].values))
                item.extend(list(temp_df.drop_duplicates('Book-Title')['Image-URL-M'].values))
                data.append(item)

        return data
    else:
        return f"'{book_name}' not found in the dataset."

# Example Recommendation
print(recommend('1984'))

# Save the data and models using pickle
pickle.dump(popular_df, open(r'C:\Users\Welcome\Documents\Book recommendetion
system\popular.pkl', 'wb'))
pickle.dump(pt, open(r'C:\Users\Welcome\Documents\Book recommendetion system\pt.pkl',
'wb'))
pickle.dump(books, open(r'C:\Users\Welcome\Documents\Book recommendetion
system\books.pkl', 'wb'))
pickle.dump(similarity_scores, open(r'C:\Users\Welcome\Documents\Book recommendetion
system\similarity_scores.pkl', 'wb'))
```

7.1.1 Import Libraries

```
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
import pickle
```

- **Purpose:** Import necessary libraries for data manipulation, similarity calculations, and model saving.

7.1.2 Load the Datasets

```
books = pd.read_csv('Books.csv', sep=";", on_bad_lines='skip', encoding='latin-1')
users = pd.read_csv('Users.csv', sep=";", on_bad_lines='skip', encoding='latin-1')
ratings = pd.read_csv('Ratings.csv', sep=";", on_bad_lines='skip', encoding='latin-1')
```

- **Purpose:** Load Books, Users, and Ratings datasets from CSV files with specific parameters to handle formatting and bad lines.

7.1.3 Check for Missing Values and Duplicates

```
print(books.isnull().sum())
print(users.isnull().sum())
print(ratings.isnull().sum())

print(books.duplicated().sum())
print(ratings.duplicated().sum())
print(users.duplicated().sum())
```

- **Purpose:** Identify any missing values and duplicates in each dataset for data quality checks.

7.1.4 Merge Ratings with Book Titles

```
ratings_with_name = ratings.merge(books, on='ISBN')
```

- **Purpose:** Merge ratings with corresponding book titles using the ISBN identifier.

7.1.5 Clean the Ratings Data

```
ratings_with_name['Book-Rating'] = pd.to_numeric(ratings_with_name['Book-Rating'],
errors='coerce')
ratings_with_name = ratings_with_name.dropna(subset=['Book-Rating']).astype({'Book-
Rating': 'float'})
```

- **Purpose:** Convert the Book-Rating column to numeric format, remove rows with NaN ratings, and ensure the ratings are in float format.

7.1.6 Calculate Number and Average Ratings per Book

```
num_rating_df = ratings_with_name.groupby('Book-Title').count()['Book-
Rating'].reset_index()
num_rating_df.rename(columns={'Book-Rating': 'num_ratings'}, inplace=True)

avg_rating_df = ratings_with_name.groupby('Book-Title')['Book-
Rating'].mean().reset_index()
avg_rating_df.rename(columns={'Book-Rating': 'avg_rating'}, inplace=True)

popular_df = num_rating_df.merge(avg_rating_df, on='Book-Title')
```

- **Purpose:** Create dataframes to count the number of ratings and calculate the average rating per book, then merge them.

7.1.7 Filter and Sort Popular Books

```
popular_df = popular_df[popular_df['num_ratings'] >= 250].sort_values('avg_rating',
ascending=False).head(50)
```

- **Purpose:** Filter the merged dataframe to include only books with more than 250 ratings and sort by average rating, selecting the top 50.

7.1.8 Merge with Book Information

```
popular_df = popular_df.merge(books, on='Book-Title').drop_duplicates('Book-Title')
popular_df = popular_df[['Book-Title', 'Book-Author', 'Image-URL-M', 'num_ratings',
'avg_rating']]
popular_df['avg_rating'] = popular_df['avg_rating'].round(1)
```

- **Purpose:** Merge with the books dataframe to include author and image information, and retain only relevant columns.

7.1.9 Collaborative Filtering Preparation

```
x = ratings_with_name.groupby('User-ID').count()['Book-Rating'] > 200
padhe_likhe_users = x[x].index

filtered_rating = ratings_with_name[ratings_with_name['User-ID'].isin(padhe_likhe_users)]
y = filtered_rating.groupby('Book-Title').count()['Book-Rating'] >= 50
famous_books = y[y].index
```

- **Purpose:** Identify users with more than 200 ratings and filter ratings for these users, selecting books rated by at least 50 users.

7.1.10 Create a Pivot Table

```
final_ratings = filtered_rating[filtered_rating['Book-Title'].isin(famous_books)]
pt = final_ratings.pivot_table(index='Book-Title', columns='User-ID', values='Book-Rating')
pt.fillna(0, inplace=True)
```

- **Purpose:** Construct a pivot table where rows are book titles, columns are user IDs, and values are the respective ratings, filling missing values with 0.

7.1.11 Calculate Similarity Scores

```
similarity_scores = cosine_similarity(pt)
```

- **Purpose:** Compute cosine similarity scores between books based on the user ratings pivot table.

7.1.12 Recommendation Function

```
def recommend(book_name):
    if book_name in pt.index:
        index = np.where(pt.index == book_name)[0][0]
        similar_items = sorted(list(enumerate(similarity_scores[index])), key=lambda x: x[1],
reverse=True)[1:5]
```

```
        data = []
        for i in similar_items:
            temp_df = books[books['Book-Title'] == pt.index[i[0]]]
            if not temp_df.empty:
                item = []
                item.extend(list(temp_df.drop_duplicates('Book-Title')['Book-Title'].values))
                item.extend(list(temp_df.drop_duplicates('Book-Title')['Book-Author'].values))
                item.extend(list(temp_df.drop_duplicates('Book-Title')['Image-URL-M'].values))
                data.append(item)

        return data
    else:
        return f"'{book_name}' not found in the dataset."
```

- **Purpose:** Define a function that takes a book title as input and returns the top 4 similar books based on similarity scores.

7.1.13 Example Recommendation

```
print(recommend('1984'))
```

- **Purpose:** Test the recommendation function using the book title '1984' to demonstrate the functionality.

7.1.14 Saving the Data and Models

```
pickle.dump(popular_df, open(r'C:\Users\Welcome\Documents\Book recommendetion system\popular.pkl', 'wb'))
pickle.dump(pt, open(r'C:\Users\Welcome\Documents\Book recommendetion system\pt.pkl', 'wb'))
pickle.dump(books, open(r'C:\Users\Welcome\Documents\Book recommendetion system\books.pkl', 'wb'))
pickle.dump(similarity_scores, open(r'C:\Users\Welcome\Documents\Book recommendetion system\similarity_scores.pkl', 'wb'))
```

- **Purpose:** Save the processed dataframes and similarity scores to pickle files for efficient loading in future sessions.

7.2 Main Application (app.py)

```
from flask import Flask, render_template, request
import pickle
import numpy as np
```

```
# Load the pickled data
popular_df = pickle.load(open('popular.pkl', 'rb'))
pt = pickle.load(open('pt.pkl', 'rb'))
books = pickle.load(open('books.pkl', 'rb'))  # Fixed typo
similarity_scores = pickle.load(open('similarity_scores.pkl', 'rb'))

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html',
                           book_name=list(popular_df['Book-Title'].values),
                           author=list(popular_df['Book-Author'].values),
                           image=list(popular_df['Image-URL-M'].values),
                           votes=list(popular_df['num_ratings'].values),
                           rating=list(popular_df['avg_rating'].values)
                           )

@app.route('/recommend')
def recommend_ui():
    return render_template('recommend.html')

@app.route('/recommend_books', methods=['POST'])
def recommend():
    user_input = request.form.get('user_input')

    # Step 1: Search the entire `books` dataset for the book title
    searched_book = books[books['Book-Title'].str.contains(user_input, case=False,
na=False)]

    # If book is found in the books dataset
    if not searched_book.empty:
        # Fetch book details
        book_title = searched_book['Book-Title'].values[0]
        book_author = searched_book['Book-Author'].values[0]
        book_image = searched_book['Image-URL-M'].values[0]

        # Step 2: Display the found book
        data = [[book_title, book_author, book_image]]

        # Step 3: Also suggest similar books if the book is present in the pivot table (`pt`)
        if book_title in pt.index:
            index = np.where(pt.index == book_title)[0][0]
            similar_items = sorted(list(enumerate(similarity_scores[index])), key=lambda x: x[1],
reverse=True)[1:5]
            for i in similar_items:
                temp_df = books[books['Book-Title'] == pt.index[i[0]]]
                if not temp_df.empty:
                    similar_book = []
                    similar_book.extend(list(temp_df.drop_duplicates('Book-Title')['Book-
Title'].values))
                    similar_book.extend(list(temp_df.drop_duplicates('Book-Title')['Book-
Author'].values))
                    similar_book.extend(list(temp_df.drop_duplicates('Book-Title')['Image-URL-
M'].values))
                    data.append(similar_book)

        return render_template('recommend.html', data=data)

    else:
        # If book is not found in the dataset
        return render_template('recommend.html', message="Book not found. Please try another
book.")

if __name__ == '__main__':
    app.run(debug=True)
```

7.2.1 Importing Required Libraries

```
from flask import Flask, render_template, request
import pickle
import numpy as np
```

1. **Flask**: A lightweight web framework used to build web applications. render_template is used to render HTML templates, and request is used to handle form inputs from the user.

2. **pickle**: Used for loading pre-saved Python objects such as datasets and models.

3. **numpy**: A numerical library used here to work with arrays for recommendation calculations.

7.2.2 Loading the Pickle Files

```
popular_df = pickle.load(open('popular.pkl', 'rb'))
```

```
pt = pickle.load(open('pt.pkl', 'rb'))
books = pickle.load(open('books.pkl', 'rb'))
similarity_scores = pickle.load(open('similarity_scores.pkl', 'rb'))
```

1. **popular.pkl**: Contains a DataFrame with the most popular books, including their title, author, image URL, and ratings.
2. **pt.pkl**: This is a pivot table created during collaborative filtering, where rows represent books and columns represent user ratings.
3. **books.pkl**: Contains the complete dataset of books, including metadata like title, author, and image URL.
4. **similarity_scores.pkl**: Contains pre-calculated similarity scores for books based on user ratings or content features, which will be used to suggest similar books.

7.2.3 Setting up the Flask Application

```
app = Flask(__name__)
```

1. This initializes the Flask app instance, making it the central object for your web application.

7.2.4 Index Route (Homepage)

```
@app.route('/')
def index():
    return render_template('index.html',
                        book_name=list(popular_df['Book-Title'].values),
                        author=list(popular_df['Book-Author'].values),
                        image=list(popular_df['Image-URL-M'].values),
                        votes=list(popular_df['num_ratings'].values),
                        rating=list(popular_df['avg_rating'].values)
                        )
```

1. This defines the homepage of the application (/ route). When accessed, it renders index.html, passing the following data:
   - **book_name**: List of popular book titles from popular_df.
   - **author**: List of authors of these books.
   - **image**: Image URLs of the books.

  - **votes**: Number of user ratings for each book.
  - **rating**: Average ratings for each book.

2. These values are passed into the HTML template for display.

7.2.5 Recommendation Page Route (UI)

```
@app.route('/recommend')

def recommend_ui():

    return render_template('recommend.html')
```

1. This route simply renders the recommend.html template, which provides a search interface for users to input a book title for recommendations.

7.2.6 Book Recommendation Logic

```
@app.route('/recommend_books', methods=['POST'])

def recommend():

    user_input = request.form.get('user_input')
```

1. This route handles the POST request from the user when they submit a book title for recommendation. It retrieves the user’s input from the form using request.form.get('user_input').

7.2.7 Searching for the Book in the Dataset

```
searched_book = books[books['Book-Title'].str.contains(user_input, case=False, na=False,
regex=False)]
```

1. It checks if the book title provided by the user exists in the books dataset by searching the Book-Title column using str.contains() for a case-insensitive match.

7.2.8 If Book is Found

```
if not searched_book.empty:
    book_title = searched_book['Book-Title'].values[0]
    book_author = searched_book['Book-Author'].values[0]
    book_image = searched_book['Image-URL-M'].values[0]
```

1. If the book is found, it extracts the book’s title, author, and image URL and stores them in book_title, book_author, and book_image, respectively.

7.2.9 Suggesting Similar Books

```
if book_title in pt.index:
    index = np.where(pt.index == book_title)[0][0]
    similar_items = sorted(list(enumerate(similarity_scores[index])), key=lambda x: x[1],
reverse=True)[1:5]
```

1. If the searched book exists in the pivot table (pt), it finds the index of the book and retrieves similar books based on pre-calculated similarity scores from similarity_scores.
2. **sorted()** sorts the similar books in descending order of similarity, and the top 4 similar books are selected.

7.2.10 Fetching Details of Similar Books

```
for i in similar_items:
    temp_df = books[books['Book-Title'] == pt.index[i[0]]]
    if not temp_df.empty:
        similar_book = []
        similar_book.extend(list(temp_df.drop_duplicates('Book-Title')['Book-Title'].values))
        similar_book.extend(list(temp_df.drop_duplicates('Book-Title')['Book-Author'].values))
        similar_book.extend(list(temp_df.drop_duplicates('Book-Title')['Image-URL-
M'].values))
        data.append(similar_book)
```

1. For each similar book, it fetches the details (title, author, image) from the books dataset and appends them to the data list.
2. The data list will contain both the original searched book and the similar recommended books.

7.2.11 Rendering the Recommendations

return render_template('recommend.html', data=data)

1. After collecting the book details, the data list is passed to recommend.html for display, allowing the user to see their searched book and similar recommendations.

7.2.12 Handling the Case of No Matching Book

```
else:
    return render_template('recommend.html', message="Book not found. Please try another book.")
```

1. If the book is not found in the books dataset, the user is shown a message in the recommend.html template stating that the book was not found, asking them to try another search.

7.2.13 Running the Application

```
if __name__ == '__main__':
    app.run(debug=True)
```

1. The app starts in debug mode, which allows for live reloading and provides detailed error logs during development.

7.3 HTML Templates (templates/index.html)

index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Book Recommender System</title>
    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap.min.css"
integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">
    <style>
        body {
            background-color: #1b1b2a;
            color: #eaeaea; /* Light grey text */
        }
        .navbar {
            background-color: #ff6f61
        }
        .navbar-brand, .nav > li > a {
            color: #ffffff; /* White text for links */
        }
        .navbar-brand:hover, .nav > li > a:hover {
            color: #f2f2f2;
        }
        .card {
            background-color: #252850; /* Darker shade for cards */
            border: none;
            margin-bottom: 30px;
            border-radius: 8px;
        }
        .card-body {
            text-align: center;
            padding: 20px;
        }
        .card-img-top {
            max-width: 100%;
            height: auto;
            border-radius: 5px;
        }
        h1 {
            text-align: center;
            margin: 40px 0;
            font-size: 50px;
        }
    </style>
</head>
<body>

    <nav class="navbar">
        <div class="container-fluid">
            <div class="navbar-header">
                <a class="navbar-brand">My Book Recommender</a>
            </div>
            <ul class="nav navbar-nav">
                <li><a href="/">Home</a></li>
                <li><a href="/recommend">Recommend</a></li>
            <li><a href="#">Contact</a></li>
          </ul>
        </div>
    </nav>

    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <h1>Top 50 Books</h1>
            </div>

            {% for i in range(book_name|length) %}
                <div class="col-md-3">
                    <div class="card">
                        <div class="card-body">
                            <img class="card-img-top" src="{{ image[i] }}" alt="Cover of {{
book_name[i] }}">
                            <p>{{ book_name[i] }}</p>
                            <h4>{{ author[i] }}</h4>
                            <h4>Votes: {{ votes[i] }}</h4>
                            <h4>Rating: {{ rating[i] }}</h4>
                        </div>
                    </div>
                </div>
            {% endfor %}

        </div>
    </div>

</body>
</html>
```

7.3.1 Including Bootstrap CSS

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap.min.css"
integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">
```

1. This link imports the Bootstrap CSS framework, which helps in styling and making the layout responsive. The integrity attribute provides a way to ensure that the file has not been tampered with.

7.3.2 Main Content Section

```
<div class="container">
   <div class="row">
      <div class="col-md-12">
         <h1>Top 50 Books</h1>
      </div>
```

1. **Container**: The container class provides a responsive fixed-width container for the content.
2. **Row and Column**: Defines a grid layout where the header for "Top 50 Books" spans all 12 columns of the grid.

7.3.3 Displaying Books with Jinja2

```
{% for i in range(book_name|length) %}
   <div class="col-md-3">
      <div class="card">
         <div class="card-body">
            <img class="card-img-top" src="{{ image[i] }}" alt="Book Cover">
            <p>{{ book_name[i] }}</p>
            <h4>{{ author[i] }}</h4>
            <h4>Votes: {{ votes[i] }}</h4>
            <h4>Rating: {{ rating[i] }}</h4>
         </div>
      </div>
   </div>
{% endfor %}
```

1. **Looping through Books**: The Jinja2 {% for %} loop iterates through the book_name list.
2. **Grid Layout for Cards**: Each book is displayed in a card format, using Bootstrap's grid system (4 cards per row).
3. **Card Structure**:
   - **Image**: Displays the book cover using the src attribute from the image list.
   - **Book Details**: Shows the book title, author, number of votes, and average rating in a structured manner.

7.4 HTML Templates (recommend.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Book Recommender System</title>
    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap.min.css"
integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">
</head>
<style>
    body {
        background-color: #1a1a2e;
        color: #e94560;
    }
    .navbar {
        background-color: #16213e;
    }
    .navbar-brand, .nav > li > a {
        color: #e94560;
    }
    .navbar-brand:hover, .nav > li > a:hover {
        color: #f8f8f8;
    }
    .card {
        background-color: #0f3460;
        border: none;
    }
    .card-body {
        padding: 20px;
    }
    .btn-warning {
        background-color: #e94560;
        border: none;
    }
    .btn-warning:hover {
        background-color: #ff6f61;
    }
    h1 {
        text-align: center;
        margin-bottom: 30px;
    }
</style>
<body>
```

```
    <nav class="navbar">
        <div class="container-fluid">
            <div class="navbar-header">
                <a class="navbar-brand">My Book Recommender</a>
            </div>
            <ul class="nav navbar-nav">
                <li><a href="/">Home</a></li>
                <li><a href="/recommend">Recommend</a></li>
                <li><a href="#">Contact</a></li>
            </ul>
        </div>
    </nav>

    <div class="container">
        <div class="row">
            <div class="col-md-12">
                <h1>Recommend Books</h1>
                <form action="/recommend_books" method="post">
                    <input name="user_input" type="text" class="form-control" placeholder="Enter
a book title or genre">
                    <br>
                    <input type="submit" class="btn btn-lg btn-warning" value="Find
Recommendations">
                </form>
            </div>

            {% if data %}
                {% for i in data %}
                    <div class="col-md-3" style="margin-top:50px">
                        <div class="card">
                            <div class="card-body text-center">
                                <img class="card-img-top" src="{{i[2]}}" alt="Cover of {{i[0]}}"
style="max-width: 100%; height: auto;">
                                <p>{{i[0]}}</p>
                                <h4>{{i[1]}}</h4>
                            </div>
                        </div>
                    </div>
                {% endfor %}
            {% endif %}
        </div>
    </div>

</body>
```

```
</html>
```

7.4.1 Main Content Section

```
<div class="container">
   <div class="row">
      <div class="col-md-12">
         <h1>Recommend Books</h1>
         <form action="/recommend_books" method="post">
            <input name="user_input" type="text" class="form-control" placeholder="Enter a
book title or genre">
            <br>
            <input type="submit" class="btn btn-lg btn-warning" value="Find
Recommendations">
         </form>
      </div>
```

1. **Container and Row**: The container class provides a responsive fixed-width container, and the row class helps in laying out the elements properly.
2. **Header**: Displays the title "Recommend Books" in a centered manner.
3. **Form Element**:
   - **Input Field**: Allows users to enter a book title or genre, styled with Bootstrap's form control.
   - **Submit Button**: A button styled with the btn-warning class, which is customized for a distinct look, prompting users to find book recommendations.

7.4.2 Displaying Recommended Books

```
{% if data %}
   {% for i in data %}
      <div class="col-md-3" style="margin-top:50px">
         <div class="card">
            <div class="card-body text-center">
               <img class="card-img-top" src="{{i[2]}}" alt="Cover of {{i[0]}}" style="max-
width: 100%; height: auto;">
               <p>{{i[0]}}</p>
               <h4>{{i[1]}}</h4>
            </div>
         </div>
      </div>
   {% endfor %}
{% endif %}
```

1. **Conditional Display**: The {% if data %} checks if there are any recommendations to display.
2. **Looping through Recommendations**: The {% for %} loop iterates through the data list, where each item i contains the book details.
3. **Grid Layout for Cards**: Each recommended book is displayed in a card format, structured to occupy 3 columns in the Bootstrap grid system.
4. **Card Structure**:
   - **Image**: Displays the book cover, sourced from the data and ensuring it is responsive.
   - **Book Title and Author**: Displays the book's title and author within the card, centered for aesthetics.

8. How the Recommender System Works

The Book Recommender System is designed to provide personalized book recommendations based on user preferences and historical data. The underlying mechanics of the system involve several key components and methodologies that allow it to function effectively. This section outlines how the system works, including the data processing, recommendation algorithms, and user interaction.

8.1 Overview of the Recommendation Process

The recommendation process can be broken down into the following steps:

1. **Data Collection:** The system collects data from various sources, including book information, user ratings, and interaction history. This data is stored in a database and forms the basis for generating recommendations.
2. **Data Preprocessing:** Before making recommendations, the collected data undergoes preprocessing to clean and format it. This step may involve handling missing values, normalizing ratings, and structuring the data for analysis.
3. **User Input:** Users interact with the system by providing ratings for books they have read. This input is crucial as it directly influences the recommendations generated for that user.
4. **Recommendation Generation:** Based on the user input and the available data, the system employs algorithms to calculate book recommendations. These algorithms can include collaborative filtering, content-based filtering, or a hybrid approach.
5. **Displaying Recommendations:** Once the recommendations are generated, they are presented to the user in a user-friendly format through the web interface.

8.2 Data Collection

The system collects the following types of data:

- **Book Data:** Information about books, including titles, authors, genres, and descriptions. This data is typically stored in a database, such as SQLite.
- **User Ratings:** Ratings submitted by users for books they have read. This data helps the system understand user preferences.
- **User Profiles:** Information about users, which may include demographic data or reading history, although this is optional.

8.3 Data Preprocessing

Data preprocessing is a critical step in ensuring that the system operates effectively. It involves:

- **Cleaning Data:** Removing duplicates and irrelevant entries, handling missing values, and ensuring data consistency.
- **Normalizing Ratings:** Adjusting user ratings to a common scale to ensure fair comparisons. For instance, if one user rates on a scale of 1-5 while another uses a scale of 1-10, the ratings may need to be normalized.
- **Structuring Data:** Transforming the data into a format suitable for the recommendation algorithms, such as a user-item matrix.

8.4 Recommendation Algorithms

The core of the recommender system lies in the algorithms used to generate recommendations. The following methods can be employed:

- **Collaborative Filtering:**
  - **User-Based Collaborative Filtering:** This approach identifies users with similar preferences and recommends books that those similar users liked. It relies on the assumption that if two users have similar ratings, they will also agree on other books.
  - **Item-Based Collaborative Filtering:** This method focuses on finding similarities between items (books). If a user likes a particular book, the system will recommend other books that are similar based on the ratings of other users.
- **Content-Based Filtering:**
  - This technique recommends books based on their attributes and the user's past preferences. For example, if a user enjoys science fiction novels, the system will recommend other books within the same genre or with similar themes.
- **Hybrid Approach:**
  - A combination of collaborative and content-based filtering to leverage the strengths of both methods, potentially increasing the accuracy of recommendations.

8.5 User Interaction

The user interface is designed to facilitate easy interaction with the system. The key components include:

- **Home Page:** Displays a list of available books, allowing users to browse and select books to rate.
- **Rating Submission:** Users can provide ratings for the books they have read. This interaction updates the user’s profile and influences future recommendations.
- **Recommendations Page:** After submitting ratings, users are redirected to a page displaying personalized book recommendations based on the input provided.

9. Web Interface (Flask)

The web interface of the Book Recommender System is built using the Flask framework. Flask is a lightweight web framework for Python that enables rapid development of web applications. This section covers the essential aspects of the Flask framework, including its introduction, user interface design, routing, displaying recommendations, and the interaction between the frontend and backend.

9.1 Introduction to Flask Framework

Flask is a micro web framework for Python that is designed to be simple and flexible. It allows developers to create web applications quickly and with minimal code. Key features of Flask include:

- **Lightweight and Modular:** Flask provides a minimalistic core that can be extended with various plugins and libraries to suit the needs of the application.
- **Built-in Development Server:** Flask comes with a built-in development server, making it easy to test applications during development.
- **Support for RESTful Request Dispatching:** Flask makes it easy to create RESTful APIs, enabling smooth interaction between the frontend and backend.
- **Templating Engine:** Flask uses Jinja2 as its templating engine, allowing developers to create dynamic HTML pages by rendering templates with data from Python.

Installation: To get started with Flask, install it using pip:

```
pip install Flask
```

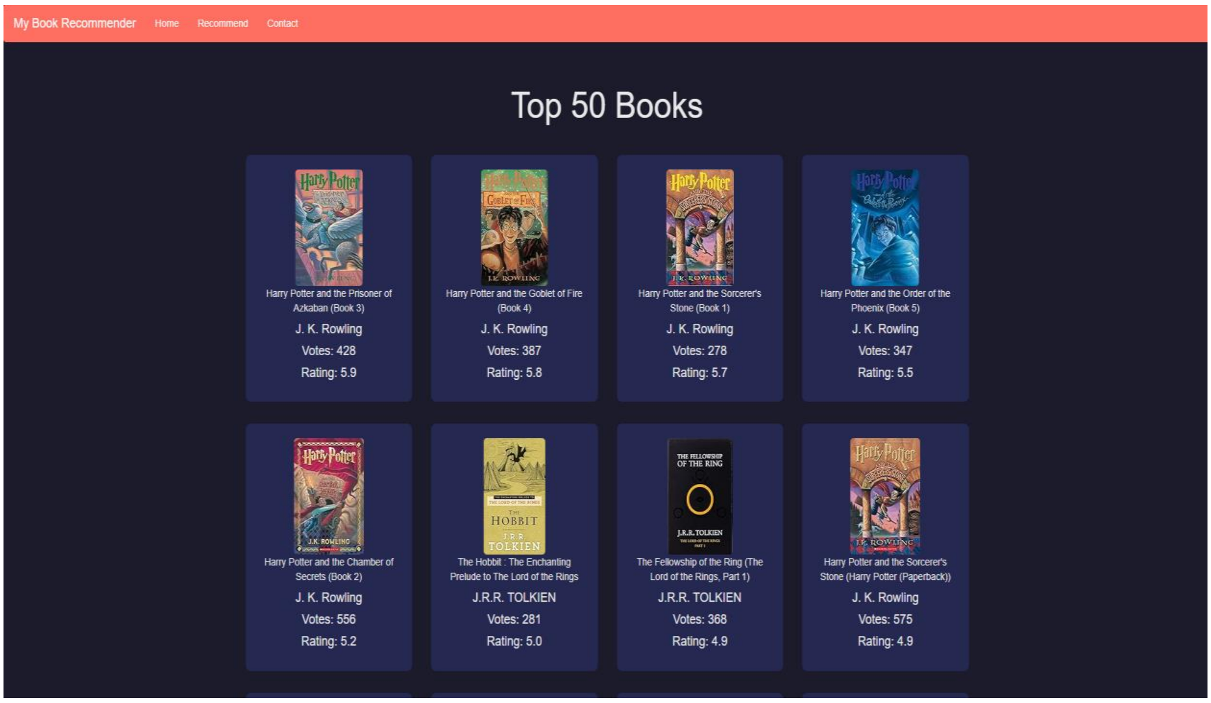
9.2 Web Pages (User Interface Design)

The user interface design of the Book Recommender System focuses on creating an intuitive and user-friendly experience. Key elements include:

- **Home Page:** The main landing page displays a welcome message, a brief description of the recommender system, and a list of available books for users to browse.
- **Recommendations Page:** After submitting ratings, users are redirected to a page that displays personalized book recommendations based on their input. This page should showcase book titles, authors, and cover images for easy browsing.

- **Navigation:** The interface should include a navigation bar with links to the home page, rating submission, and recommendations, allowing users to navigate seamlessly.

Home page



My Book Recommender Home Recommend Contact

Top 50 Books

Harry Potter and the Prisoner of Azkaban (Book 3)
J. K. Rowling
Votes: 428
Rating: 5.9

Harry Potter and the Goblet of Fire (Book 4)
J. K. Rowling
Votes: 387
Rating: 5.8

Harry Potter and the Sorcerer's Stone (Book 1)
J. K. Rowling
Votes: 278
Rating: 5.7

Harry Potter and the Order of the Phoenix (Book 5)
J. K. Rowling
Votes: 347
Rating: 5.5

Harry Potter and the Chamber of Secrets (Book 2)
J. K. Rowling
Votes: 556
Rating: 5.2

The Hobbit : The Enchanting Prelude to The Lord of the Rings
J.R.R. TOLKIEN
Votes: 281
Rating: 5.0

The Fellowship of the Ring (The Lord of the Rings, Part 1)
J.R.R. TOLKIEN
Votes: 368
Rating: 4.9

Harry Potter and the Sorcerer's Stone (Harry Potter (Paperback))
J. K. Rowling
Votes: 575
Rating: 4.9

9.3 Routing in Flask

Routing in Flask is the mechanism that allows developers to map URLs to functions. This enables users to access different pages of the application by entering specific URLs.

- **Defining Routes:** Routes are defined using the @app.route() decorator. Each route is associated with a specific function that returns the content to be displayed on that page.

Example Route Definition:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def home():
    return render_template('home.html')

@app.route('/rate')
def rate():
    return render_template('rate.html')

@app.route('/recommendations')
def recommendations():
    return render_template('recommendations.html')

if __name__ == '__main__':
    app.run(debug=True)
```

- **Dynamic Routing:** Flask supports dynamic routing, allowing parameters to be passed in the URL. This can be useful for displaying specific book details based on user input.

Example Dynamic Route:

```
@app.route('/book/<book_title>')

def book_detail(book_title):

    # Logic to fetch and display details for the specific book

    return render_template('book_detail.html', title=book_title)
```

9.4 Displaying Recommendations

Displaying recommendations involves rendering a template that shows the recommended books to the user. This can be achieved by passing the recommended book data to the template.

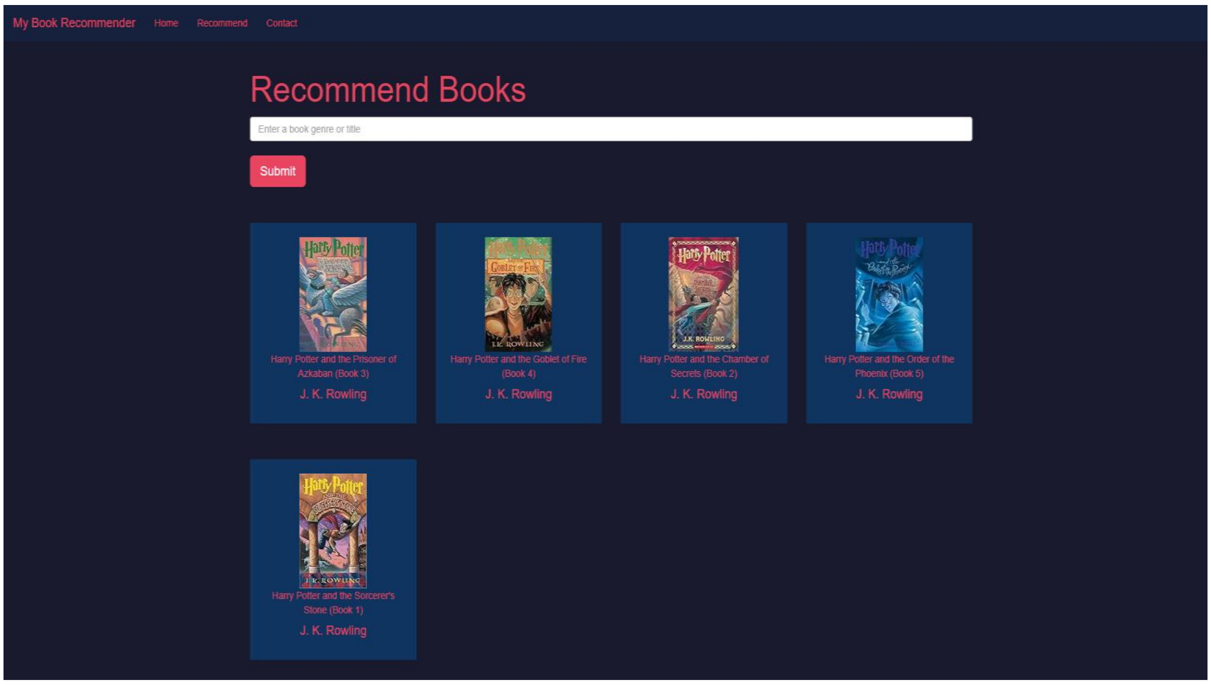
Example Rendering Recommendations:

```
@app.route('/recommendations', methods=['POST'])
def get_recommendations():
    user_input = request.form['user_input']
    recommendations = recommend(user_input)  # Call the recommend function
    return render_template('recommendations.html', recommendations=recommendations)
```

Example HTML for Recommendations:

```
<h2>Your Book Recommendations</h2>
<ul>
    {% for book in recommendations %}
        <li>
            <h3>{{ book[0] }}</h3>
```

```
      <p>Author: {{ book[1] }}</p>
      <img src="{{ book[2] }}" alt="{{ book[0] }} cover image">
    </li>
  {% endfor %}
</ul>
```


My Book Recommender
Home
Recommend
Contact
Recommend Books
Enter a book genre or title
Submit
Harry Potter and the Prisoner of Azkaban (Book 3)
J. K. Rowling
Harry Potter and the Goblet of Fire (Book 4)
J. K. Rowling
Harry Potter and the Chamber of Secrets (Book 2)
J. K. Rowling
Harry Potter and the Order of the Phoenix (Book 5)
J. K. Rowling
Harry Potter and the Sorcerer's Stone (Book 1)
J. K. Rowling

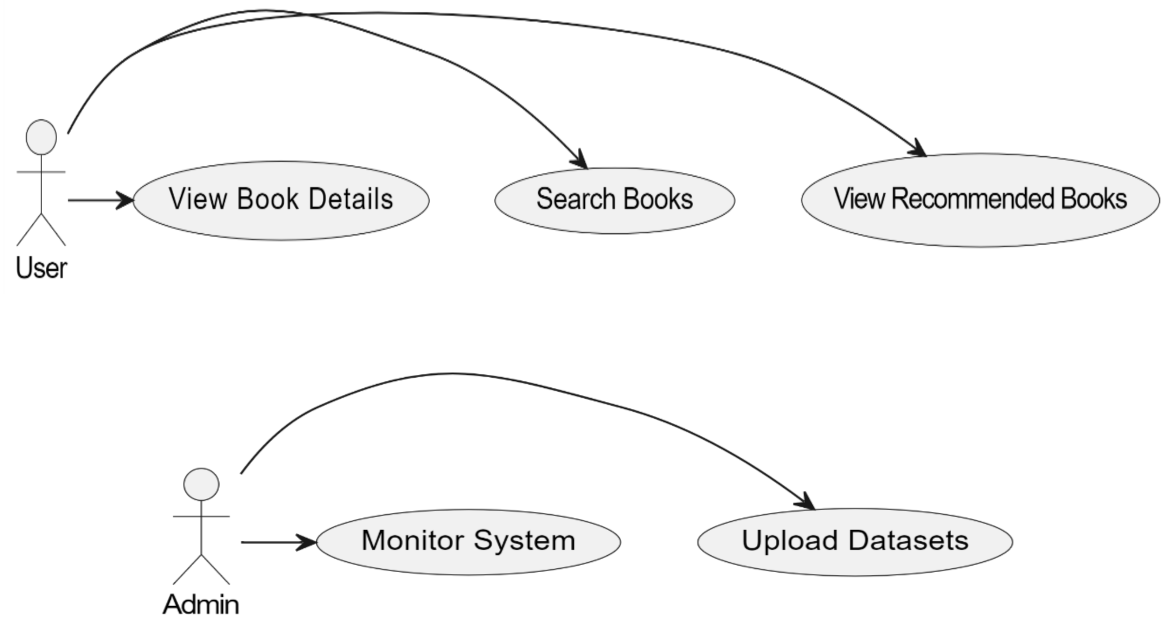
9.5 Interaction between Frontend and Backend

The interaction between the frontend and backend is crucial for the functionality of the recommender system. This typically involves:

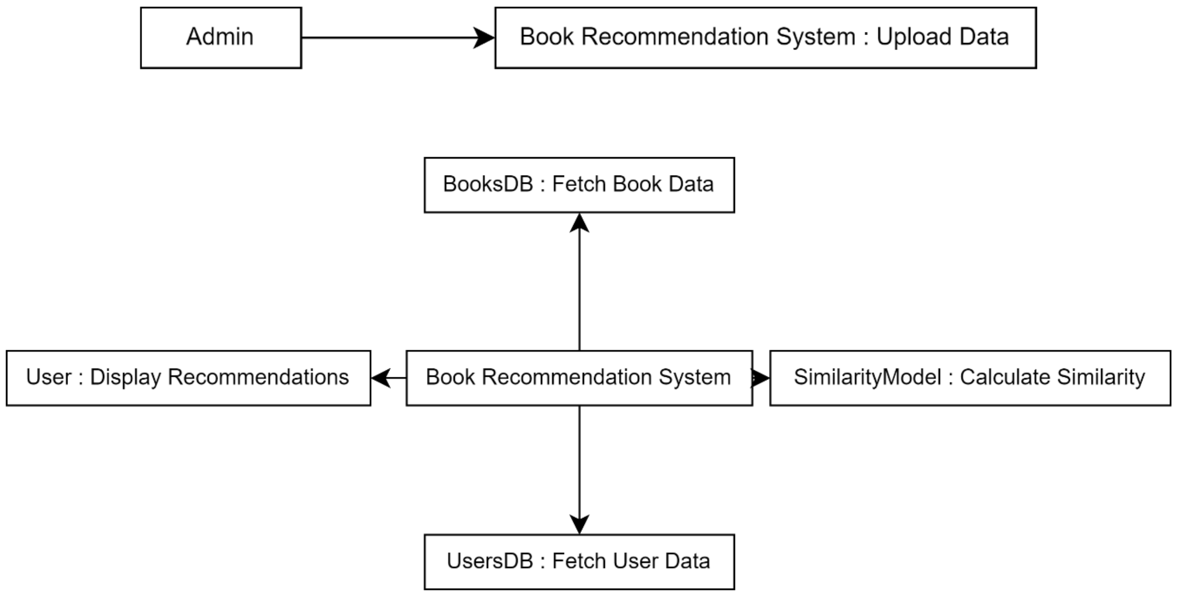
- **Form Submission:** Users submit their ratings through a form on the frontend, which sends the data to the backend for processing.
- **API Calls:** The frontend can make API calls to the backend to fetch recommendations dynamically, allowing for a more interactive user experience.
- **Data Handling:** The backend processes the input data, performs the recommendation logic, and sends the results back to the frontend for display.

10. Diagram

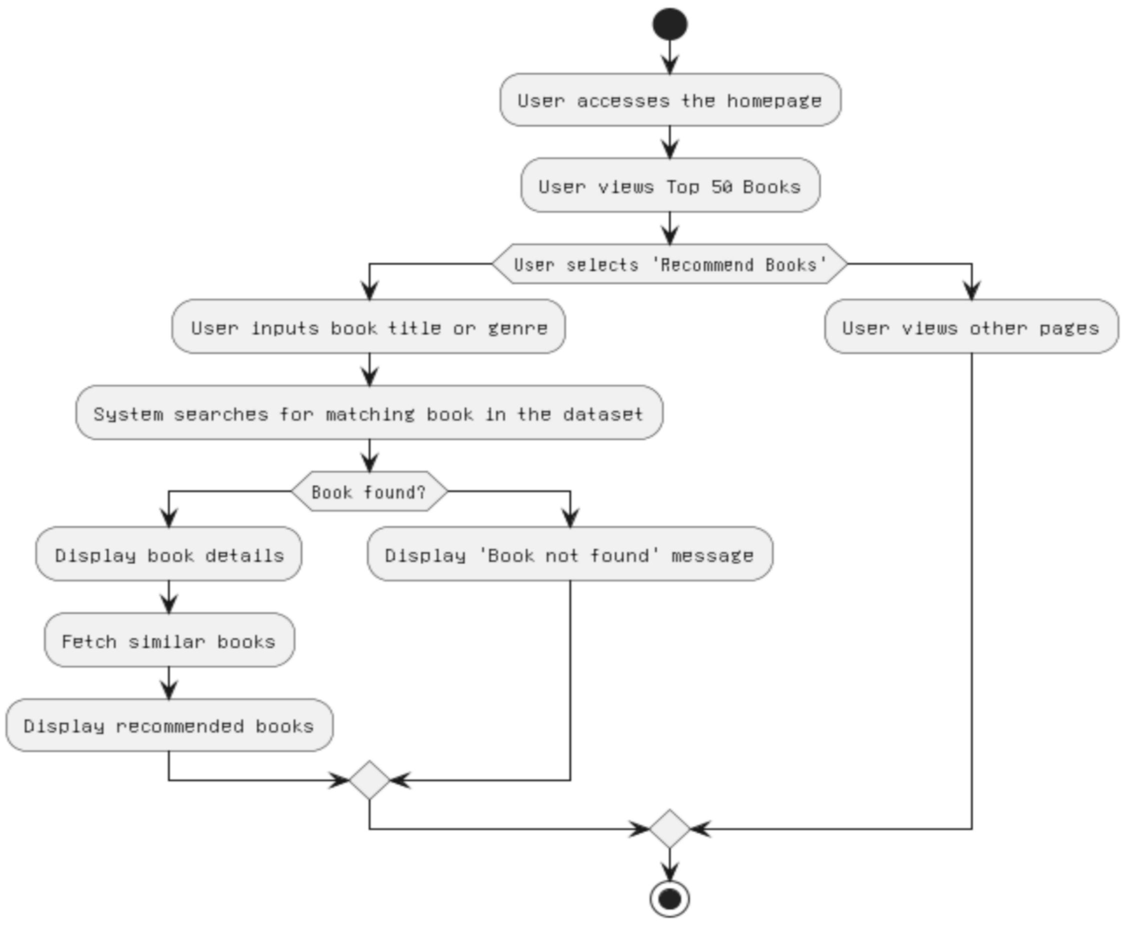
10.1 Use Case Diagram:



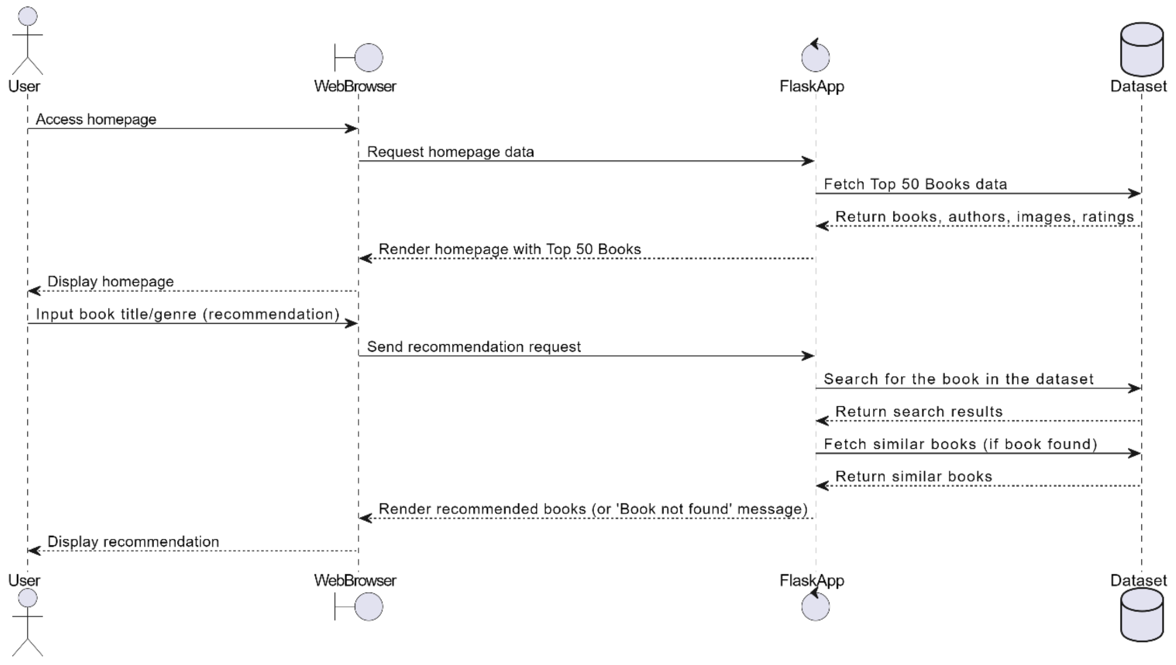
10.2 Data Flow Diagram (DFD):


Admin
Book Recommendation System : Upload Data
BooksDB : Fetch Book Data
User : Display Recommendations
Book Recommendation System
SimilarityModel : Calculate Similarity
UsersDB : Fetch User Data

10.3 Activity Diagram:

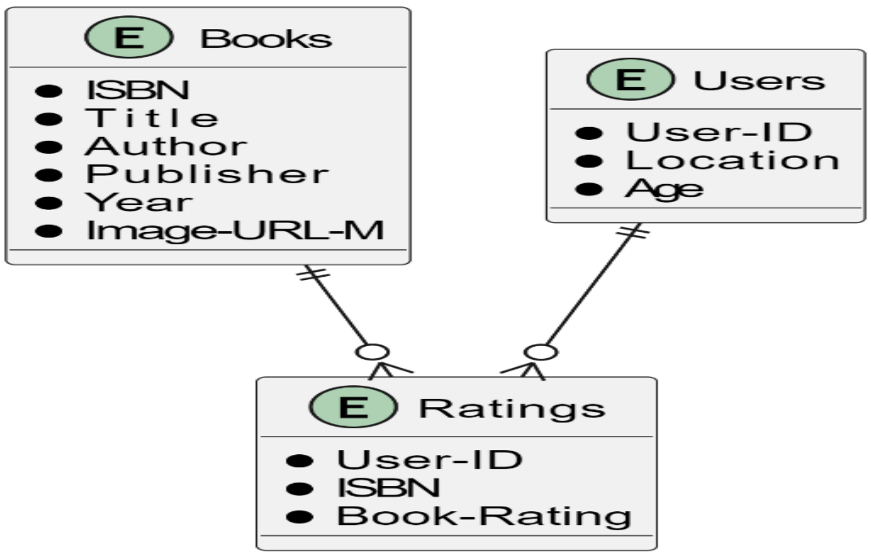

User accesses the homepage
User views Top 50 Books
User selects 'Recommend Books'
User inputs book title or genre
User views other pages
System searches for matching book in the dataset
Book found?
Display book details
Display 'Book not found' message
Fetch similar books
Display recommended books

10.4 Sequence Diagram:

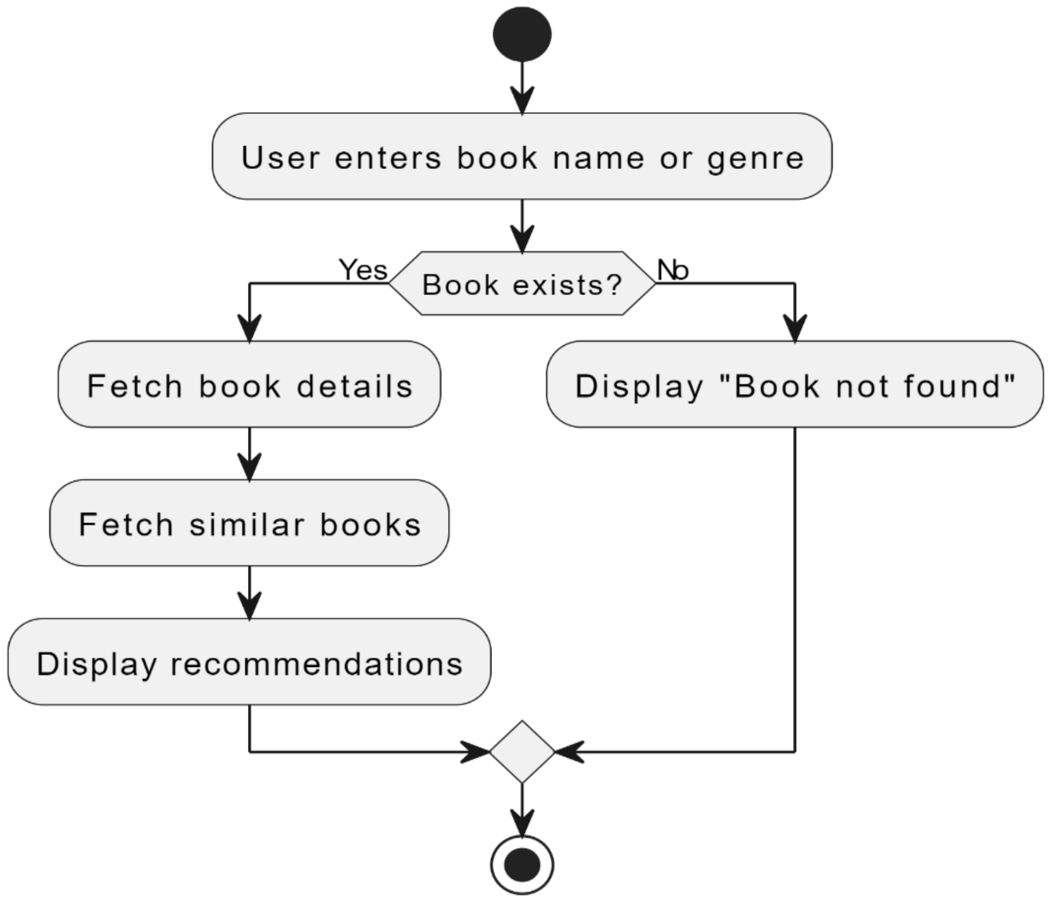


User
WebBrowser
FlaskApp
Dataset
Access homepage
Request homepage data
Fetch Top 50 Books data
Return books, authors, images, ratings
Render homepage with Top 50 Books
Display homepage
Input book title/genre (recommendation)
Send recommendation request
Search for the book in the dataset
Return search results
Fetch similar books (if book found)
Return similar books
Render recommended books (or 'Book not found' message)
Display recommendation
User
WebBrowser
FlaskApp
Dataset

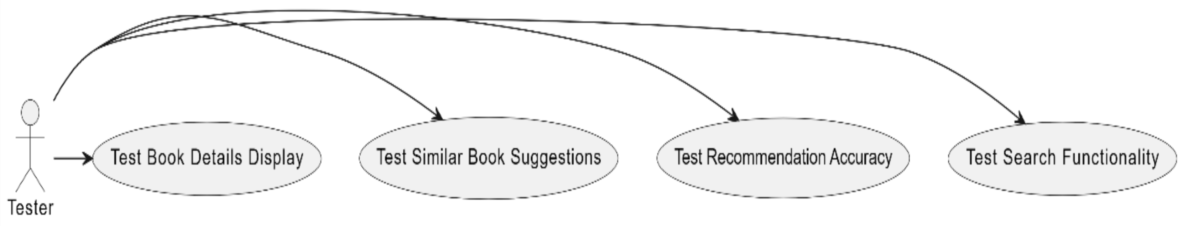
10.5 Database Schema Diagram:



10.6 Logic Flow Diagram:



10.7 Test Case Diagram


Tester
Test Book Details Display
Test Similar Book Suggestions
Test Recommendation Accuracy
Test Search Functionality

11. References

1. Kaggle. (n.d.). *Book Recommendation Dataset*. Retrieved from https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset

2. YouTube. (2023). *Building a Book Recommendation System | Machine Learning Project | Python | Data Science* [Video]. Retrieved from https://www.youtube.com/watch?v=1YoD0fg3_EM&t=3644s

3. diagrams.net. (n.d.). *Diagrams Tool*. Retrieved from https://app.diagrams.net/#G1e1Os5-EziWmKrTjjvciawA1RxctJOjXC#%7B%22pageId%22%3A%22xkXZdXrxeT9EktWEo8N2%22%7D

4. YouTube. (2023). *Deploying Machine Learning Models with Flask | Full Stack Development* [Video]. Retrieved from https://www.youtube.com/watch?v=WuEGXlokpuQ&t=1157s