**Terna Engineering College**

**Computer Engineering Department**

**Class: BE Sem.: VII**

**Course: Natural Language Processing [NLP]**

**Experiment No. 09- MINI PROJECT**

**PART A**

(PART A : TO BE REFFERED BY STUDENTS)

**A.1 Aim:** Book Recommendation System

**A.2 Prerequisite:** Python, Flask, scikit-learn, Pandas, Numpy, Google Colab, Pickle libraries.

**A.3 Outcome:**

Students will learn how to implement a book recommendation system using popularity-based recommendations and collaborative filtering. The project demonstrates the application of NLP and machine learning techniques to personalize user experiences in web applications.

**A.4 Theory:**

The Book Recommendation System is designed to assist users in discovering new books by providing personalized recommendations based on their past behavior or general popularity. Recommendation systems are integral to modern web applications, and this project focuses on two primary recommendation strategies: popularity-based and collaborative filtering. Using Natural Language Processing (NLP) and machine learning, the system analyzes user interactions to generate suggestions. The backend is built using Flask, while Python handles the machine learning algorithms.

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

|  |  |
| --- | --- |
| **Roll. No.:** | **Name:** |
| **Class:** | **Batch:** |
| **Date of Experiment:** | **Date of Submission:** |
| **Grade:** | |

**B.1 Software Code written by student:**

The **Book Recommendation System** integrates Python, Flask, and machine learning models to create a dynamic and interactive platform for suggesting books based on popularity and collaborative filtering. This project demonstrates the successful implementation of two recommendation strategies and showcases the combined power of web development and data science.

**Project Overview:**

The code is designed to offer users recommendations based on two main approaches: popularity-based suggestions and collaborative filtering. Flask is used as the backend framework to handle user interactions, while pre-trained machine learning models handle the recommendation logic. The models and data, preprocessed and serialized using pickle, ensure the system operates efficiently and responds quickly to user queries.

**Explanation of Code Functionality:**

1. **Initialization and Libraries:**

The project begins by importing necessary libraries, including Flask, pickle, numpy, and machine learning utilities. The serialized models (stored using pickle) are loaded at runtime to provide the recommendation logic. This ensures that the system avoids retraining models on every execution, leading to faster and more efficient responses.

1. **Popularity-Based Recommendations (Homepage):**

The homepage uses a popularity-based approach to list the top 50 books based on user feedback (number of ratings and average rating). This is a static recommendation that does not change with individual users but offers a snapshot of popular content across the entire dataset. This information is served using Flask’s / route, which loads data from the pre-trained popularity model (popular.pkl) and displays it using HTML templates (index.html).

* + **Purpose:** To allow users to see top books that have received widespread attention, providing an easy entry point for book discovery.
  + **Frontend Integration:** The index.html template renders this data in a responsive layout using Bootstrap, ensuring it adapts well to different screen sizes.

1. **Collaborative Filtering-Based Recommendations:**

The real value of this system is derived from its personalized recommendations, which are generated using collaborative filtering. Collaborative filtering works by analyzing similarities between users and books based on past interactions (ratings). When a user inputs a book title on the recommendation page (/recommend\_books route), the system uses cosine similarity to find books that are most similar to the selected title. The similarity scores are precomputed and stored in a file (similarity\_scores.pkl) for efficient retrieval.

* + **Purpose:** To recommend books that are likely to align with a user’s taste, based on their previous choices or similar users' preferences.
  + **Backend Logic:** Flask handles the user’s input, processes it using the precomputed similarity matrix, and returns a list of similar books. This dynamic output is presented to the user on the recommend.html page.

1. **Data Preprocessing and Models:**

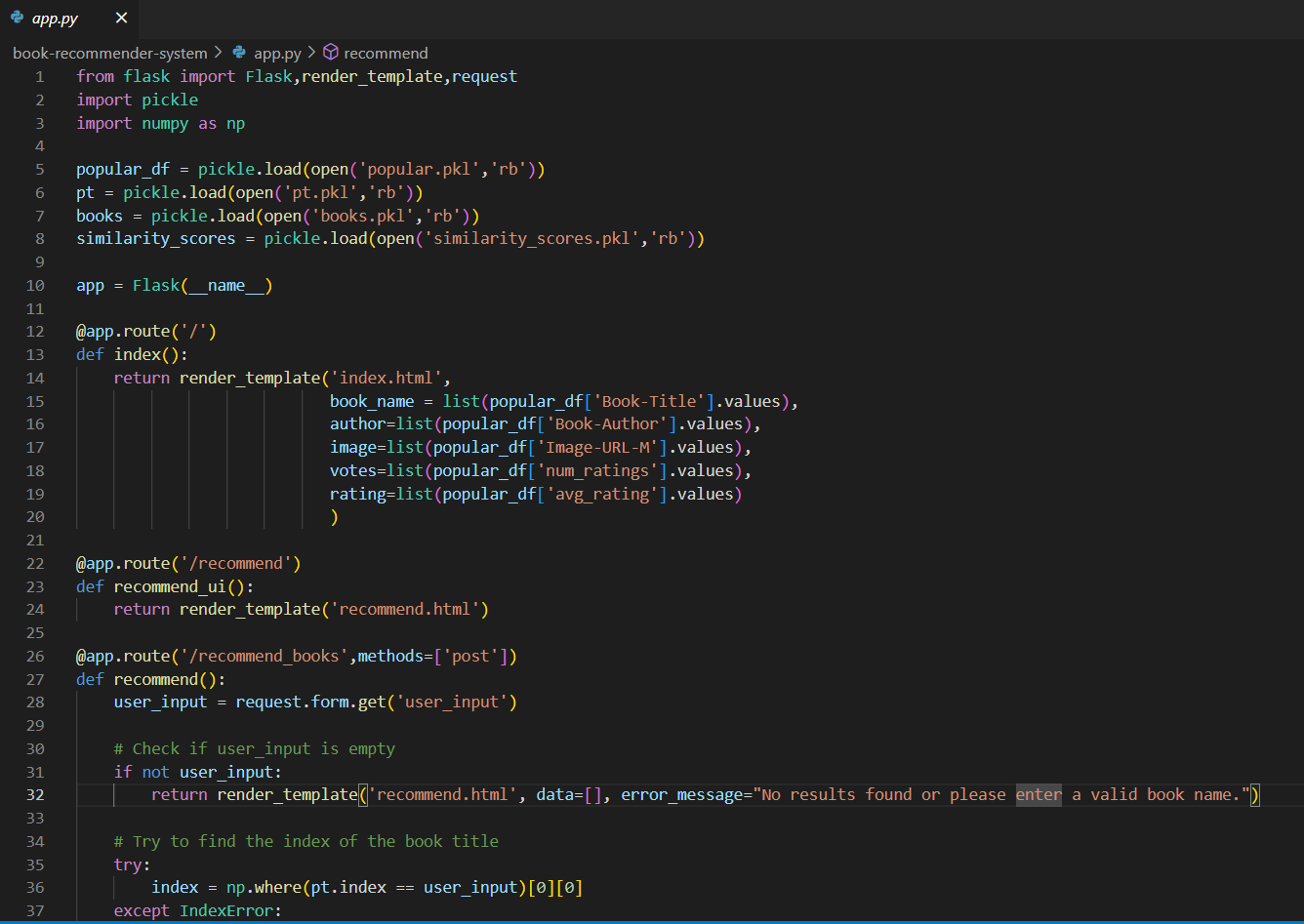
Preprocessing and training of the models occur offline, allowing the web app to serve pre-built models for recommendation generation. The data (ratings, books, and user preferences) are cleaned, normalized, and processed into matrices that allow the system to compute similarities efficiently. The pivot table (pt.pkl) and similarity matrix ensure that recommendations are tailored based on user interaction with the books.

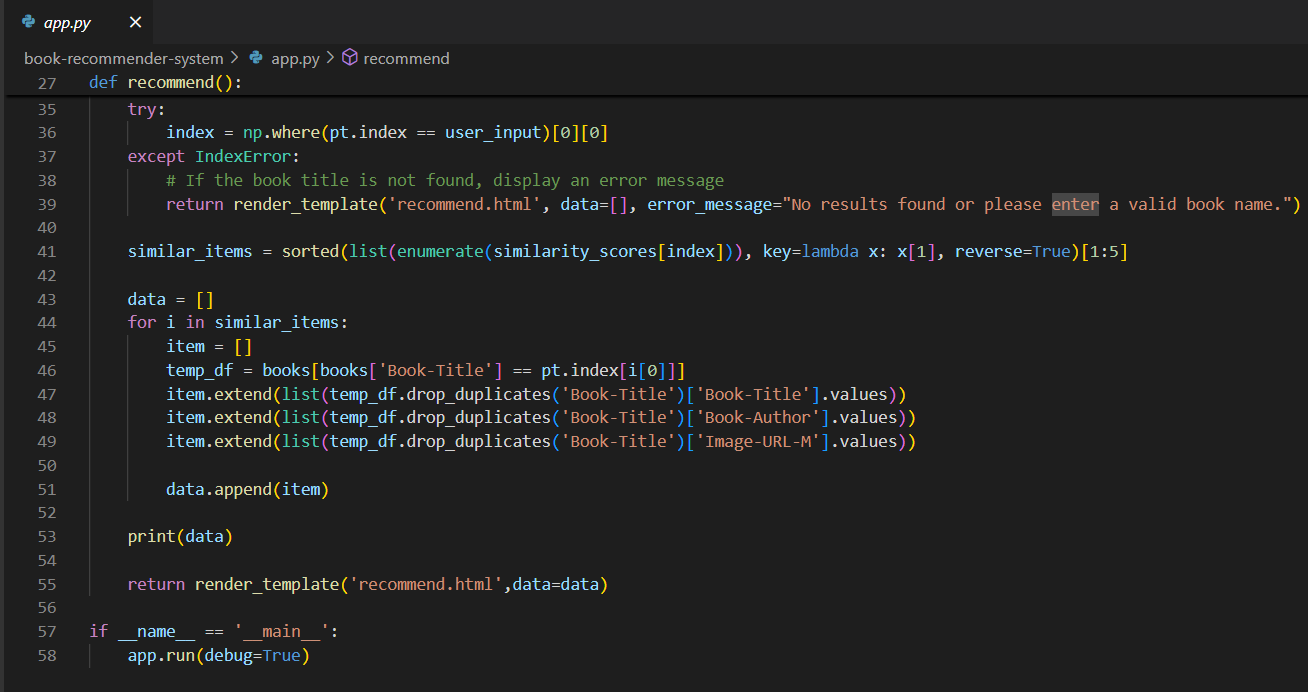
1. **Frontend Display:**

The project leverages Bootstrap to create a clean and interactive frontend where book recommendations are displayed. Users can input their book preferences and receive suggestions in real-time, improving user engagement. The responsive design ensures that the layout adapts well across devices, making the system accessible to a wide audience.

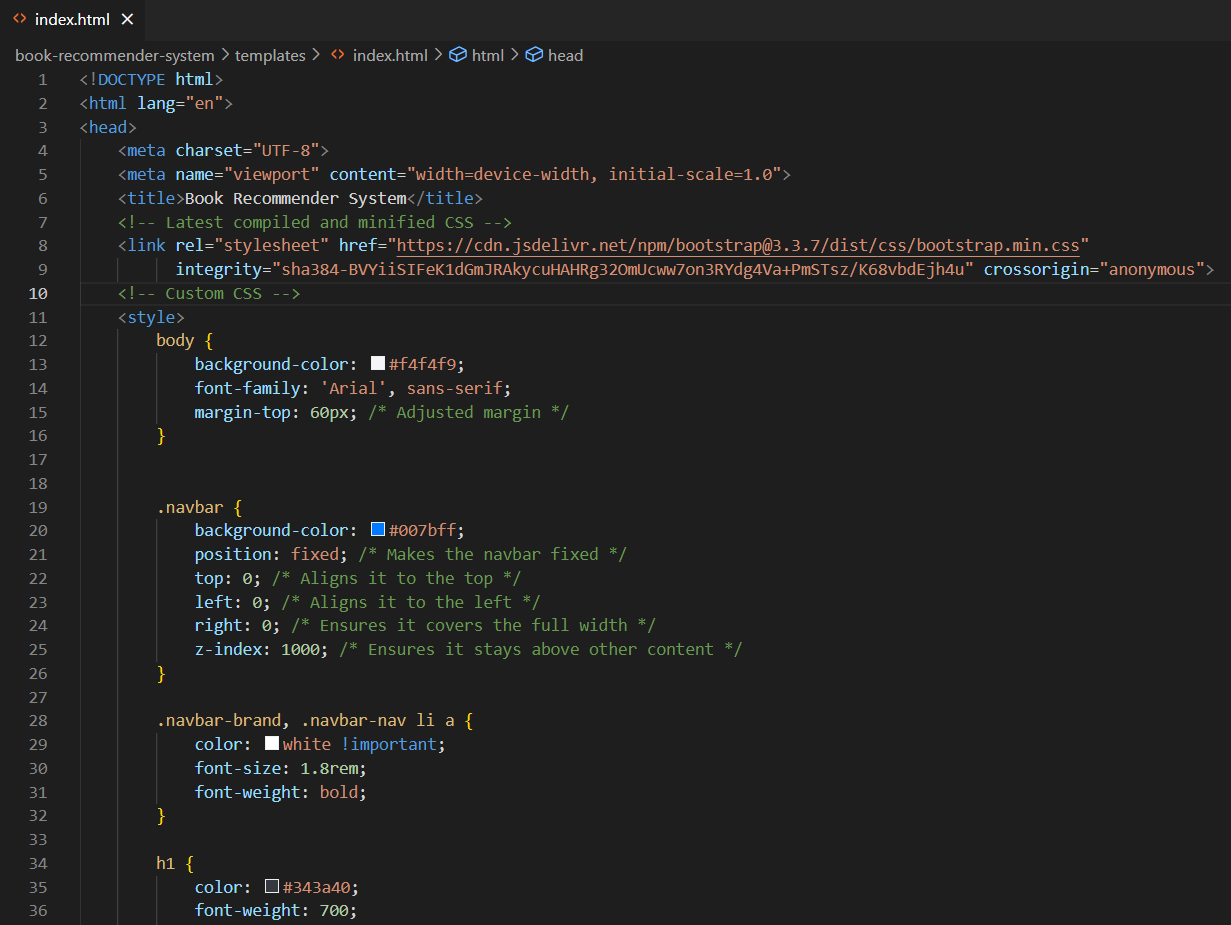
1. **Program Flow:**
   * The user navigates to the homepage, where they see a list of the top 50 books based on popularity.
   * On the recommendation page, they enter the name of a book they like. The system processes the input, calculates similar books based on collaborative filtering, and displays them in a user-friendly format.
   * The system provides an engaging experience by dynamically generating personalized recommendations, enhancing book discovery for the user.

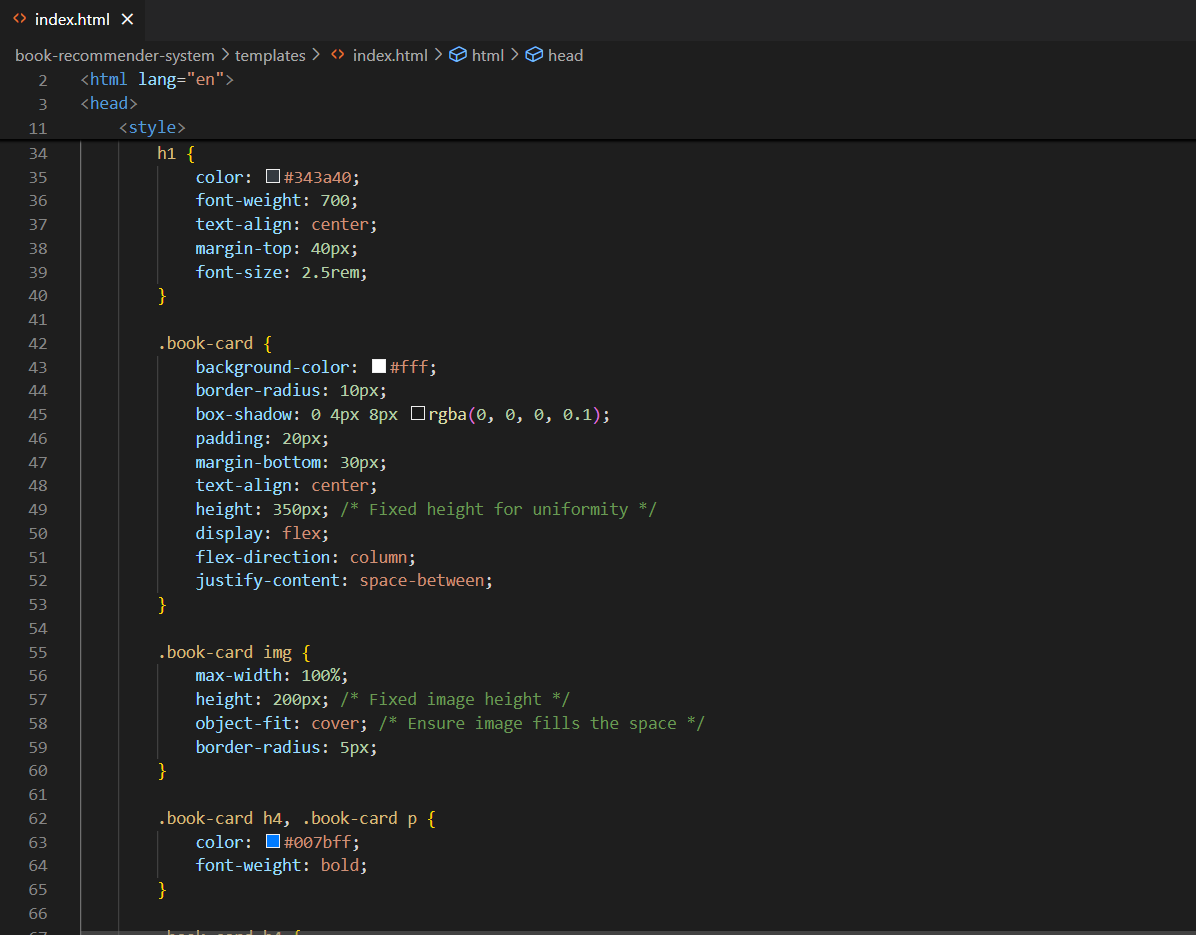
**App.py :**

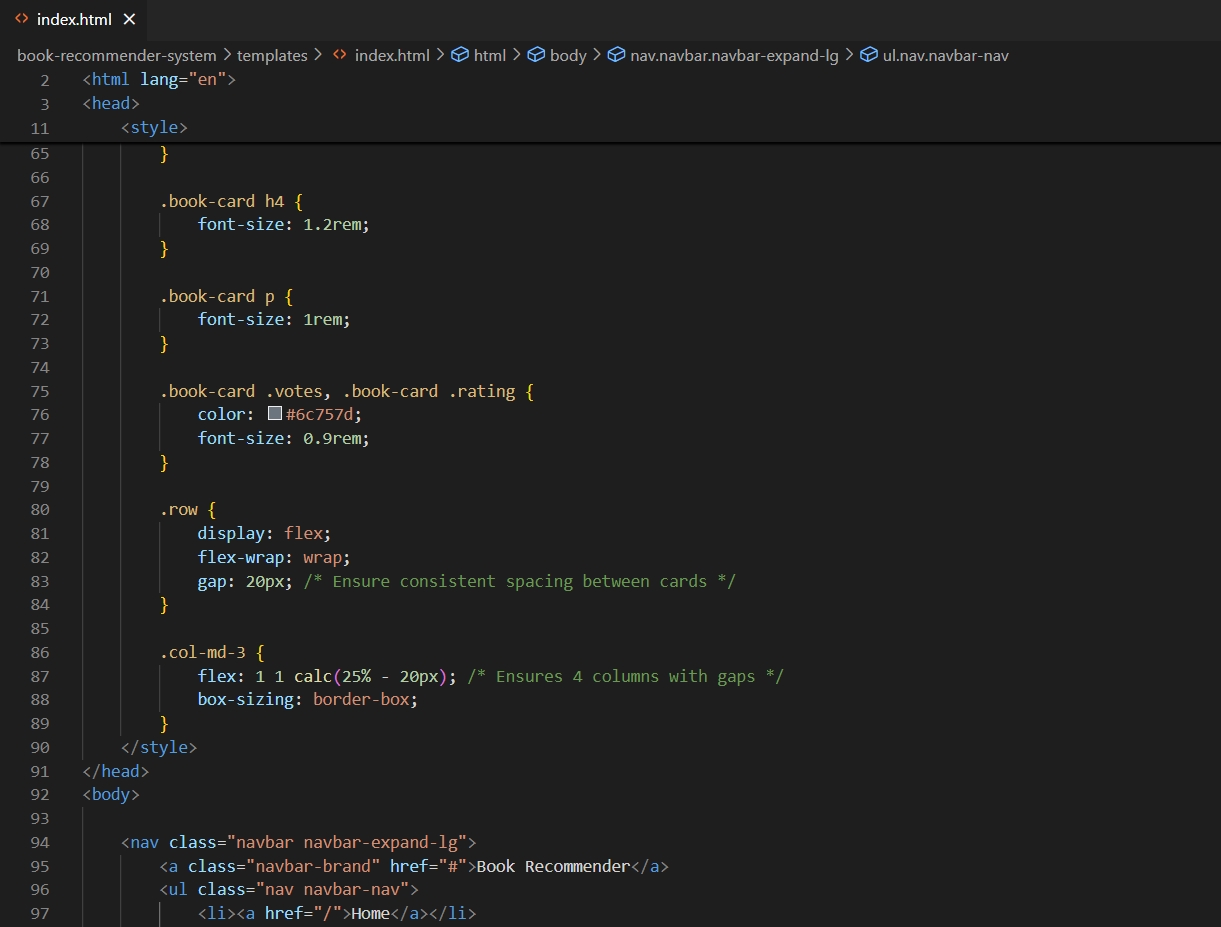


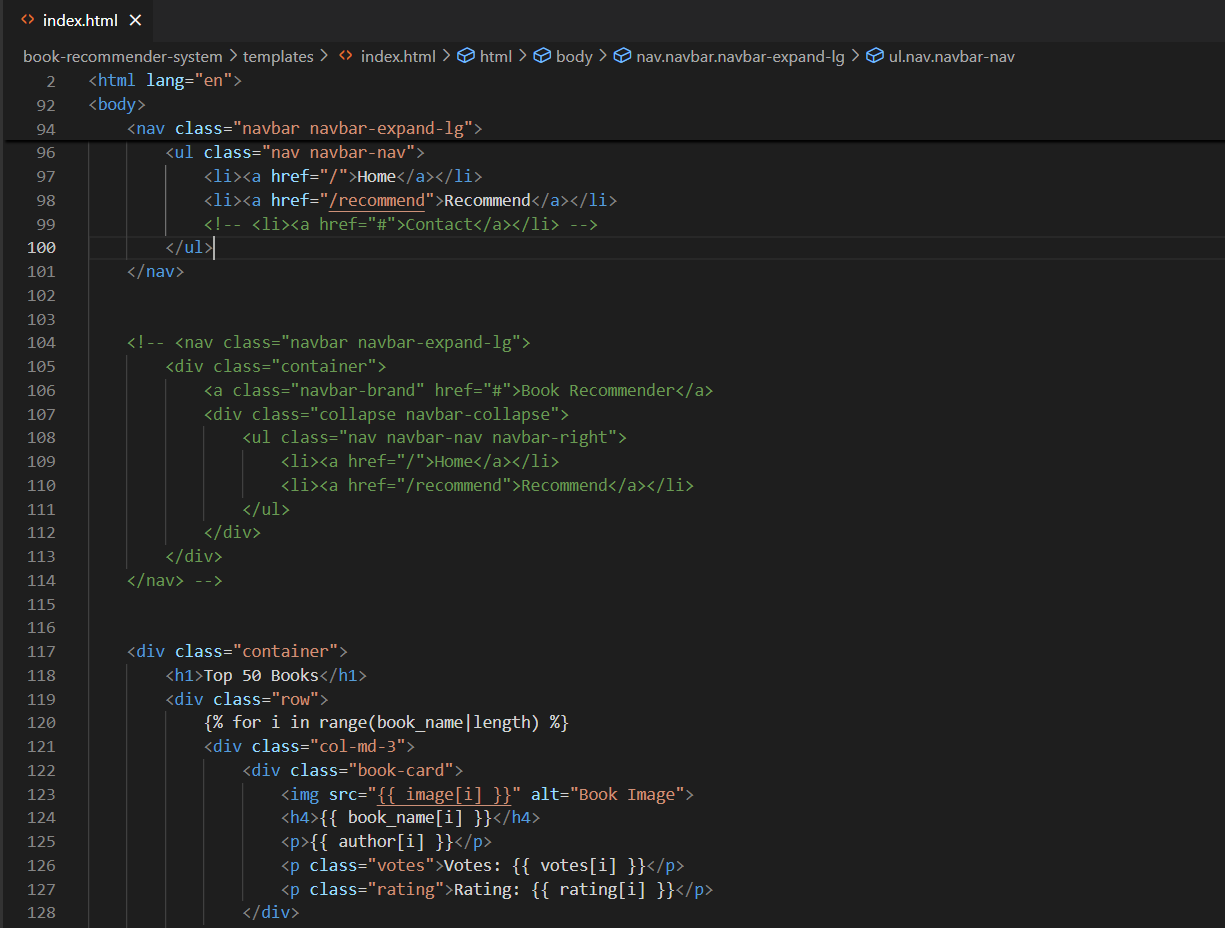


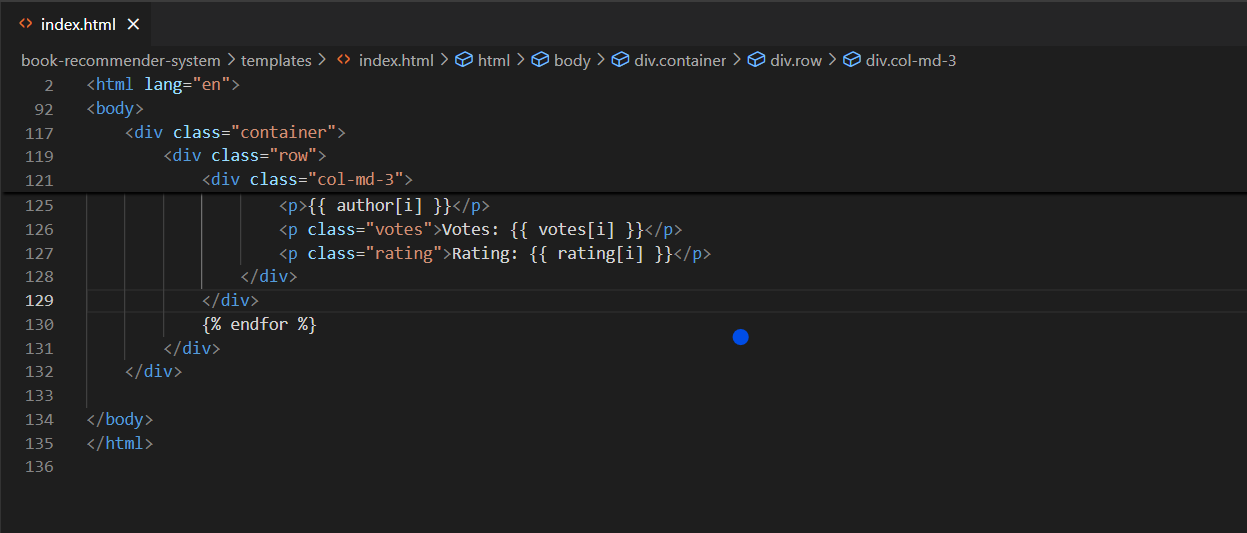
**Index.html :**

****

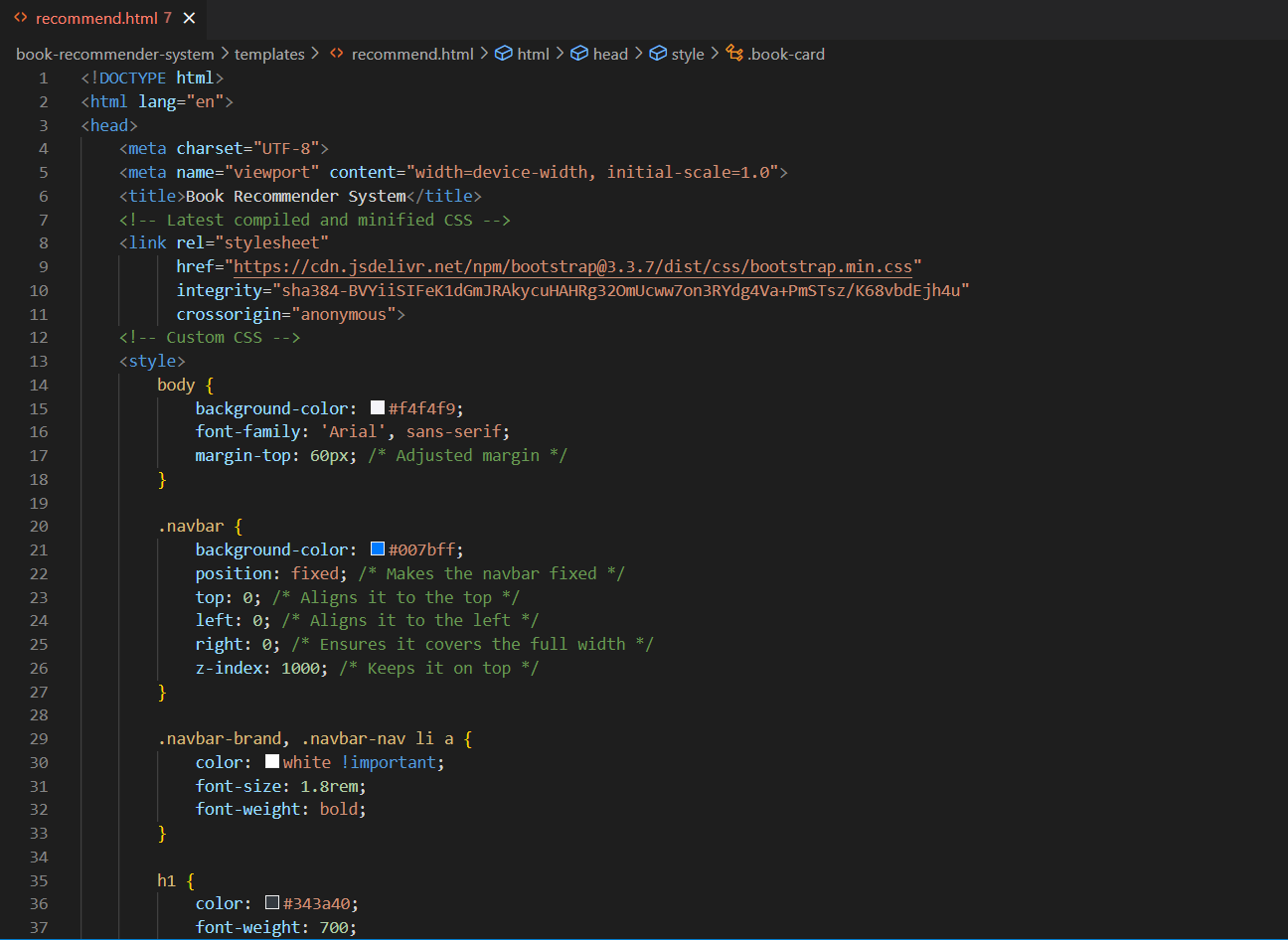
****

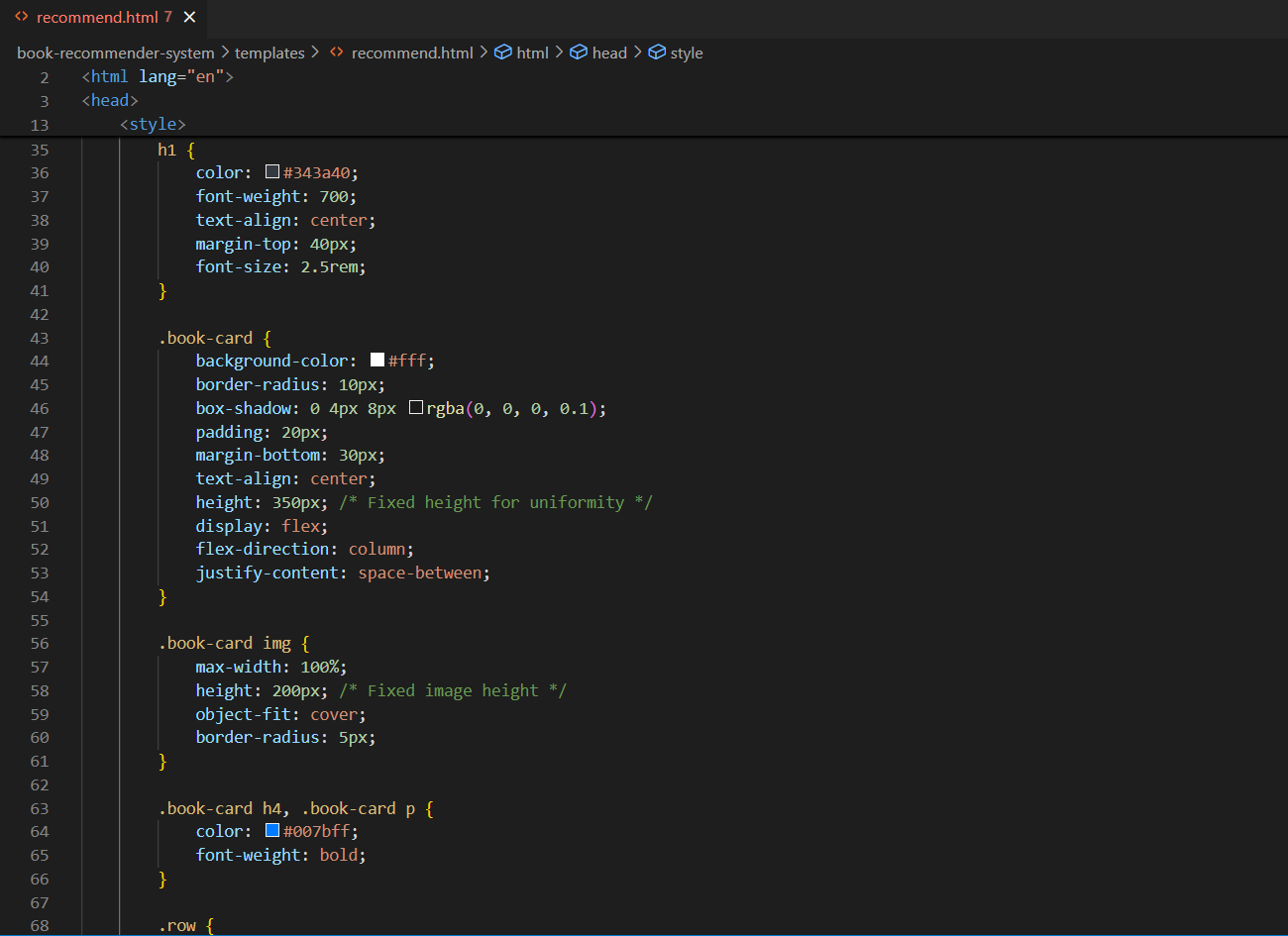
****

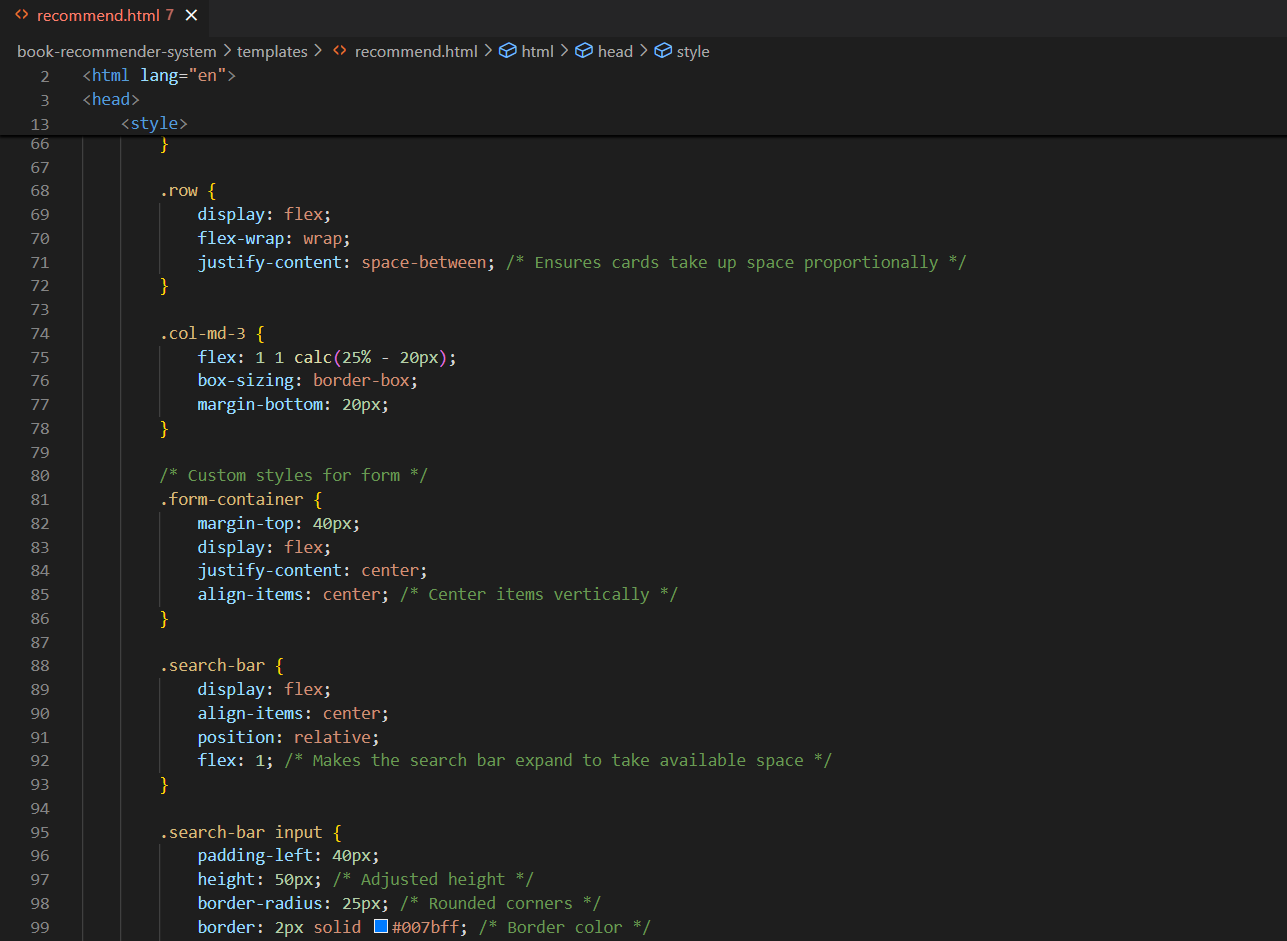
****

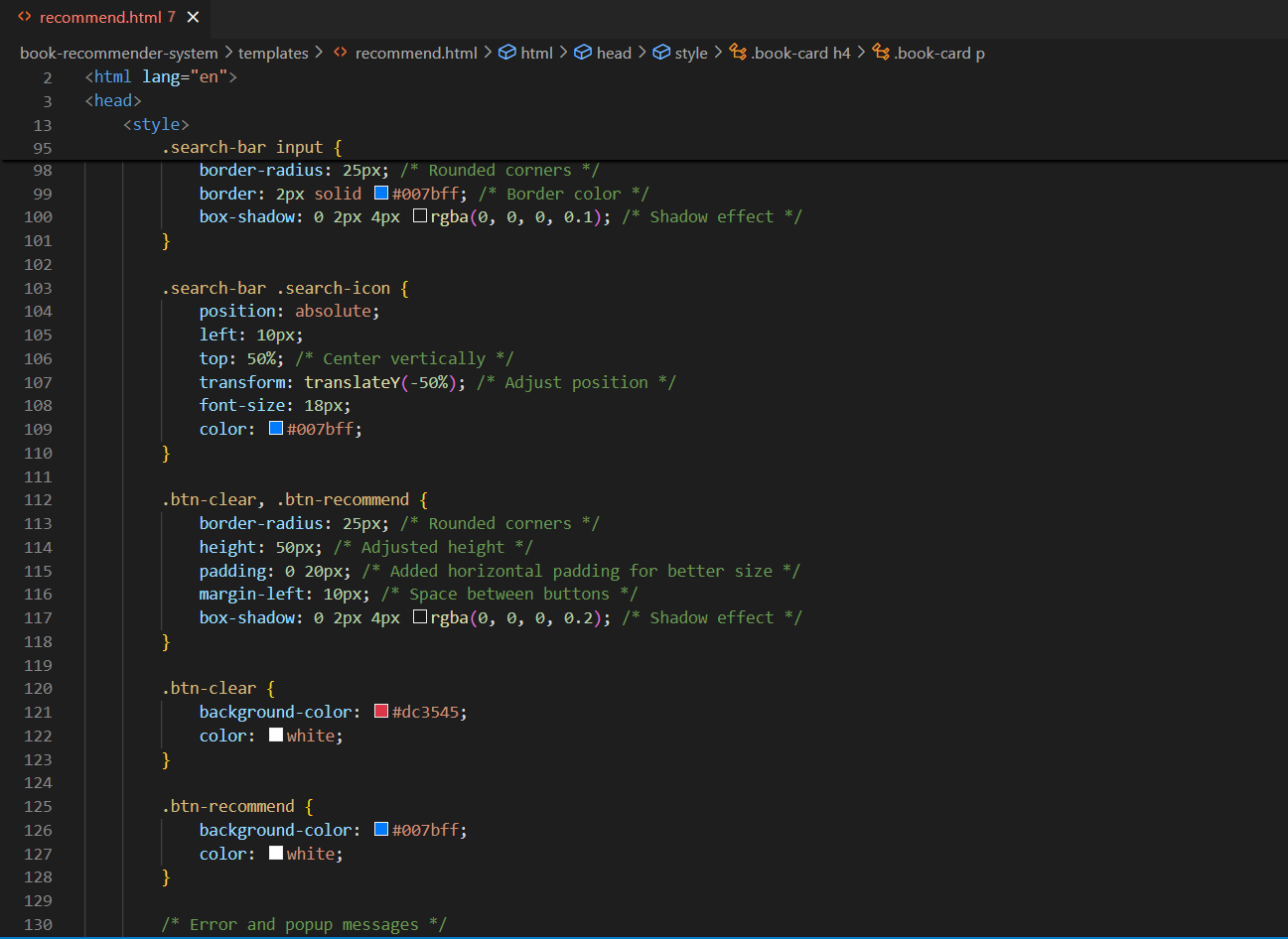
****

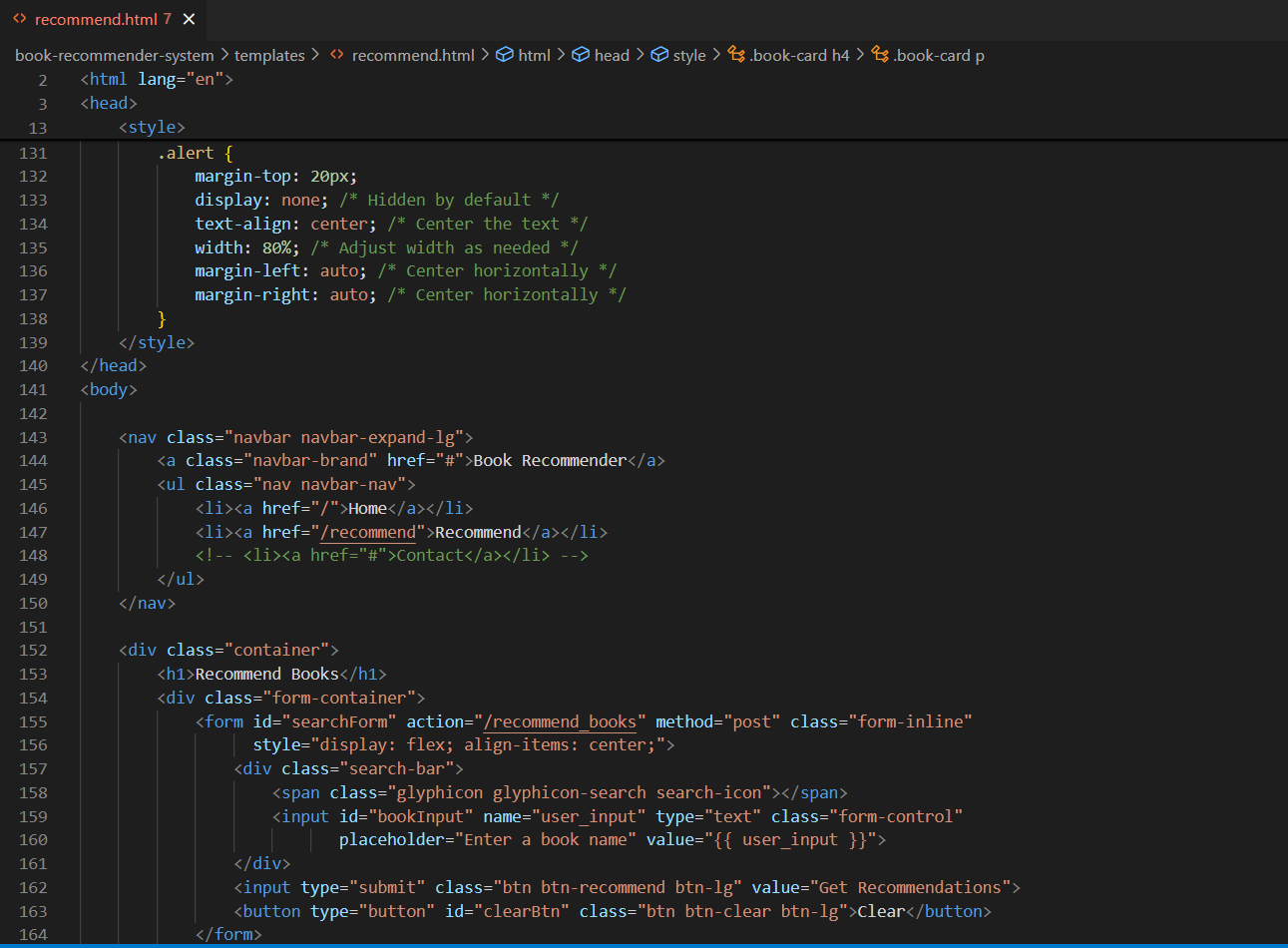
**Recommend.html :**

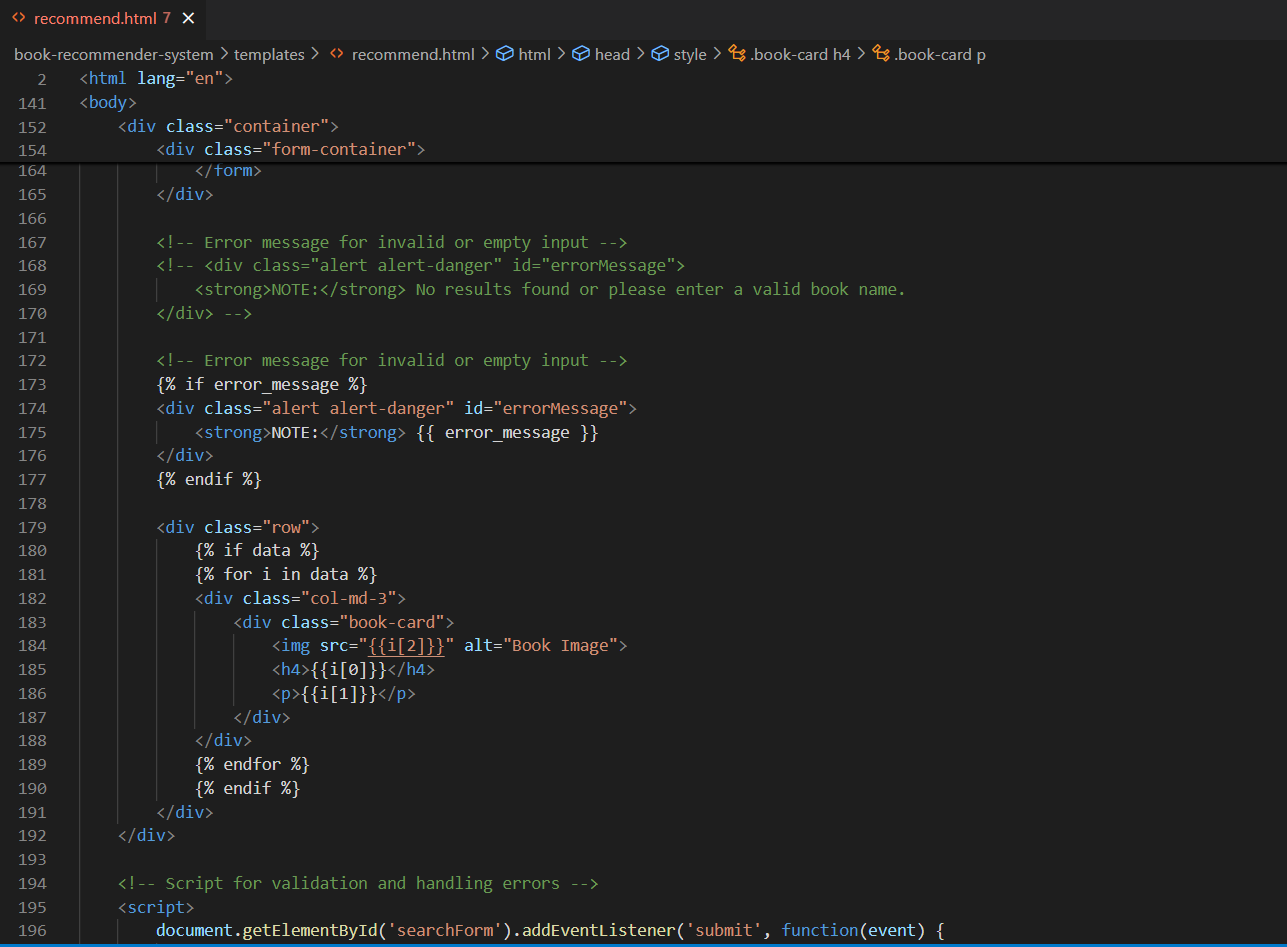
****

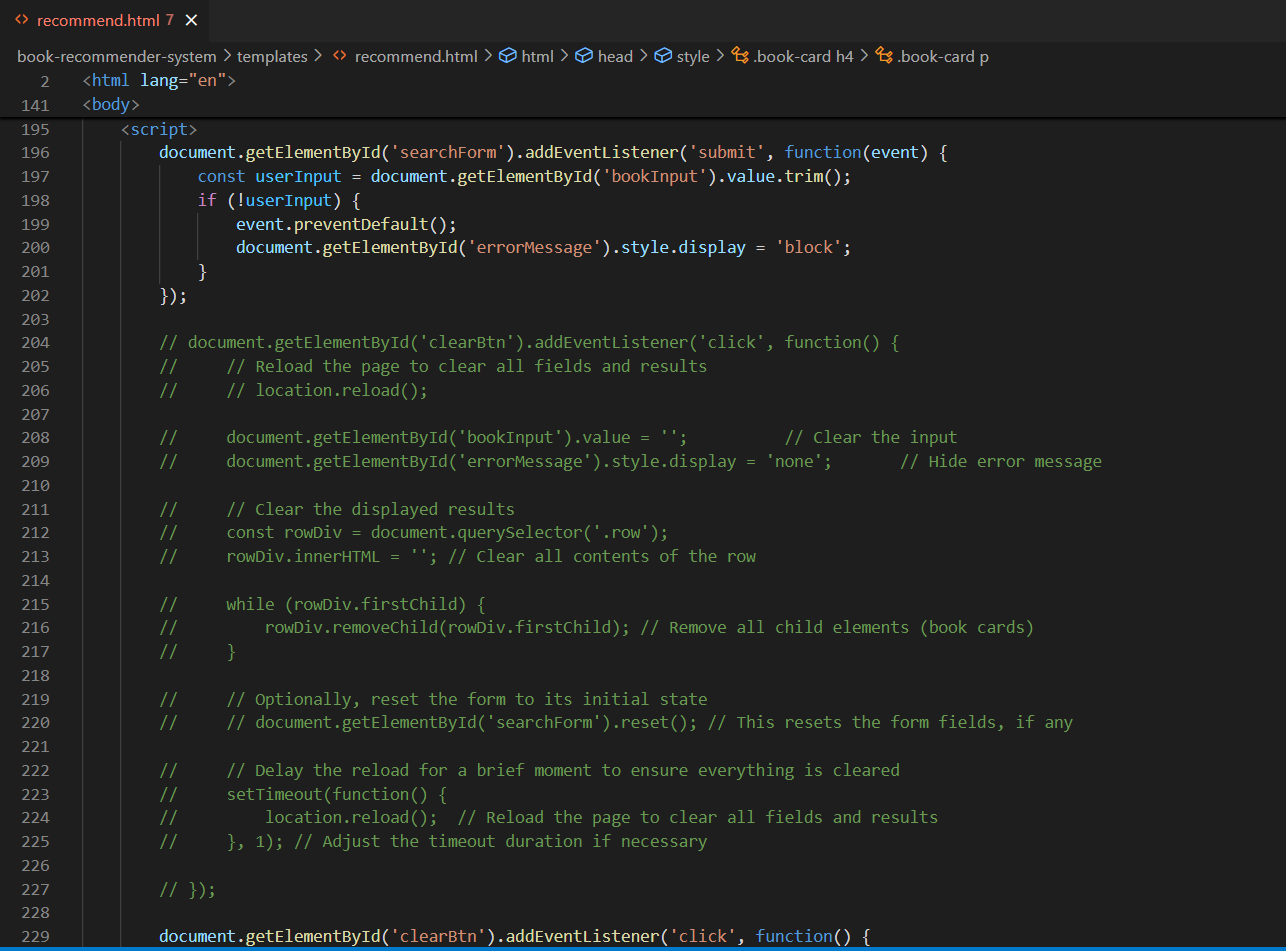
****

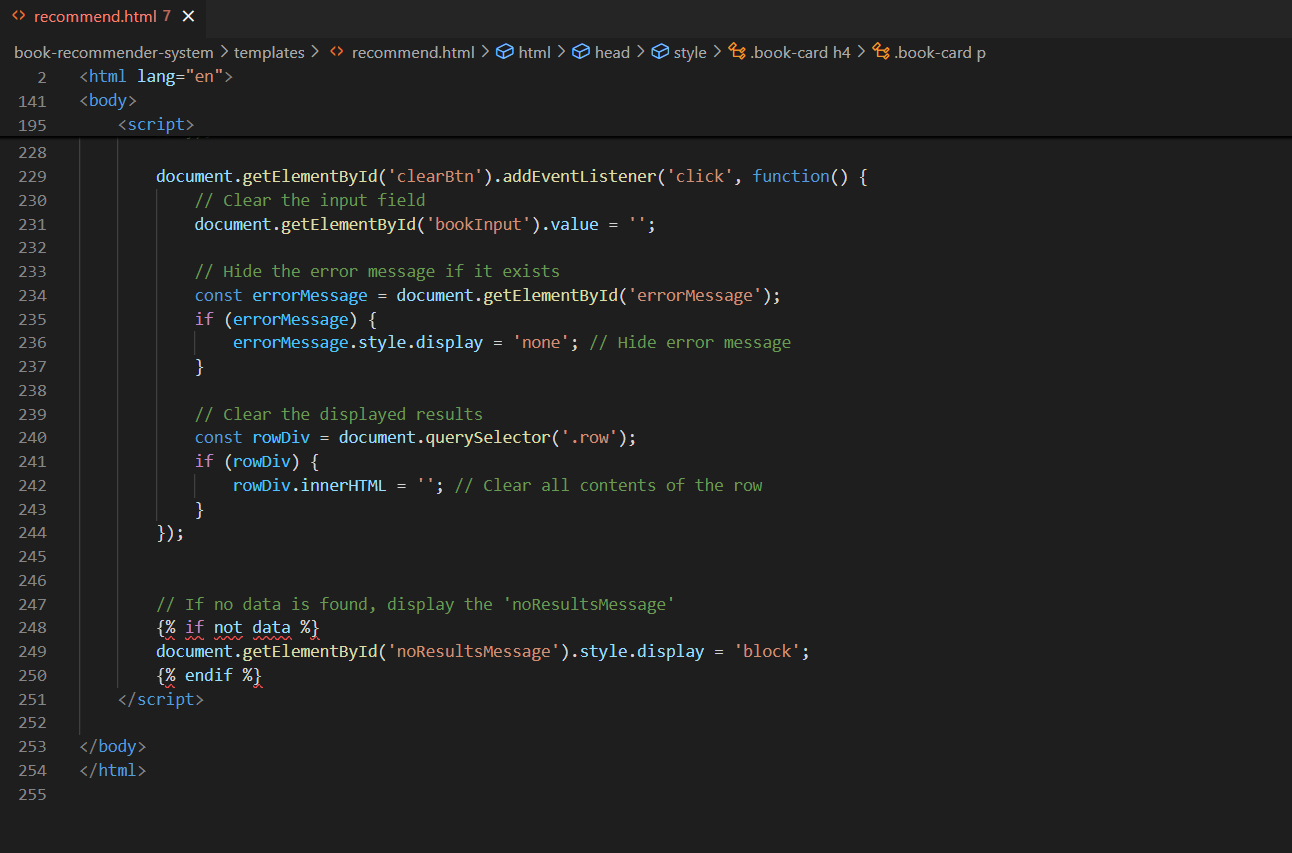
****

****

****

****

****

****

**B.2 Input and Output:**

**Input Process:**

1. **Homepage Interaction:**
   * Upon landing on the homepage (index.html), the user is presented with a static list of the top 50 books. This list is derived from the popularity-based model, which calculates the books with the highest number of ratings and best average ratings.
   * **Example Input:**

A user views the homepage and browses through the top 50 books. Each book's details, including title, author, rating, and votes, are displayed in an organized grid.

1. **Recommendation Input:**
   * On the recommendation page (recommend.html), users are prompted to enter the name of a book. For instance, the user might type "Harry Potter" into the search bar. This input is passed to the Flask backend, which queries the collaborative filtering model for similar books.
   * **Example Input:**

The user types in a book title like "Harry Potter" into the search bar and submits the form.

**Output Process:**

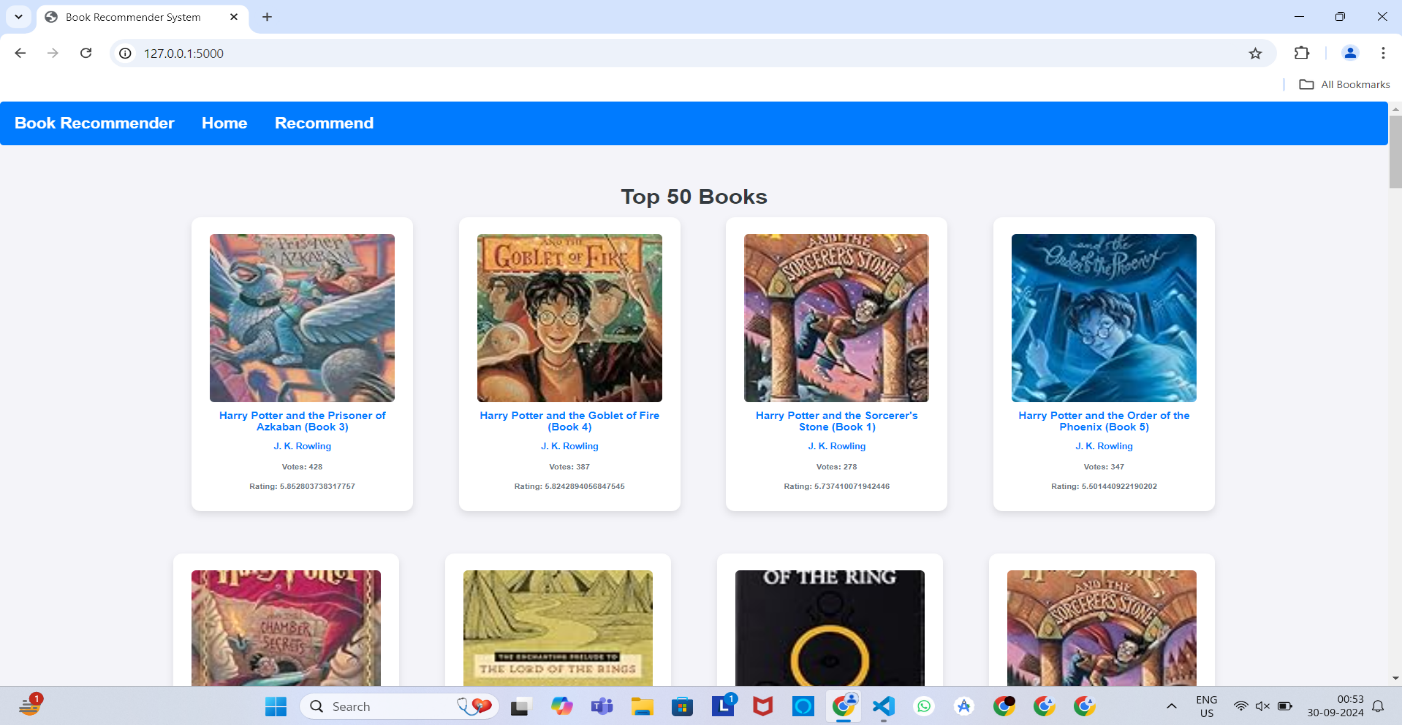
1. **Popularity-Based Output:**
   * On the homepage, the user sees a pre-generated list of the top 50 books. These are presented with details such as book title, author, rating, and number of votes. This output is static and consistent for all users, as it reflects general trends rather than personalized suggestions.
   * **Example Output:**

The user sees "The Alchemist" by Paulo Coelho with a rating of 4.7 and 1200 votes. This provides an overview of popular choices within the system.

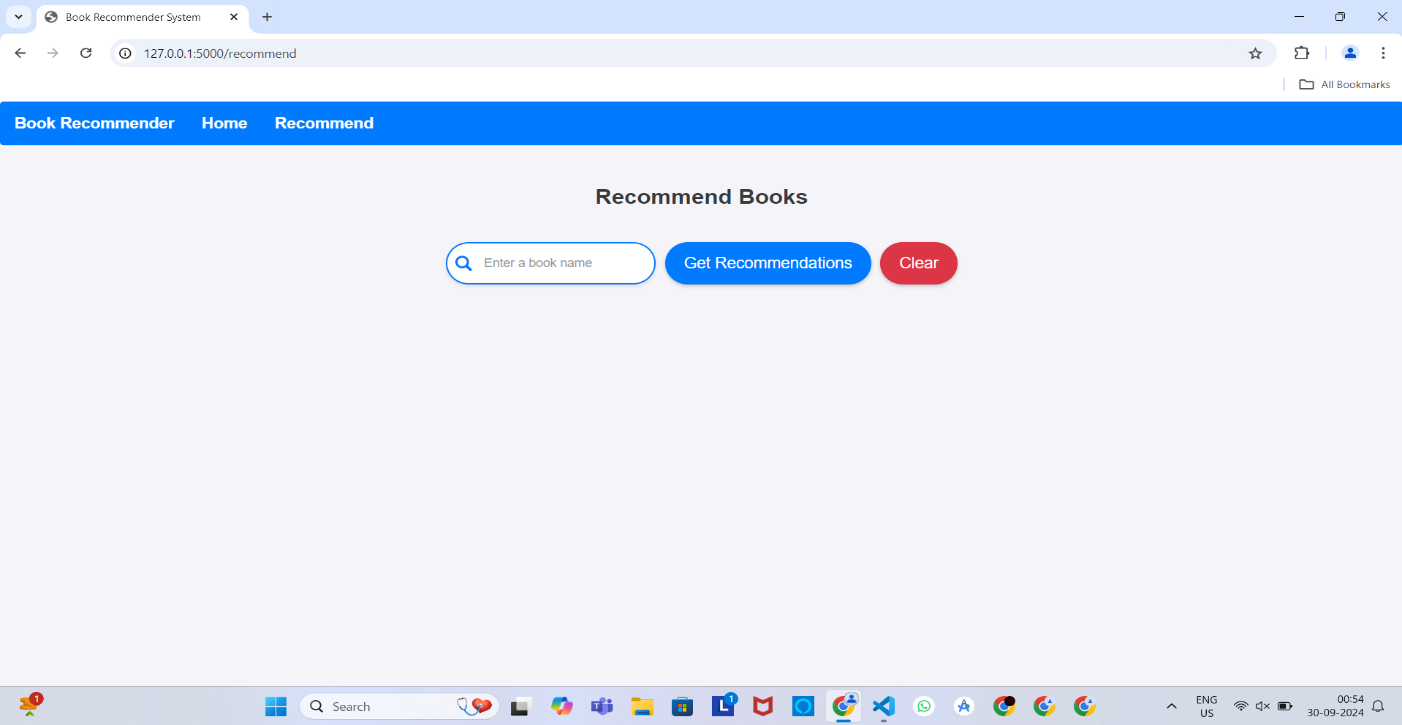
1. **Collaborative Filtering-Based Output:**
   * Once the user submits their chosen book title on the recommendation page, the system retrieves books that are most similar to the input title based on collaborative filtering. The results are presented in a similar card format to the homepage, but the content is tailored specifically to the user’s input.
   * **Example Output:**

If the user enters "Harry Potter", the system might return recommendations such as "Percy Jackson", "The Chronicles of Narnia", and other books with similar themes or audience preferences. Each recommendation includes the book’s cover image, title, and author, providing a visually consistent and intuitive experience.

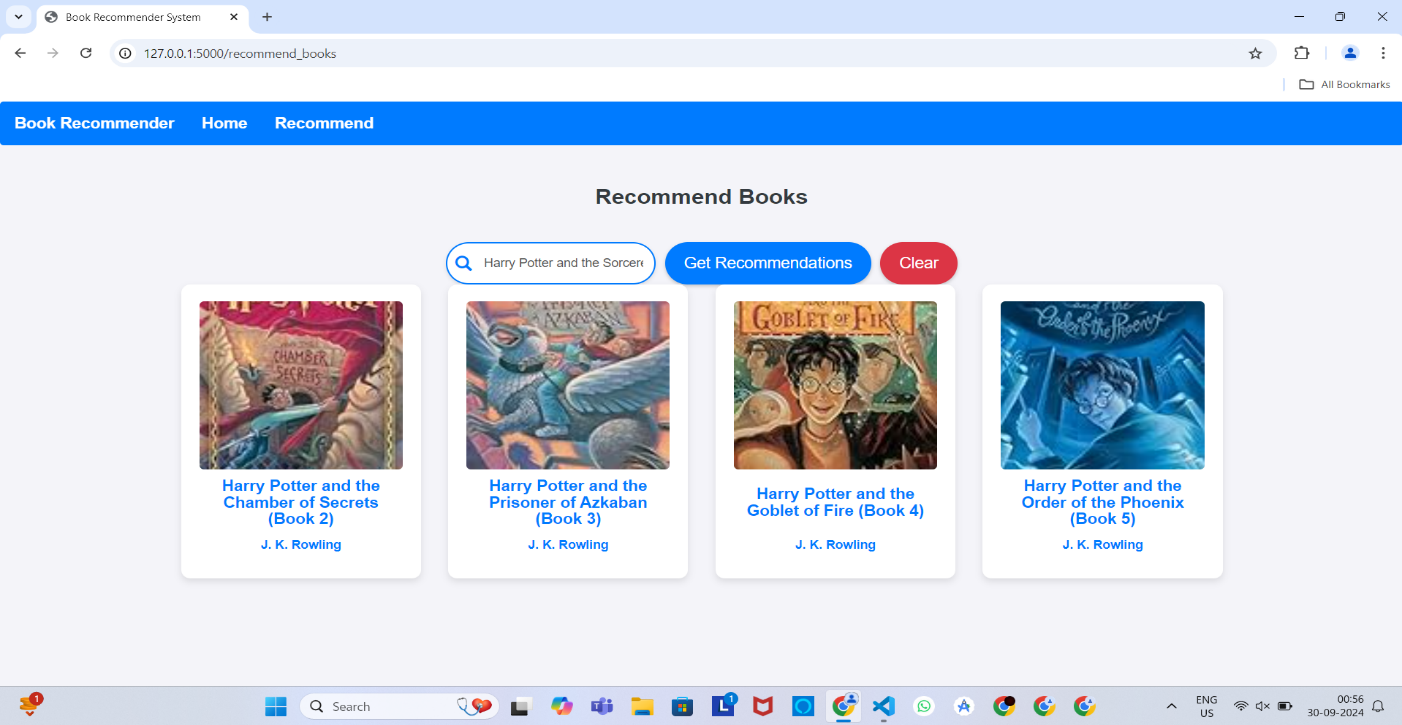
### Initial view of Top Books:



### Get recommendation page:



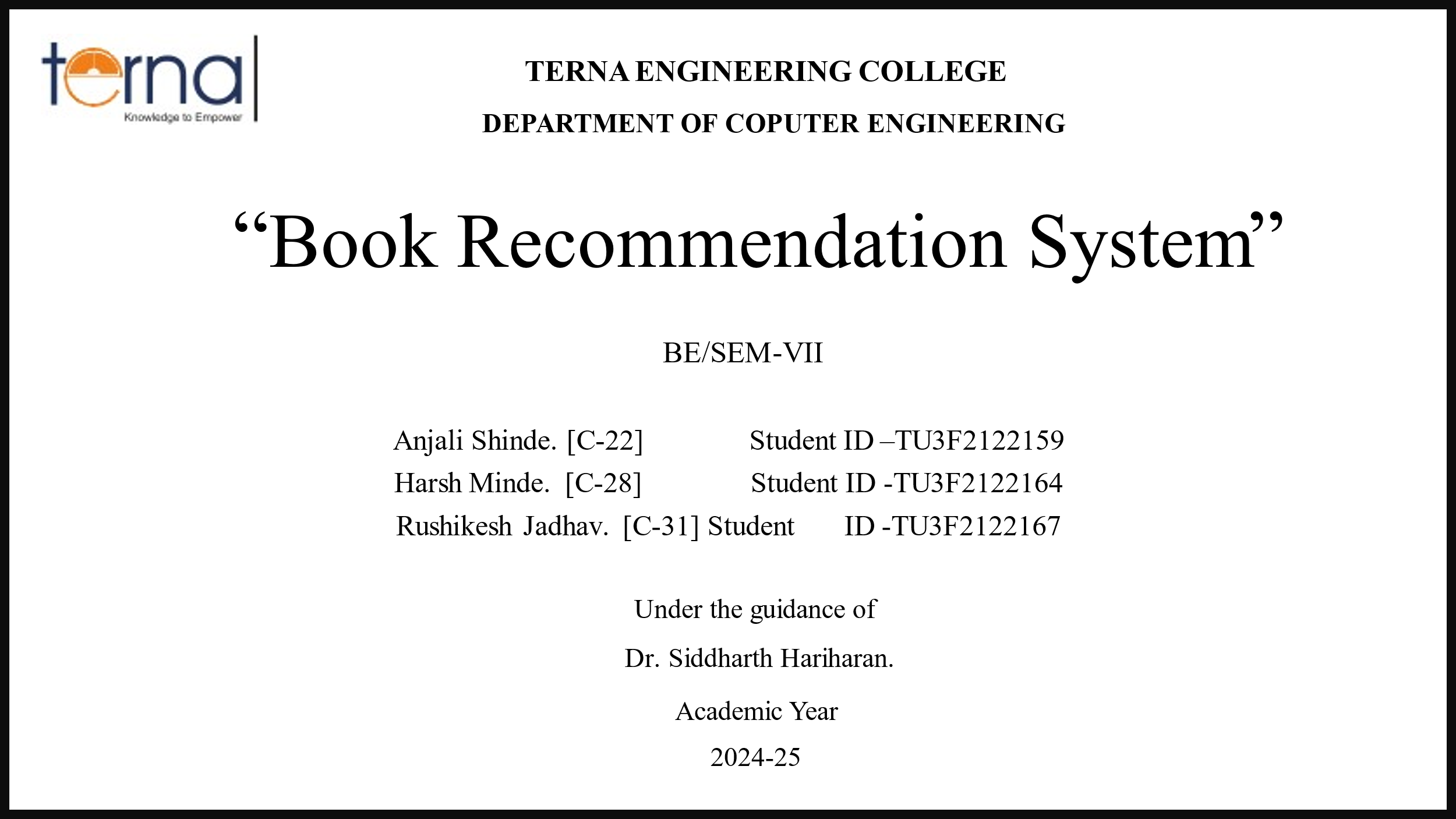
### Recommended results:

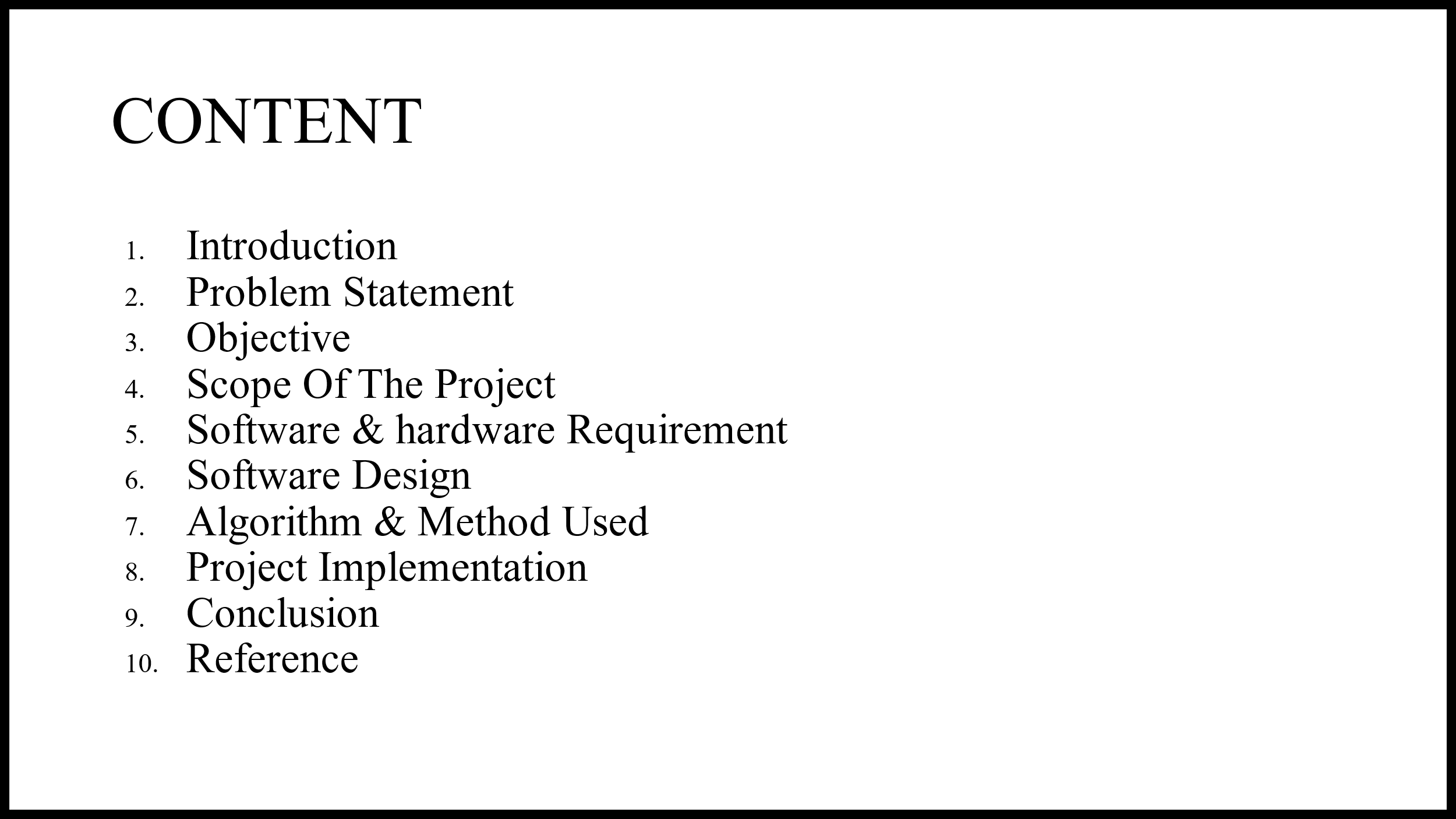


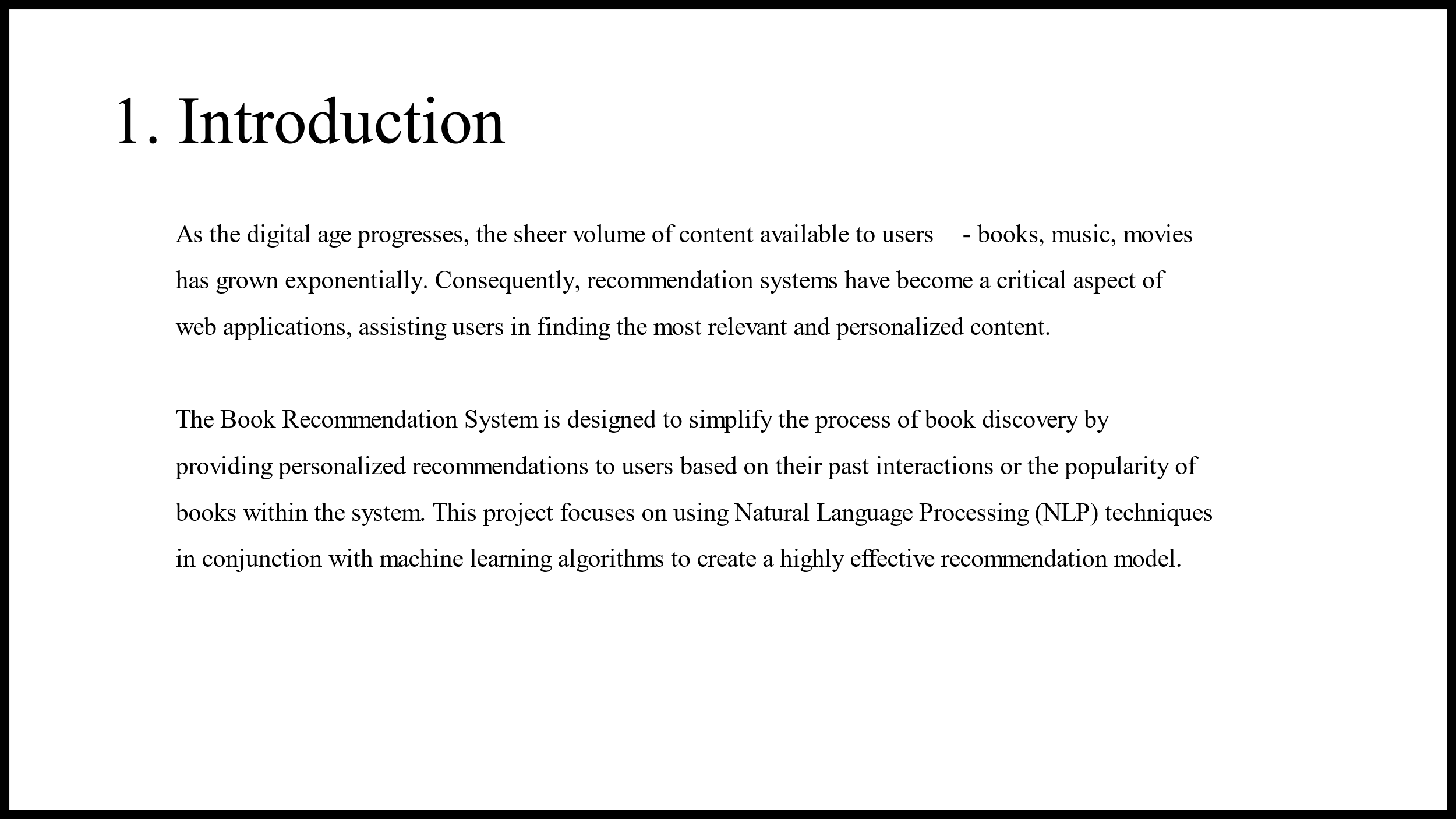
### Recommended results:

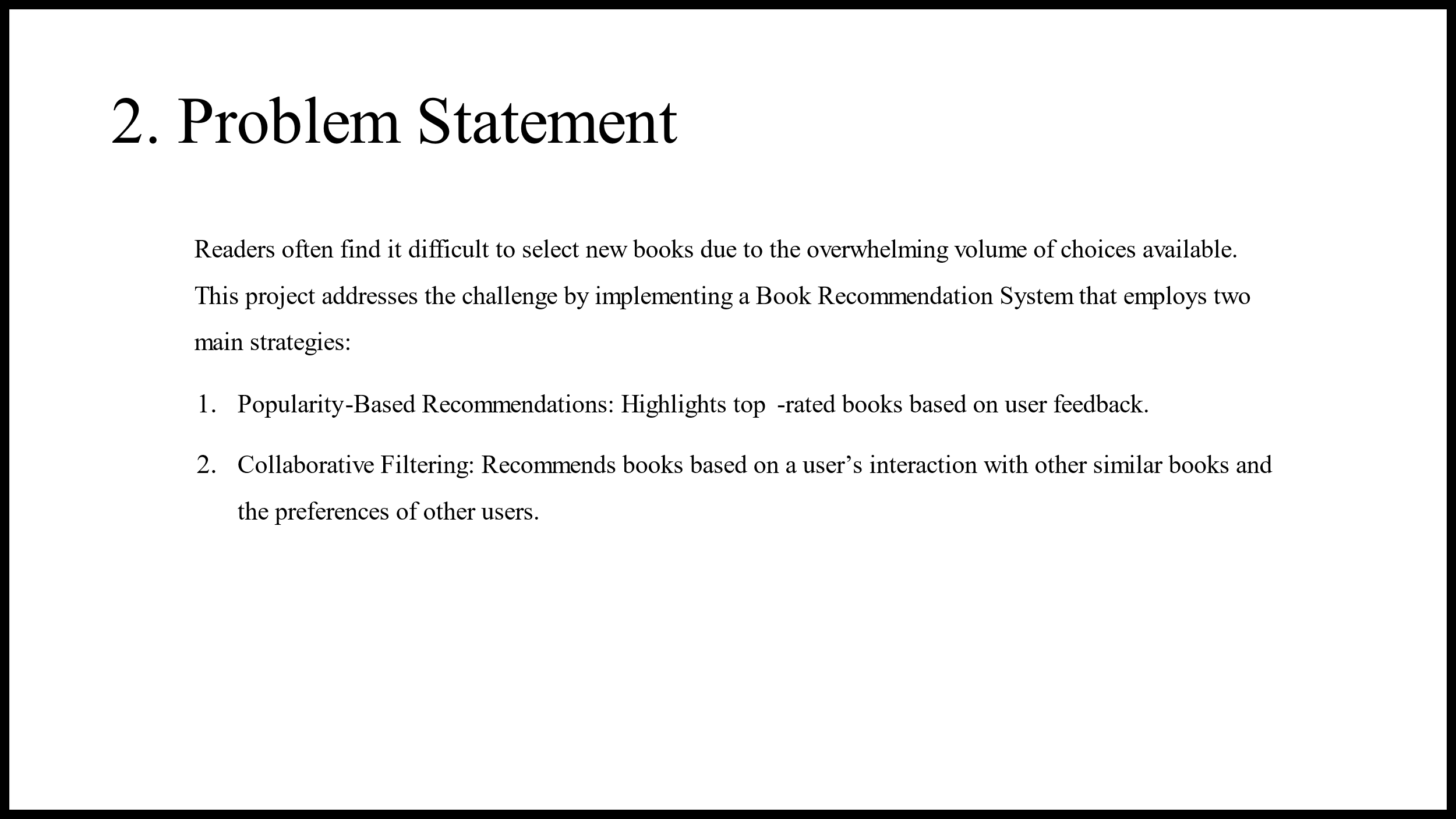
### :

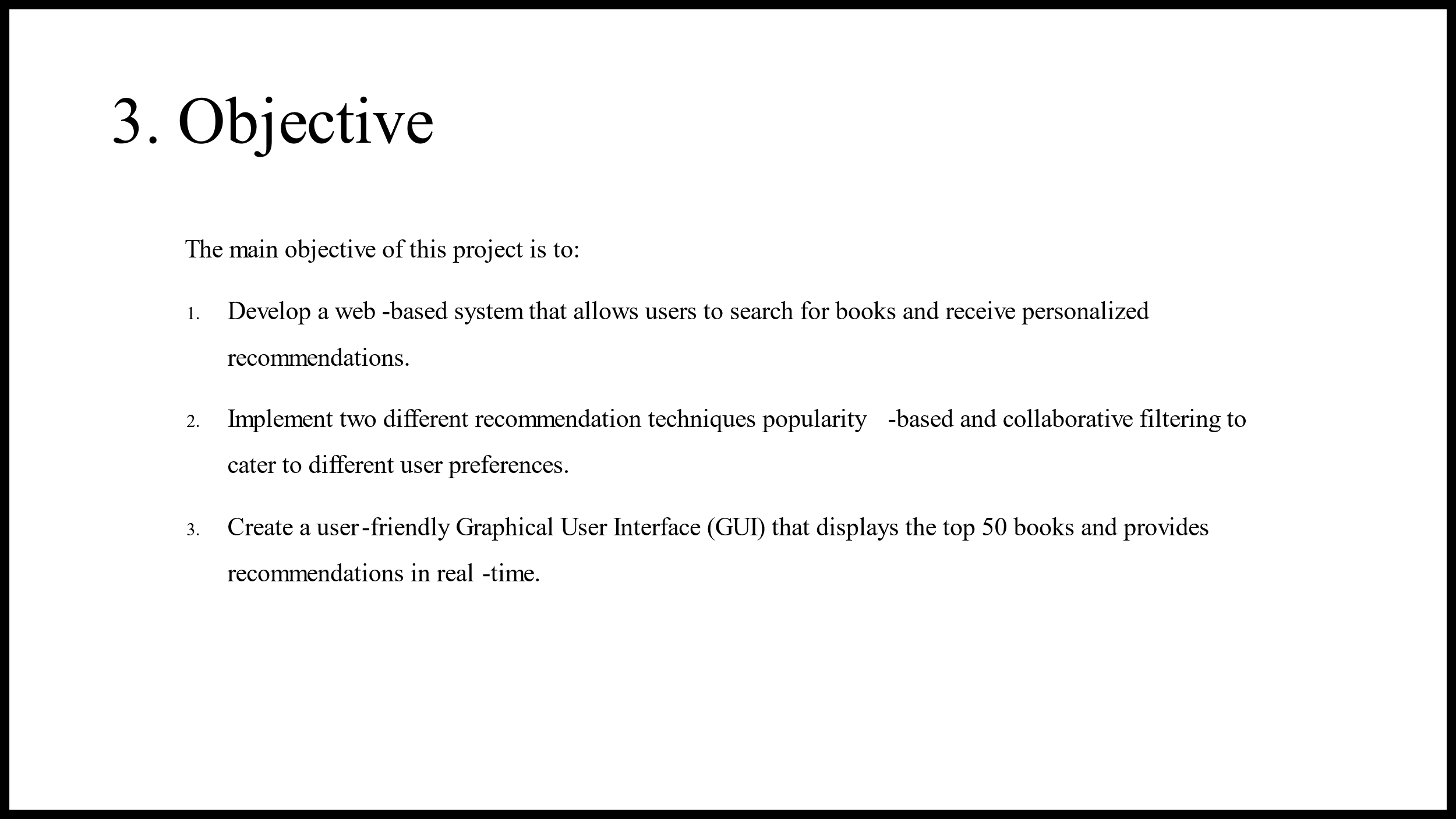
**B.3 Observations and learning:**

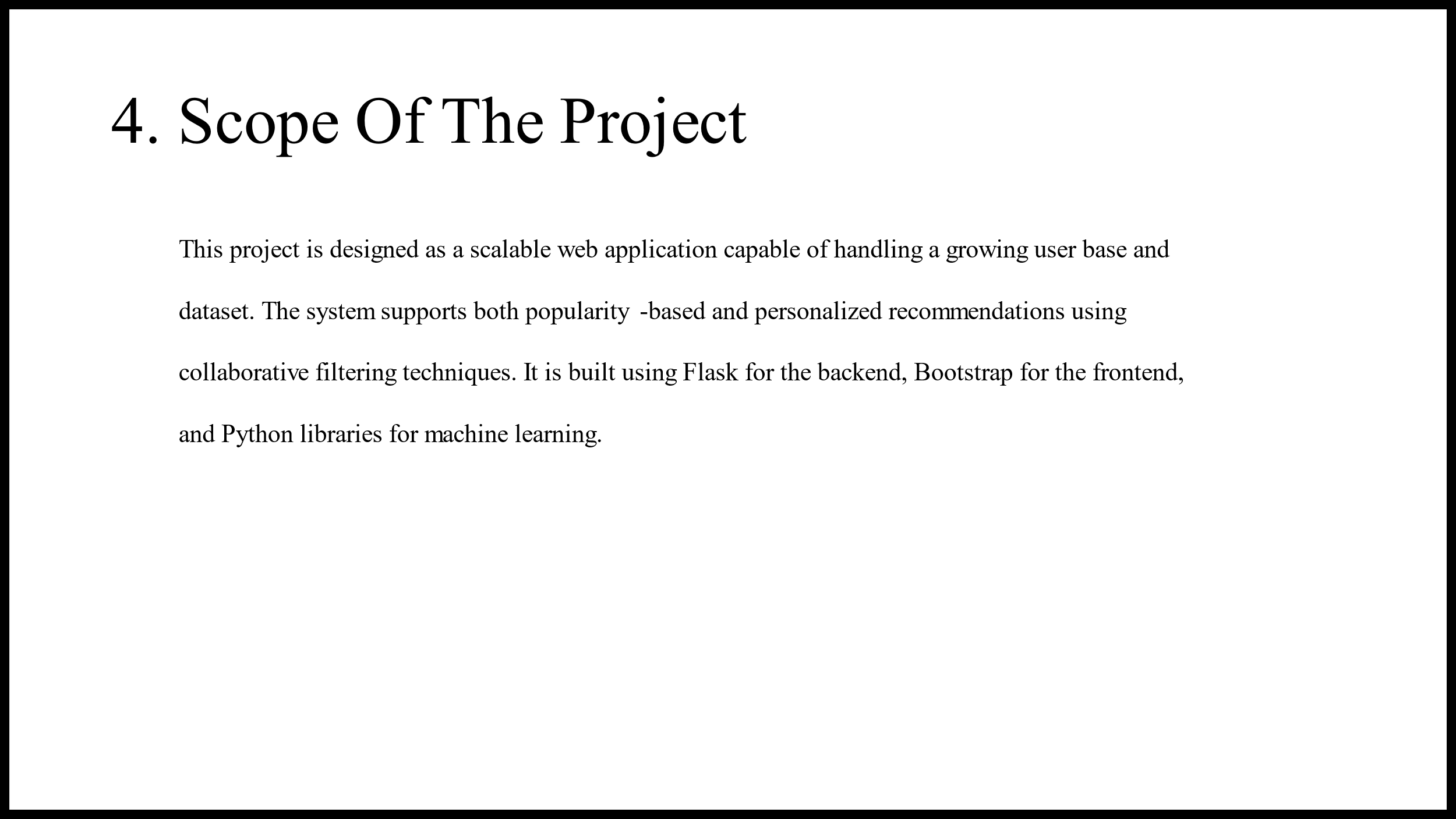


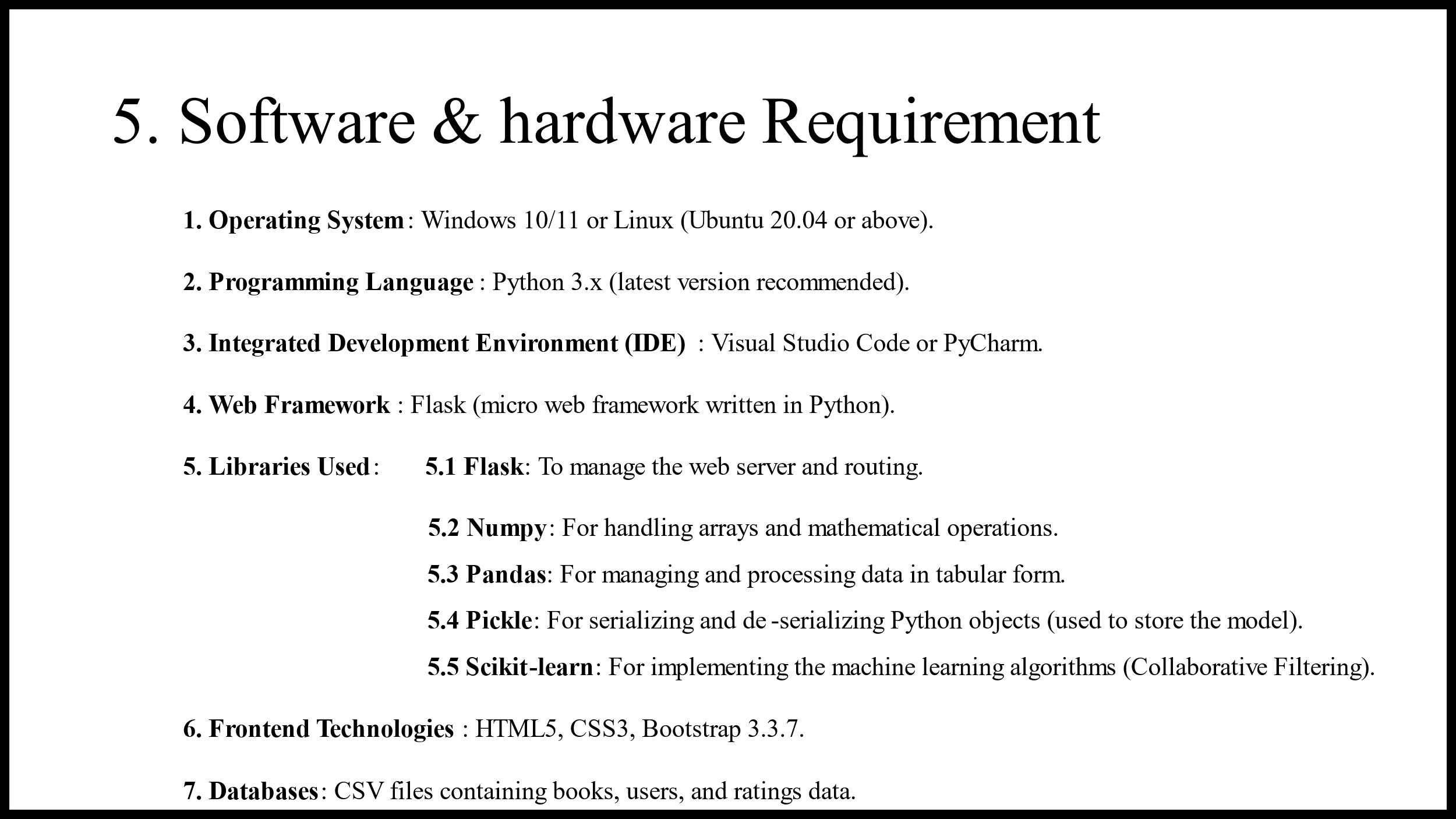


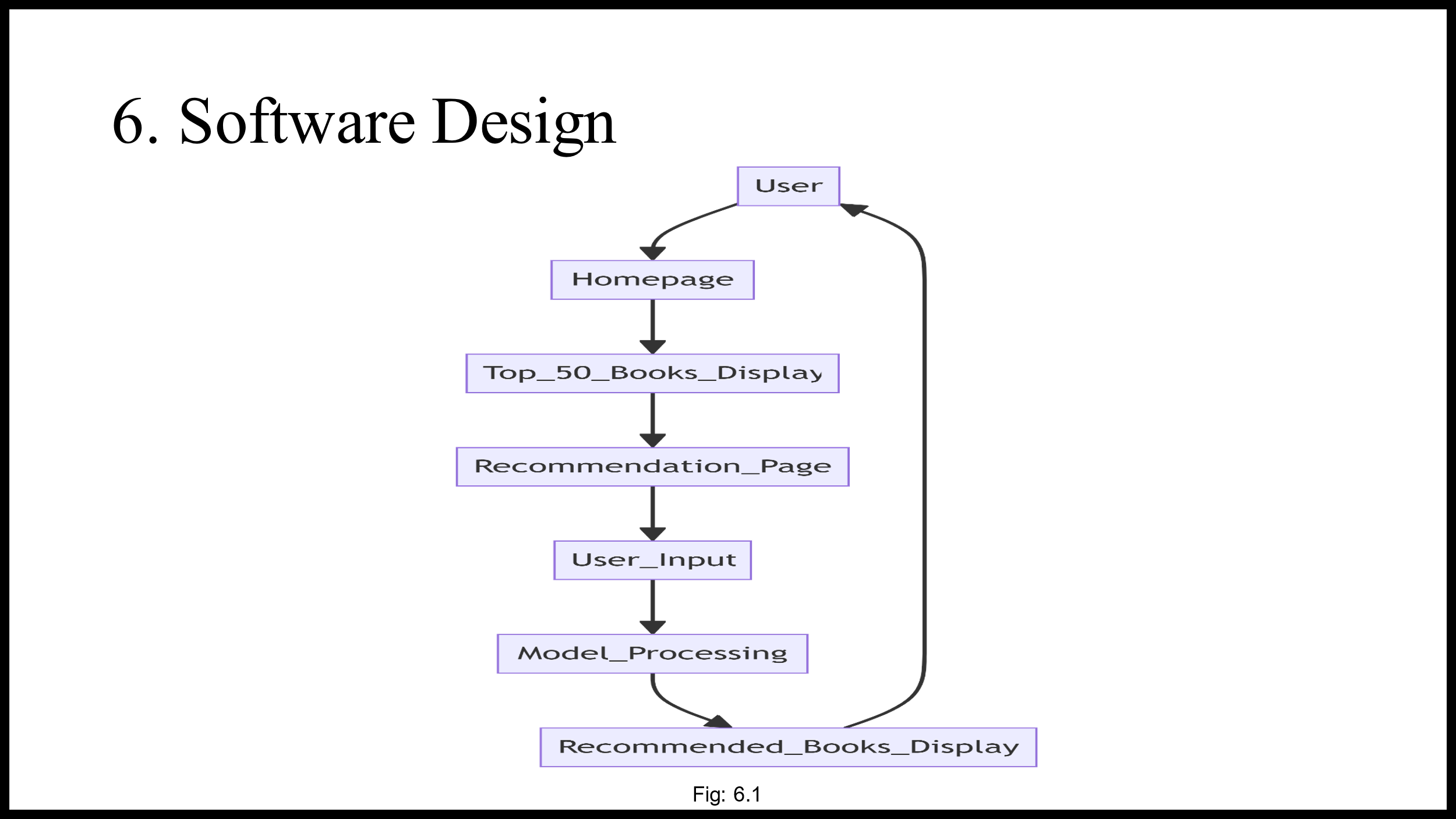


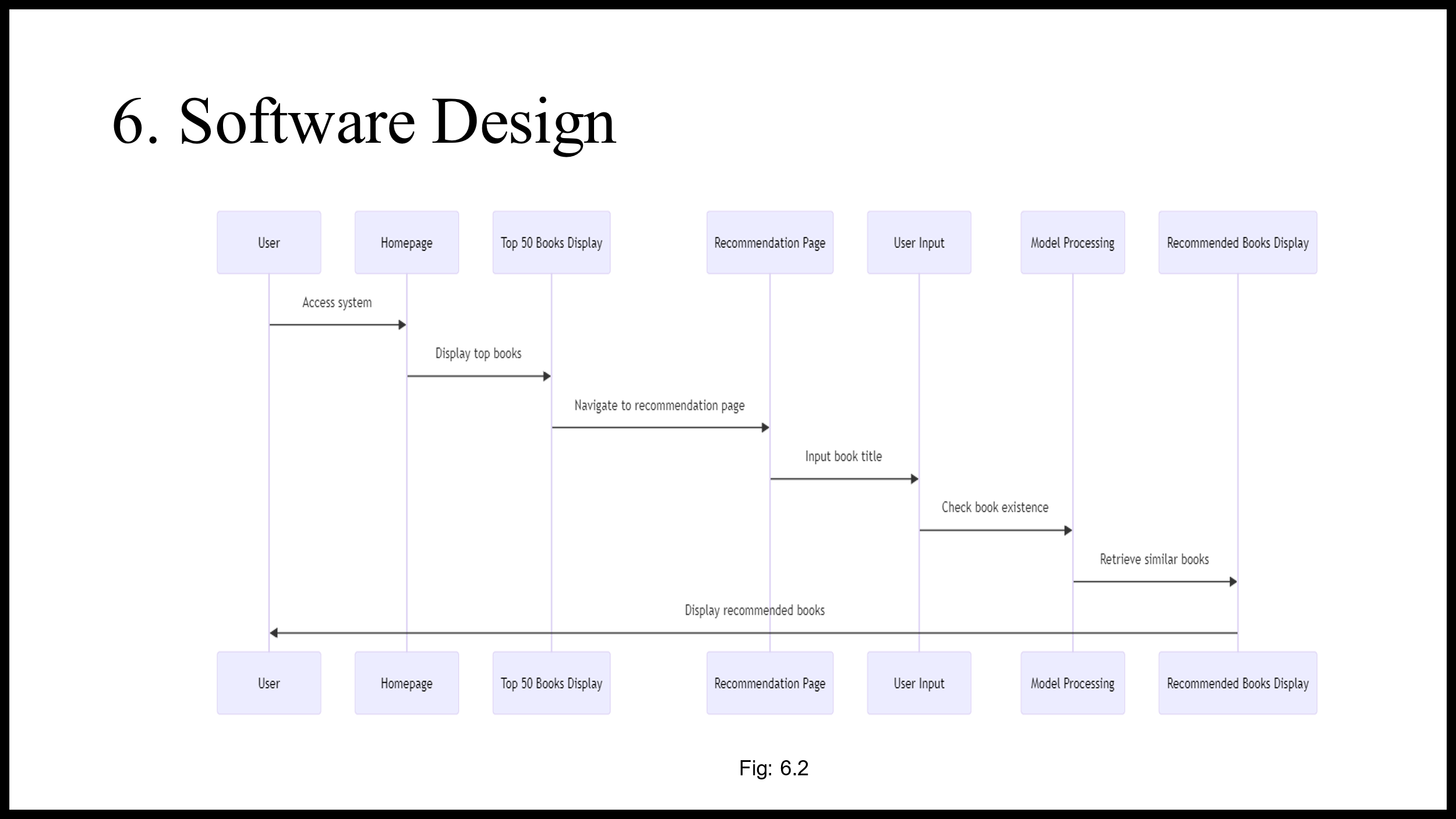


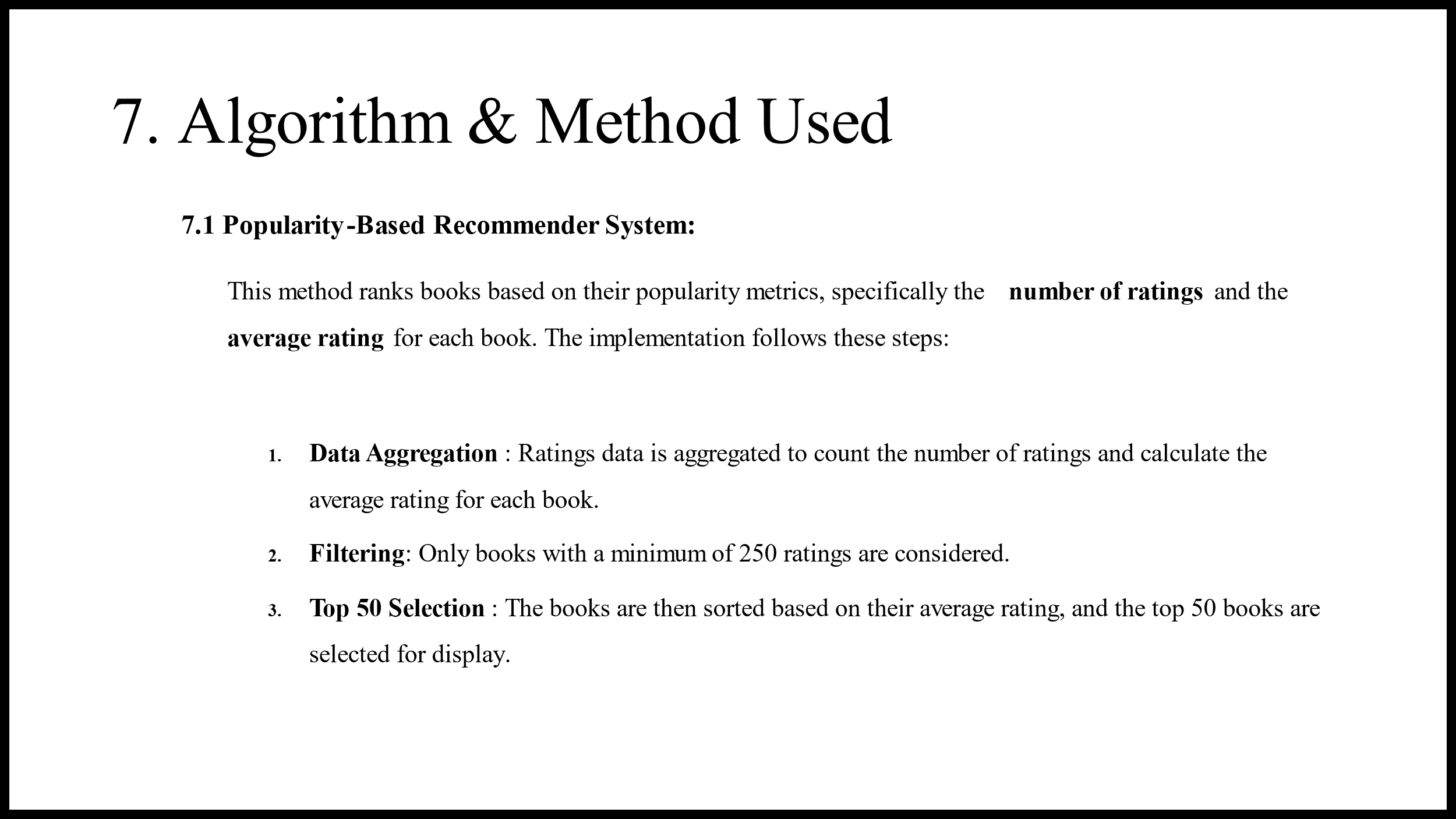


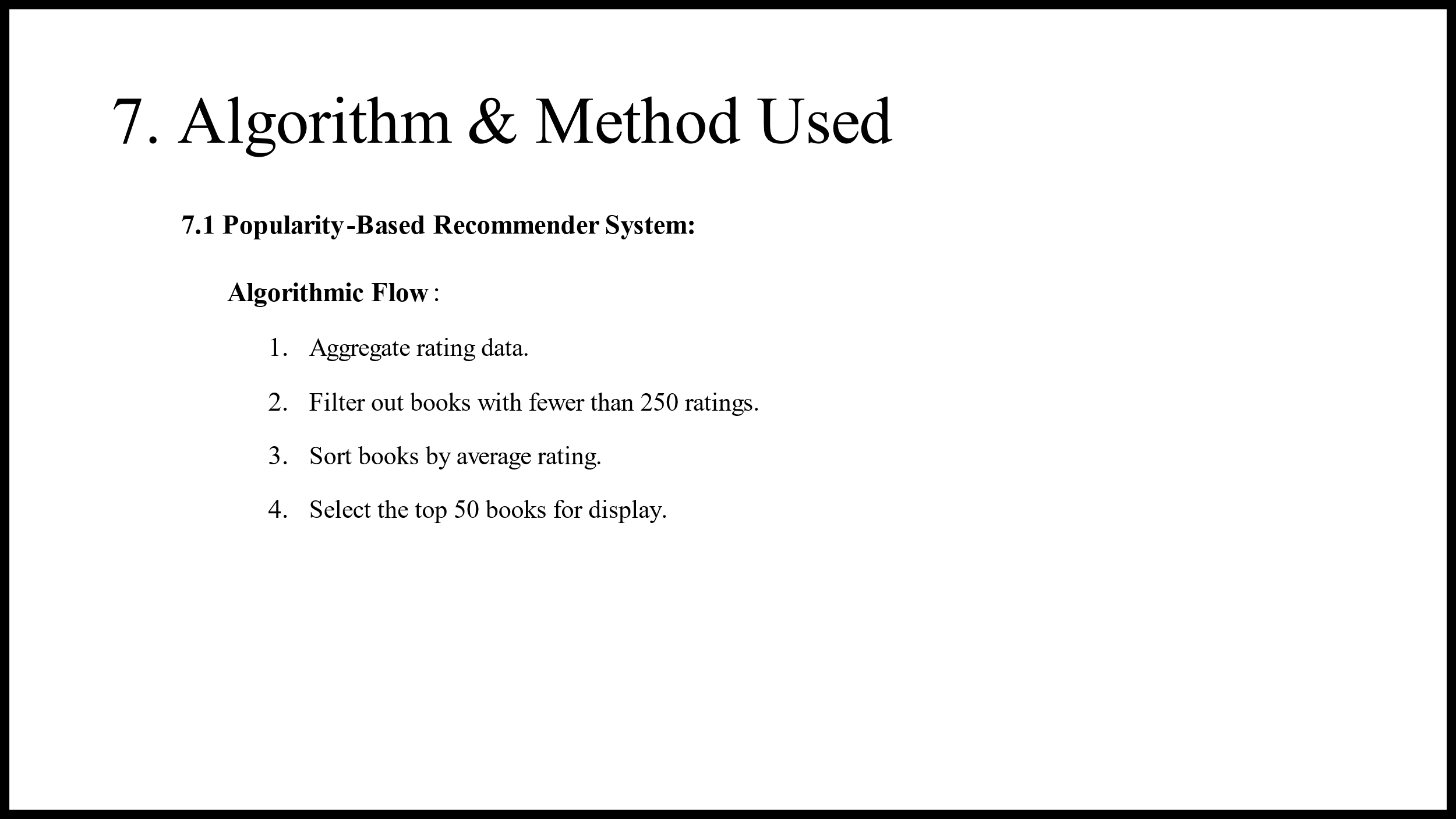




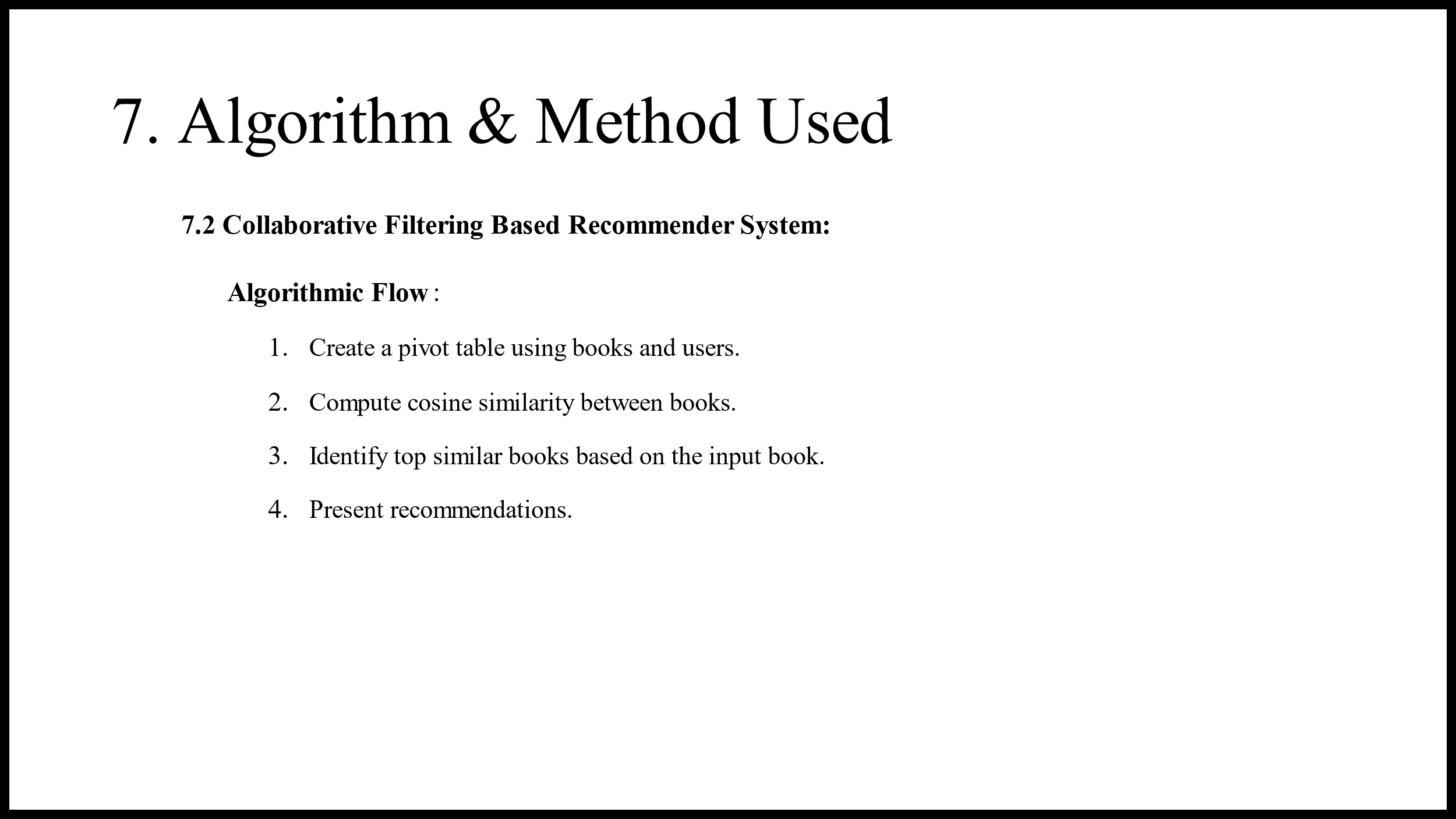


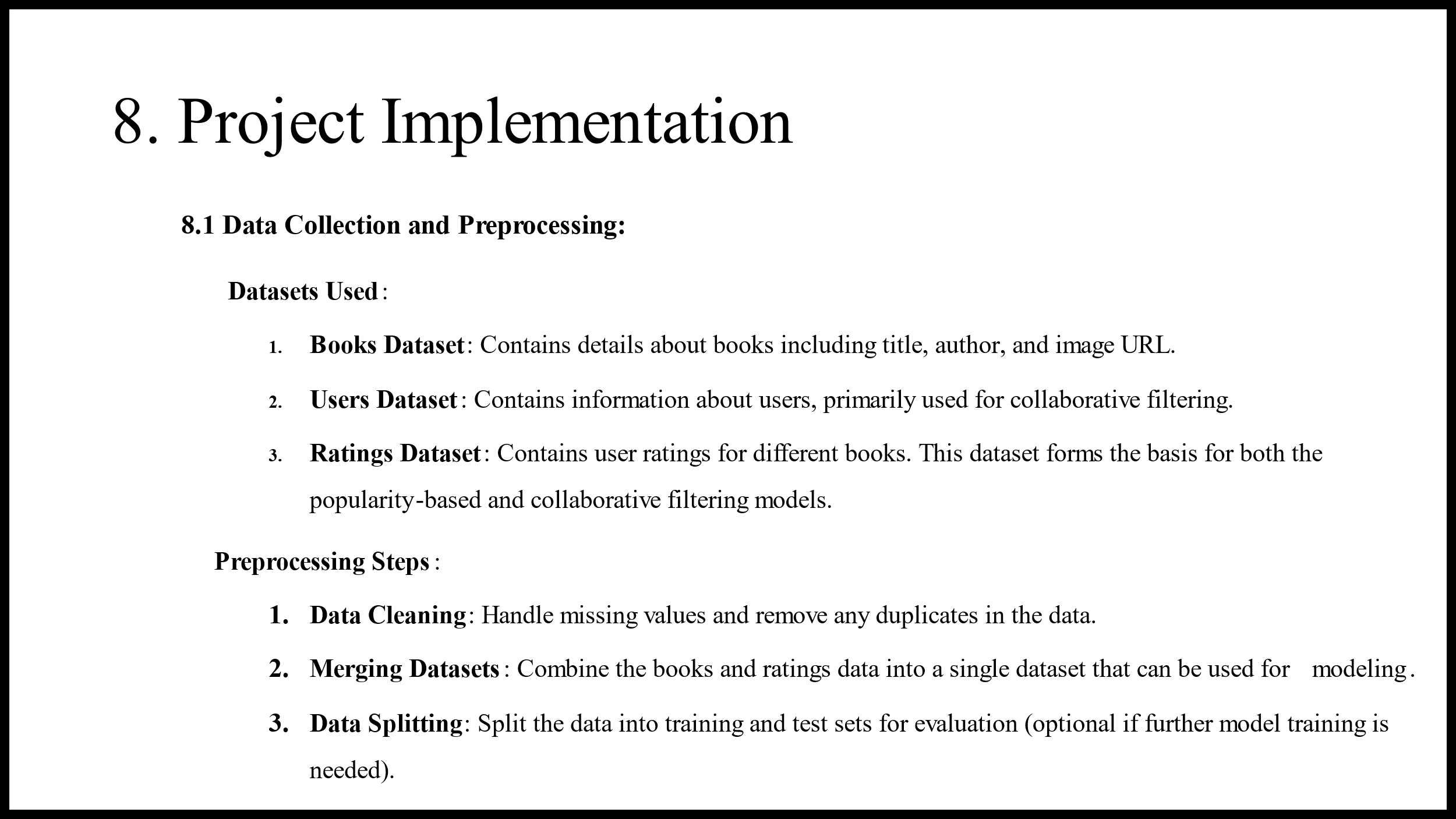


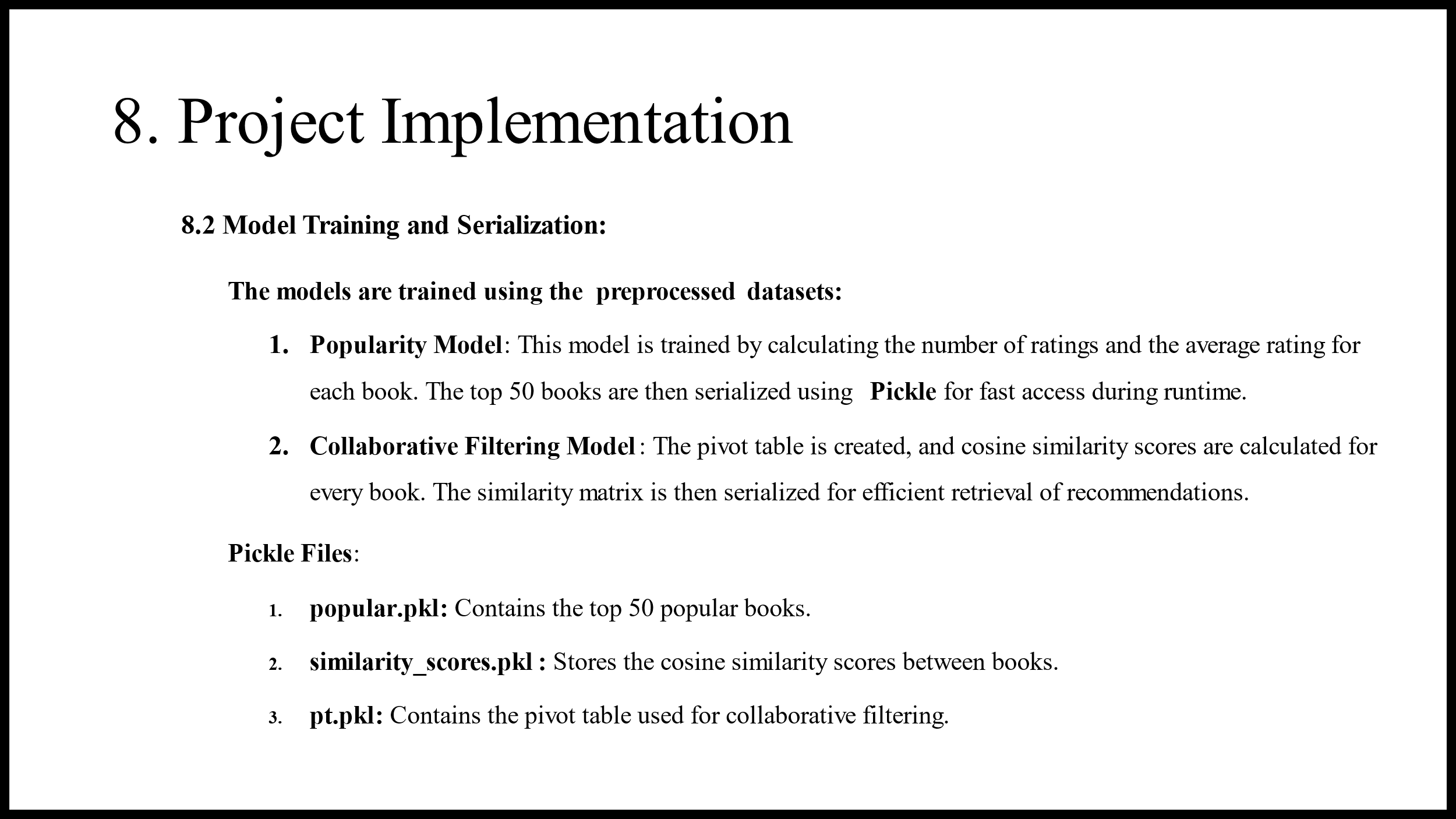


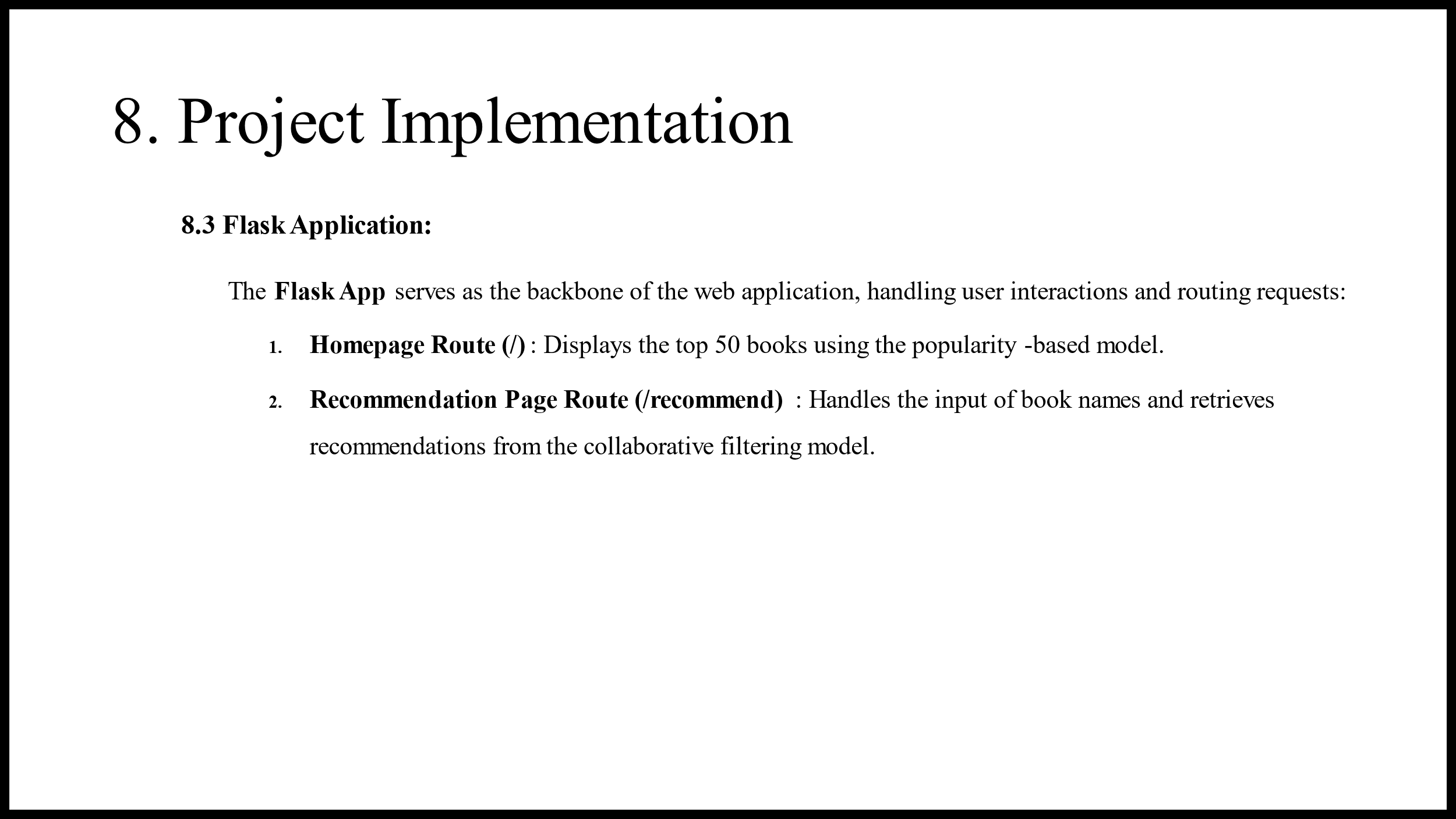


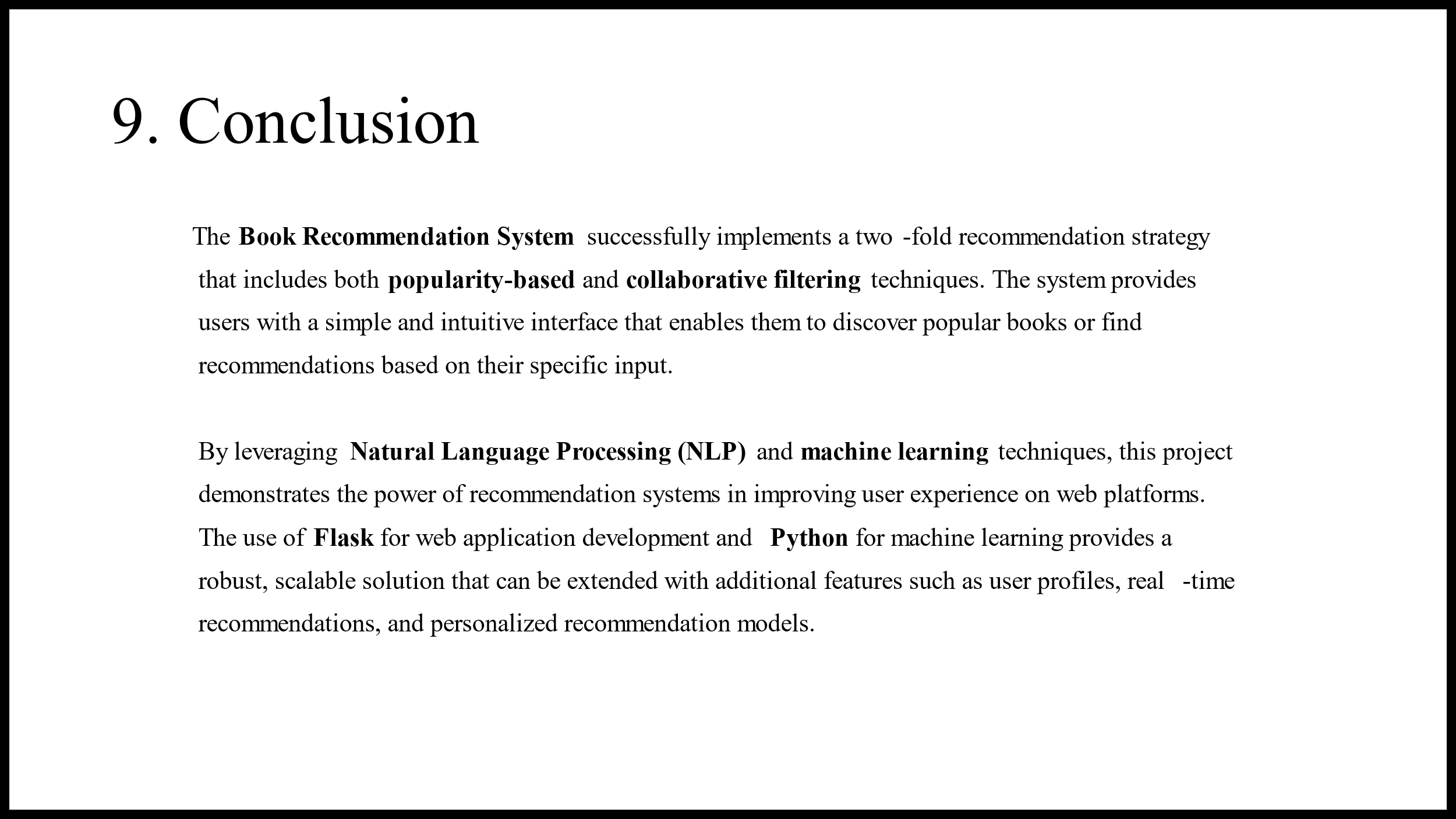


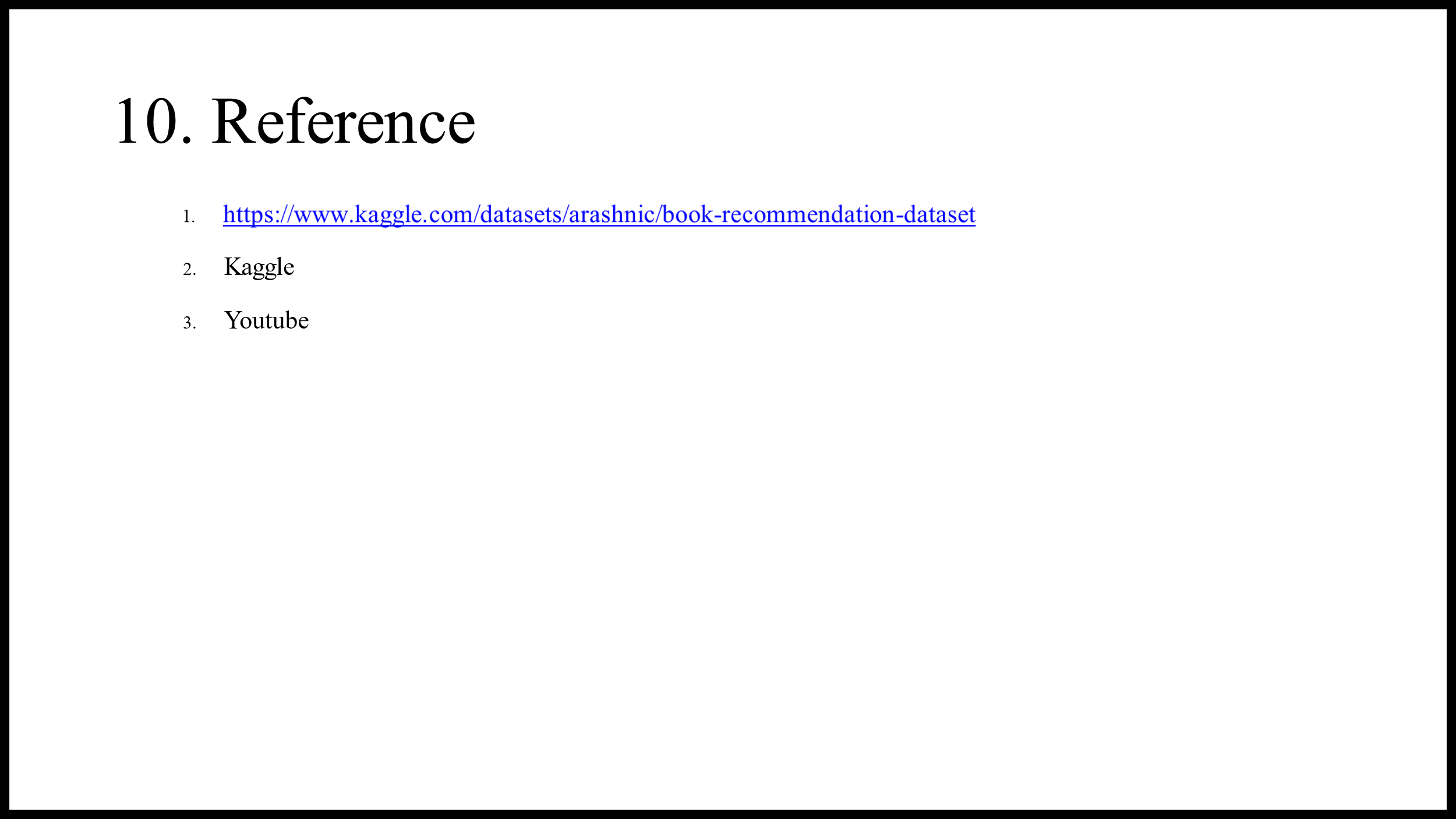












**B.4 Conclusion:**

The **Book Recommendation System** effectively implements a dual recommendation strategy, combining popularity-based recommendations and collaborative filtering techniques. By utilizing **Natural Language Processing (NLP)** and **machine learning**, the system demonstrates the potential of modern recommendation systems to significantly improve user experiences on web platforms. The simplicity and intuitiveness of the user interface allow users to easily discover popular books or obtain personalized recommendations based on their specific input.

**Key aspects of the system include:**

* **User Interaction**:

The system enables seamless interaction, where users input book titles and receive tailored recommendations. Flask handles these interactions efficiently, ensuring the system remains responsive even under different user requests.

* **Efficient Computation**:

By precomputing models and leveraging serialized files for data storage, the system quickly processes large datasets and delivers recommendations without unnecessary delays, ensuring a smooth user experience.

* **Engaging User Experience**:

Balancing static, popularity-based suggestions with dynamic, collaborative filtering results, the system effectively caters to a wide range of user preferences, enhancing engagement and satisfaction.

The project showcases the integration of **Flask** for web application development and **Python** for machine learning, creating a scalable and robust solution. Additionally, the system provides opportunities for future extensions, such as adding user profiles, real-time recommendation updates, and advanced personalization techniques. Through a successful combination of web development and data science, this recommendation platform delivers both functional and personalized results, making it a powerful tool for book discovery and recommendation.

**THANK YOU !!**