A Project Report Submitted In Partial Fulfillment Of The Requirement For The Degree Of Master Of Computer Application

On
Book Recommendation
Mini Project KCA-353

MCA-II year/ III Semester

Submitted By

Kartikay Srivastava	2301640140096
Shivam Nishad	2301640140169
Yash Chaurasia	2301640140213

Under the Supervision of

Mr. Rahul Bajpai
(Assistant Professor)

Pranveer Singh Institute Of Technology, Kanpur

To The

Dr. A. P. J. Abdul Kalam
Technical University, Uttar Pradesh

(2023-24)

Certificate

This is to certify that the report entitled **“Book Recomendation”** , submitted by **Kartikay Srivastava** to the Pranveer Singh Institute of Technology Kanpur, for the award of the degree of **Master of Computer Applications**, is a record of the original, bona fide work carried out by him under our supervision and guidance. The report has reached the standards fulfilling the requirements of the regulations related to the award of the degree.

The results contained in this report have not been submitted in part or in full to any other University or Institute for the award of any degree or diploma to the best of our knowledge.

Mr. Rahul Bajpai	Mr. Rahul Dev Shukla
(Project Mentor)	(Head of Department)

Declaration

I hereby declare that the project work entitled **"Book Recommendation"** was submitted to the MCA Department , PSIT, Kanpur. It is a record of an original work done by me under the mentorship of **Mr. Rahul Bajpai**, and this project work is submitted in the partial fulfillment of the requirements for the award of the degree of Master of Computer Applications. The results embodied in this project have not been submitted to any other University or Institute for the award of any degree or diploma.

. . . . . .

Kartikay Srivastava
Roll No.: 2301640140096
Date:
Place: PSIT Kanpur

Acknowledgements

Our deepest gratitude goes out to our department head, **Mr. Sumit Chandra,** for giving us the chance to work on this project and for encouraging a culture of academic achievement in our department. We also incredibly appreciative to our project mentor, **Mr. Rahul Bajpai** and we also want to express our gratitude to the MCA Department instructors for all of their help and inspiration with this effort. Without these people's assistance and combined efforts, this project would not have been feasible. We are grateful for your support and belief in us.

Abstract

This project report outlines the development and implementation of a book recommendation system designed to assist college students in discovering new and engaging reads. The system employs a combination of data analysis techniques and user input to generate personalized book recommendations. The report details the system's architecture, including data collection, feature extraction, and recommendation algorithms. We discuss the challenges encountered during development, such as data sparsity and the diversity of student reading preferences. Through user testing and evaluation, we assess the effectiveness and usability of the recommendation system in helping students find books that align with their interests, ultimately fostering a more engaging and personalized reading experience within the college community.

Contents

Certificate

Acknowledgements

Abstract

Contents

List of Figures

1 Introduction 11
- 1.1 Problem Definition 11
- 1.2 Purpose 12
- 1.3 Hardware and software specification 12
- 1.4 Problem Statement 14
- 1.5 Proposed Solution 15
- 1.6 Scope 16

2 Project Analysis **17**
- 2.1 Introduction 17
  - 2.1.1 Data Collection and Preprocessing 17
  - 2.1.2 System Development 18
  - 2.1.3 Conclusion 18
- 2.2 Study of Existing System 18
  - 2.2.1 Strengths of Existing Systems 18
  - 2.2.2 Limitations of Existing Systems 19
  - 2.2.3 Our Approach 19
- 2.3 Feasibility Study 19
  - 2.3.1 Technical Feasibility 19
  - 2.3.2 Economic Feasibility 20
  - 2.3.3 Operational Feasibility 20

2.4 Tools Used To Gather Information 20

3 Project Design 21
3.1 Software Requirement Specification 21
3.2 Software Functional Specification 22
3.3 Data Flow Diagram....23
3.3.1 Zero Level DFD 23
3.3.2 Level 1 DFD (DFD-L1)....24
3.3.3 Level 2 DFD (DFD-L2)....25
3.4 UML Diagram....25
3.4.1 Use Case Diagram....26
3.4.2 Class Diagram....27

4 System Implementation 28
4.1 Model Explanation....28
4.2 Module Description 29
4.3 Work Plan 31

5 Testing 32

6 System Input And Output 34
6.1 App Execution Code 34
6.2 Login Page Code....39
6.3 Home Page Code....41
6.4 Recommendation Page Code 48
6.5 Search Page Code....55

7 Limitation and Scope of Project 56

8 Conclusions 57

References 59

Chapter 1

Introduction

In today's fast-paced world, discovering new books that align with individual interests can be a daunting task. With a vast sea of literature available, sifting through titles and genres to find something captivating can feel overwhelming. This project aims to address this challenge by developing a book recommendation system specifically tailored for college students.

The system leverages the power of data analysis and machine learning techniques to provide personalized book recommendations. By analyzing user preferences, reading history, and book attributes, the system aims to suggest relevant and engaging reads that cater to individual tastes. This not only enhances the reading experience for students but also promotes a love for literature within the college community.

The project will explore various recommendation algorithms, such as collaborative filtering and content-based filtering, to determine the most effective approach for generating accurate and relevant recommendations. Additionally, the system will be designed with a user-friendly interface, making it easy for students to interact with and navigate.

Through this project, we aim to demonstrate the potential of data-driven solutions in enhancing the student experience and fostering a vibrant reading culture within the college environment.

1.1 Problem Definition

College students today face a significant challenge in discovering new and engaging books. With limited time and a vast ocean of literature, finding books that align with their diverse interests can be a time-consuming and often frustrating endeavor. Traditional methods of book discovery, such as browsing library shelves or relying on word-of-mouth recommendations, are often inefficient and may not always lead to truly personalized suggestions. This lack of an effective book discovery mechanism can lead to:

- **Reading Slumps:** Students may become stuck in reading ruts, repeatedly gravitating towards familiar genres or authors, hindering their literary exploration.
- **Time Waste:** Significant time can be wasted searching for books that ultimately fail to capture their interest.
- **Limited Exposure:** Students may miss out on hidden gems or diverse genres that could broaden their literary horizons.

1.2 Purpose

Book Recommendation Systems (BRS) have become indispensable tools in the digital era, serving as intelligent companions that help users discover, explore, and select books based on their interests and preferences. This report aims to explore the purpose and significance of BRS in modern society.

Convenience and Efficiency: The primary purpose of BRS is to simplify the process of finding relevant books for users. By leveraging data-driven algorithms and user preferences, BRS eliminate the need for extensive manual searches, saving time and effort while enhancing the overall reading experience.

Accessibility: BRS play a crucial role in making literature accessible to a diverse audience. They bridge the gap between users and vast book collections by offering personalized recommendations that cater to varied reading levels, genres, and cultural contexts.

Information Retrieval: Another key purpose of BRS is to facilitate the discovery of books and authors. Users can explore curated lists, reviews, and book summaries, gaining insights without needing to manually sift through libraries or online stores.

Enhanced Reading Experience: BRS enable users to enhance their reading experience by recommending books based on their mood, interests, or goals. They provide suggestions for similar reads, trending titles, or classics, ensuring users always have a tailored reading list.

Personalization: Many Book Recommendation Systems (BRS) offer personalized experiences by learning from user interactions and preferences over time. They can tailor book suggestions, genres, and curated lists based on individual reading habits and profiles, making the discovery process more intuitive and enjoyable.

1.3 Hardware and software specification

Python:- Python is an OOPs (Object Oriented Programming) based, high level, interpreted programming language. It is a robust, highly useful language focused on rapid application development (RAD).

Wikipedia:- Knowledge bases play an increasingly important role in enhancing the intelligence of web and enterprise search and supporting information integration. Wikipedia leverages this gigantic source of knowledge by extracting structured information from Wikipedia and making this information accessible on the web.

NumPy: NumPy is a Python library used for numerical computations. It provides support for arrays, matrices, and a wide range of mathematical functions to perform operations on these data structures. It is highly efficient and widely used in scientific computing and data analysis.

MySQL: MySQL is a powerful, open-source relational database management system designed for efficiently storing and managing large-scale data. It supports a broad range of SQL features, including advanced data integrity, transactions, and scalability. MySQL operates as a standalone database server and can be integrated into various applications. It is highly popular for web applications and data-driven projects, offering compatibility with Python through libraries like mysql-connector and SQLAlchemy. Unlike SQLite, MySQL requires installation and configuration, making it well-suited for larger, distributed systems.

Software Requirements:- Any OS with clients and access the internet Wi-Fi Internet or cellular Network Create and design Data Flow and Context Diagram Versioning Control Medium to find reference to do system testing, display and run.

Hardware Requirements:-The software is designed to be light-weighted so that it doesn't be a burden on the machine running it. This system is being build keeping in mind the generally available hardware and software compatibility. Here are the minimum hardware and software requirement for virtual assistant.

Memory (RAM): Sufficient RAM is important for running the voice assistant software smoothly, especially if it involves processing large amounts of data or running multiple tasks simultaneously.

Storage: You'll need storage space to store the voice assistant's software, resources, and any user data it collects.

Internet Connection: Many voice assistants rely on cloud-based services for natural language processing and other functionalities. A stable internet connection is therefore essential during development and deployment.
For running the application: -

The software is designed to be light-weighted so that it doesn't be a burden on the machine running it. This system is being build keeping in mind the generally available hardware and software compatibility. Here are the minimum hardware and software requirement for book recommendation.

Hardware:

- Pentium-pro processor or later.
- RAM 512MB or more.

Software:

- Windows 7(32-bit) or above.
- Python 2.7 or later
- Chrome Driver
- Selenium Web Automation
- MySQLWindows

1.4 Problem Statement

Develop a personalized book recommendation system capable of understanding and responding to user preferences and queries in natural language. The system should be able to perform a range of tasks, such as:

- **Recommending Books**: Provide book suggestions based on user preferences (e.g., favorite genres, authors, or previous reads) using advanced algorithms.

- **Search and Filtering**: Allow users to search for specific books, authors, or genres and apply filters like publication year, ratings, or availability.

- **Managing Personal Libraries**: Enable users to create, organize, and track their personal reading lists, including marking books as “Read” or “To Read.”

- **Interactive Book Exploration**: Allow users to explore book summaries, reviews, and ratings to make informed choices about their next read.

- **Engaging Features**: Provide engaging content such as book-related trivia, fun facts about authors, or quotes from famous books to make the experience interactive.

The system would leverage natural language processing (NLP) to interpret user inputs and offer intuitive and dynamic recommendations.

1.5 Proposed Solution

1. Dataset:

This dataset consists of audio recordings of spoken words or phrases, along with cor- responding transcriptions (text). It's used to train the speech recognition component of the voice assistant, which converts spoken words into text. Common datasets for speech recognition include:

2. System Architecture:

Frontend:- Developed using HTML, CSS, JavaScript .
Backend:-Integrates machine learning models .
Data Preprocessing: Data preprocessing in a Book Recommendation involves preparing the input data for further analysis and processing

3. Machine Learning Models:

Deep Learning Models:

Convolutional Neural Networks (CNNs):- In a book recommendation system, CNNs can be used for tasks such as image-based feature extraction, where they analyze book cover images to derive genre or theme information. This visual data can enhance recommendation accuracy by combining it with user preferences and textual features.

Recurrent Neural Networks (RNNs):- RNNs, especially Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) variants, are ideal for sequential data modeling. They can be used to process user interaction sequences, such as reading history or search patterns, to predict and recommend books that align with the user's evolving interests.

Transformer-based Models:- Transformer architectures, such as the original Transformer or its advanced variants like BERT, can significantly enhance book recommendation systems by leveraging contextual understanding from textual data. These models can process user reviews, book summaries, and metadata to generate meaningful embeddings that represent both user preferences and book attributes.Classical Models:

Semantic Parsing

- **Rule-based Systems:-** For simpler applications, rule-based systems can effectively parse user queries and extract structured information. For instance, rules can identify genres, authors, or specific book titles in user queries like "Find books by Agatha Christie" or "Show me science fiction novels."

- **Sequence-to-Sequence Models:-** Encoder-decoder architectures with attention mechanisms can handle more complex semantic parsing tasks, such as interpreting nuanced user requests like "Suggest books similar to Harry Potter with a mix of fantasy and adventure." These models convert natural language queries into actionable search or recommendation commands.

Dialogue Management

- **Reinforcement Learning (RL)**: RL techniques can be employed to optimize dialogue flows in a conversational book recommendation system. For example, the system could learn how to ask clarifying questions ("Do you prefer fiction or non-fiction?") to ensure accurate recommendations, focusing on maximizing long-term user satisfaction.
- **Deep Reinforcement Learning**: Advanced algorithms like Deep Q-Networks (DQN) or Policy Gradient methods can improve dialogue management by learning directly from user interactions. These methods allow the system to refine its responses dynamically, ensuring that conversations remain engaging and effective in understanding user preferences.

Deployment:-Integrate a pre-trained recommendation model with a relational database (e.g., MySQL) for storing book data and user interactions, ensuring seamless performance and user access.

Personalization: Incorporate personalization features to tailor responses and recommendations based on individual user preferences, past interactions, and user pro- files. Integrate a pre-trained recommendation model with a relational database (e.g., MySQL) for storing book data and user interactions, ensuring seamless performance and user access.

Integration with IoT Devices: Extend the assistant's capabilities to control and interact with a wider range of Internet of Things (IoT) devices, such as smart home appliances, wearables, and connected vehicles.

Knowledge Graph Integration: Enhance the assistant's knowledge base by integrating with knowledge graphs like Wikidata or Freebase, enabling it to provide more comprehensive and accurate answers to user queries.

1.6 Scope

A book recommendation system can harness user-specific data, much like a voice assistant recognizes individual voices. By understanding user preferences, such as favorite genres, authors, or reading habits, the system can deliver personalized book suggestions tailored to each user. This approach ensures that recommendations resonate with the unique tastes and reading journeys of every use.

Chapter 2

Project Analysis

2.1 Introduction

The **book recommendation system project** aims to build an intelligent interface powered by **machine learning algorithms** and **data analysis techniques** to assist users in discovering books that match their preferences. The system is designed to offer personalized recommendations, improve user engagement, and simplify the process of finding books by leveraging user behavior, reviews, and other metadata.

2.1.1 Data Collection and Preprocessing

Data Collection

Sources of data include:
Book Metadata: Information about books, such as titles, authors, genres, summaries, publication dates, and ISBNs.

User Interaction Data: Logs of user actions, such as books viewed, searched, purchased, or rated.

Ratings and Reviews: User feedback on books to inform the system about preferences and sentiment.

External Datasets: Publicly available datasets from platforms like Kaggle or Goodreads.

Data Preprocessing

Cleaning:
- Remove duplicate entries and incorrect data.
- Handle missing values in critical fields like ratings or genres.

Standardization: Normalize data formats, such as author names and genre labels, to ensure consistency.

Feature Engineering: Extract useful attributes such as average ratings, number of pages, and keywords from book summaries.

Text Processing: Tokenize and process textual data (e.g., book descriptions and reviews) using techniques like TF-IDF or word embeddings for advanced semantic understanding.

Normalization of Ratings: Scale ratings to a standard range (e.g., 0–5) for uniform analysis.

2.1.2 System Development

We developed a user-friendly web application using python Flask. This application takes user input, runs it through our models, and displays the results.

2.1.3 Conclusion

The **book recommendation system** we have developed is capable of assisting users in discovering books based on their preferences and commands. It offers a range of functionalities, from personalized book recommendations to advanced filtering options, ensuring users can easily find their next read. The system employs a straightforward and efficient approach using Python, focusing on simplicity and effectiveness. Currently, the system is fully operational and provides reliable recommendations based on a combination of collaborative and content-based filtering techniques. Looking ahead, we aim to incorporate **deeper Artificial Intelligence** capabilities, such as enhanced Natural Language Processing (NLP) for understanding user sentiments in reviews and predictive modeling to anticipate future, reading preferences.

2.2 Study of Existing System

2.2.1 Strengths of Existing Systems

Existing book recommendation systems, such as those used by **Goodreads**, **Amazon**, and **Google Books**, exhibit several strengths that contribute to their widespread adoption and user satisfaction. Here are some key strengths:

- **Personalization:-**Book recommendation systems excel at delivering personalized suggestions by analyzing user preferences, reading history, and ratings. This ensures that users receive recommendations tailored to their specific tastes and interests**.**
- **Diverse Filtering Options:-**Users can search and filter books by various attributes such as genre, author, publication year, and average ratings, making it easier to discover books that meet their criteria.
- **User Reviews and Ratings:-** By aggregating reviews and ratings from a large user base, these systems provide valuable insights into book popularity and quality, enabling informed decision-making.

2.2.2Limitations of Existing Systems

While book recommendation systems provide valuable insights and suggestions, they also have some limitations that can affect user experience and overall effectiveness:

Accuracy and Relevance of Recommendations:-Recommendation systems may struggle to provide highly accurate or relevant suggestions, especially for users with limited interaction history or niche preferences. This can result in generic or repetitive recommendations that fail to engage the user.

Limited Contextual Understanding:-Many systems lack a deep contextual understanding of user preferences. For example, they may fail to differentiate between a user's temporary interest (e.g., exploring a specific genre) and long-term preferences, leading to recommendations that feel less personalized.

Privacy Concerns:- Storing and analyzing user data to generate recommendations poses potential privacy risks. Users may be concerned about how their data is collected, used, and shared.

2.2.3Our Approach

- The system will remain responsive to user interactions, allowing for seamless access to book recommendations and search results.
- If the system is unable to process or understand the user's query (e.g., ambiguous or incomplete input), it will prompt the user to clarify or rephrase their request.
- The system will feature an intuitive, visually appealing interface with clearly organized categories such as genres, top-rated books, and personalized suggestions, ensuring a smooth and engaging user experience.

2.3 Feasibility Study

Preliminary investigation examine project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resource

2.3.1 Technical Feasibility

The technical feasibility of the **book recommendation system** involves addressing key considerations to ensure the project can be implemented effectively. These include:

- Does the necessary technology exist to do what is suggested?
- Do the proposed equipment have the technical capacity to hold the data required to use the new system?
- Will the proposed system provide adequate response to inquiries,regardless of the number or location of users?
- Can the system be upgraded if developed?

2.3.2Economic Feasibility

While a **book recommendation system** can be developed technically and is expected to be widely used, it is essential to ensure that it represents a sound financial investment for the organization. Economic feasibility evaluates the cost of development against the anticipated benefits derived from the system. (Cost benefit analysis ,Use of Existing Resources, Nominal Expenditure) The system is economically viable, as the benefits, including improved user retention, higher user satisfaction, and the potential for monetization (e.g., through partnerships or advertisements), are expected to exceed the development and operational costs.

2.3.3Operational Feasibility

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization's operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. Some of the important issues raised are to test the operational feasibility of a project includes the following:

- Is there sufficient support for the management from the users?
- Will there be any resistance from the user that will undermine the possible application benefits?

2.4 Tools Used To Gather Information

Developing a **book recommendation system** involves gathering data from various sources to ensure its accuracy, relevance, and user-centric design. Below are some tools and methods commonly employed during the development process:

Data Collection Platforms

Kaggle Datasets: Provides rich datasets of books, ratings, and user interactions.
Public APIs: APIs such as Goodreads, Google Books, or Open Library are used to access extensive metadata, reviews, and ratings for books.

Data Analysis Tools

Pandas and NumPy: Essential Python libraries for cleaning, organizing, and analyzing large datasets.
Matplotlib and Seaborn: Visualization libraries used to understand user preferences and patterns in the data.

Machine Learning Frameworks

Scikit-learn: Provides algorithms for collaborative and content-based filtering for personalized recommendations.
TensorFlow and PyTorch: Used for advanced recommendation models, such as deep learning-based recommendation systems.

Chapter 3

Project Design

If the available book recommendation systems don’t cater to all your needs, it is possible to design your own. For a basic recommendation system, you don’t even need advanced programming skills—there are tools and frameworks that help build personalized systems that recommend books based on user preferences.

Here are some key elements to consider when building a book recommendation system:

- Remember the end user.
- Choose useful features.
- Give it personality.
- Integrate it with various platforms.

System Testing

Finally, we tested our system to ensure that it works as expected. This involved unit testing of individual components, integration testing of the whole system, and usability testing to ensure that the user interface is intuitive and user-friendly.

3.1 Software Requirement Specification

Introduction

The purpose of this document is to outline the software requirements for our Book Recommendation System project.

Functional Requirements

User Input and Interaction

- The system should accept multiple forms of input, including typed queries and selection through graphical interfaces.
- Support for natural language input to allow users to ask questions such as "Recommend me a mystery novel" or "Show me books by Agatha Christie."

Recommendation Engine

- The system should leverage user data such as ratings, past searches, or preferences to provide personalized book recommendations.
- It should support recommendation algorithms like rating-based filtering or collaborative filtering.

Search and Filters

- Users should be able to search for books using keywords, authors, or genres.
- The system should provide filtering options such as genre, language, publication year, and user ratings.

.

Error Handling

- The system should handle situations like ambiguous queries or lack of relevant results by providing suggestions or clarifications.
- In cases where a recommendation cannot be made, it should suggest alternative options or direct the user to popular or trending books.

Software and Hardware Requirements Software requirements:

Python: For implementing the machine learning algorithms and Flask application. Speech Recognition: Python libraries for implementing machine learning algorithms.

Pycharm IDE: An open-source code editor that we used for writing and debugging our code.

Chrome: A web browser that we used for testing and interacting with our web application.

Terminal: A command-line interface that we used for running our application and scripts.

Hardware requirements:

For application development, the following Software Requirements are:

Processor: Intel or high RAM: 1024 MB
Space on disk: minimum 100mb For running the application
Device: Any device that can access the internet Minimum space to execute 20 MB.

3.2 Software Functional Specification

User Interface

The user interface, developed using HTML,CSS,JavaScript and Python library, provides an intuitive and user-friendly way for users to interact with our application. Users can give the command, which is then processed by our machine models and perform the task.

Data Pre-processing

The system takes the user command and preprocesses it using Python and its libraries. This involves cleaning the data, handling missing values and outliers.

Recognition

The book recommendation system integrates machine learning models to adapt and improve its suggestions over time. By analyzing user interactions, ratings, and feedback. Collaborative filtering for personalized suggestions based on similar users' preferences. Hybrid approaches to combine multiple recommendation strategies for improved accuracy.

3.3 Data Flow Diagram

Data Flow Diagrams (DFDs) are graphical representations of the flow of data through an information system. They are used to visualize the data processing in a structured way. DFDs can be divided into different levels like a zero level DFD also known as a context diagram, a Level 1 DFD, and so on, based on the amount of detail they provide.

3.3.1 Zero Level DFD

A zero level Data Flow Diagram (DFD) provides a high-level overview of the sys- tem, showing the interactions between the system (the virtual voice assistant) and external entities (such as users and external systems).

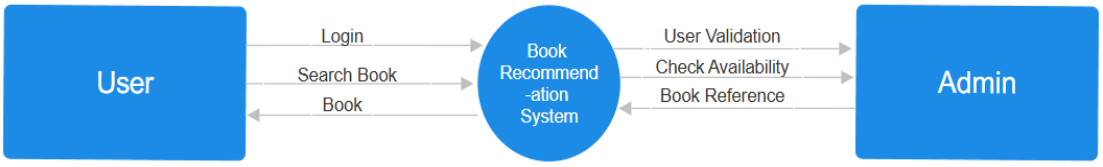

FIGURE 3.1: DFD level Zero

Components Involved:

User: Initiates the process by inputting details.
Process (Book Recommendation System): This is represented as a single bubble in the center, labeled "Book Recommendation System." It encompasses the entire system's func- tionality.
Inputs: These are internal entities or data sources that provide input to the System. Examples include text inputs.
Outputs: These are external entities or data destinations that receive output from the Book Recommendation System. Examples include Redirect to book API or references.

3.3.2 Level 1 DFD (DFD-L1)

The Level 1 DFD provides a more detailed view of the system's functioning. In our case, it shows the user interacting with a server through various processes. The user inputs details which go through a registration process before feeding values to the server. The server then matches these values with a database to predict a disease.

Components Involved:

USER: The rectangular box labeled "USER" represents the starting point. Users initiate voice commands, which flow from the USER box.

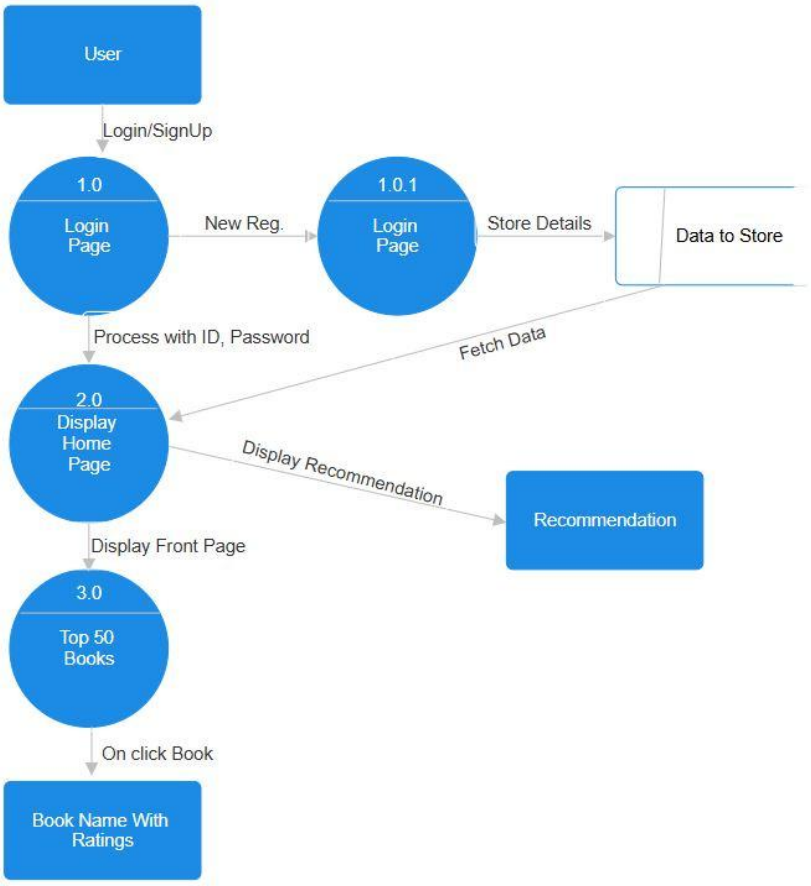

FIGURE 3.2: DFD level One

3.3.3 Level 2 DFD (DFD-L2)

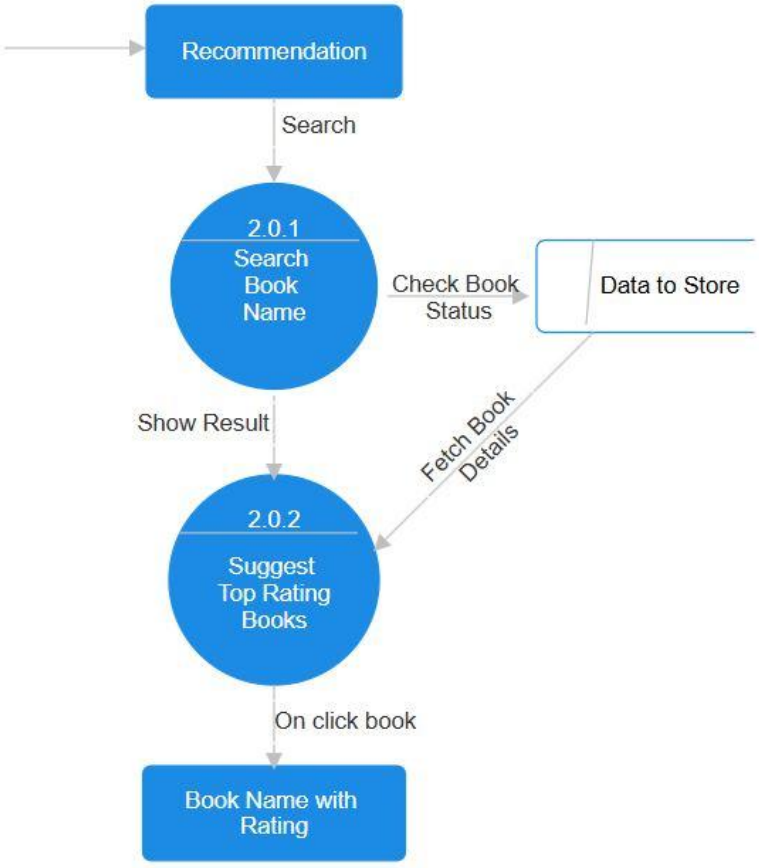

FIGURE 3.3: DFD level Second

3.4 UML Diagram

UML stands for Unified Modeling Language. UML is a standardized general- purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation.

The Unified Modeling Language is a standard language for specifying, Visualiza- tion, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.

- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.

3.4.1 Use Case Diagram

A Use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

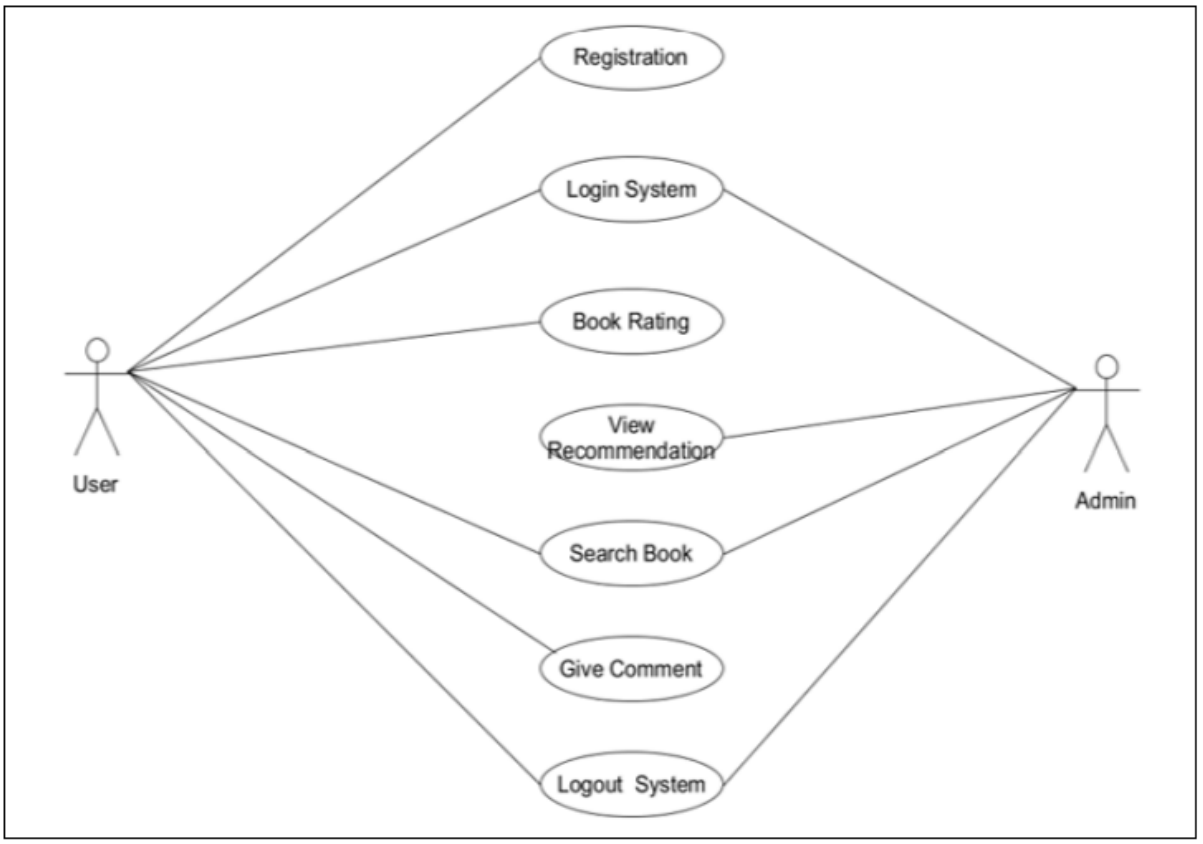

FIGURE 3.3: Use Case Diagram

3.4.2 Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

Chapter 4

System Implementation

4.1 Model Explanation

Recommendation Engine

The core component of the system, the recommendation engine, analyzes user preferences and behaviors to provide personalized book suggestions.

Natural Language Processing (NLP) Model-

The NLP model processes user text inputs and extracts relevant information, such as search terms, preferences, or specific book requests. Techniques used include:

- **Entity Recognition:** Identifying book titles, author names, or genres in user queries.
- **Intent Detection:** Understanding whether the user wants to "search for a book," "get recommendations," or "filter by genre."
- **Contextual Understanding:** Leveraging pre-trained transformer models (e.g., BERT or GPT) to understand nuanced queries like "Show me books similar to *1984* by George Orwell."

Implementation Steps

1. **Setup Environment**
    - Install necessary libraries and frameworks, such as Scikit-learn, TensorFlow, or PyTorch, for recommendation algorithms.
    - Use libraries like NLTK or SpaCy for natural language processing.

2. Build and Train Models

Develop collaborative and content-based filtering models using user interaction data and book metadata. Fine-tune transformer models for intent recognition and query understanding.

3. Integrate NLP Components

- Define user intents, such as "search for books," "filter by genre," or "rate a book."
- Extract entities like book titles, authors, and genres from user inputs.

4. Recommendation Pipeline

- Combine user preferences, search history, and metadata to generate personalized book suggestions.
- Implement a feedback loop where user interactions (e.g., ratings, likes) continuously improve recommendations.

5. Develop the Front-End Interface

- Build a user-friendly interface with PyQt5 or web frameworks to enable seamless interaction.
- Include features like search bars, filters, and personalized dashboards.

Integration with External Services for Book Recommendation System-

Description:

Integration with external services, such as Goodreads APIs, public library databases, or online bookstores, to provide comprehensive and enriched book details, availability status, and user reviews. This allows users to explore curated content and make informed decisions.

Implementation: Details on how news sources are selected, fetched, and presented to users in a structured format.

Example Interaction:- Searching for trending books, e.g., "What are the top fantasy books this month?"

conclusion-

Assess whether the system meets user expectations and achieves its primary goal of providing a personalized, efficient, and engaging book discovery platform. Provide accurate and relevant recommendations based on user preferencese. Seamlessly integrate external data to enrich user experience..

1. "Designing Bots: Creating Conversational Experiences" by Amir Shevat

This book provides a comprehensive guide to designing conversational interfaces, including virtual voice assistants. It covers best practices for interaction design, dialogue management, and integrating voice-based features into applications. Shevat also addresses important topics like user experience and the underlying AI systems that drive these assistants.

2. "Speech and Language Processing" by Daniel Jurafsky and James H. Martin
If you're looking to understand the technical side of **speech recognition** and **natural language processing (NLP)**, this book is a classic. It's a detailed resource on how these technologies work, which are at the core of virtual voice assistants. It provides both theoretical explanations and practical examples, making it a must-read for anyone involved in the development of voice-driven applications.

3. "Python for Data Analysis" by Wes McKinney
For those who want to dive into the technical details of implementation, particularly in Python (a popular language for AI projects), this book is excellent. It covers various libraries and tools, including those used in machine learning and data manipulation, which are essential for building intelligent virtual assistants. It's a great starting point for learning how to process and analyze data from voice interactions.

4. "Building Intelligent Systems: A Guide to Machine Learning Engineering" by Geoff Hulten
This book focuses on the practical side of implementing AI systems, from design to deployment. It’s particularly helpful for understanding how machine learning can be applied to build and enhance virtual assistants. It covers key concepts like data preprocessing, model training, and the integration of external APIs.

5. "Artificial Intelligence: A Modern Approach" by Stuart Russell and Peter Norvig
As a comprehensive textbook on AI, this book dives deep into the theories and algorithms that power modern AI systems, including virtual voice assistants. It’s a thorough exploration of topics such as **machine learning**, **natural language processing**, and **planning**, all of which are essential components of intelligent assistants.

Model Development:

Needs Assessment and User Requirements: the process of identifying user needs and requirements to inform the development of the assistant’s functionality and features. It includes methods such as user surveys, interviews, and usability testing to gather insights into user preferences, pain points, and expectations.

Model Selection and Architecture Design: The model selection and architec- ture design section detail the process of selecting suitable algorithms and designing the overall architecture of the virtual voice assistant system.

Training Data Collection and Annotation: In this section, the focus is on the collection and annotation of training data necessary for training and fine-tuning the various models used in the virtual voice assistant.

4.2 Work Plan

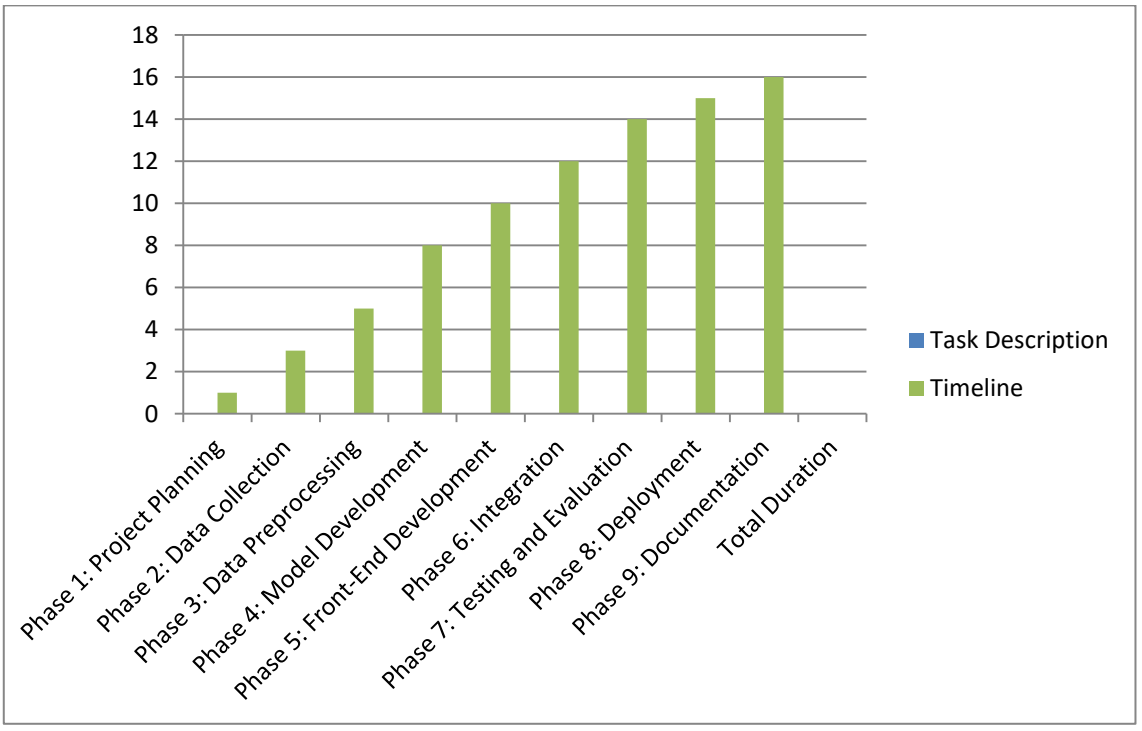

FIGURE 4.1: Graph of Work Plan

Chapter 5

Testing

Testing is a crucial part of any software development process, including machine learning projects. It ensures that the developed system is working as expected and helps in identifying any potential issues or bugs.

Functional Testing: This involves testing the basic functionality of the virtual assistant, such as: Verifying that it accurately recognizes and responds to user inputs. Ensuring that it correctly executes requested tasks or actions. Testing various features such as setting reminders, playing music, providing information, etc.

Usability Testing: This type of testing focuses on the user experience and interface design: Assessing how intuitive and easy-to-use the virtual assistant is for users. Identifying any usability issues or pain points in the interaction flow. Collecting feedback from users on their overall experience with the assistant.

Performance Testing: This involves evaluating the performance and responsive- ness of the virtual assistant: Testing its response time to user inputs. Assessing its ability to handle multiple requests concurrently. Monitoring resource usage to ensure efficient operation.

Compatibility Testing: Verifying that the virtual assistant works correctly across different devices and platforms: Testing compatibility with various operating systems (e.g., iOS, Android, Windows). Ensuring compatibility with different web browsers or voice-enabled devices (e.g., smart speakers, smartphones, smartwatches).

Load Testing: Assessing the virtual assistant's performance under heavy loads: Simulating a large number of concurrent users to test scalability. Monitoring system behavior and response times to identify performance bottlenecks.

Accessibility Testing: Assessing the accessibility of the virtual assistant for users with disabilities: Testing compatibility with assistive technologies such as screen readers or voice input devices. Ensuring that the assistant's interface and interactions are accessible

to users with visual or motor impairments. Testing the Models Each of the models (Decision Tree, Linear Regression, Logistic Regression, SVM, Random Forest, XGBoost, Standard Scaler, Gradient Boost) was trained using the training set and then evaluated using the testing set. The performance of each model was then assessed using the evaluation metrics described above.

Discussion

Overall, the testing results provide valuable insights into the performance, reliability, and user experience of the virtual voice assistant, highlighting areas of strength and opportunities for further enhancement. .

Chapter 6

System Input And Output

1.1 App Execution Code

```
from flask import Flask, render_template, request, redirect, url_for, flash
import pickle
import numpy as np
from flask_mysqldb import MySQL
import MySQLdb.cursors
from flask import session
popular_df=pickle.load(open('popular.pkl','rb'))
pt=pickle.load(open('pt.pkl','rb'))
books=pickle.load(open('books.pkl','rb'))
similarity_score=pickle.load(open('similarity_score.pkl','rb'))
app = Flask(__name__)
app.secret_key = 'your_secret_key'
# MySQL Configurations
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = ''  # XAMPP default MySQL password
# app.config['MYSQL_DB'] = 'book recommended'
app.config['MYSQL_DB'] = 'book recommended'  # Use underscores for valid naming
mysql = MySQL(app)
google_drive_base_url = "https://drive.google.com/file/d/"

@app.route('/')
def home():
```

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

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form.get('email')
        password = request.form.get('password')

        if not email or not password:
            flash('Please provide both email and password.', 'warning')
            return render_template('login.html')

        try:
            cursor = mysql.connection.cursor()
            cursor.execute('SELECT * FROM user WHERE email = %s', (email,))
            user = cursor.fetchone()
            cursor.close()
        except Exception as e:
            flash('Database error occurred. Please try again later.', 'danger')
            return render_template('login.html')

        if user and user[5] == password:  # Assuming password is stored in the 5th column
            session['loggedin'] = True
            session['id'] = user[0]       # Assuming user ID is in the 0th column
            session['email'] = user[4]    # Assuming email is in the 4th colum
            flash('Login successful!', 'success')
            return redirect(url_for('second'))
        elif user:
            flash('Incorrect password. Please try again.', 'danger')
        else:
            flash('Email not found. Please sign up.', 'warning')
            return redirect(url_for('signup'))

    return render_template('login.html')

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

```
def second():
    return render_template('second.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['Book-Rating'].values), rating
=list(popular_df['avg_ratings'].values))


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


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


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


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


@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        # Print the form data received from the user
        print(request.form)  # This will print all form fields and their values


        # Now proceed with form validation and insertion into the database
        if 'username' in request.form and 'email' in request.form and 'password' in request.form:
            username = request.form['username']
            email = request.form['email']
            # password = generate_password_hash(request.form['password'],
method='pbkdf2:sha256')
            password = request.form['password']


            # Insert into MySQL
            cursor = mysql.connection.cursor()
```

```
cursor.execute('INSERT INTO user (username, email, password) VALUES (%s, %s, %s)',
(username, email, password))
            mysql.connection.commit()
            cursor.close()

            flash('You have successfully registered!', 'success')
            return redirect(url_for('login'))  # Redirect to login page
        else:
            flash('Missing form data. Please try again.', 'danger')

    return render_template('signup.html')  # Show signup form for GET requests

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

def get_file_id(book_id):
    # Implement your logic to retrieve the file ID for a given book ID
    return "1ufRI06Gj1n4JI84MmALNv5UM6LDhP8Zr"

@app.route("/showbook")
def index():
    # ... your logic to retrieve book data ...
    books = [
        {"book_id": 1, "image": "...", "book_name": "..."},
        {"book_id": 2, "image": "...", "book_name": "..."},
    ]
    return render_template("index.html", books=books)

@app.route('/recommend_books',methods=['post'])
def recommend_book():
    user_input=request.form.get('user_input')
    index = np.where(pt.index == user_input)[0][0]

    # Find the most similar items, sorted by similarity score, excluding the first (which is the
book itself)
    similar_items = sorted(list(enumerate(similarity_score[index])), key=lambda x: x[1],
reverse=True)[1:6]
```

```
data = []
for i in similar_items:
        item = []
        temp_df = books[books['Book-Title'] == pt.index[i[0]]].drop_duplicates('Book-Title')
        item.append(temp_df['Book-Title'].values[0])
        item.append(temp_df['Book-Author'].values[0])
        item.append(temp_df['Image-URL-M'].values[0])
        data.append(item)

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

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

Output Screen

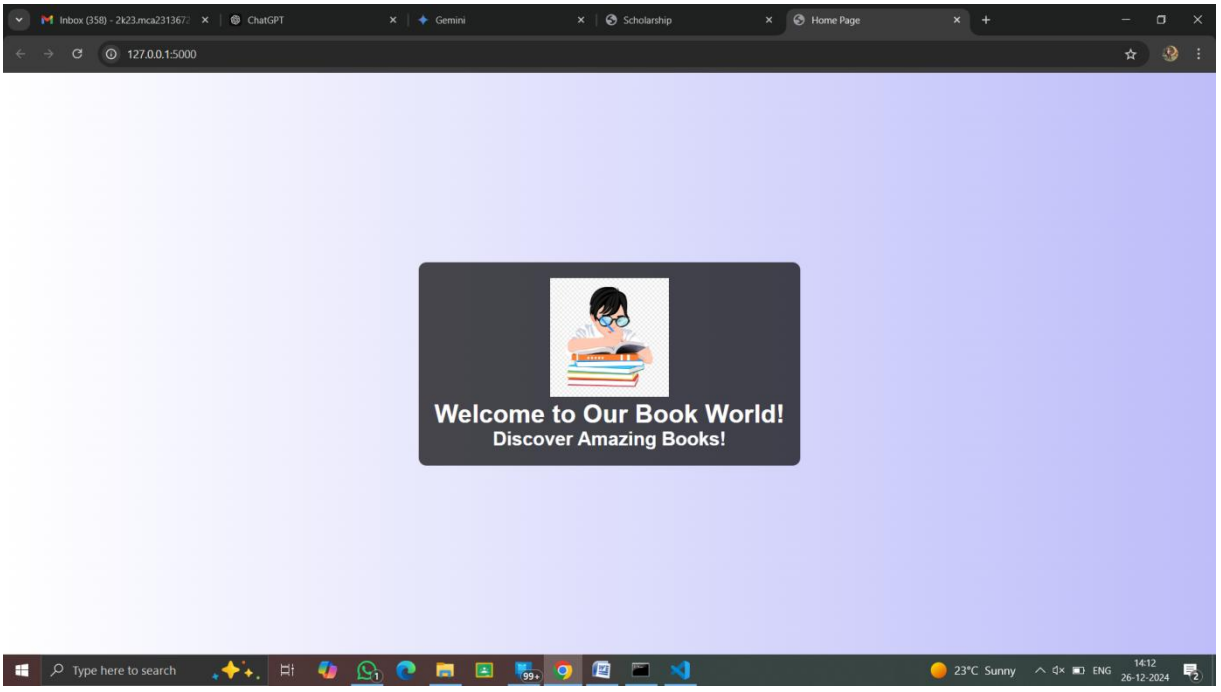

FIGURE 6.1: Output Screen 1

1.2 Login Page Code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 0;
      padding: 0;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      background-color: #f4f4f4;
    }
    .login-container {
      background: #fff;
      padding: 20px;
      border-radius: 8px;
      box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
      width: 300px;
    }
    .login-container h2 {
      margin-bottom: 20px;
      text-align: center;
    }
    .form-group {
      margin-bottom: 15px;
    }
    .form-group label {
      display: block;
      margin-bottom: 5px;
      font-weight: bold;
    }
    .form-group input {
      width: 100%;
      padding: 8px;
      border: 1px solid #ccc;
      border-radius: 4px;
    }
    .form-group button {
      width: 100%;
      padding: 10px;
      background-color: #007BFF;
      color: white;
```

```
border: none;
            border-radius: 4px;
            cursor: pointer;
            font-size: 16px;
        }
        .form-group button:hover {
            background-color: #0056b3;
        }
        .error {
            color: red;
            font-size: 14px;
            margin-bottom: 10px;
        }
    </style>
</head>
<body>
    <div class="login-container">
        <h2>Login</h2>
        <!-- Show any error messages -->
        {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
                <div class="error">
                    {% for category, message in messages %}
                        <p>{{ message }}</p>
                    {% endfor %}
                </div>
            {% endif %}
        {% endwith %}

        <form method="POST" action="{{ url_for('login') }}">
            <div class="form-group">
                <label for="email">Email:</label>
                <input type="email" name="email" id="email" placeholder="Enter your email"
required>
            </div>
            <div class="form-group">
                <label for="password">Password:</label>
                <input type="password" name="password" id="password" placeholder="Enter your
password" required>
            </div>
            <div class="form-group">
                <button type="submit">Login</button>
            </div>
        </form>
        <p>Don't have an account? <a href="{{ url_for('signup') }}">Sign up here</a>.</p>
    </div>
</body>
</html>
```

Output Screen

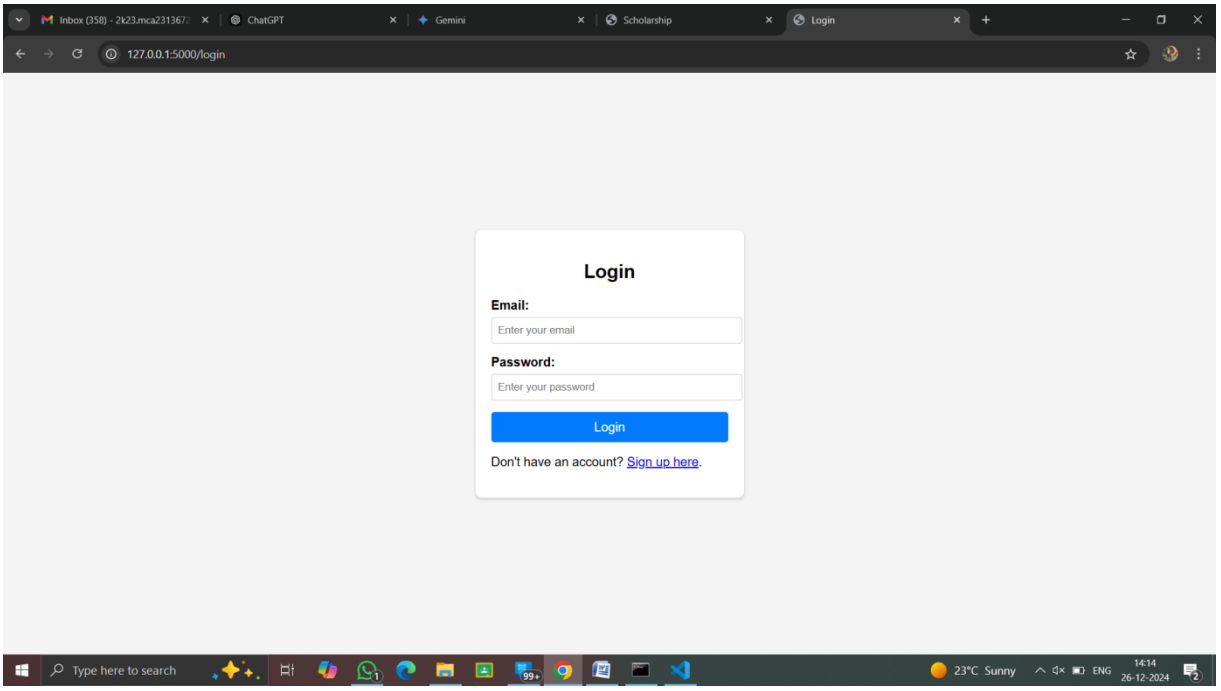

FIGURE 6.2: Output Screen 2

1.3 Home Page Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Book Recommendation</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <!-- <link rel="stylesheet" href="styles.css"> -->
    <style type="text/css">
        body {
    margin: 0;
    padding: 0;
    height: 100vh;
    display: flex;
    flex-direction: column;
    background: linear-gradient(to right,white,rgb(123, 123, 244));
    /* background: linear-gradient(to left ,rgb(35, 23, 15), rgba(221, 115, 40, 0.995)); */
    background-size: 300% 300%; /* Allow for animation */
    animation: gradientAnimation 15s ease infinite; /* Animate the gradient */
    color: #333;
    font-family: 'Arial', sans-serif;
```

```
transition: background-color 0.5s ease;
}
@keyframes gradientAnimation {
   0% {
      background-position: 0% 50%;
   }
   50% {
      background-position: 100% 50%;
   }
   100% {
      background-position: 0% 50%;
   }
}

header {
   position: absolute;
   top: 20px;
   left: 20px;
   text-align: center;
}
.header-wrapper {
   display: flex;
   align-items: center;
}
   /* this is use to so effect like typing*/  */
.header-text {
   font-size: 1.0em;
   margin: 0 8px; /* Space around the text */
   opacity: 0; /* Start hidden */
   transition: opacity 0.5s ease, transform 0.5s ease; /* Smooth transition for opacity and
movement */
}
.cursor {
   display: inline-block;
   width: 2px;
   height: 1.5em;
   background: #5cb85c; /* Cursor color */
   animation: blink 1s step-end infinite; /* Blinking effect */
}
@keyframes blink {
   50% { opacity: 0; }
}
.visible {
   opacity: 1; /* Make text visible */
}
.hidden {
   opacity: 0; /* Hide text */
   transform: translateY(-10px); /* Move text up slightly when hiding */
```

```
}
button {
   margin-bottom: 10px; /* Space between button and header */
   padding: 10px;
   border: none;
   border-radius: 5px;
   background-color: #5cb85c;
   color: white;
   cursor: pointer;
   transition: background-color 0.3s ease;
}

button:hover {
   background-color: #4cae4c;
}
.container {
   background: rgba(255, 255, 255, 0.9);
   border-radius: 15px;
   box-shadow: 0 4px 30px rgba(0, 0, 0, 0.2);
   padding: 30px;
   text-align: center;
   margin-top: 100px;
   margin-left: 35%;
   width: 90%;
   max-width: 400px;
   animation: fadeIn 1s ease;
}
#search
{
   height: 20px;
   width: 100%;
   border: 2px solid black;
   border-radius: 50%;
}
@keyframes fadeIn {
   from {
      opacity: 0;
      transform: translateY(-20px);
   }
   to {
      opacity: 1;
      transform: translateY(0);
   }
}
form {
   display: flex;
   flex-direction: column;
   gap: 15px;
}
```

```
select, input[type="text"] {
    padding: 12px;
    border: 2px solid #5cb85c;
    border-radius: 5px;
    transition: border-color 0.3s ease, box-shadow 0.3s ease;
    font-size: 1em;
}
select:focus, input[type="text"]:focus {
    border-color: #4cae4c;
    box-shadow: 0 0 8px rgba(76, 175, 80, 0.5);
    outline: none;
}
button {
    padding: 12px;
    background-color: #5cb85c;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease, box-shadow 0.3s ease;
    font-size: 1em;
}
button:hover {
    background-color: #4cae4c;
    transform: scale(1.05);
    box-shadow: 0 4px 15px rgba(76, 175, 80, 0.3);
}
/* for recommend books that shows on browser */
.book-gallery {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(180px, 1fr));
    gap: 20px;
    padding: 20px;
}
.book-box {
    background: white;
    border-radius: 10px;
    box-shadow: 0 6px 20px rgba(0, 0, 0, 0.15);
    text-align: center;
    padding: 15px;
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}
.book-box:hover {
    transform: translateY(-5px);
    box-shadow: 0 8px 25px rgba(0, 0, 0, 0.2);
}
.book-box img {
    max-width: 100%;
    border-radius: 5px;
```

```
    transition: transform 0.3s ease;
}
.book-box img:hover {
    transform: scale(1.05);
}
.heading{
    margin-left: 50px;
}
/* for the top nav bar  */
nav {
    background-color: #f8f8f8;
    padding: 10px 0;
    text-align: center;
}
.nav-links {
    display: inline-block;
}
.nav-links a {
    margin: 0 15px;
    text-decoration: none;
    color: #333;
    font-weight: bold;
}
.nav-links a:hover {
    text-decoration: underline;
}
html {
    scroll-behavior: smooth;
}
#genre{
    cursor:pointer;
}
.dark{
    background-color: black;
    color: white;
}
.light{
    background-color: white;
    color: black;
}
body {
            background-color: #ffffff;
        }
        .text-white {
            color: #0a0a0a00;
        }
        .box {
            padding: 20px;
        }
```

```
        .card {
            background-color: #f4f2f2;
            border: none;
        }
        .card img {
            max-height: 300px;
            object-fit: cover;
        }
    </style>
</head>
<body>
    <nav>
        <div class="nav-wrapper">
            <div class="nav-links">
                <a href="/second">Home</a>
                <a href="/recommend">Recommendations</a> <!-- Updated to anchor link -->
                <a href="/about">About Us</a>
                <a href="/dev">Developer</a>
                <button id="mode">Change Mode</button>
            </div>
        </div>
    </nav>
     <header>
        <div class="header-wrapper">
            <span class="bracket"></span>
            <h1 id="headerText" class="header-text">Book Recommendation</h1>
            <span class="bracket"></span>
        </div>
    </header>
    <main>
         <div class="box">
            <div class="row">
                <div class="col-md-12">
                </div>
            </div>
                <h1 class="text-white text-center" style="font-size:50px">Top 50 Books</h1>
                {% for i in range(book_name|length) %}
                    {% if i % 4 == 0 %}
                        <div class="row" style="margin-top: 50px;">
                    {% endif %}
                    <div class="col-md-3">
                        <div class="card">
                            <div class="card-body text-center">
                                <img class="card-img-top" src="{{image[i]}}" alt="{{ book_name[i]
}}">
                                <h3 class="text-black">{{ book_name[i] }}</h3>
                                <p class="text-black">{{ author[i] }}</p>
                                <p class="text-black">Votes - {{ votes[i] }}</p>
```

```
                        <p class="text-black">Rating - {{ rating[i] }}</p>
                    </div>
                </div>
            </div>
                    {% if i % 4 == 3 or i == book_name|length - 1 %}
                </div>
            {% endif %}
        {% endfor %}
            </div>
        </div>
    </div>
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.4.4/dist/umd/popper.min.js"></script>
    <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
      <section class="book-gallery" id="bookGallery"> <!-- Added ID for targeting -->
        <!-- Book boxes will be dynamically generated here -->
    </section>
  </main>
  <script>
  function searchBooks(){
    window.location.href = "/recommend";
  }
  function about(){
    window.location.href = "/about";
  }
  function dev(){
    window.location.href = "/dev";
  }

  document.getElementById("mode").addEventListener("click", function() {
       document.body.classList.toggle("dark");
    });
  </script>
</body>
</html>
```

Output Screen

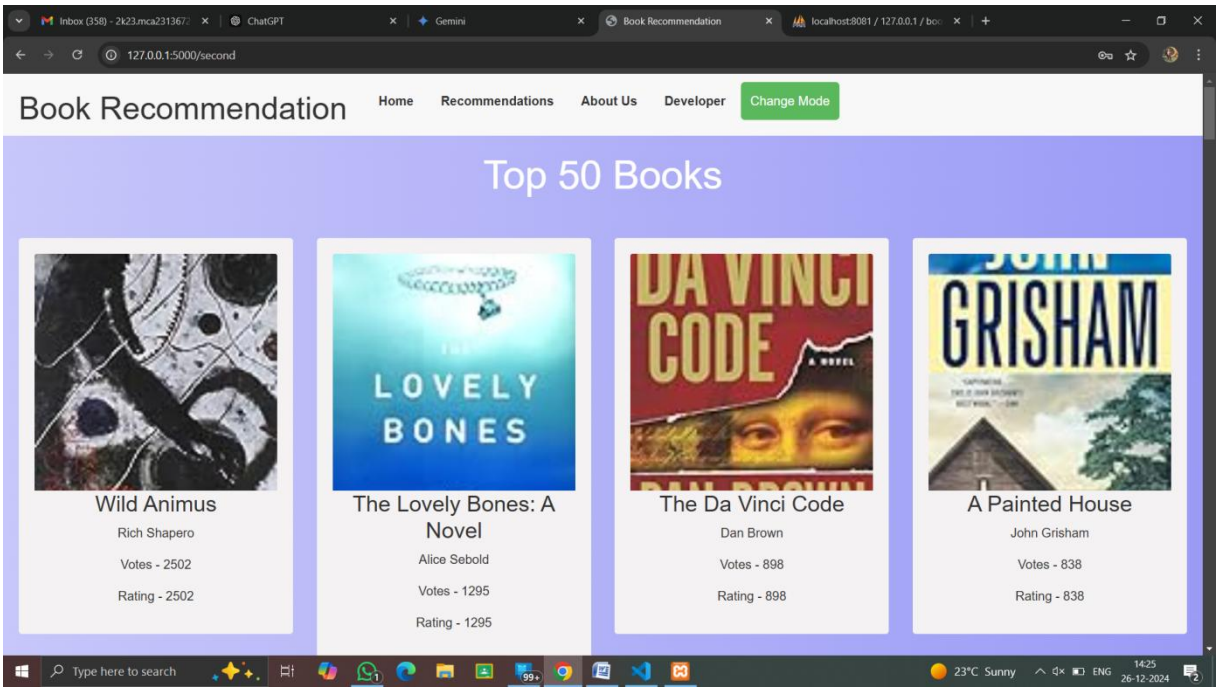

FIGURE 6.3: Output Screen 3

1.4 Recommendation Page Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Book Recommendation</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <!-- <link rel="stylesheet" href="styles.css"> -->
    <style type="text/css">
        body {
    margin: 0;
    padding: 0;
    height: 100vh;
    display: flex;
    flex-direction: column;
    background: linear-gradient(to right,white,rgb(123, 123, 244));
    background-size: 300% 300%; /* Allow for animation */
    animation: gradientAnimation 15s ease infinite; /* Animate the gradient */
    color: #333;
    font-family: 'Arial', sans-serif;
    transition: background-color 0.5s ease;
}
```

```
@keyframes gradientAnimation {
   0% {
      background-position: 0% 50%;
   }
   50% {
      background-position: 100% 50%;
   }
   100% {
      background-position: 0% 50%;
   }
}
header {
   position: absolute;
   top: 20px;
   left: 20px;
   text-align: center;
}
.header-wrapper {
   display: flex;
   align-items: center;
}
.header-text {
   font-size: 1.5em;
   margin: 0 10px; /* Space around the text */
   opacity: 0; /* Start hidden */
   transition: opacity 0.5s ease, transform 0.5s ease; /* Smooth transition for opacity and
movement */
}
.cursor {
   display: inline-block;
   width: 2px;
   height: 1.5em;
   background: #5cb85c; /* Cursor color */
   animation: blink 1s step-end infinite; /* Blinking effect */
}
@keyframes blink {
   50% { opacity: 0; }
}
.visible {
   opacity: 1; /* Make text visible */
}
.hidden {
   opacity: 0; /* Hide text */
   transform: translateY(-10px); /* Move text up slightly when hiding */
}
button {
   margin-bottom: 10px; /* Space between button and header */
   padding: 10px;
   border: none;
   border-radius: 5px;
   background-color: #5cb85c;
```

```
    color: white;
    cursor: pointer;
    transition: background-color 0.3s ease;
}
button:hover {
    background-color: #4cae4c;
}
.container {
    background: rgba(255, 255, 255, 0.9);
    border-radius: 15px;
    box-shadow: 0 4px 30px rgba(0, 0, 0, 0.2);
    padding: 30px;
    text-align: center;
    margin-top: 100px;
    margin-left: 35%;
    width: 90%;
    max-width: 400px;
    animation: fadeIn 1s ease;
}
#search
{
    height: 20px;
    width: 100%;
    border: 2px solid black;
    border-radius: 50%;
}
@keyframes fadeIn {
    from {
        opacity: 0;
        transform: translateY(-20px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}
form {
    display: flex;
    flex-direction: column;
    gap: 15px;
}
select, input[type="text"] {
    padding: 12px;
    border: 2px solid #5cb85c;
    border-radius: 5px;
    transition: border-color 0.3s ease, box-shadow 0.3s ease;
    font-size: 1em;
}
select:focus, input[type="text"]:focus {
    border-color: #4cae4c;
    box-shadow: 0 0 8px rgba(76, 175, 80, 0.5);
```

```
    outline: none;
}
button {
    padding: 12px;
    background-color: #5cb85c;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease, box-shadow 0.3s ease;
    font-size: 1em;
}
button:hover {
    background-color: #4cae4c;
    transform: scale(1.05);
    box-shadow: 0 4px 15px rgba(76, 175, 80, 0.3);
}
.book-gallery {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(180px, 1fr));
    gap: 20px;
    padding: 20px;
}
.book-box {
    background: white;
    border-radius: 10px;
    box-shadow: 0 6px 20px rgba(0, 0, 0, 0.15);
    text-align: center;
    padding: 15px;
    transition: transform 0.3s ease, box-shadow 0.3s ease;
}
.book-box:hover {
    transform: translateY(-5px);
    box-shadow: 0 8px 25px rgba(0, 0, 0, 0.2);
}
.book-box img {
    max-width: 100%;
    border-radius: 5px;
    transition: transform 0.3s ease;
}

.book-box img:hover {
    transform: scale(1.05);
}
.heading{
    margin-left: 50px;
}
nav {
    background-color: #f8f8f8;
    padding: 10px 0;
```

```
    text-align: center;
}
.nav-links {
    display: inline-block;
}
.nav-links a {
    margin: 0 15px;
    text-decoration: none;
    color: #333;
    font-weight: bold;
}
.nav-links a:hover {
    text-decoration: underline;
}
html {
    scroll-behavior: smooth;
}
#genre{
    cursor:pointer;
}
.dark{
    background-color: black;
    color: white;
}
.light{
    background-color: white;
    color: black;
}
body {
            background-color: #ffffff;
        }
        .text-white {
            color: #0a0a0a00;
        }
        .box {
            padding: 20px;
        }
        .card {
            background-color: #f4f2f2;
            border: none;
        }
        .card img {
            max-height: 300px;
            object-fit: cover;
        }
    </style>
</head>
<body>
    <nav>
        <div class="nav-wrapper">
```

```
        <div class="nav-links">
          <a href="/second">Home</a>
          <a href="/recommend">Recommendations</a> <!-- Updated to anchor link -->
          <a href="/about">About Us</a>
          <a href="/dev">Developer</a>
          <button id="mode">Change Mode</button>
        </div>
      </div>
    </nav>
    <header>
      <div class="header-wrapper">
        <span class="bracket"></span>
        <h1 id="headerText" class="header-text">Book Recommendation</h1>
        <span class="bracket"></span>
      </div>
    </header>
    <main>
      <div class="box">
        <div class="row">
          <div class="col-md-12">
          </div>
        </div>
          <h1 class="text-white text-center" style="font-size:50px">Recommend
Books</h1>
            <form action="/recommend_books" method="POST">
              <input name="user_input" type="text" class="form-control "><br>
              <input type="submit" class="btn btn-lg btn-warning"> <br>
            </form>
            {% if data %}
      <div class="row">
        {% for i in data %}
          <div class="col-md-3" style="margin-top: 20px;">
            <div class="card">
              <div class="card-body text-center">
                <img class="card-img-top" src="{{ i[2] }}" alt="Book Image">
                <h3 class="text-black">{{ i[0] }}</h3>
                <p class="text-black">{{ i[1] }}</p>
              </div>
            </div>
          </div>
        {% endfor %}
      </div>
    {% endif %}
          </div>
        </div>
      </div>

      <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
```

```
    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.4.4/dist/umd/popper.min.js"></script>
        <script
   src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>

        <section class="book-gallery" id="bookGallery">
        </section>
      </main>
      <script>
      function searchBooks(){
        window.location.href = "/second";
      }
      function about(){
        window.location.href = "/about";
      }
      function dev(){
        window.location.href = "/dev";
      }
      document.getElementById("mode").addEventListener("click", function() {
          document.body.classList.toggle("dark");
        });
      </script>
   </body>
</html>
```

Output Screen

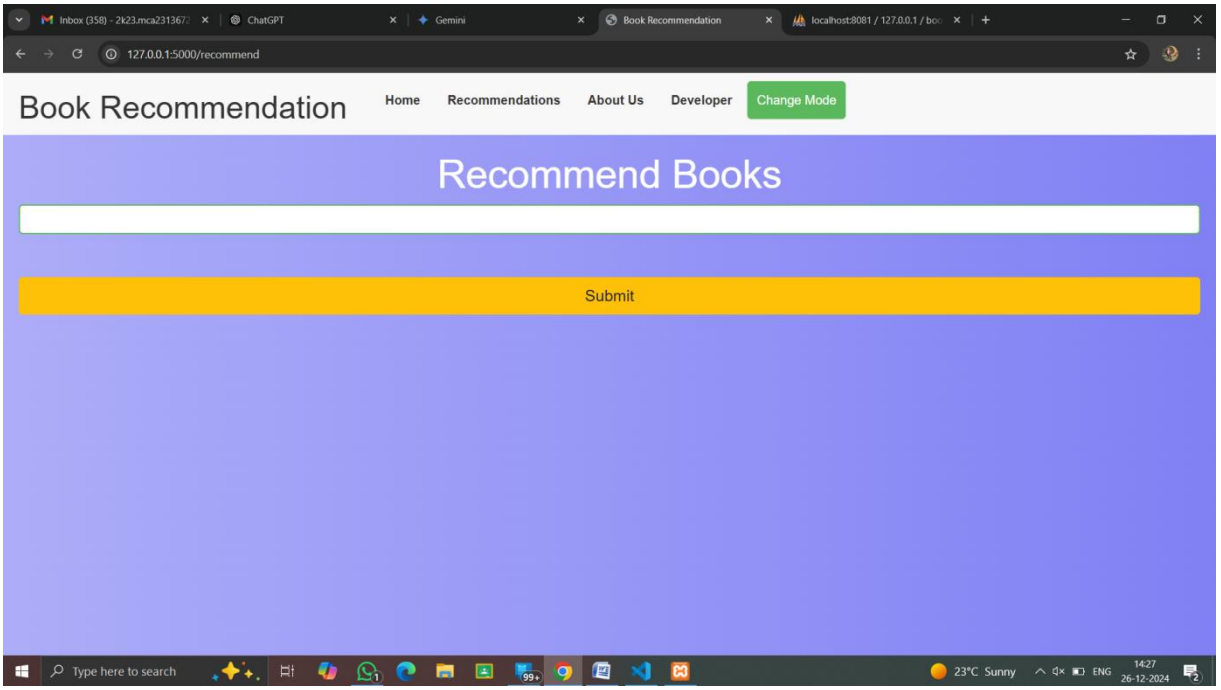

FIGURE 6.3: Output Screen 4

1.5 Search Page Code

```
@app.route('/recommend_books',methods=['post'])
def recommend_book():
    user_input=request.form.get('user_input')
    index = np.where(pt.index == user_input)[0][0]
    similar_items = sorted(list(enumerate(similarity_score[index])), key=lambda x: x[1],
reverse=True)[1:6]
    data = []
    for i in similar_items:
        item = []
        temp_df = books[books['Book-Title'] == pt.index[i[0]]].drop_duplicates('Book-Title')
        item.append(temp_df['Book-Title'].values[0])
        item.append(temp_df['Book-Author'].values[0])
        item.append(temp_df['Image-URL-M'].values[0])
        data.append(item)
    print(data)

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

Output Screen

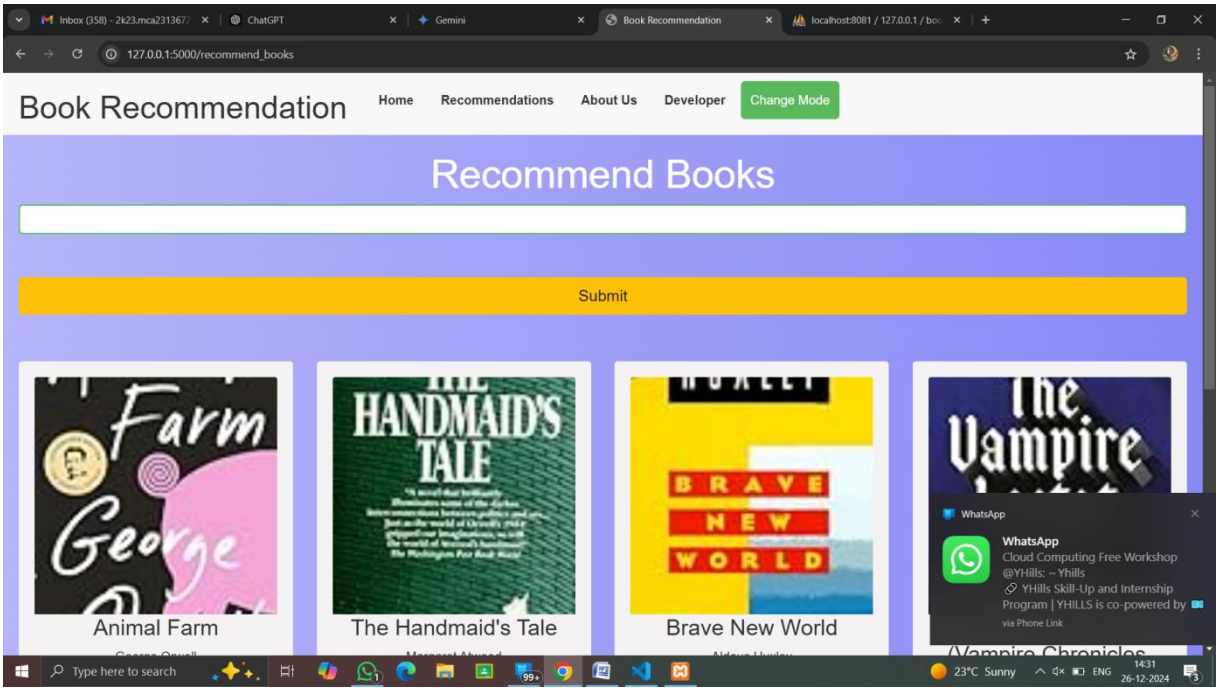

FIGURE 6.3: Output Screen 5

Chapter 7

Limitation and Scope of Project

Limitations

The limitations and scope of a virtual voice assistant project depend on various factors such as budget, time constraints, available resources, and the specific goals of the project. Here are some common limitations and considerations:

Scope of Features: Define the specific features and functionalities that the virtual assistant will offer. Due to limitations in resources and time, it may not be feasible to implement every possible feature initially. Prioritize essential features that align with the project goals and target audience needs.

Technology Constraints: The choice of technology stack may impose limitations on the project.

Data Availability: The effectiveness of the virtual assistant depends on the avail- ability and quality of training data. Limited access to relevant data sources may constrain the assistant's ability to understand user inputs accurately or provide meaningful responses.

Performance and Scalability: Consider the anticipated volume of user interactions and ensure that the virtual assistant can scale to handle increased load as the user base grows.

User Interface Design: The design of the user interface plays a significant role in the usability and user experience of the virtual assistant.

Integration with External Systems: If the virtual assistant needs to interact with external systems or APIs (e.g., third-party services, databases), consider the complexity and potential limitations of these integrations.

Privacy and Security: Protecting user privacy and data security is paramount.

Scope

As AI becomes more advanced and voice technology becomes more accepted, not only will voice controlled digital assistants become more natural, they will also be- come more integrated into more daily devices. Also, conversations will become much more natural, emulating human conversations, which will begin to introduce more complex task flows. More and more people are using voice assistants too, as it was estimated in early 2019 that 111.8 million people in the US will use a voice assistant at least monthly, from last year.

Chapter 8

Conclusions

Voice Controlled Personal Assistant System will use the Natural language process- ing and can be integrated with artificial intelligence techniques to achieve a smart assistant that can control IoT applications and even solve user queries using web searches.. It can be designed to minimize the human efforts to interact with many other subsystems, which would otherwise have to be performed manually. By achiev- ing this, the system will make human life comfortable. More specifically, this system is designed to interact with other subsystems intelligently and control these devices, this includes IoT devices or getting news from Internet, providing other informa- tion, getting personalized data saved previously on the system, etc. The android application should let the user add data such as calendar entries, set alarm, or even reminders. The software will facilitate ease of access to various other devices and platforms. The system will have the following phases: Data collection in the form of voice; Voice analysis and conversion to text; Data storage and processing; generating speech from the processed text output.

REFERENCES

2.1 **Ricci, F., Rokach, L., Shapira, B., & Kantor, P. B.** (2011). *Recommender Systems Handbook*. Springer.

2.2 **Koren, Y., Bell, R., & Volinsky, C.** (2009). Matrix Factorization Techniques for Recommender Systems. *IEEE Computer*, 42(8), 30-37.

2.3 **Linden, G., Smith, B., & York, J.** (2003). Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, 7(1), 76-80.

2.4 **Resnick, P., & Varian, H. R.** (1997). Recommender Systems. *Communications of the ACM*, 40(3), 56-58.

2.5 **Google Books API Documentation**. (2023). Available at: https://developers.google.com/books

2.6 **Scikit-learn Documentation**. (2023). Available at: https://scikit-learn.org/stable/

2.7 **Kaggle Datasets**. (2023). Available at: https://www.kaggle.com/datasets