Recommendation System using NLP and Vector Databases

Mohammad Shweiki Basil Al-Jamal

Supervisor: Dr. Hashem Tamimi

Acknowledgment

In the name of "Allah", the most beneficent and merciful who gave us strength, knowledge and helped us to get through this project. To the people that have inspired and supported us into the people that we are today, our families, friends and our supervisor. We would've never been able to reach this achievement without their support, care, and encouragement. We want to thank them all and we would like to express our gratitude to our graduation project supervisor Dr. Hashem Tamimi for his guidance, support, and encouragement throughout the project.

Moreover, we owe an immense debt of gratitude to our families, whose unwavering encouragement and continuous support have been the cornerstone of our journey. Their generosity, both in spirit and action, has shaped the very fabric of who we are. Mom, Dad, and all our family members, your belief in us has been a guiding light, and for that, we are profoundly thankful.

At last, we acknowledge the collective effort that has propelled us forward. Each person who touched our lives, leaving an imprint of care and encouragement, has played a vital role in our story. As we celebrate this milestone, we do so with gratitude for the shared moments and the countless individuals who have left an indelible mark on our hearts.

Abstract

This project aims to develop a cutting-edge web application for an online bookstore, featuring an advanced recommendation system to enhance the user experience. utilizing natural language processing (NLP) algorithms, dense vectors, and non-relational databases, this system offers users an intuitive platform to explore a vast collection of books. Users can search for titles based on keywords, authors, or genres, and receive personalized recommendations tailored to their preferences, derived from an analysis of their browsing history, to realize this project, Elastic-Search database was used for storing the documents and related vectors. Angular was used in implementing the frontend while Java Spring Boot was used for the backend.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Preface

In the dynamic landscape of e-commerce, personalized recommendations play a pivotal role in enhancing user experience and driving customer satisfaction. As the online marketplace continues to evolve, there is a growing need for intelligent and tailored suggestion systems. This graduation project addresses this demand by developing an innovative recommendation system for a virtual bookstore, leveraging the power of Natural Language Processing (NLP) and vector databases.

1.2 Project Aim

The primary aim of this project is to design and implement a web-based application for a bookstore that integrates NLP algorithms and vector databases. By harnessing the capabilities of NLP, the system aims to analyze textual data associated with books, such as content and user reviews, providing users with personalized and context-aware book recommendations. The recommendation engine will utilize vector databases to efficiently handle heavy computations, ensuring scalability and optimal performance.

1.3 Objectives

The specific objectives of the project are as follows:

1. Develop an NLP-driven book recommendation system from purchase history and books Descriptions.

2. Scale for real-time suggestions with optimized vector computations.

3. Design a visually intuitive interface for transparent book recommendations.

1.4 Problem Statement

Online bookstores face challenges in delivering personalized recommendations to users, relying on conventional methods that lack depth and scalability. This project addresses these issues by developing a recommendation system using Natural Language Processing (NLP) for content understanding. The system aims to provide personalized book suggestions based on NLP analysis, overcoming scalability challenges through the use of efficient vector databases. The primary goal is to enhance the online book-shopping experience, offering users tailored recommendations and fostering a more engaging literary exploration.

1.5 System Description

The system is designed to revolutionize the online book-shopping experience. It comprises a user-friendly web interface that facilitates effortless book browsing, searching, and shopping cart management. The heart of the system lies in its advanced recommendation engine, which employs Natural Language Processing (NLP) algorithms to analyze book content and user reviews. This enables the generation of personalized suggestions based on the contents of the user's shopping cart, ensuring a more tailored and engaging reading journey.

Furthermore, the system integrates vector databases to efficiently handle the storage and retrieval of book-related data. This vector database integration addresses scalability concerns, allowing the system to seamlessly scale as the number of books and users grows. The emphasis on real-time or near-real-time recommendations aims to optimize user experience and responsiveness.

1.6 Context Diagram

Here is an overview of the system's interaction with external entities, guiding stakeholders in aligning project requirements and facilitating analysis and design processes crucial for the graduation project's success.

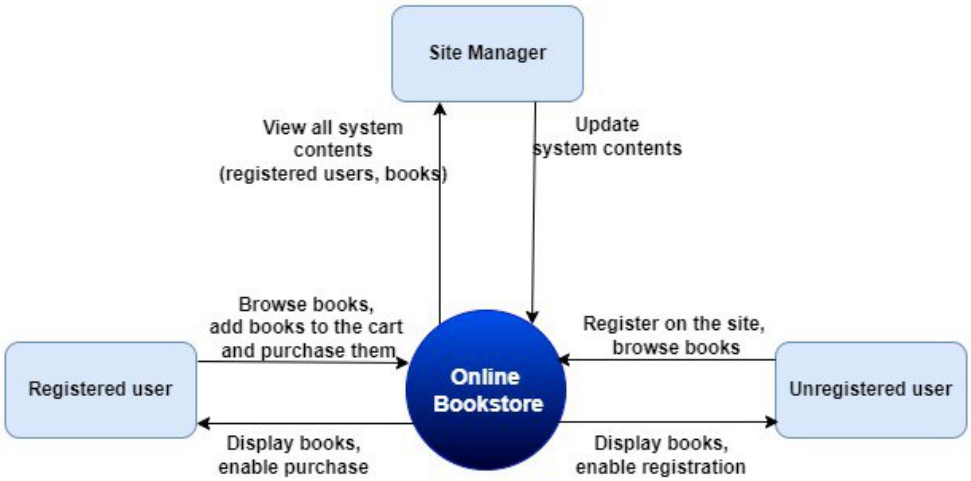

Figure 1: Context Diagram

1.7 Project Limitations/Constraints

While our project aims to deliver an advanced and personalized book-shopping experience, certain limitations and constraints need to be acknowledged:

- **Language Limitation:** The recommendation system focuses on books with English prefaces or descriptions. Recommendations may not be as accurate for books with limited or no English content.
- **Genre Specialization:** The system specializes in certain genres, and recommendations may be more accurate within these genres. Books outside the specified genres may not receive optimal suggestions.

1.8 Time Table

The system implementation and operation tasks are distributed along the first and second semesters, summarised in 1.1.

Table 1: Summary of Literature on Book Recommendation Systems

Title	**Team**	**Year**	**Main Idea**
NLP-based Book Recommendation: Building AI-based products from PoC to production-ready	Francisco Espiga	2022	This series explores the development of an NLP-based book recommendation system, focusing on creating book description embeddings and integrating them with a backend system.
NLP for Book Recommendation	Amoli Rajgor (AmoliR)	2022	This GitHub project demonstrates a content-based recommender system using NLP techniques, particularly BERT-embeddings and TF-IDF vectorization, for book descriptions.
Movie Popularity and Target Audience Prediction Using the Content-Based Recommender System	S. Sahu, R. Kumar, M. S. Pathan, J. Shafi, Y. Kumar, M. F. Ijaz	2022	Researchers achieved 96.8% accuracy in predicting movie success using a multiclass classification model, emphasizing the importance of predictive data analysis in decision-making.

The table above outlines the various tasks involved in the project along with the time allocated for each task. It is important to note that the project begins with information planning and collecting, which takes 2 weeks. This is followed by a similar period for background and alternatives analysis. The determination of system requirements is a relatively quick task, taking only 1 week. A more detailed analysis and design of system requirements require 3 weeks. The most time-consuming task is system programming and configuration, which takes 8 weeks. System testing also takes a significant amount of time, lasting 3 weeks. Finally, system documentation is an ongoing task that spans the entire project period, ensuring that all processes and changes are well-documented for future reference.

2 Theoretical Background

In the dynamic landscape of e-commerce, personalized recommendation systems have become essential in enhancing user experiences and driving customer satisfaction. As online marketplaces evolve, the need for intelligent and tailored suggestion systems becomes increasingly crucial. This section delves into the theoretical underpinnings of the technologies and methodologies employed in developing a sophisticated recommendation system for a virtual bookstore. We explore the foundational concepts of recommendation systems, the pivotal role of Natural Language Processing (NLP) in content analysis, and the implementation of word embedding techniques. Furthermore, we examine advanced models like BERT and Sentence-BERT, and their applications in semantic search and embedding vectors. The discussion also extends to non-relational databases and dense vectors, with a particular focus on vector databases and KNN search algorithms. The integration of Elasticsearch, and the use of Python libraries such as sentence-transformers, alongside the FastAPI framework, are reviewed to provide a comprehensive understanding of the system's architecture. Additionally, the section covers the implementation of Angular for the user interface and considers alternative handling approaches. This thorough exploration sets the stage for the design and implementation of a scalable, real-time book recommendation engine, leveraging cutting-edge NLP and vector database technologies.

2.1 Recommendation Systems

Recommendation systems, also known as recommender systems, are a subclass of information filtering systems that seek to predict the "rating" or "preference" a user would give to an item. These systems are designed to provide personalized suggestions to users based on various data sources and algorithms. They are widely used in various domains such as e-commerce, social media, streaming services, and more.

2.1.1 General Overview of Recommendation Systems

- **Collaborative Filtering**:

- **User-Based Collaborative Filtering**: Recommends items based on the preferences of similar users. If two users have a high overlap in their past interactions, items liked by one user are recommended to the other.
- **Item-Based Collaborative Filtering**: Recommends items similar to those a user has liked in the past. This method computes the similarity between items and suggests items that are similar to those the user has previously rated highly.

- **Content-Based Filtering**:
  - Recommends items based on the characteristics of the items and the user's past preferences. If a user has shown interest in certain attributes (e.g., genre, author), items with similar attributes are recommended.
  - This approach requires detailed item descriptions and user profiles to match users with relevant items.

- **Hybrid Systems**:
  - Combines multiple recommendation strategies to utilize the strengths of each. For example, a system might use both collaborative and content-based filtering to improve recommendation accuracy.
  - Hybrid methods can address some of the limitations of individual techniques, such as the cold start problem (difficulty in making recommendations for new users or items).

- **Knowledge-Based Systems**:
  - These systems use domain-specific knowledge and user requirements to recommend items. They are often used in cases where users have specific preferences or constraints that need to be considered.
  - Examples include recommendation systems for travel, real estate, and financial services, where user needs can be explicitly defined.

- **Context-Aware Systems**:
  - These systems consider additional contextual information (e.g., time, location, social context) to make more relevant recommendations.
  - They can adapt to changing user preferences based on the context in which recommendations are made.

2.2 Natural Language Processing (NLP)

Natural Language Processing is a subfield of artificial intelligence that focuses on the interaction between computers and human language[2]. In our project, NLP algorithms play a crucial role in analyzing book content and user reviews. By extracting semantic meaning and patterns from textual data, we aim to enhance the system's ability to generate contextually relevant book recommendations.

2.2.1 Word Embedding

Word embedding is a foundational technique in Natural Language Processing (NLP) utilized in our recommendation system. It involves representing words as dense vectors in a continuous vector space. This method captures semantic relationships and contextual meanings, enhancing the system's ability to understand and recommend books based on their textual content[3].

2.2.2 BERT

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a pre-trained natural language processing (NLP) model developed by Google. It revolutionized the field of NLP when it was introduced in 2019 by Devlin et al. [4]. BERT is based on the Transformer architecture, a deep learning model specifically designed for sequence-to-sequence tasks. What sets BERT apart is its ability to capture bidirectional contextual information from input text by using a technique called masked language modeling. During pre-training, BERT is exposed to vast amounts of text data and learns

to predict missing words within a sentence, considering both the words that come before and after the masked word. This bidirectional approach allows BERT to generate rich, context-aware word embeddings that capture the meaning of words based on their surrounding context. These pre-trained embeddings can then be fine-tuned on specific downstream NLP tasks, such as text classification, named entity recognition, sentiment analysis, and more, often leading to state-of-the-art performance across various domains.

2.2.3 Sentence-BERT

Sentence-BERT, abbreviated as SBERT, is an extension of the BERT (Bidirectional Encoder Representations from Transformers) model, which is specifically adapted for generating fixed-dimensional embeddings for sentences rather than individual words[5]. While the original BERT model is primarily designed for word-level tasks, SBERT focuses on encoding entire sentences into dense vectors, also known as embeddings. SBERT achieves this by fine-tuning the BERT architecture on tasks that require understanding sentence-level semantics, such as semantic textual similarity (STS) tasks or sentence classification. By training on such tasks, SBERT learns to generate embeddings that capture the semantic meaning of sentences, enabling applications like sentence similarity computation, clustering, and information retrieval. SBERT embeddings can be used in various natural language processing (NLP) tasks where understanding the semantic similarity between sentences is essential, leading to improved performance compared to traditional methods.

2.2.4 Semantic Search

Semantic search is a sophisticated search approach that aims to enhance search accuracy by interpreting the context and meaning of user queries. Unlike traditional keyword-based search engines, semantic search engines utilize natural language processing (NLP) techniques to analyze the semantic structure of text and understand the relationships between words, phrases, and concepts. By recognizing synonyms, related terms, and contextual clues, semantic search systems can infer the user's intent more accurately and generate results that are contextually relevant, even if they don't precisely match the keywords

used in the query. This capability allows semantic search engines to provide more nuanced and precise search results, improving user satisfaction and enabling better access to information across various domains such as web search, e-commerce, and enterprise knowledge management. Additionally, semantic search systems often incorporate machine learning algorithms to continuously refine their understanding of language patterns and user preferences, leading to further improvements in search quality over time.

2.2.5 all-MiniLM-L6-v2

The 'all-MiniLM-L6-v2' is a robust sentence-transformers model designed to map sentences and short paragraphs into a 384-dimensional vector space, suitable for tasks like clustering and semantic search. utilizing self-supervised contrastive learning, the model was fine-tuned on an extensive dataset comprising over 1 billion sentence pairs sourced from diverse sources like Reddit comments, S2ORC citations, and Stack Exchange questions. Training benefited from efficient hardware infrastructure and utilized hyperparameters such as a batch size of 1024, sequence length limitation of 128 tokens. With intended applications including information retrieval and sentence similarity tasks, the model provides a powerful tool for encoding semantic information from textual inputs.

2.3 Non-relational Databases

Non-relational databases [8], also known as NoSQL databases, play a crucial role in efficiently managing and storing diverse data within our recommendation system. These databases are utilized to store book-related information, user data, and other regular (non-vector) data integral to the system's functionality, and we chose non-relational databases over relational ones due to their scalability advantages, particularly in handling large and diverse datasets with varying structures and formats.

2.4 Dense Vectors

Dense vectors, in the context of machine learning and natural language processing (NLP), refer to vectors where most of the elements are non-zero. These vectors are often used to

represent high-dimensional data, such as words, sentences, or documents, in a continuous vector space. Dense vectors are typically composed of real numbers and can capture rich semantic information about the underlying data. In NLP tasks, dense vectors are commonly employed as embeddings, where words or sentences are mapped to dense vector representations using techniques like word2vec, GloVe, or BERT. These embeddings encode semantic relationships between words or sentences, facilitating various downstream tasks such as text classification, sentiment analysis, machine translation, and more [5]. Dense vectors contain information in every dimension, making them suitable for capturing complex patterns and relationships within data.

2.4.1 Vector Databases

Vector databases are optimized for storing and retrieving vector-based data efficiently. In our project, we utilize vector databases to handle the storage and retrieval of book-related data. This ensures scalability as our system grows, enabling real-time or near-real-time recommendations by optimizing vector computations.

2.4.2 KNN Search

KNN Search is a method in machine learning and information retrieval where data points are represented as vectors in a high-dimensional space, indexed for efficient nearest neighbor retrieval. Upon receiving a query vector, the algorithm retrieves the K nearest neighbors based on a chosen distance metric, like Euclidean distance or cosine similarity [5]. These neighbors are then ranked or scored according to their similarity to the query vector, with the top-ranked results returned as the search output. Widely applied in recommendation systems, image recognition, and various other fields, KNN search efficiently identifies similar items or documents, facilitating tasks such as personalized recommendations and content retrieval.

2.5 Elasticsearch

Elasticsearch is a distributed, RESTful search and analytics engine renowned for its horizontal scalability, real-time search capabilities, and high reliability. Built on top of Apache Lucene and developed in Java, it excels in storing, searching, and analyzing vast amounts of data quickly and efficiently. With features like near real-time search, schema-free JSON document storage, a RESTful API, powerful analytics and aggregation capabilities, and high scalability and performance, Elasticsearch is widely used for various applications such as log analysis, full-text search, real-time analytics, and more. Additionally, it offers the ability to store vector data and serves as a non-relational database solution, making it versatile for diverse data needs. Its distributed architecture ensures high availability and fault tolerance, making it a robust solution for organizations dealing with large-scale data challenges.

2.6 Python

Python is a general-purpose programming language. Its high-level data structures, dynamic typing, and many other features make it equally useful for scripting or "glue code" that ties components together as they do for developing complex applications. Additionally, it can be enhanced to execute code written in languages like C, C++, or Rust and make system calls on almost all operating systems. It is especially useful in data science and machine learning thanks to its concise syntax, interpreted nature, and extensive library collection.

For our project, when comparing Python and NodeJS in the context of our project Python surpasses NodeJS in natural language processing (NLP) and Machine learning tasks. With libraries like SKlearn, NLTK, Python streamlines text data handling. Its clear syntax and strong community support enable efficient NLP feature development.

2.6.1 sentence-transformers

The 'sentence-transformers' Python library is a versatile tool for generating dense vector representations of sentences, utilizing transformer models like BERT and RoBERTa. With an extensive collection of pre-trained models and straightforward integration into existing projects, it enables users to effortlessly compute embeddings for various natural language processing tasks such as semantic similarity, clustering, and classification. Its support for customization through fine-tuning on specific datasets or tasks further enhances its utility, catering to diverse application requirements. Benefiting from an active community and continual development, 'sentence-transformers' stands as a valuable resource for researchers and practitioners seeking effective solutions for understanding and processing text at the sentence level.

2.6.2 FastAPI

FastAPI is a python back-end framework that can be used to build RESTFul and GraphQL APIs, it has built in support for authentication and authorization, and validation based on user defined schema.

2.7 Angular

Angular is a comprehensive open-source web application framework developed and maintained by Google, designed for building dynamic and interactive single-page web applications (SPAs). Built on TypeScript, it promotes a component-based architecture, enabling developers to create reusable and modular components encapsulating HTML templates, CSS styles, and TypeScript code. Angular offers features like two-way data binding for seamless synchronization between model and view, dependency injection for managing component dependencies, built-in directives for extending HTML functionality, routing for multi-view applications, and powerful form handling capabilities. With its robust ecosystem and extensive tooling, Angular is a popular choice for developing modern web applications that require scalability, maintainability, and performance.

2.8 Literature Review

The literature on book recommendation systems highlights significant contributions from various projects and research papers. This review synthesizes key methodologies, models, and findings from these works, providing a comprehensive overview of the state of the art in this field.

Table 2: Summary of Literature on Book Recommendation Systems

Title	**Team**	**Year**	**Main Idea**
NLP-based Book Recommendation: Building AI-based products from PoC to production-ready	Francisco Espiga	2022	This series explores the development of an NLP-based book recommendation system, focusing on creating book description embeddings and integrating them with a backend system.
NLP for Book Recommendation	Amoli Rajgor (AmoliR)	2022	This GitHub project demonstrates a content-based recommender system using NLP techniques, particularly BERT-embeddings and TF-IDF vectorization, for book descriptions.
Movie Popularity and Target Audience Prediction Using the Content-Based Recommender System	S. Sahu, R. Kumar, M. S. Pathan, J. Shafi, Y. Kumar, M. F. Ijaz	2022	Researchers achieved 96.8% accuracy in predicting movie success using a multiclass classification model, emphasizing the importance of predictive data analysis in decision-making.

- **NLP-based Book Recommendation: Building AI-based products from PoC to**

production-ready

  - *Team:* Francisco Espiga
  - *Year:* 2022
  - *Main Idea:* This work explores the construction of a book recommendation system using NLP. It focuses on creating deep learning models to generate book description embeddings and discusses the integration of these models into a backend system.

- **NLP for Book Recommendation**
  - *Team:* Amoli Rajgor (AmoliR)
  - *Year:* 2022
  - *Main Idea:* This project, hosted on GitHub, illustrates a content-based book recommender system using NLP techniques. It employs BERT embeddings to extract keywords from book descriptions and applies TF-IDF for feature extraction. The system then uses cosine similarity to recommend books based on these features.

- **Movie Popularity and Target Audience Prediction Using the Content-Based Recommender System**
  - *Team:* S. Sahu, R. Kumar, M. S. Pathan, J. Shafi, Y. Kumar, M. F. Ijaz
  - *Year:* 2022
  - *Main Idea:* This research paper demonstrates the use of a multiclass classification model to predict movie success with a high accuracy of 96.8%. The study highlights the capabilities of predictive and prescriptive data analysis in enhancing decision-making within information systems. Despite its success, the paper notes the need for expert systems to predict a movie's probability of success with reasonable accuracy.

Table 2 provides a summary of key projects and their contributions to the fields of personalized recommendation systems, natural language processing (NLP), and vector databases. Each project listed has made significant advancements in understanding and applying these technologies to enhance user experiences in various domains, particularly in e-commerce and online recommendations.

2.9 Handling alternatives

From the above discussion we can summarize the alternatives that we had and the recommended ones. as in Table3:

Table 3: Summary of different alternatives

Subject	**Alternatives**	**Recommended alternative**
Database	Relational vs. Non-relational	Non-relationalis are more scalable
Vector Database	Pinecone vs. Elasticsearch	Elasticsearch is self-hosted
Non-relational Database	Elasticsearch vs MongoDb	Elasticsearch provides fast Full-text search
Backend	Python vs NodeJS	Python is more suitable for NLP
Frontend	Angular vs VueJS	Angular has better support

2.10 Summary

This chapter explored the key technologies and theories behind our personalized book recommendation system. We covered the fundamentals of recommendation systems and the role of NLP techniques such as word embedding, BERT, and Sentence-BERT in content analysis. We discussed non-relational databases and vector databases for scalability, the use of Elasticsearch for data retrieval, and the implementation of Python libraries like sentence-transformers with FastAPI. Additionally, we reviewed the use of Angular for the user interface. These insights provide a solid foundation for building an efficient and scalable recommendation system.

3 System Design and Analysis

3.1 Preface

This chapter provides an in-depth exploration of the architectural and design decisions that form the foundation of our recommendation system. We delve into the intricacies of system components, data flow, and the technologies employed to create a robust and scalable project.

3.2 Requirements

3.2.1 Functional Requirements

- **Registration and Authentication:** Unregistered users should be able to register for an account securely, providing necessary details such as username, email address, and password. The system should validate user input and implement measures to protect user information. Registered users should be able to log in securely using their credentials.

- **Book Browsing and Searching:** All users should have the ability to browse the bookstore's collection of books through categories, genres, or featured titles. A search functionality should enable users to find books by entering keywords, titles, authors, or ISBN numbers. The system should provide relevant search results quickly and accurately, with options to filter and sort the results.

- **Shopping Cart Management:** Registered users should be able to add books to their shopping cart and view a summary of their selected items. The shopping cart interface should allow users to modify the quantity or remove items before proceeding to checkout. The system should calculate the total cost, including taxes and any applicable discounts, and provide secure payment options.

- **Personalized Book Recommendations:** The system should analyze user interactions, browsing history, and preferences to generate personalized book recommen-

dations. Recommended books should be prominently displayed, with options for users to dismiss suggestions or provide feedback.

- **Feedback Mechanism:** Registered users should be able to provide feedback on recommended books by rating them, writing reviews, or leaving comments. Administrators should have access to a moderation interface to manage user feedback and respond to comments. User feedback should be analyzed to enhance the recommendation algorithm.

- **Admin Dashboard:** Administrators should have access to a secure dashboard to manage system settings, users, and content. The dashboard should provide an overview of key metrics and allow administrators to perform CRUD operations on users, books, and orders. Advanced features such as data visualization and reporting tools should be available.

- **Content Management:** Administrators should be able to add, edit, and remove books from the bookstore's collection. The content management interface should be intuitive, allowing administrators to update book details, upload cover images, and apply tags for categorization.

- **Order Management:** Administrators should have access to an order management interface to view, process, and fulfill orders. Each order should contain essential details such as customer information, order items, and payment status. Administrators should be able to update order statuses and communicate with customers regarding order updates.

- **View Registered Users:** Administrators should be able to access and view a list of registered users within the system. This functionality allows administrators to review user profiles, monitor user activity, and manage user accounts as needed. The view should include relevant user information such as usernames, email addresses, registration dates, and any additional details captured during user registration.

3.2.2 Non-functional Requirements

- **Scalability:** The architecture should be scalable to accommodate a growing number of books, users, and transactions without compromising performance.
- **Reliability:** The system should be reliable and available 24/7, minimizing downtime and ensuring uninterrupted access for users.
- **Compatibility:** The system should be compatible with various devices and web browsers, ensuring a consistent experience across different platforms like IOS, Android..etc.

3.3 Analysis and use cases

Here is an illustration on the various interactions between users and the system, highlighting the functionalities provided by the system. Figure 2 shows the use case diagram of our project.

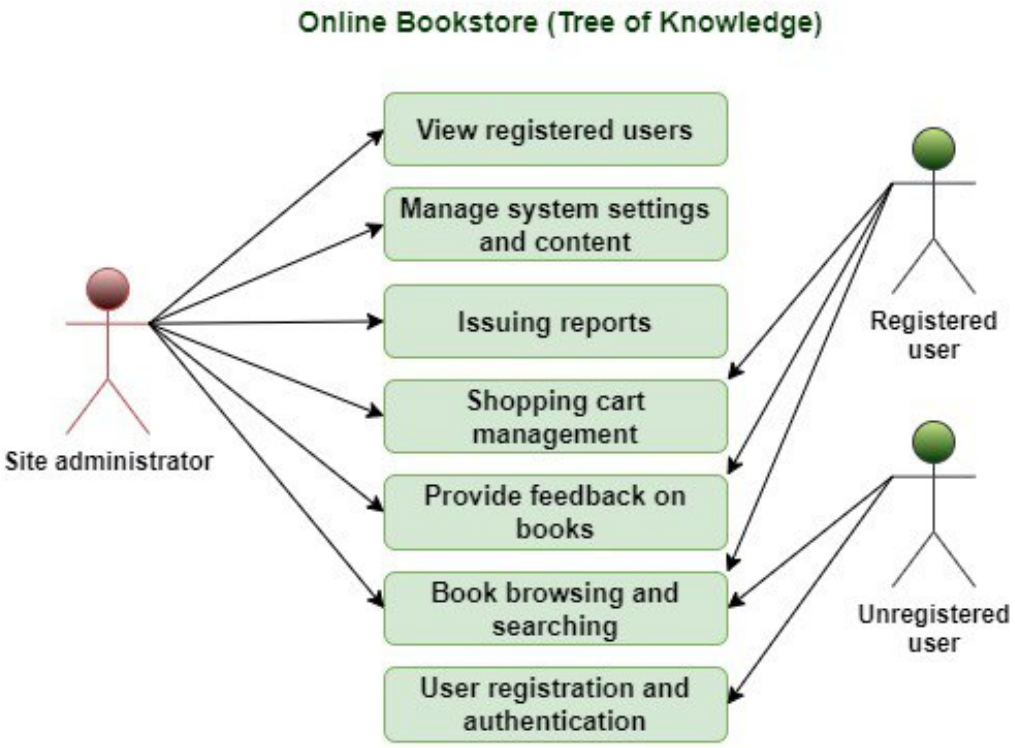

Figure 2: Use Case diagram

The two main functionalities that we focus on in our project are the *Search Books* and the *View recommendation*. Here are the detailed analysis for each one:

3.3.1 Book Addition

- **Aim**: Allow Admin to add new books to the system.
- **Use Case Description**: This use case describes the process of a registered user adding a new book to the system. The user provides information about the book, including title, author, ISBN, and description. The system validates the input, generates an embedding for the description, and stores the book information along with the embedding (Dense Vector) in the database.
- **Actors**: Admin.
- **User Interactions**:
  - User enters complete book details, including description, on the user interface.
  - User submits the form. (Validation can be implicit or handled client-side)
  - The system retrieves the user's input and sends it for processing.
  - The system validates the data, generates an embedding for the description, and stores the book with its embedding in the database.
  - The system provides feedback to the user (success/error) regarding the book addition.
- **Pseudocode**:

```
User enters book information (title, author, ISBN, description,...)

If validation fails:
    Client displays error message to User

If no duplicate ISBN is found:
    Client sends request to add book to API
    API adds book information to Database, including embedding
    If duplicate ISBN found:
        Database returns error message to API
    Else:
        API sends book description to Model for embedding generation
        Model generates embedding for description
        Database creates new book entry with all information
        Database sends confirmation message to API
        API sends confirmation message to Client
        Client displays success message to User
```

Figure 3: Book Addition Pseudocode

Figure 4 shows the sequence diagram for the addition.

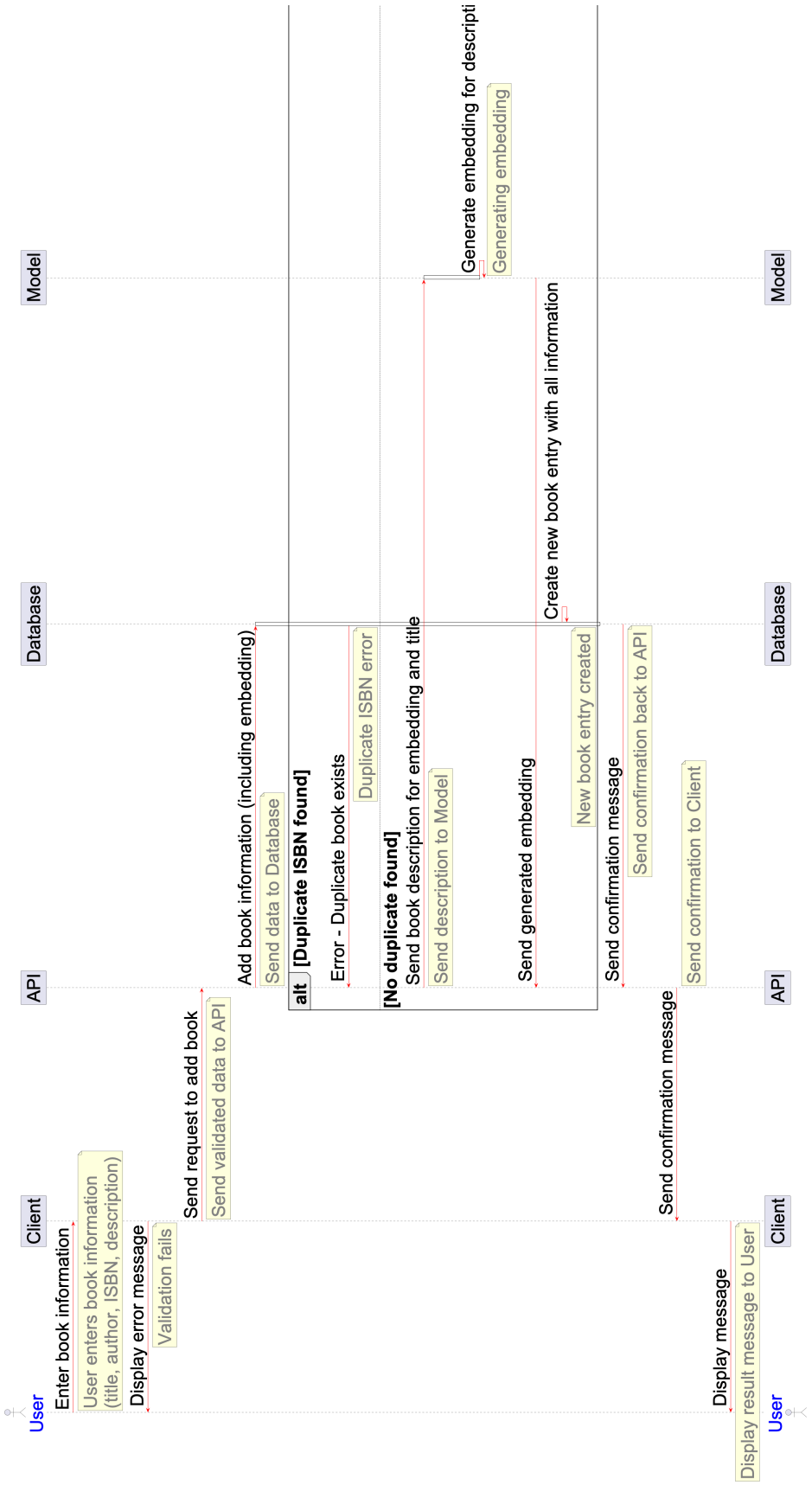

Figure 4: Book Addition Sequence Diagram

3.3.2 Search Books

- **Aim**: Users efficiently discover books in the bookstore's collection by utilizing the search feature. By entering keywords, titles, or authors, they swiftly locate desired literature, streamlining the book exploration process.
- **Use Case Description**: Users navigate to the bookstore's search feature and enter search terms. The system retrieves books matching the criteria, and users review the results to select desired books for further action.
- **Actors**: Both registered and unregistered users.
- **User Interactions**:
  - User navigates to the search feature.
  - User enters search terms.
  - System retrieves matching books.
  - User reviews search results and selects desired book.
- **Pseudocode**:

```
User inputs search text

If there is a network error:
    Show User: "Network Error"
Else:
    Get results from API (search text)
    Show User: "Search Results:"
    For each book in results:
        Show User: book.title + book.image + book.authors + book.price

API receives search text:
    Get embedding from Model (search text)
    Find nearest neighbors in Database (embedding)
    Send sorted results back to User

Model receives search text:
    Create embedding from text

Database receives embedding:
    Find closest entries (embedding)
    Send results back to API
```

Figure 5: Book Search Pseudocode

Figure 6 shows the sequence diagram for the search.

Figure 6: Search Books Sequence Diagram

3.3.3 View Recommendations

- **Aim**: Users access personalized book recommendations based on their preferences and browsing history.
- **Use Case Description**: Users navigate to the recommendations section, where the system analyzes their history and presents relevant book suggestions. Users explore the recommendations and select books to view more details or add to their shopping cart.
- **Actors**: Both registered and unregistered users.
- **User Interactions**:
  - User navigates to the recommendations section.
  - System analyzes user history and presents book suggestions.
  - User explores recommendations and selects desired books.
- **Pseudocode**:

```
User clicks "Recommend me" button

If user preferences are unavailable:
    Client retrieves user preferences
    User sends user preferences to Client

Client sends user preferences to API
API requests recommendations based on preferences from Database

If user preferences are available:
    Database identifies user preferences embedding
    Database performs KNN search on book embeddings
    Database retrieves nearest neighbor books

Database returns list of recommended books with details to API
API sends sorted list of recommendations to Client
Client displays recommendations page to User

User interacts with recommendations (e.g., click book, add to wishlist)
```

Figure 7: View Recommendation Pseudocode

Figure 8 shows the sequence diagram for the recommendations.

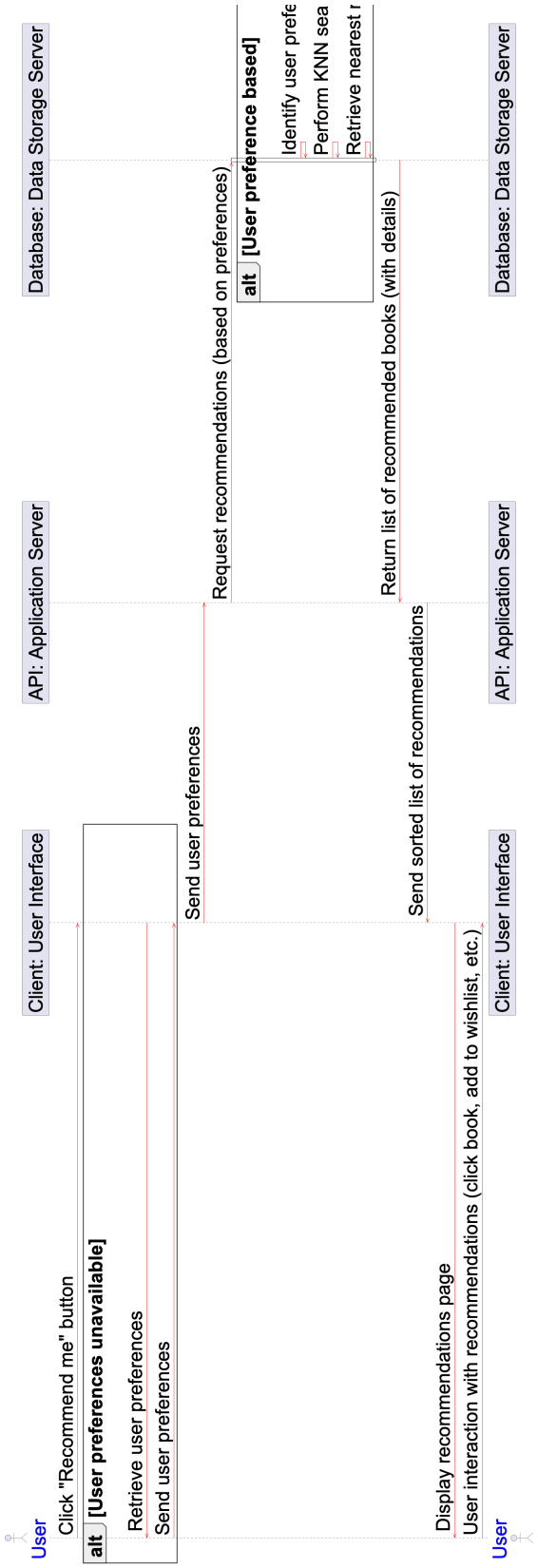

Figure 8: View Recommendations sequence diagram

3.3.4 Books Purchase

- **Aim**: Streamlined book purchasing with user preference-based recommendations.
- **Use Case Description**: Registered users browse books, add them to a virtual cart, and view cart details. The system retrieves or generates a user preference embedding to suggest relevant books during cart interaction.
- **Actors**: Registered User.
- **User Interactions**:
  - Browse books (optional).
  - Add book to cart (ISBN, quantity).
  - System retrieves/generates user preference embedding.
  - View cart contents (books, quantities, subtotal).
- **Pseudocode**:

```
User browses book listings

User clicks "Add to Cart" for a book
Client adds book to cart (book ISBN, quantity)

If user preference embedding exists:
    API retrieves user preference embedding from Database
Else:
    API analyzes purchase history (books, genres, etc.) and generates user preference embedding using Model
    If new embedding:
        API stores user preference embedding in Database

Model sends book information and user preference embedding to API

API confirms success/failure and updates cart
Client updates user interface (confirmation, cart view)

User views cart contents
Client retrieves cart data from API
API gets user's cart information from Database
API sends cart information (books, quantities, subtotal) to Client
Client displays cart contents to User
```

Figure 9: Book Purchase Pseudocode

Figure 10 shows the sequence diagram for the book purchasing process.

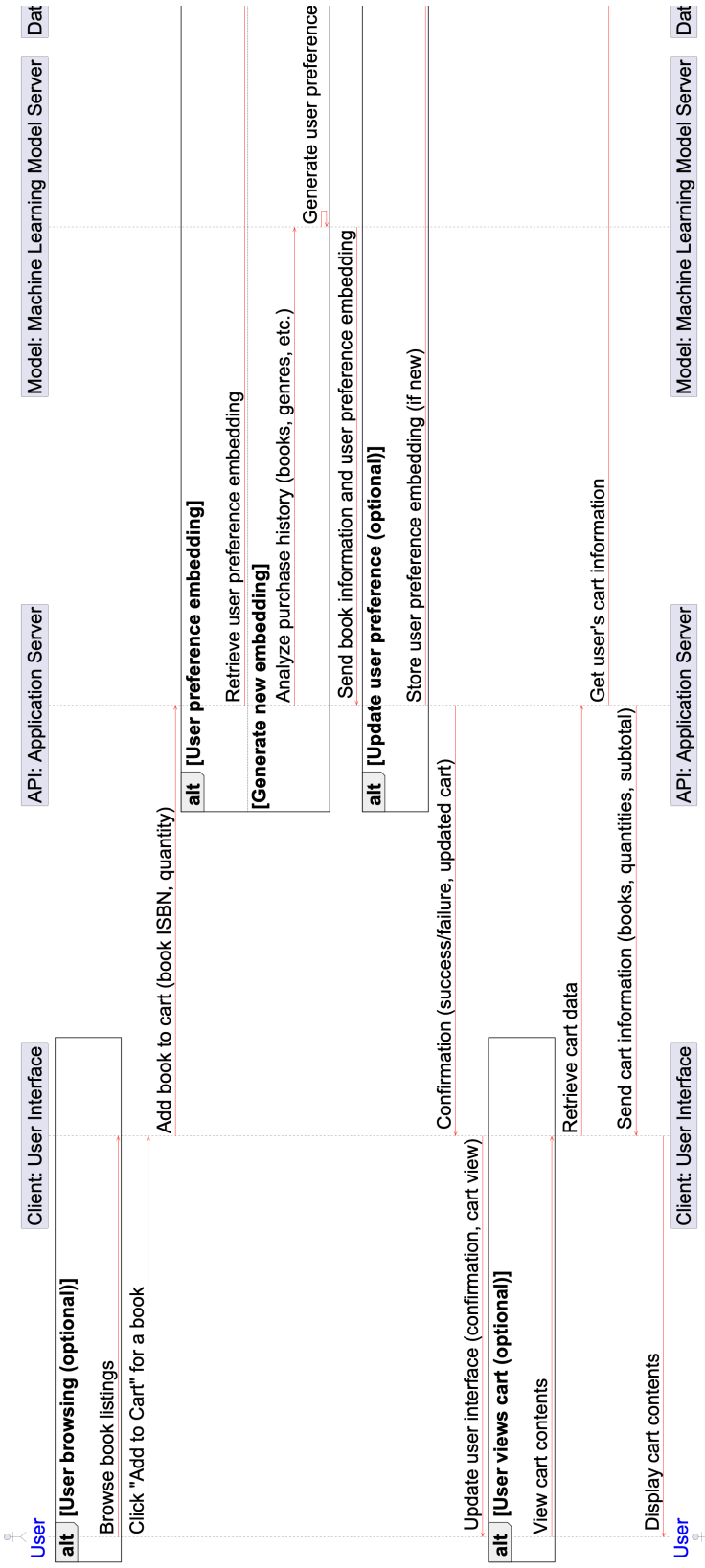

Figure 10: Purchase Books sequence diagram

3.4 Database Design

3.4.1 Introduction

The database design for our web application bookstore is a crucial component that ensures the system's scalability, performance, and reliability. By utilizing Elasticsearch for non-relational and dense vector storage, we utilize its powerful full-text search capabilities, real-time indexing, and efficient handling of large datasets. This chapter outlines the design principles, schema structure, and integration of Elasticsearch in our recommendation system.

3.4.2 Design Principles

Our database design is guided by the following principles:

- **Scalability**: To accommodate a growing number of books and users without performance degradation.
- **Performance**: To ensure fast search and retrieval times, providing a seamless user experience.
- **Flexibility**: To support complex queries and personalized recommendations.
- **Reliability**: To maintain data integrity and consistency across operations.

3.4.3 Schema Design

The database schema serves as the blueprint for organizing and storing information within the system. It defines the structure of data entities, their attributes, and the relationships between them. This schema is crucial for ensuring the effective management and retrieval of data, providing a solid foundation for the bookstore application's functionality.

In Elasticsearch, data is stored in indices, each containing multiple documents. Each document represents a book and includes various fields for search and recommendation purposes.

In this section, we outline the database schema, focusing on two primary indices: `Books` and `Users`. The `Books` index contains information about the bookstore's collection, including details about individual books and their attributes. On the other hand, the `Users` index stores information about the application's users, facilitating user management and personalized experiences.

- **Books**

  The `Books` index serves as a repository for information about the bookstore's collection. Each entry in this index represents a unique book and includes various attributes to describe its characteristics. These attributes encompass essential details such as the book's ISBN, title, authors, genres, price, publication date, and more. Additionally, the index incorporates nested attributes to provide deeper insights into each book's popularity and reception.

  Attributes Table:

Attribute	**Type**	**Description**
isbn	String	Unique identifier for the book
title	String	Title of the book
authors	List of Strings	List of authors contributing to the book
genres	List of Strings	List of genres associated with the book
price	Float	Price of the book
description	String	Brief description of the book
publicationDate	String	Date of publication of the book
ratings	`RatingEntity`	Ratings and average rating of the book

Continued on next page

Table 4 – *Continued from previous page*

Attribute	**Type**	**Description**
awards	List of Strings	List of awards received by the book
publisher	String	Publisher of the book
language	String	Language in which the book is written
pages	Integer	Number of pages in the book
available	Integer	Number of available copies of the book
sold	Integer	Number of copies sold
imageLink	String	URL link to the image of the book
embedding	List of Floats	Dense vector representing the content of the book
onSale	Boolean	Flag indicating if the book is on sale
salePrice	Float	Sale price of the book if it's on sale

Table 4: Attributes of the `Book` table

Nested Attributes:

RatingEntity:

Table 5: Attributes of the `RatingEntity` attribute

Attribute	**Type**	**Description**
ratingList	List of `Rating`	List of ratings given by users
averageRating	Float	Average rating of the book
ratingSum	Integer	Sum of all ratings

Rating:

Table 6: Attributes of the `Rating` attribute

Attribute	**Type**	**Description**
userId	String	User identifier who gave the rating
rating	Integer	Rating given by the user
description	String	Description/comment provided by the user

Sample Document Image: Below is a JSON sample illustrating how a Book document is structured and stored within the database.

```
{
    "isbn": "9780385495325",
    "title": "The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography",
    "authors": [
        "Simon Singh"
    ],
    "genres": [
        "Nonfiction",
        " Science",
        " History",
        " Mathematics",
        " Computer Science",
        " Technology",
        " Popular Science",
        " Computers",
        " Microhistory",
        " Espionage"
    ],
    "price": 3.0,
    "description": "The Code Book tells the story of the most powerful intellectual weapon ever known: secrecy.Throughout the text are
      clear technical and mathematical explanations, and portraits of the remarkable personalities who wrote and broke the world's most
      difficult codes. Accessible, compelling, and remarkably far-reaching, this book will forever alter your view of history and what
      drives it. It will also make you wonder how private that e-mail you just sent really is.",
    "publicationDate": "2000-08-29",
    "ratings": [{
        "userId" : "9780618680",
         "rating" : 5,
         "description" : "Well written, accessible, and rather thorough, this book remains one of the best books on encryption for
           laypersons. Highly recommended."
    }],
    "awards": [
        "Corine Internationaler Buchpreis for Sachbuch 2001"
    ],
    "publisher": "Anchor",
    "language": "English",
    "pages": 412,
    "embedding" : [
        0.058759086,
        0.101835065,
        -0.00959089,
        0.015944667,
        0.002372314,
        0.060027108,
        0.050473057,
        -0.14160548,
        0.016236814,
        -0.02043557,
        0.04264645],
    "available": 10,
    "sold": 10,
    "imageLink": "https://i.gr-assets.com/images/S/compressed.photo.goodreads.com/books/1403181687l/17994.jpg",
    "onSale": true,
    "salePrice": 2.0
}
```


Figure 11: Sample document of the `Book` index

- **Users**

  The `Users` index stores information about users of the bookstore application. Each entry in this index represents a unique user and includes various attributes to describe their profile and interactions within the platform. These attributes encompass essential details such as the user's ID, username, email, password, and more. Additionally, the index incorporates nested attributes like the user's wishlist, shopping cart, and order history, providing a comprehensive view of user engagement and preferences.

  Attributes Table:

Attribute	**Type**	**Description**
userId	String	Unique identifier for the user
username	String	Username of the user
firstName	String	First name of the user
lastName	String	Last name of the user
email	String	Email address of the user
password	String (encrypted)	Password of the user (encrypted)
wishlist	Set of Strings	Set of book identifiers in the user's wishlist
cart	`Cart`	Shopping cart of the user
orders	`OrdersEntity`	Orders placed by the user
role	`UserRole`	Role of the user (admin, customer, publisher)
favoriteGenres	Set of Strings	Set of favorite genres chosen by the user

Continued on next page

Table 7 – *Continued from previous page*

Attribute	Type	Description
embedding	List of Floats	Dense vector representing the user's characteristics
wallet	Double	Wallet balance of the user

Table 7: Attributes of the `User` index

Nested Attributes:

Cart:

Attribute	Type	Description
items	HashMap	Items in the cart
totalPrice	Double	Total price of all items in the cart

Table 8: Nested attributes of the `Cart` attribute

CartItem:

Table 9: Attributes of the `CartItem` attribute

Attribute	Type	Description
bookTitle	String	Title of the book in the cart
bookAuthors	List of Strings	List of authors of the book in the cart
quantity	Integer	Quantity of the book in the cart
price	Float	Price of a single unit of the book in the cart
totalPrice	Float	Total price of the book in the cart

Sample Document Image: Below is a JSON sample illustrating how a User document is structured and stored within the database.

```
{
  "userId": "3JdmoI8BBhQGvT0K_uoo",
  "username": "Basil_",
  "firstName": "Basil",
  "lastName": "Al-Jamal",
  "email": "201067@ppu.edu.ps",
  "wishlist": [
    "9781451648539",
    "9780141000510"
  ],
  "cart": {
    "items": {
      "9780735611313": {
        "bookTitle": "Code: The Hidden Language of Computer Hardware and Software",
        "bookAuthors": [
          "Charles Petzold"
        ],
        "quantity": 1,
        "price": 10.0,
        "totalPrice": 10.0
      }
    },
    "totalPrice": 10.0
  },
  "orders": {
    "orders": [
      {
        "date": "2024-06-04",
        "cart": {
          "items": {
            "9780804139298": {
              "bookTitle": "Zero to One: Notes on Startups, or How to Build the Future",
              "bookAuthors": [
                "Peter Thiel",
                "Blake Masters (Goodreads Author)"
              ],
              "quantity": 1,
              "price": 6.0,
              "totalPrice": 6.0
            }
          },
          "totalPrice": 6.0
        },
        "totalPrice": 6.0,
        "status": "CONFIRMED",
        "address": "ahmad mohsen street",
        "phoneNo": "+972435352"
      }
    ],
    "total": 6.0
  },
  "favoriteGenres": [
    "Economy",
    "Computer Sciennce",
    "Programming",
    " Nonfiction",
    " Economics",
    " Technology",
    " Finance",
    " Entrepreneurship",
    " Self Help",
    " Management",
    " Buisness",
    "Business",
    " Leadership"
  ],
  "wallet": 19.0,
  "authorities": [
    {
      "authority": "CUSTOMER"
    }
  ]
}
```


Figure 12: Sample document of the `User` index

3.5 System Architecture

The bookstore application is designed using a hybrid architecture that combines the flexibility and scalability of microservices with the clear separation of concerns provided by layered architecture principles. This design ensures that each component of the system is both independently manageable and highly cohesive, facilitating easier development, deployment, and maintenance. The system architecture is composed of four key components: the frontend, backend, AI NLP microservice, and the database. These components interact seamlessly to deliver a robust, scalable, and efficient application capable of handling complex user interactions and data processing. Below is a detailed description of each component and their interactions within the system.

3.5.1 Frontend

The frontend is developed using Angular, a powerful framework for building dynamic web applications. It serves as the presentation layer, handling all user interactions and providing a responsive user interface. Users can browse books, input their preferences, and view recommendations seamlessly through this layer. The frontend communicates with the backend through HTTP requests, ensuring a smooth and interactive user experience.

3.5.2 Backend

The backend is implemented using Java Spring Boot, forming the core of the business logic layer. It acts as the central orchestrator of the application, managing user requests, processing business logic, and interacting with other services and the database. The backend exposes RESTful APIs that the frontend consumes and handles communication with the AI NLP microservice and the Elasticsearch database.

3.5.3 AI NLP Microservice

The AI NLP microservice is built using FastAPI, a modern web framework for building APIs with Python. This microservice specializes in natural language processing (NLP), where it processes book descriptions or user preferences and converts them into dense

vectors. These vectors encapsulate semantic information, enabling the backend to provide personalized recommendations. The microservice operates independently, allowing for scalable and modular development.

3.5.4 Database

The database layer is powered by Elasticsearch, a highly scalable and distributed search and analytics engine. Elasticsearch stores non-relational indexes of books and user data, as well as the dense vector embeddings generated by the AI NLP microservice. This setup ensures efficient storage, retrieval, and querying of large datasets, supporting the application's need for fast and accurate search capabilities.

3.5.5 Component Interactions

The interactions between the components are designed to ensure seamless data flow and processing across the system:

- The Angular frontend sends HTTP requests to the Java Spring Boot backend to fetch and display data.
- The backend communicates with the AI NLP microservice to process and embed book descriptions or user preferences into dense vectors.
- The backend stores and queries data from Elasticsearch, including both non-relational indexes and vector embeddings, to deliver precise search results and personalized recommendations.

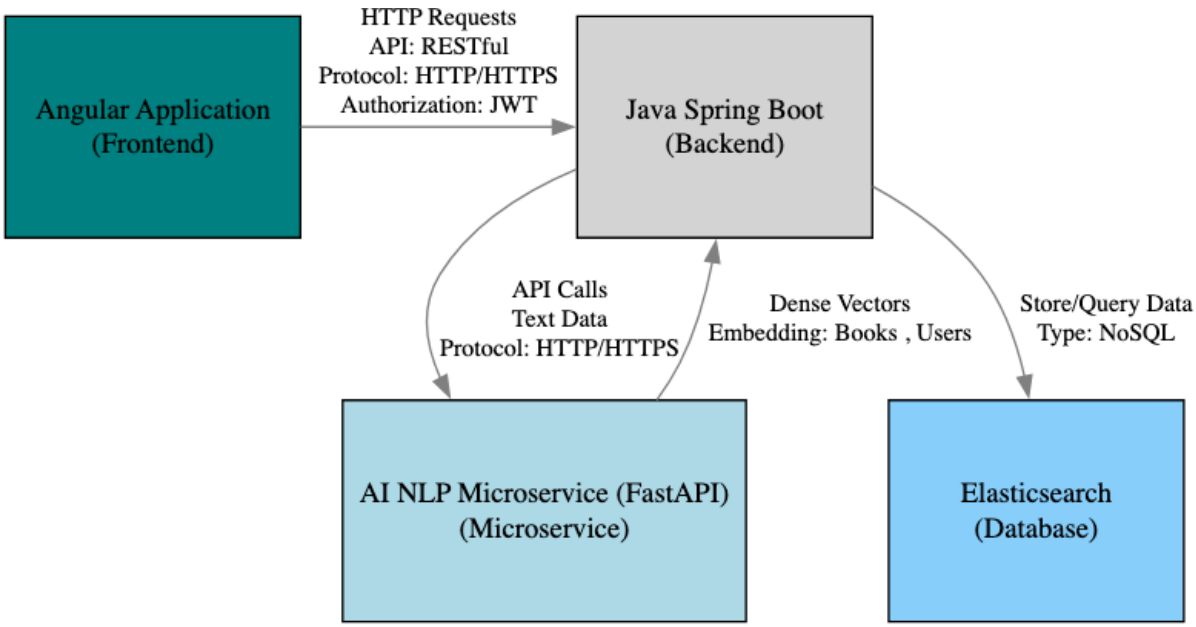

Figure 13: General Overview of System Components

This architecture utilizes the benefits of microservices for independent scalability and deployment, while maintaining the clear separation of concerns offered by a layered approach. The combination ensures that the bookstore application is robust, scalable, and maintainable, capable of handling complex user interactions and data processing efficiently.

3.6 Summary

In summary, the system design and description chapter provides a comprehensive overview of the requirements, analysis, and architectural design of the bookstore application. The chapter begins with a detailed examination of both functional and non-functional requirements, followed by an analysis of key use cases such as book addition, search functionality, view recommendations, and book purchase. Additionally, the chapter delves into the database design principles and schema design to ensure efficient data management. Finally, the system architecture section outlines the components of the frontend, backend, AI NLP microservice, and database, along with their interactions, laying the foundation for the implementation of the system. Overall, this chapter serves as a blueprint for

the development and deployment of the bookstore application, ensuring alignment with stakeholder expectations and project objectives.

4 Implementation, Testing and Results

This chapter is dedicated to the comprehensive results of our bookstore application. It covers two critical aspects: the visual validation of the application's user interface and the performance evaluation of its core functionalities. Detailed screen captures are provided to demonstrate the various user interactions and features, ensuring that the application meets the expected design and usability standards. Additionally, this chapter includes a thorough analysis of the system's performance, particularly focusing on the efficiency of embedding generation and retrieval operations. Through rigorous testing, we aim to ensure that the application not only functions correctly and efficiently but also delivers a seamless user experience.

4.1 Screen Captures

Here are Screen Captures provided for the main pages we have in the system:

4.1.1 Registration

- Registration Page

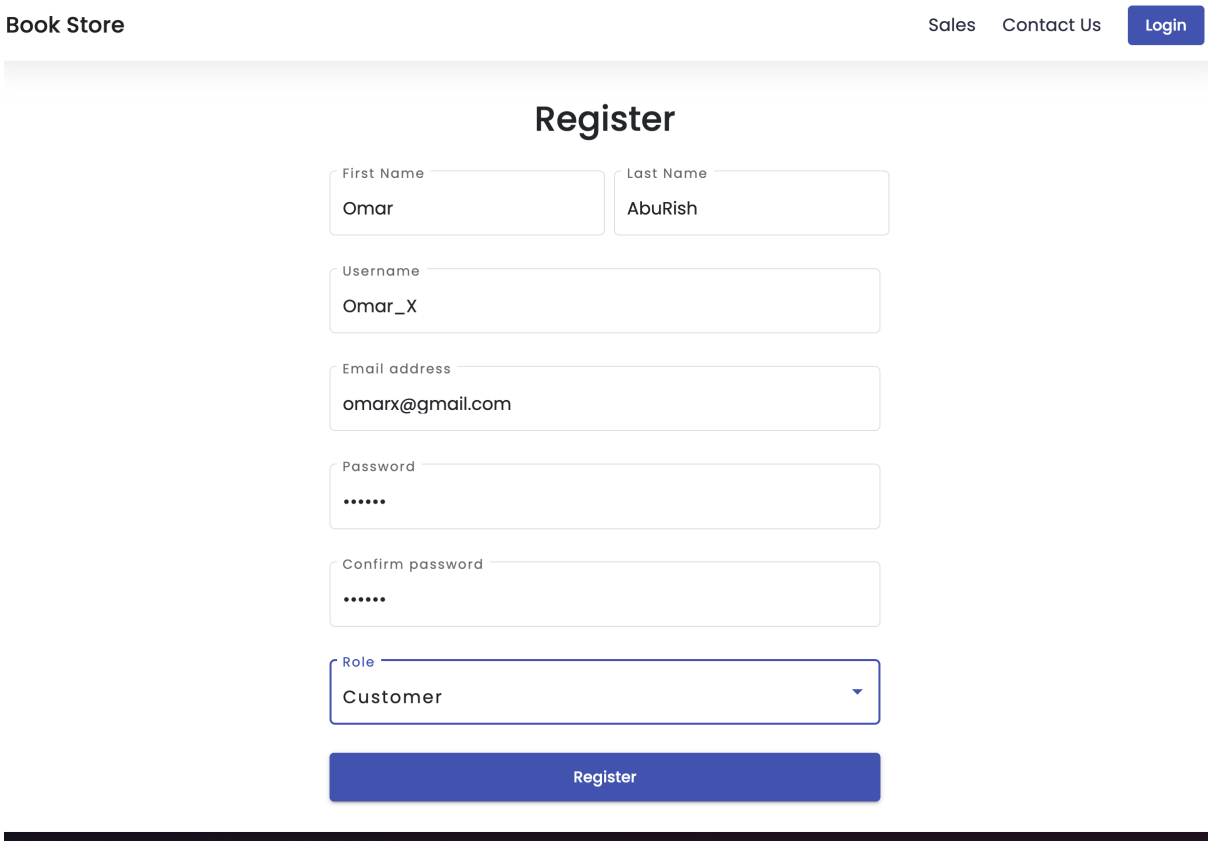

Figure 14: User Registration Page

- Login Page

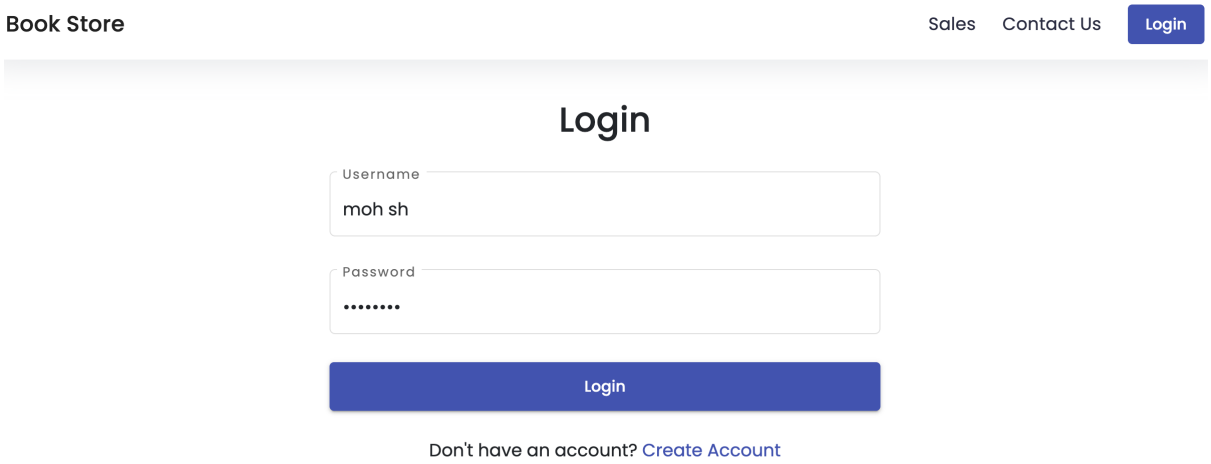

Figure 15: User Login Page

- Admin Home Page

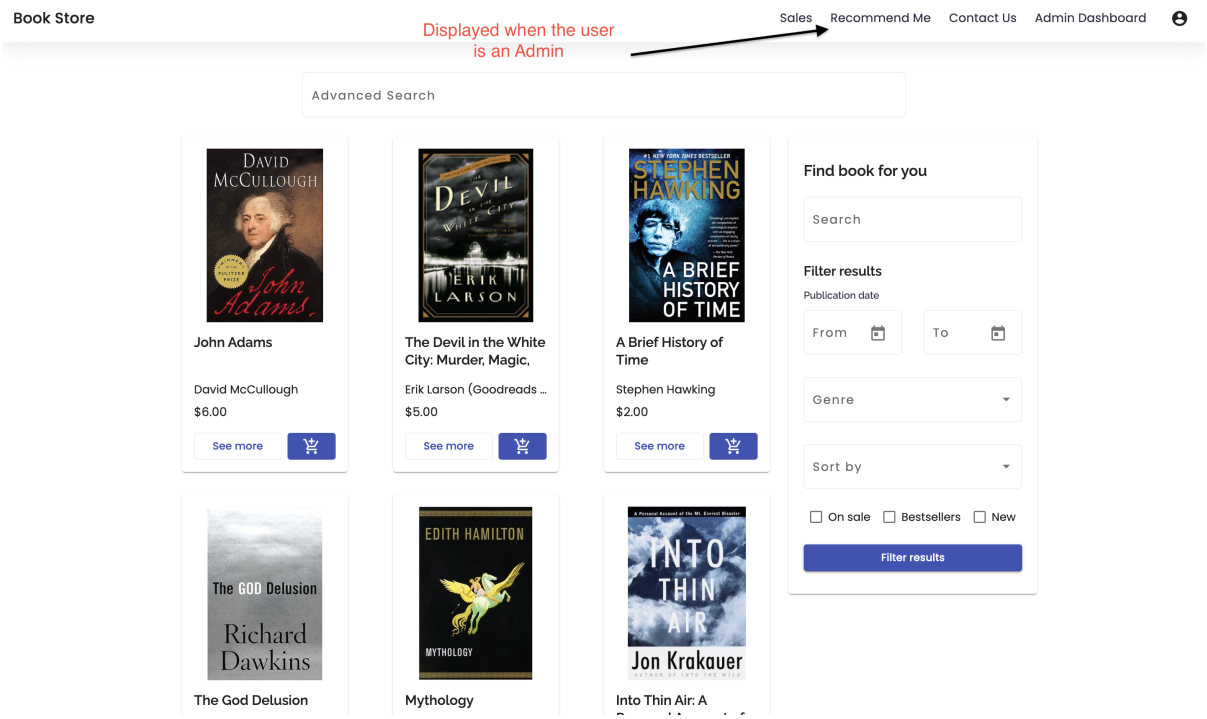

Figure 16: Admin Home Page

4.1.2 Main Pages

- Home Page

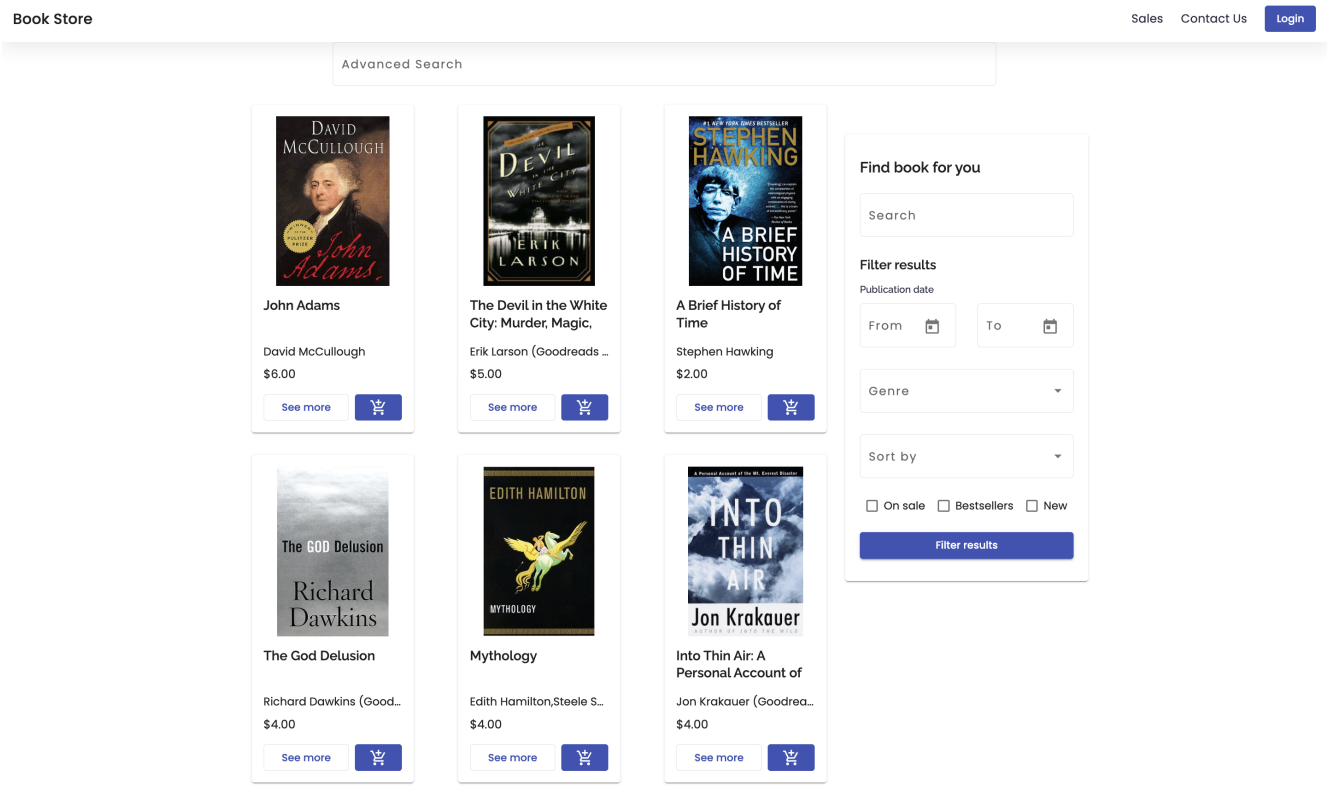

Figure 17: Book Store Home Page

- Advanced Search

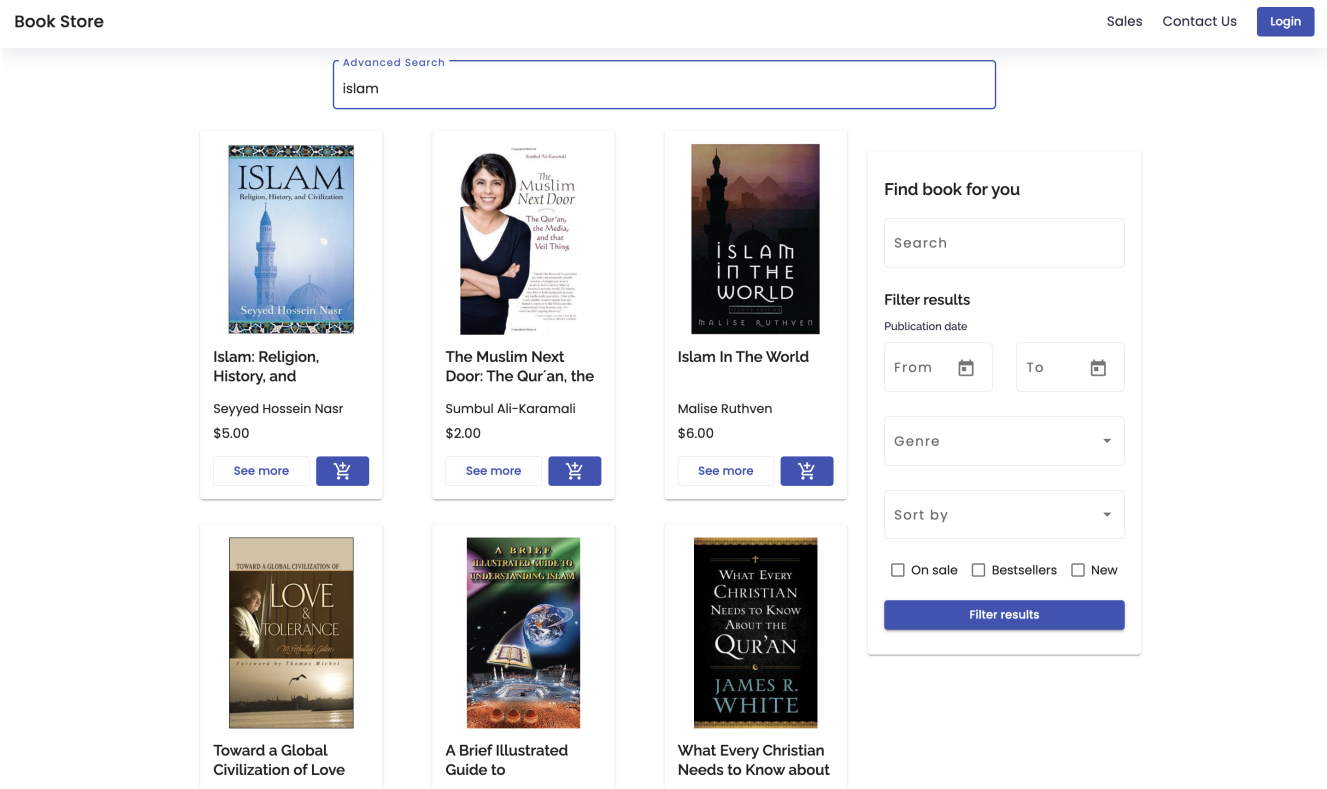

Figure 18: Advanced Search Page

- Recommend Me

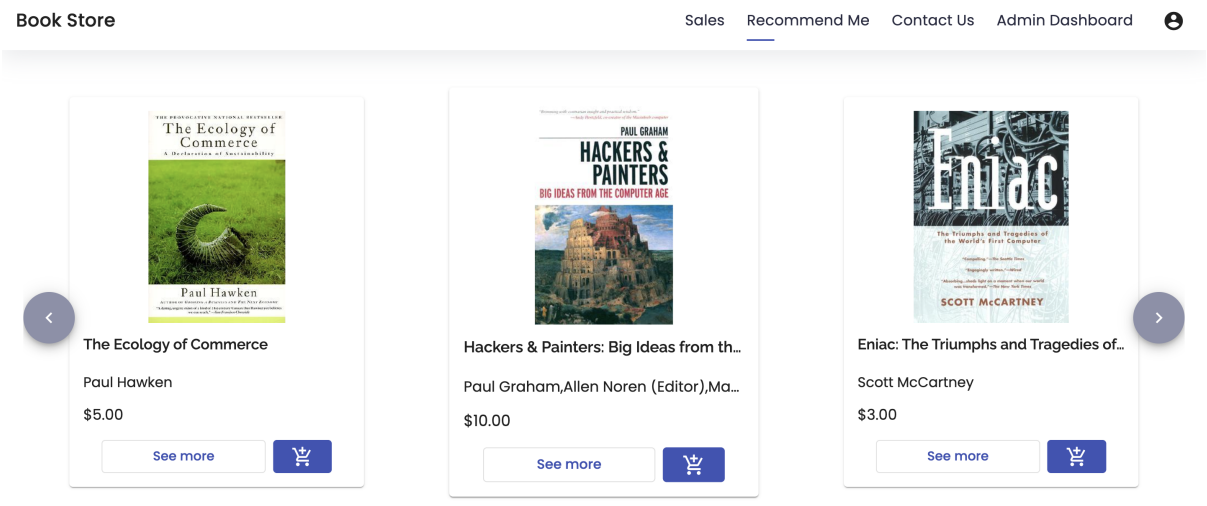

Figure 19: User Personally Recommended Books Page

- Book Preview

- Sales Page

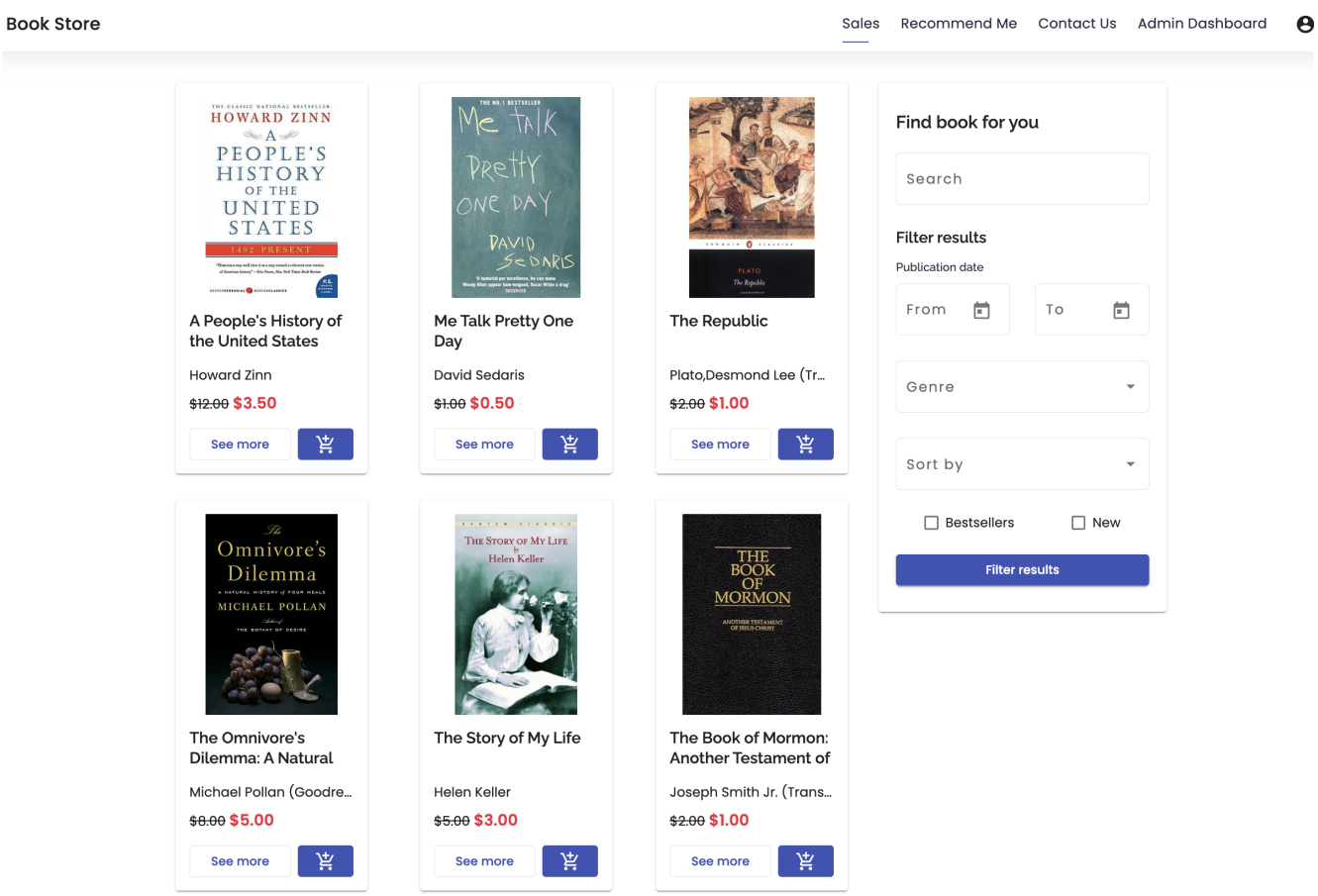

Figure 21: Sales Page

4.1.3 Profile

- Navigation

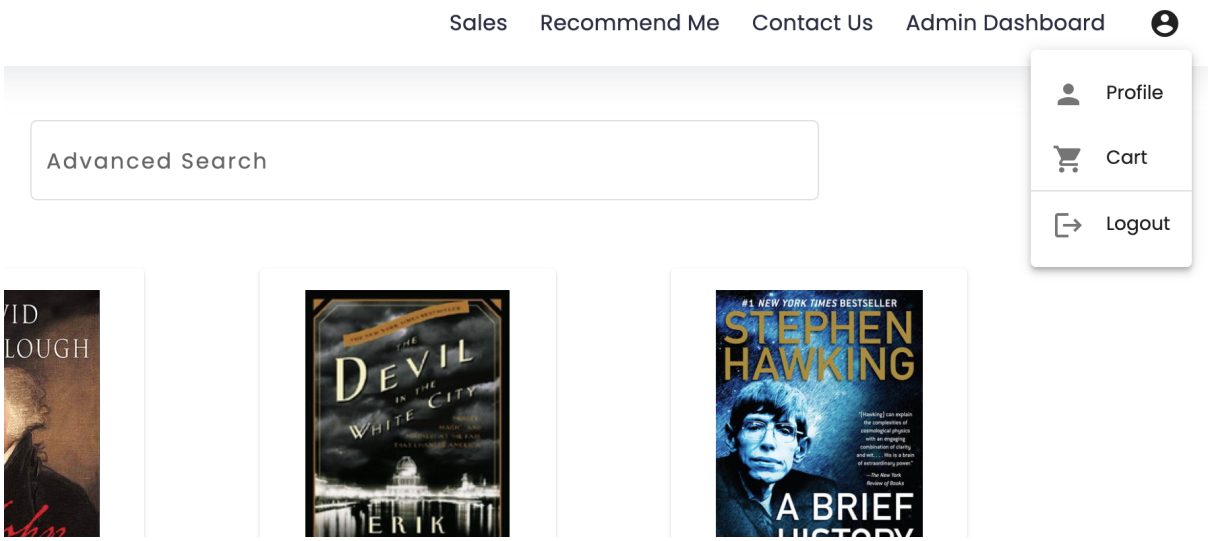

Figure 22: User Profile Navigation

- Profile

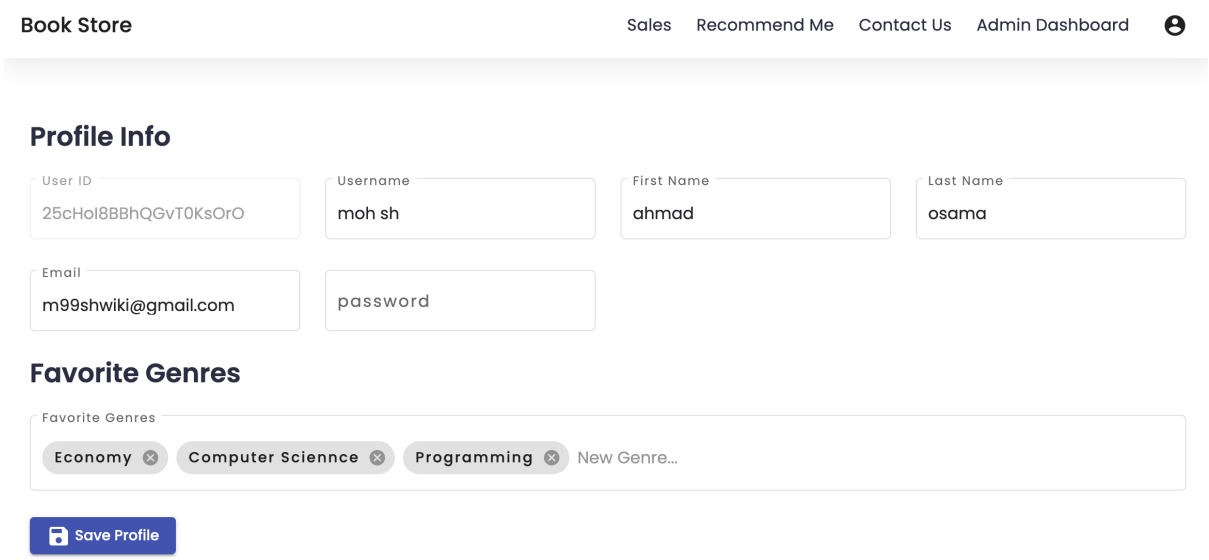

Figure 23: User Profile Page

- Profile Update

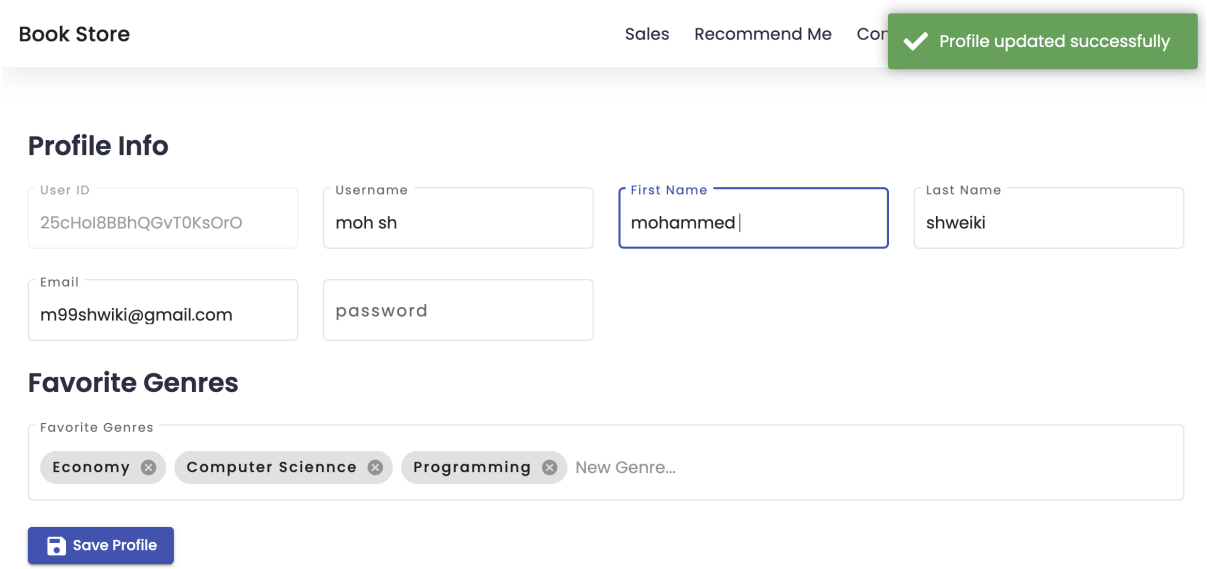

Figure 24: User Profile Updated Successfully

- Cart

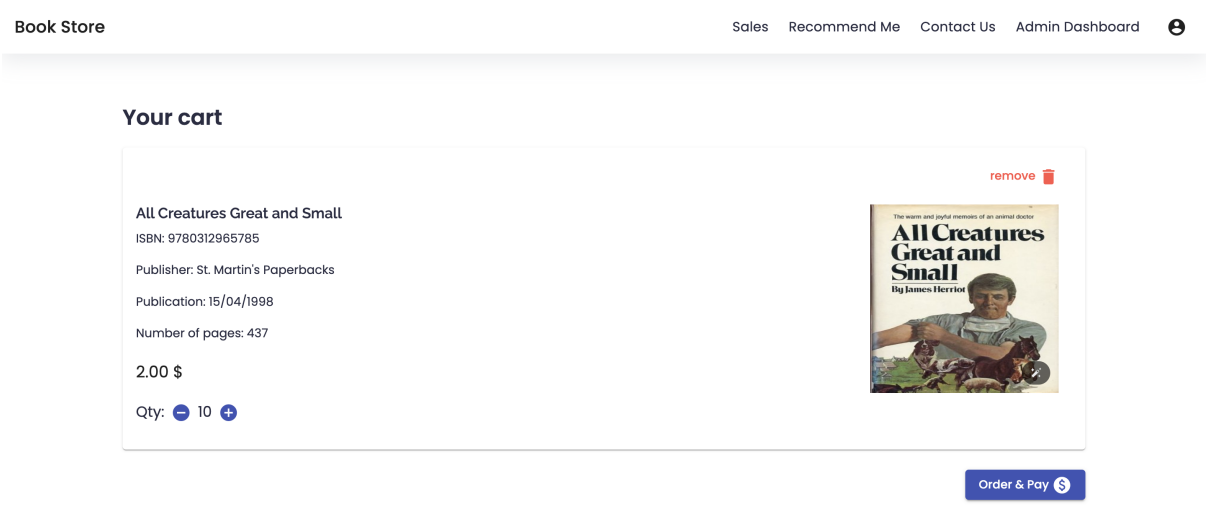

Figure 25: User Cart Page

4.1.4 Admin Dashboard

- Users

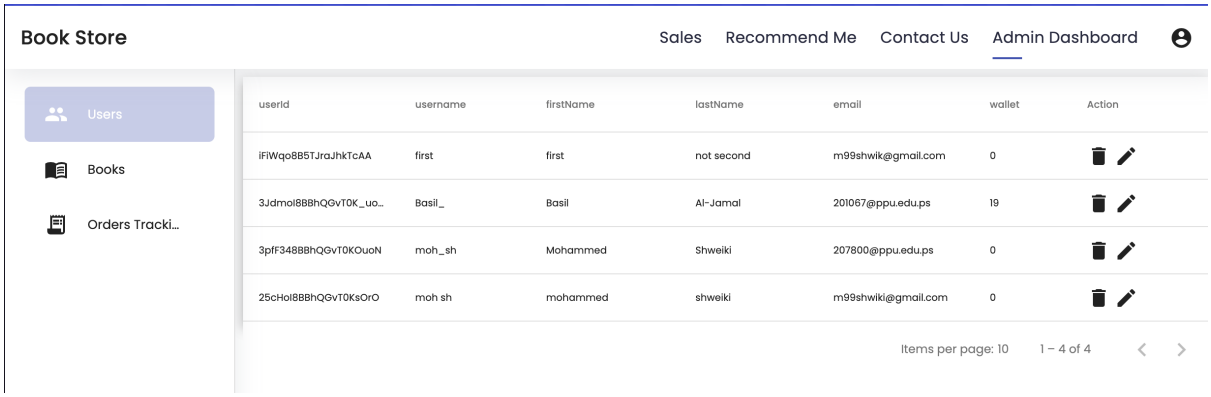

Figure 26: Admin Users View Page

- Books

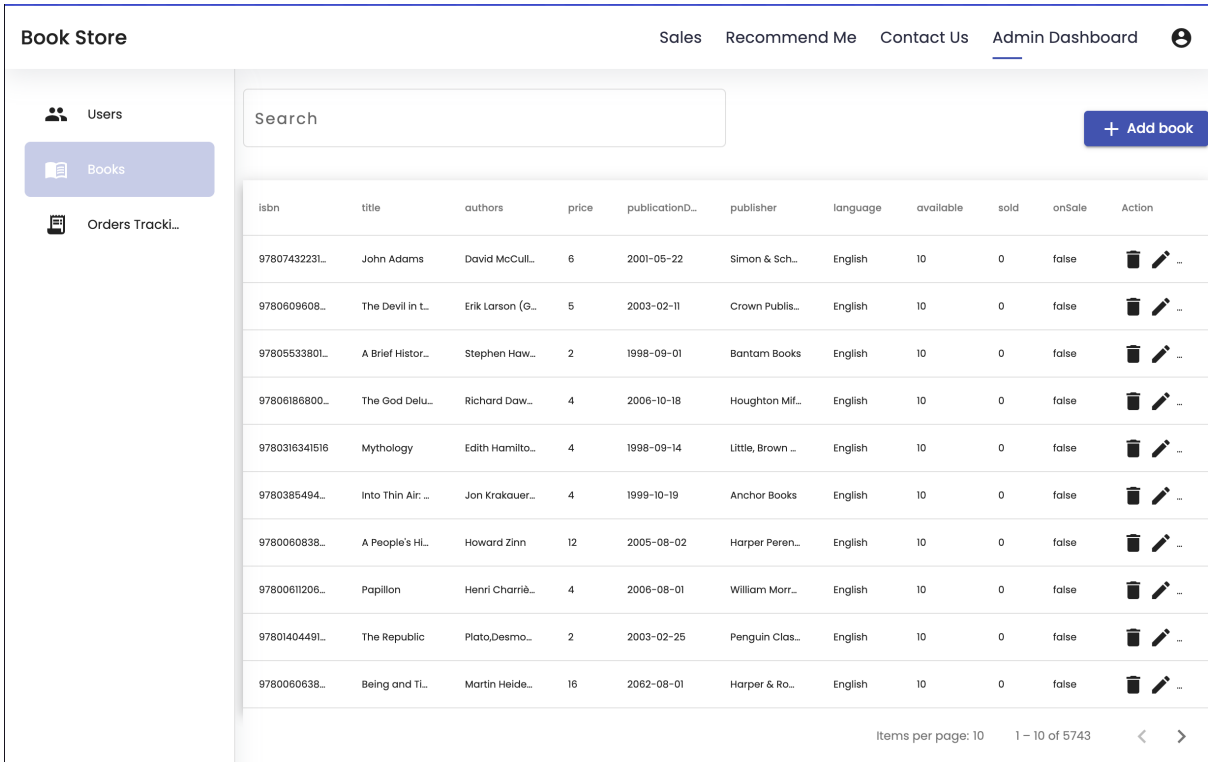

Figure 27: Admin Books View Page

- Orders Tracking

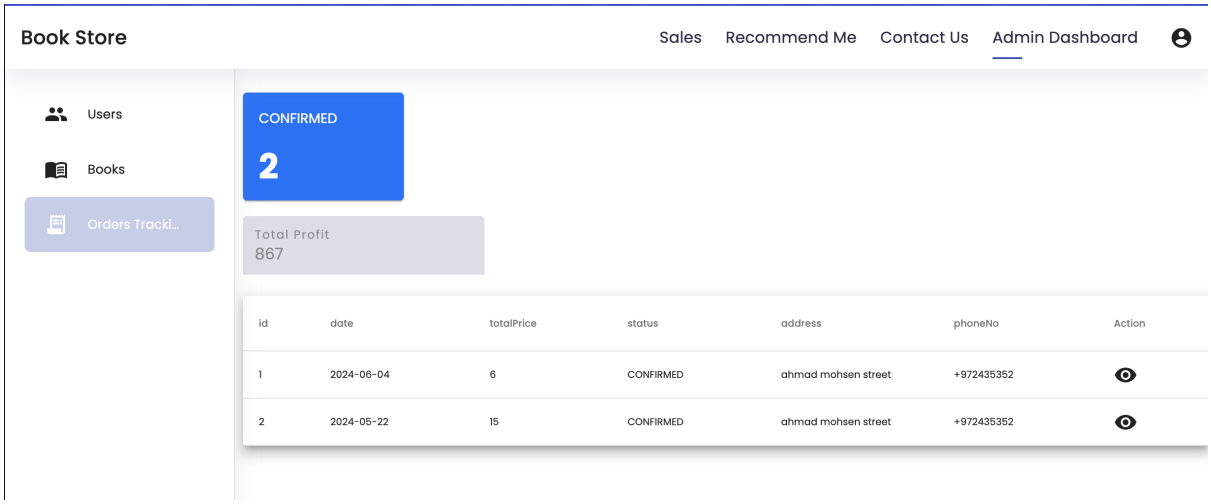

Figure 28: Admin System Orders Tracking Page

4.2 Performance Evaluation

4.2.1 Stress Testing

We conducted stress testing on various endpoints within the system to evaluate and measure their reliability under high load conditions. This process involved subjecting the endpoints to an increasing volume of requests to determine how well they perform and maintain functionality when pushed beyond their usual operational capacity. By doing so, we aimed to identify potential bottlenecks, ensure the robustness of the system, and verify that the endpoints can handle extreme workloads without failure.

- **Book Addition**:
  - **URL**: `http://book-store/books/`
  - **Method**: POST
  - **Description**: This endpoint allows the addition of a new book to the database. It processes the book data, generates embeddings for book's title and description using embedding method in the NLP microservice, and then stores the book in the database.
  - **Test Results**:

* **Total Requests**: 6184
* **Successful Requests**: 6181
* **Errors**: 3
* **Total Response Time**: 3363.068 seconds
* **Average Response Time**: 0.544 seconds

Test Results:

Table 10: Stress Test Results for Book Addition Endpoint

Total Requests	Total Response Time (s)	Average Response Time (s)	Errors
100	153.990	1.540	0
150	88.847	0.880	0
200	99.691	0.987	0
250	98.614	0.976	0
300	218.480	2.163	0
350	136.286	1.349	0
400	100.735	0.997	0
450	91.731	0.908	0
500	101.809	1.008	0
550	117.205	1.160	0

Continued on next page

Table 10 – *Continued from previous page*

Total Requests	**Total Response Time (s)**	**Average Response Time (s)**	**Errors**
600	104.054	1.030	0
650	94.613	0.937	0
700	72.494	0.718	0
750	89.631	0.887	0
800	104.344	1.033	0
850	100.801	0.998	0
900	120.977	1.198	0
950	104.158	1.031	0
1000	82.695	0.819	0

- **Note on Embedding Method**: This endpoint consumes an embedding method in the NLP microservice. The embedding method is applied to book's title and description to generate vector representations. These embeddings are used to enhance search and recommendation functionalities. After embedding, the book data is stored in the database.

Analysis: The stress test results indicate that the book addition endpoint handles concurrent requests efficiently, with an average response time of 0.544 seconds and only 3 errors encountered out of 6184 requests.

Conclusion: The book addition endpoint demonstrates reliable performance under stress conditions, with the majority of requests successfully processed within a reasonable time frame.

- **Advanced Search**
  - **URL**: `http://book-store/advanced-search?query="string"`
  - **Method**: POST
  - **Description**: This endpoint facilitates advanced search functionality by allowing users to search for specific items using various criteria. The search can be customized with query terms and filters to refine the results.

Test Results:

Analysis: The stress test results indicate that the advanced search endpoint handles concurrent requests efficiently up to 1000 requests without any errors. The average response time remains below 1 second for most increments, with a slight increase observed at 300 concurrent requests (2.163 seconds). Despite this increase, the system quickly stabilized, and response times returned to an acceptable range. No errors were encountered throughout the test, demonstrating the reliability of the endpoint under high load conditions.

Conclusion: The advanced search endpoint exhibits robust performance and reliability under stress conditions, handling up to 1000 concurrent requests efficiently. The system's ability to maintain low average response times and zero errors suggests that it is well-suited for high-traffic scenarios.

4.3 Summary

In summary, this chapter delves into the detailed implementation and rigorous testing of the bookstore application. It highlights the successful integration of a

Table 11: Stress Test Results for Advanced Search Endpoint

Total Requests	**Total Response Time (s)**	**Average Response Time (s)**	**Errors**
100	153.990	1.540	0
150	88.847	0.880	0
200	99.691	0.987	0
250	98.614	0.976	0
300	218.480	2.163	0
350	136.286	1.349	0
400	100.735	0.997	0
450	91.731	0.908	0
500	101.809	1.008	0
550	117.205	1.160	0
600	104.054	1.030	0
650	94.613	0.937	0
700	72.494	0.718	0
750	89.631	0.887	0
800	104.344	1.033	0
850	100.801	0.998	0
900	120.977	1.198	0
950	104.158	1.031	0
1000	82.695	0.819	0

user-friendly interface and the robust performance of key features under stress conditions. By presenting visual validations and performance metrics, we demonstrate the application's ability to handle high loads efficiently while ensuring a seamless and engaging user experience. This comprehensive evaluation underscores the reliability and effectiveness of the system in real-world scenarios.

5 Conclusion

5.1 Summary

In this project, we have developed a comprehensive bookstore application that seamlessly integrates a user-friendly frontend with a powerful backend, augmented by an AI-driven NLP microservice and an efficient database system. The application facilitates smooth user interactions for searching, adding, and recommending books, ensuring an enriching user experience. Through detailed requirements analysis, robust system design, and extensive testing, we have ensured the application's functionality, performance, and scalability. The system's architecture, utilizing Angular for the frontend, Java Spring Boot for the backend, FastAPI for the AI microservice, and Elasticsearch for data storage, demonstrates a well-balanced use of modern technologies to achieve our objectives. The performance evaluation confirmed the system's ability to handle various loads and scenarios effectively. Overall, this project showcases our ability to create a scalable, efficient, and user-centric application that meets both functional and non-functional requirements.

5.2 Future Work

To further enhance the recommendation system application, several avenues for future work can be explored:

5.2.1 Enhanced Personalization

Objective: Improve the recommendation system by incorporating more advanced machine learning algorithms and deeper user behavior analysis.

Approach: Implement deep learning techniques and real-time data processing to provide more accurate and personalized book recommendations.

5.2.2 Mobile Application Development

Objective: Extend the application's reach by developing native mobile applications for iOS and Android.

Approach: Use frameworks such as React Native or Flutter to create cross-platform mobile applications, ensuring a consistent user experience across all devices.

5.2.3 Improved Search Capabilities

Objective: Enhance the search functionality to support advanced query options and natural language processing.

Approach: Implement more sophisticated search algorithms and NLP techniques to understand and process complex user queries more effectively.

5.2.4 User Analytics Dashboard

Objective: Provide administrators with insights into user behavior and application usage through detailed analytics.

Approach: Develop an analytics dashboard that visualizes data such as user engagement, search patterns, and sales metrics, using tools like Kibana or Tableau.

5.2.5 Handling Arabic Books in Shopping and Recommendations

Objective: Expand the bookstore's inventory and recommendation system to include Arabic books.

Approach: Utilize NLP techniques tailored for Arabic text processing, such as Arabic word embeddings and sentiment analysis. Integrate Arabic language support in search functionalities and recommendation algorithms to cater to Arabic-speaking users effectively.

References

1. Alharthi, H. (2019). Natural language processing for book recommender systems [Master's thesis, University of Ottawa]. *ruor.uottawa.ca* (accessed June 2, 2024).

2. Berbatova, M. (2019, June). Overview on NLP techniques for content-based recommender systems for books. In *Proceedings of the Student Research Workshop at the Conference on Information Technology Education and Training* (pp. 232-239).

3. Kukreja, S., Kumar, T., Bharate, V., Purohit, A., Kapoor, A., Sharma, R., Singh, M., & Patel, A. (2023, May). Vector Databases and Vector Embeddings-Review. In *2023 International Conference on Electrical, Electronics, Communication, Computer, and Informatics Engineering (EECEE 2023)* (pp. 1-6). IEEE.

4. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics

5. Birunda, S. S., & Devi, R. K. (2021, February). A review on word embedding techniques for text classification. In *Innovative Data Communication Technologies and Security* (pp. 1-8). Springer.

6. Ottinger, J. B., & Lombardi, A. (2024). *Beginning Spring 6: From Beginner to Pro*. Springer.

7. Lü, L., Medo, M., Yeung, C. H., Zhang, Y. C., Zhang, Z. K., Zhang, X., Zeng, A., Zho, W., & Zhou, T. (2012). Recommender systems. *Physics Reports*, 515(1), 1-49.

8. Bhat, U., & Jadhav, S. (2010, January). Moving towards non-relational databases. *International Journal of Computer Applications*, 1(2).

9. Selfa, D. M., Carrillo, M., & Boone, M. D. R. (2006, June). A database and web application based on MVC architecture. In *2006 IEEE Conference on Electronics, Communications and Instrumentation (CECI)* (Vol. 2, pp. 1422-1427). IEEE.

10. Jorgensen, P. C. (2013). Software Testing: A Craftsman's Approach (4th ed.). CRC Press.