Bookipedia

An Interactive Reading Platform Using Artificial Intelligence

By:

Hoda Atya Ahmed

Nada Hossam El din Ali

Nayra Ashraf Hamdy

Youssef Mohamed Rashad

Ahmed Mohamed El-Hindawi

Mohamed Alaa Mohamed

Abdelhakiem Osama Eita

Mohamed Hazem Mohamed

Mohamed Emad El-Awady

Mustafa Ibrahim Abdelhameed

Under the supervision of

Dr. Hanaa ZainEldin

Eng. Manar Ahmed Hesham

Faculty of Engineering, Mansoura University
Mansoura, Egypt
2024

Bookipedia

An Interactive Reading Platform Using Artificial Intelligence

By:

Hoda Atya Ahmed

Nada Hossam El din Ali

Nayra Ashraf Hamdy

Youssef Mohamed Rashad

Ahmed Mohamed El-Hindawi

Abdelhakiem Osama Eita

Mohamed Hazem Mohamed

Mohamed Alaa Mohamed

Mohamed Emad El-Awady

Mostafa Ibrahim Abdelhameed

A Graduation Project Document Submitted in Partial Fulfillment of the Requirements for the BSc. Degree in Computer Engineering
(Computer Engineering and Control Systems)

Under the Supervision of:

Dr. Hanaa ZainEldin

Eng. Manar Ahmed Hesham

Faculty of Engineering, Mansoura University
Mansoura, Egypt

2024

Acknowledgement

We would like to express our deepest gratitude to all those who have supported and guided us throughout the journey of completing this graduation project.

First and foremost, We would like to thank our supervisors, Dr. Hanaa ZainEldin and Eng. Manar Ahmed, for their invaluable guidance, encouragement, and insightful feedback. Their expertise and support have been instrumental in shaping this project and ensuring its successful completion.

We are also grateful to the faculty members of CSE Department at Mansoura University for their continuous support and for providing a stimulating academic environment that fostered our growth and learning.

A special thank you to our industry partners and collaborators, including the publishers, technology providers, and academic institutions, whose cooperation and resources were crucial in the development of this project, especially Egyptian Academy of Scientific Research and Technology and CodeScalers-Egypt cloud provider.

We extend our heartfelt thanks to our fellow students and friends, especially Eng. Ahmed Salah, for his constant encouragement, constructive criticism, and for sharing this journey with us. Your companionship and support have made this experience both enriching and enjoyable.

We would also like to acknowledge our families for their unwavering support and belief in us. Your love and encouragement have been our source of strength throughout this endeavor.
Lastly, We are thankful to all those who contributed directly or indirectly to the successful completion of this project. Your support and encouragement have been invaluable.
Thank you all for making this journey a memorable and fulfilling one.

Table of Contents

List of Tables

List of Figures

Nomenclature

UI	User Interface
UX	User Experience
AI	Artificial Intelligence
RAG	Retrieval Augmented Generation
LLMs	Large Language Models
OCR	Optical Character Recognition
TTS	Text To Speech
HNSW	Hierarchical Navigable Small World
GANs	Generative Adversarial Networks
VAEs	Variational Autoencoders
CRUD	Create, Read, Update, Delete
SDK	Software Development Kit
HTML	HyperText Transfer Protocol
CSS	Cascading Style Sheets
I/O	Input / Output
BLoC	Business Logic Component
URL	Uniform Resource Locator
ARM	Advanced RISC Machine
JWT	JSON Web Token
ERD	Entity Relationship Diagram
API	Application Programming Interface
JSON	JavaScript Object Notation
VPS	Virtual Private Server

Abstract

In today's fast-paced digital world, the need for efficient and insightful reading tools has never been greater. This project presents the development of an innovative application designed to enhance the reading experience for users by leveraging advanced AI technologies. The application allows users to easily upload and read their books and documents, offering an AI-enhanced reading experience that is both smooth and user-friendly.

Key features of the application include the ability to highlight any text within a document to receive instant, referenced insights from both within and outside the text. This is complemented by an interactive AI chat feature, allowing users to delve deeper into any topic of interest directly within the reading interface. The platform is designed to be intuitive, responsive, and visually appealing, providing a comprehensive reading environment.

Our application stands out through its integration of Large Language Models (LLMs) for context-aware question answering, as well as intelligent in-document and web retrieval. This combination enables an immersive and efficient reading experience, allowing users to gain deeper insights and engage more fully with their reading material. By merging cutting-edge AI capabilities with a seamless user interface, this project aims to redefine the way users interact with digital texts, making reading more engaging and informative than ever before.

Chapter 1: Project Description

1.1 Introduction

In recent years, advancements in artificial intelligence and machine learning have revolutionized various aspects of our daily lives. From virtual assistants to personalized recommendations, AI technologies are increasingly becoming integral to the way we interact with digital content. One area that stands to benefit significantly from these advancements is digital reading. Traditional e-readers and document viewers have remained relatively static, focusing primarily on displaying text with limited interactivity or contextual support. However, as our consumption of digital content grows, so does the need for more sophisticated tools that can enhance our reading experience, making it more immersive, interactive, and insightful.

This chapter explores the potential of AI-driven enhancements to digital reading platforms. It addresses the current limitations of conventional e-readers and introduces innovative solutions designed to transform the way users engage with written content. By leveraging AI capabilities, these solutions aim to provide instant contextual insights, facilitate deeper understanding through interactive chats, and ensure a user-friendly interface that adapts seamlessly across devices. The subsequent sections will delve into the specific challenges faced by readers, outline the proposed services and requirements of the advanced reading platform, and define the expected outcomes that will ultimately shape the future of digital reading.

1.2 Problem Statement

In the current digital age, the way we consume and interact with written content has evolved significantly. Despite the abundance of digital reading materials available, existing reading platforms often fall short in providing an immersive and insightful reading experience. Traditional e-readers and document viewers primarily focus on displaying text, with limited interactive features and inadequate support for contextual understanding.

Readers frequently encounter challenges when they need to understand complex topics, verify information, or delve deeper into the context of the material. This often requires manually searching for additional information, which can be time-consuming and disruptive to the reading flow. Moreover, the lack of integrated tools for highlighting, referencing, and discussing content within the document further hampers the efficiency and engagement of the reading process.

The core problem is the absence of a comprehensive reading platform that seamlessly integrates advanced AI capabilities to enhance the reader's experience. Specifically, there is a need for an application that not only displays digital texts but also provides intelligent, context-aware insights and facilitates interactive learning. Such a platform should support the following functionalities:

1. **Instant Referenced Insights:** The ability to highlight any text and receive immediate, relevant information from within the document and external sources.
2. **Interactive AI Chat:** An AI-driven interactive chat feature to expand on queries and explore topics in greater depth.
3. **User-Friendly Interface:** An attractive, intuitive, and responsive interface that enhances user engagement and accessibility.

Addressing these challenges will create a more efficient, engaging, and insightful reading experience, transforming the way users interact with digital texts.

1.3 List of Services

1. **AI-Enhanced Reading:**
    a. Enable users to upload any document (PDFs, DOCX, etc.) for analysis.
    b. Leverage cutting-edge AI models to enhance the reading experience:
        i. Text Highlighting: Users can highlight text within the document.
        ii. Instant Insights: Upon highlighting, provide context-aware insights from both within and outside the document.
        iii. Interactive Chat: Allow users to engage in real-time conversations with our AI for deeper exploration.
2. **Comprehensive Platform:** Create an integrated reading platform that combines:
    a. Document Repository: Store and organize user-uploaded content.
    b. AI Engine: Process and analyze documents for insights.
    c. Chat Interface: Facilitate interactive discussions.
    d. Reference Retrieval: Fetch relevant external information.
3. **Responsive Design:**
    a. Ensure the platform adapts to various screen sizes and orientations.
    b. Prioritize responsiveness for an optimal reading experience.
4. **User-Friendly Interface:**
    a. Develop an interface that users can navigate effortlessly across devices (mobile and web).
    b. Prioritize readability, aesthetics, and ease of use.
5. **Cross-Device Syncing:**
    a. Seamless synchronization of documents, highlights, and annotations across multiple devices.
    b. Cloud storage support for access to reading materials anywhere, anytime.

1.4 Requirements

1.4.1 Functional Requirements

User requirements:

1. The user can upload various document formats
2. Simple way for User Authentication and login
3. Users need a simple User Interface (intuitive and attractive interface for both mobile and web.Prioritize readability, ease of navigation, and aesthetics.)
4. Users can highlight specific text within documents and Upon highlighting, the system provides instant insights in document and in External References
5. Users can engage in conversation with an AI chatbot to provide additional context, offer deeper insights and ask it more about the book context to help him understand it easier.
6. Users can search for specific documents within their library and create their own categories

System requirements:

1. **Document Upload and Storage:**
2. **The system shall accept PDF files**
3. **Uploaded documents shall be securely stored and associated with the user's account.**
4. **The system shall validate uploaded files to ensure they adhere to the specified formats.**
5. **In case of invalid or unsupported formats, the system shall provide appropriate error messages to users.**
6. **Text Highlighting and Insights:**
    a. Text Highlighting:
        i. Users shall be able to select and highlight specific portions of text within documents.
        ii. The system shall visually indicate the highlighted text (e.g., by changing its background color).
    b. Instant Insights: When users highlight text, the system shall analyze the content and provide relevant insights. These insights may include:
        i. Definitions of key terms.
        ii. Contextual explanations.
        iii. Related concepts.
        iv. Statistical data.
        v. Sentiment analysis (if applicable).
        vi. Suggestions for further reading.

c. External References: Alongside instant insights, the system shall offer external references or links to authoritative sources. These references may include:
    i. Hyperlinks to relevant articles, research papers, or websites.
    ii. Citations from reputable sources.
    iii. Cross-references to related documents within the system.
d. User Interaction:
    i. Users shall trigger the insights and external references by actively highlighting text.
    ii. The system shall display insights in a non-intrusive manner (e.g., tooltips, sidebars, or pop-ups).
e. Security and Privacy:
    i. The system shall handle user-highlighted content securely and not store sensitive information.
    ii. External references shall be vetted to ensure reliability and accuracy.

7. Interactive Chat with AI:

a. Chatbot Interaction:
    i. Users shall be able to initiate conversations with the AI chatbot.
    ii. The chatbot shall respond promptly and engage in a natural dialogue.
b. Contextual Input:
    i. Users can provide context related to the book they are reading. This context may include:
        1. Book title.
        2. Author.
        3. Genre.
        4. Specific chapters or sections.
        5. Personal interpretations or reflections.
c. Deeper Insights:
    i. When users share context, the chatbot shall offer deeper insights into the book. These insights may include:
        1. Literary analysis.
        2. Historical context.
        3. Thematic exploration.
        4. Character motivations.
        5. Symbolism.
        6. Critical perspectives.
d. Clarifications:
    i. Users can ask the chatbot specific questions about the book. The chatbot shall provide clear and concise answers to enhance comprehension.
e. User-Friendly Experience:
    i. The chatbot's responses shall be user-friendly, avoiding jargon or overly technical language.

f. Integration with Book Content:
    i. The chatbot shall reference specific parts of the book when providing insights or clarifications.
    ii. If available, the chatbot may link to relevant pages or chapters within the book.
g. Privacy and Security:
    i. User interactions with the chatbot shall be confidential.
    ii. The chatbot shall not store or share personal information.

8. Responsive User Interface:

a. Minimalistic Design:
    i. The UI shall follow a clean and minimalistic design approach.
    ii. Extraneous elements, clutter, and unnecessary features shall be avoided.
b. Clear Navigation:
    i. Users shall be able to find their way around the system without confusion.
    ii. Navigation menus, buttons, and links shall be logically organized and labeled.
c. Consistent Layout:
    i. UI components (buttons, forms, icons, etc.) shall maintain consistent placement and appearance.
    ii. Consistency across different screens or modules shall be maintained.
d. Intuitive Controls:
    i. Buttons, checkboxes, input fields, and other controls shall behave as expected.
    ii. Users shall easily understand how to interact with each control.
e. Responsive Design:
    i. The UI shall adapt gracefully to different screen sizes (e.g., desktop, tablet, mobile).
    ii. Responsiveness shall ensure usability across various devices.
f. Error Handling:
    i. Clear error messages shall be displayed when users encounter issues (e.g., invalid input, network errors).
    ii. Error messages shall guide users toward corrective actions.
g. Accessibility:
    i. The UI shall meet accessibility standards to accommodate users with disabilities.
    ii. Keyboard navigation, alt text for images, and proper color contrast shall be implemented.
h. User Feedback:
    i. The UI shall provide visual feedback (e.g., loading spinners, success indicators) during interactions.
    ii. Users shall receive confirmation messages for completed actions.

i. Help and Documentation:
    i. the UI shall offer contextual help or tooltips.
    ii. A concise user guide or documentation shall be available for reference.
j. User Preferences:
    i. Users may customize certain UI elements (e.g., theme, font size) based on personal preferences.

9. User Authentication and Profiles:

a. User Authentication:
    i. Users shall be able to create accounts and log in securely.
    ii. Authentication methods may include:
        1. Email and password.
        2. Social login (e.g., Google, Facebook).
    iii. Failed login attempts shall be limited to prevent brute-force attacks.
    iv. Passwords shall be hashed and stored securely.
b. User Profiles:
    i. Each user shall have a profile associated with their account.
    ii. Profile information may include:
        1. Full name.
        2. Email address.
        3. Profile picture.
    iii. Users shall be able to update their profiles.
c. Access Control:
    i. The system shall enforce role-based access control (RBAC).
    ii. Roles (e.g., admin, user) shall determine access to specific features or resources.
    iii. Admins may have additional privileges (e.g., user management).
d. Session Management:
    i. User sessions shall be maintained securely.
    ii. Sessions shall expire after a specified period of inactivity.
    iii. Users shall be automatically logged out upon session expiration.
e. Forgot Password Flow:
    i. Users who forget their passwords shall be able to reset them via email
    ii. The system shall verify the user's identity before allowing password reset.
f. Profile Privacy Settings:
    i. Users shall control the visibility of their profile information (public, private, friends-only).
    ii. Privacy settings shall apply to profile pictures, bio, and other details.
g. Profile Customization:
    i. Users may customize their profiles by adding additional information (e.g., interests, location).
    ii. Customization options shall be user-friendly.

h. Profile Picture Upload:
    i. Users shall be able to upload profile pictures.
    ii. The system shall validate image formats and size.
i. User Deactivation:
    i. Admins shall have the ability to deactivate or suspend user accounts.
    ii. Deactivated users shall not be able to log in.
j. Audit Trail:
    i. The system shall maintain an audit trail of user authentication and profile-related actions (e.g., login, profile updates).

10. Document Retrieval and Search:

a. Document Search:
    i. Users shall be able to search for documents based on keywords, titles, or other relevant criteria.
    ii. The search feature shall provide relevant results quickly and accurately.
    iii. Search results shall display document names, metadata, and a brief preview.
b. Advanced Search Options:
    i. Users may refine their search using filters (e.g., date range, file type, author).
    ii. Advanced search options shall be user-friendly and intuitive.
c. Custom Categories:
    i. Users shall create their own categories or folders to organize documents.
    ii. Each category may have a name and description.
    iii. Categories can be nested (subcategories within main categories).
d. Document Tagging:
    i. Users may tag documents with relevant keywords or labels.
    ii. Tags shall facilitate quick categorization and retrieval.
e. Category Management:
    i. Users shall add, edit, or delete categories as needed.
    ii. Categories shall be sortable and rearrangeable.
f. Document Association:
    i. Users shall associate documents with one or more categories.
    ii. A document may belong to multiple categories simultaneously.
g. User-Friendly Interface:
    i. The UI for search and category management shall be intuitive.
    ii. Clear instructions or tooltips shall guide users through the process.
h. Search Performance:
    i. The system shall optimize search queries for efficiency.
    ii. Indexing and caching mechanisms may be employed to enhance search speed.

i. Security and Privacy:
    i. User-specific search results and categories shall be private.
    ii. Documents shall be accessible only to authorized users.
j. Cross-Platform Support:
    i. The search and category features shall work seamlessly across different devices (e.g., desktop, mobile, web).

1.4.2 Non-Functional Requirements

1. **Performance and Responsiveness:**
    a. Fast response times for document uploads, highlighting, and chat interactions.
    b. Optimize for low latency to enhance user experience.
2. **Security and Privacy:**
    a. Implement robust data encryption during transmission and storage.
    b. Safeguard user data and prevent unauthorized access.
3. **Scalability and Load Handling:**
    a. Design the system to handle increasing user traffic.
    b. Consider load balancing and auto-scaling mechanisms.
4. **Compatibility and Accessibility:**
    a. Ensure compatibility across different devices and browsers.
    b. Prioritize accessibility for users with disabilities.
5. **Reliability and Availability:**
    a. Minimize system downtime.
    b. Implement backup and recovery mechanisms.
6. **Usability and User Experience:**
    a. Conduct usability testing to ensure an intuitive and smooth user journey.
    b. Provide clear instructions and tooltips.
7. **Maintainability and Extensibility:**
    a. Develop clean, well-documented code.
    b. Plan for future enhancements and feature additions.
8. **Integration with AI Models:**
    a. Seamlessly integrate context-aware question answering (LLM) and referenced retrieval.
    b. Ensure efficient communication between the reading platform and AI components.

1.5 Expected Outcomes

Outcomes provide a tangible measure of the project's success. They define the specific results that need to be achieved and establish clear expectations for the project team. In the following section we will list what we expected from Bookipedia at the end:

1. Develop an Intuitive and User-Friendly Reading Platform.
2. Integrate Advanced AI Capabilities.
3. Enhance Reader Engagement and Learning.
4. Improve Information Retrieval and Accessibility.
5. Enable Efficient Document Management.
6. Deliver a Comprehensive Reading Solution.

1.6 Business Model Canvas

The Business Model (Figure 1.1) Canvas is a strategic management template used for developing new business models and documenting existing ones.It offers a visual chart with elements describing a firm's or product's value proposition, infrastructure, customers, and finances, assisting businesses to align their activities by illustrating potential tradeoffs.

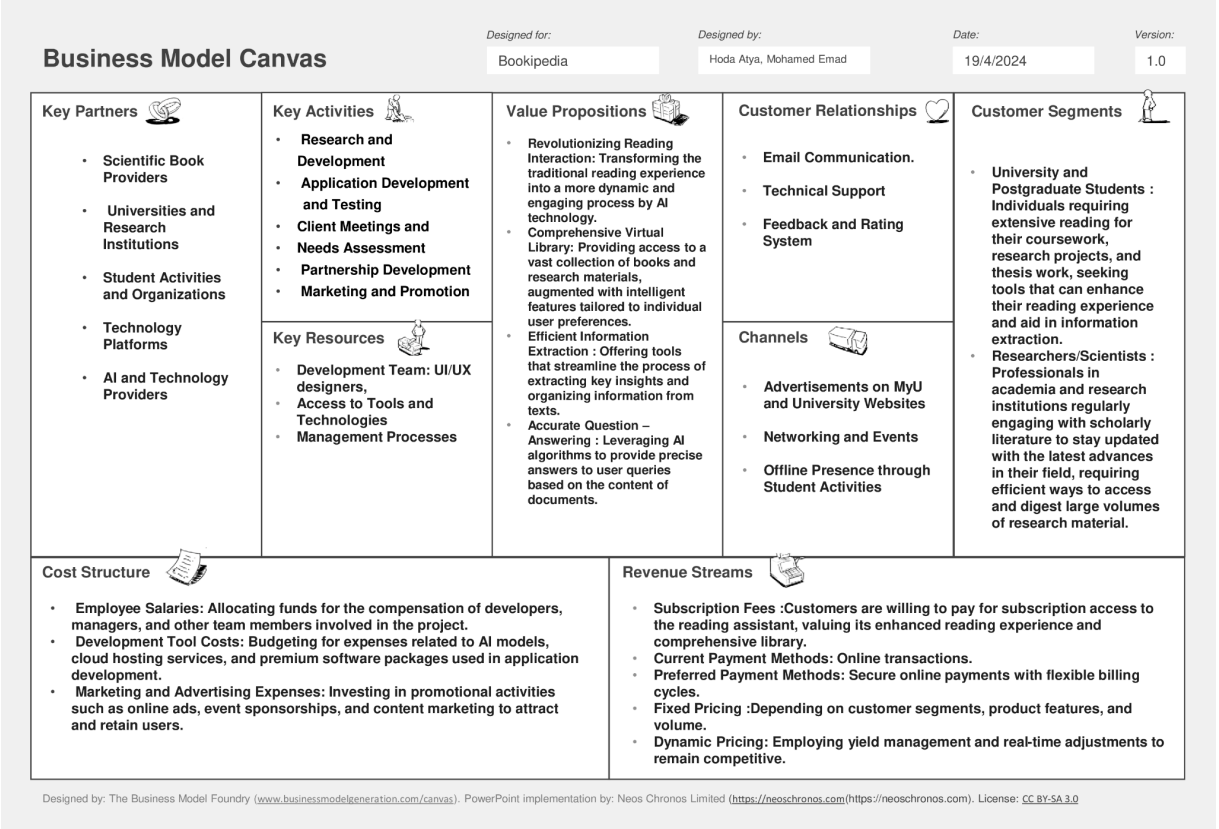

Figure 1.1: Bookipedia Business Model Canvas

Chapter 2: Building Blocks: Tools and Technologies

2.1 Introduction

The purpose of this chapter is to provide an in-depth understanding of the building blocks, tools, and technologies utilized in our graduation project, Bookipedia, which aims to develop an application that provides an insightful, interactive, and highly effective reading experience. While the previous chapter has briefly touched upon the project's description and goal, this section will focus on the high-level architecture and the main blocks that constitute our solution.

To provide a robust solution and a seamless, effective reading and studying user experience, numerous critical architectural decisions were made in selecting and integrating the system's components. The architecture diagram (Figure 2.1) illustrates the system architecture, showing the interconnected components and their functionalities.

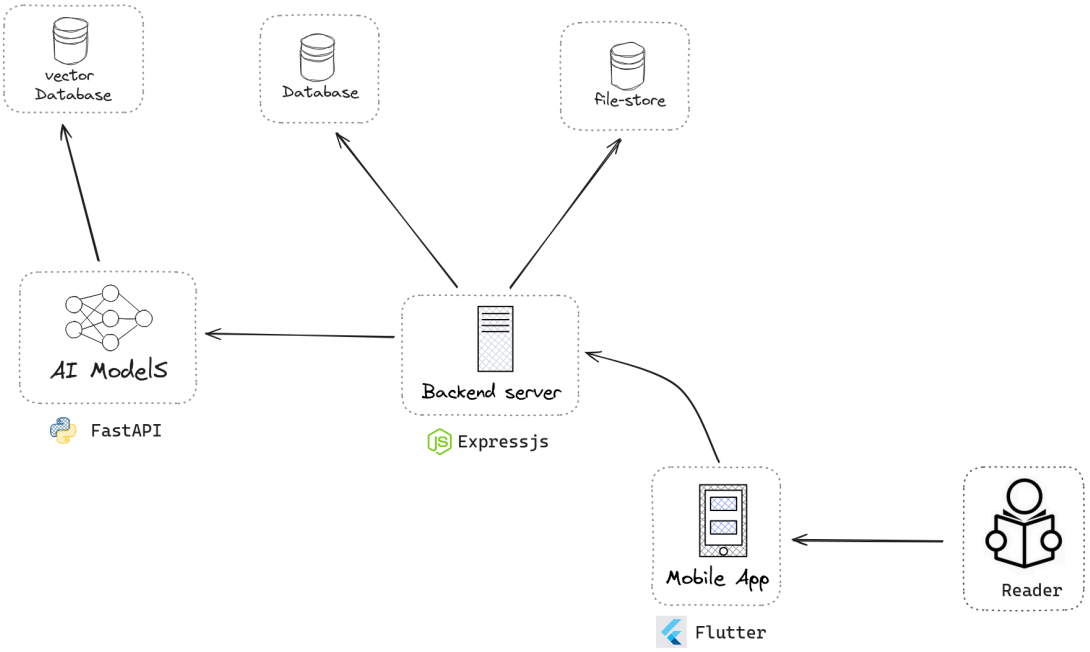

Figure 2.1: High-Level Architecture

This architecture comprises several key components:

User Interaction:

- **Mobile App**: Built with Flutter, the mobile app allows users to upload PDF documents from their devices, access a library of books, and add books to their personal bookshelves.
- **Reader Interface**: Users can interact with the app to read, ask questions and receive referenced insightful answers in chat, listen to text-to-speech, and get text summaries.

Backend Server:

- **ExpressJS**: Acts as the central hub, handling requests from the mobile app and coordinating with various services. It manages user interactions, processes data, and orchestrates communication between different components.

AI Models:

- **FastAPI**: The AI models, served via FastAPI, enable users to ask questions and receive insightful answers with references from both the book/document and the web. They also provide text-to-speech conversion and text summarization. Additionally, these models include OCR capabilities to handle scanned documents and convert them into textual formats, allowing seamless interaction with various document types.

Data Storage:

- **Vector Database:** Maintains embeddings and vectorized representations of documents for efficient retrieval of information based on user queries.
- **Database:** Holds user data, book metadata, and other structured information.
- **File-Store:** Manages cloud document storage, enabling users to upload, save, and access their files across devices.

By seamlessly integrating these components, Bookipedia ensures a cohesive and enhanced reading experience. This high-level architecture provides a comprehensive overview of our solution's structure. The subsequent sections of this chapter will delve into each building block, detailing the decisions made and the tools and technologies employed.

2.2 UI/UX Design

UI/UX Design plays a crucial role in creating an intuitive and engaging user experience for our application. This section explores the tools and technologies employed in the design process, highlighting their strengths and weaknesses.

2.2.1 Adobe XD

Adobe XD is a popular design and prototyping tool known for its user-friendly interface and powerful features. It offers a comprehensive set of tools for creating interactive prototypes, wireframes, and visual designs.

Pros of Adobe XD:

- Robust prototyping capabilities, allowing for interactive and dynamic user experiences.
- Seamless integration with other Adobe Creative Cloud applications, facilitating asset sharing and collaboration.
- Extensive library of UI kits and plugins, enabling rapid design iterations. Cons of Adobe XD:
- Limited collaboration features in the free version, requiring a subscription for full team collaboration.
- Slower performance when working with complex designs or larger projects.
- Smaller community compared to some other design tools, resulting in fewer online resources and community support.

2.2.2 Figma

Figma is a cloud-based design tool known for its collaborative features and versatility. It allows multiple designers to work simultaneously on the same project, fostering seamless collaboration and version control.

Pros of Figma:

- Real-time collaboration, enabling multiple team members to work on the same design project simultaneously.
- Cross-platform compatibility, as Figma is browser-based and can be accessed on any operating system.
- Robust design component system, facilitating reusable and consistent design elements.
- Extensive plugin ecosystem, providing additional functionality and customization options.

Cons of Figma:

- Performance issues may arise when working with very large design files or slower internet connections.
- Limited offline functionality compared to desktop-based design tools.
- The learning curve may be slightly steeper for designers transitioning from other design tools.

2.2.3 Why Figma over Adobe XD

After careful consideration, we chose to utilize Figma as our primary design tool due to its collaborative features and cross-platform compatibility. The ability for multiple team members to work together in real time greatly enhanced our design workflow and facilitated efficient communication. Furthermore, Figma's extensive plugin ecosystem allowed us to customize and extend its functionality based on our project's unique requirements. While Adobe XD is a powerful design tool, Figma's collaborative capabilities and platform flexibility ultimately aligned better with our project's collaborative design approach.

2.3 Mobile Application Development

Mobile development is a fundamental aspect of our application, allowing us to deliver an interactive and accessible experience to users. This section explores the decision-making process, trade-offs, and the tools and technologies employed in mobile app development.

2.3.1 Decision Paradigms and Trade-offs

During the planning and development phases, several decision paradigms and trade-offs were considered to select the most suitable framework for our mobile app development. Factors such as development efficiency, performance, cross-platform compatibility, and community support were considered.

2.3.2 Tools and Technologies

In our mobile development process, we utilized the following tools and technologies:

Flutter

The core framework used to develop our mobile app, provides a comprehensive set of tools and libraries for UI development and app logic.
Advantages include cross-platform compatibility, fast development cycles, and a rich set of UI capabilities. However, the framework may have a slightly larger app size compared to native development, and certain platform-specific features may require additional configuration.

Dart

The programming language used with Flutter, known for its simplicity, performance, and robust typing system.
Dart's simplicity, performance, and strong typing system contributed to our development efficiency. However, as it is less popular compared to some other programming languages, finding extensive resources and community support may sometimes be challenging.

2.3.3 Why Flutter

After careful evaluation, we chose Flutter as the framework for our mobile app development. Flutter is an open-source UI toolkit developed by Google that allows us to build natively compiled applications for multiple platforms from a single codebase. Here are some of the reasons behind our choice:

1. **Cross-platform development:** Flutter enables us to develop a single codebase that can be deployed on both Android and iOS platforms, saving development time and effort.
2. **Fast development and hot-reload:** Flutter's hot-reload feature allows for quick iteration and experimentation during the development process, resulting in faster development cycles.
3. **Rich UI capabilities:** Flutter provides a rich set of pre-designed widgets and customizable UI elements, enabling us to create visually appealing and interactive interfaces.
4. **Strong community support:** Flutter has a vibrant and growing community, offering extensive resources, packages, and libraries that enhance development productivity.

2.4 Back-End Development

The backend development aspect of our application plays a critical role in handling the logic and data processing required for a seamless user experience. In this section, we will delve into the decision-making process, trade-offs, and the chosen backend framework for our project.

2.4.1 Decision Paradigms and Trade-offs

When deciding on the backend framework for our project, we evaluated several available options, considering factors such as programming language, architectural pattern, community support, security, scalability, and performance. Each framework has its own advantages and trade-offs, and the decision-making process involves weighing these factors against our project's specific needs.

2.4.2 Available Backend Frameworks

In our evaluation process, we considered various backend frameworks, including Django (Python), Ruby on Rails (Ruby), and Express.js (JavaScript). Each framework offers its own set of features and benefits:

Django (Python)

Django is a high-level Python web framework that follows the Model-View-Controller (MVC) architectural pattern. It provides a robust set of tools and libraries for rapid development, security, and scalability. Django's batteries-included approach, extensive documentation, and active community make it a popular choice for backend development.

Ruby on Rails (Ruby)

Ruby on Rails is a full-stack web application framework written in Ruby. It follows the Convention over Configuration (CoC) principle and emphasizes developer productivity and simplicity. Ruby on Rails provides a streamlined development experience, focusing on convention-based development and code readability.

Express.js (JavaScript)

Express.js is a minimalist web application framework for Node.js, written in JavaScript. It follows a lightweight and flexible approach, allowing developers to create scalable and efficient backend systems. Express.js provides a wide range of middleware and modules, making it highly customizable and suitable for building RESTful APIs.

2.4.3 Why We Have Chosen Express.js

1. **Simplicity and Minimalism**:
    a. **Lightweight**: Express is minimalistic and un opinionated, meaning it doesn't impose any specific project structure or dependencies. This gives developers more freedom and flexibility in how they build their applications.
    b. **Simple to Set Up**: Getting an Express server up and running is quick and easy, which is beneficial for prototyping and small projects.
2. **Performance**:
    a. **Fast**: Express is built on Node.js, which is known for its non-blocking, event-driven architecture. This makes Express applications capable of handling a large number of simultaneous connections with high throughput.
3. **Middleware**:
    a. **Extensibility**: Express uses a middleware pattern, allowing developers to add functionality to the request/response pipeline.
4. **Routing**:

   a. **Flexible Routing**: Express offers a robust routing mechanism that allows developers to define routes for their application in a clean and organized manner. This is especially useful for creating RESTful APIs.
5. **Community and Ecosystem:**
   a. **Large Community:** As one of the most popular Node.js frameworks, Express has a large and active community. This means a wealth of tutorials, documentation, and community support.
   b. **Ecosystem:** There are numerous libraries, tools, and frameworks built around Express that can help speed up development. This includes tools for testing, debugging, and performance monitoring.
6. **Compatibility:**
   a. **Integration with Other Tools:** Express works well with various databases (like MongoDB, MySQL, PostgreSQL), view engines (like EJS, Pug), and other Node.js packages.
   b. **Microservices and APIs:** It's well-suited for creating microservices and APIs, making it a good choice for modern, scalable applications.
7. **Support for Latest JavaScript Features:**
   a. **ES6/ES7 and Beyond:** Being built on Node.js, Express supports the latest JavaScript features, enabling developers to write modern, clean, and efficient code.

2.4.4 Databases

In our backend development, we needed to choose a suitable database system to handle the storage and retrieval of data. Databases can be categorized into several types based on their underlying data models and the way they store and manage data.
The common types used in graduation projects:

1. **Relational Database:**
   Store data in tables with rows and columns. Tables can be related to each other through foreign keys.
   Examples: MySQL, PostgreSQL, SQLite, Oracle, Microsoft SQL Server.
   Advantages: Strong ACID (Atomicity, Consistency, Isolation, Durability) compliance, SQL query language support, well-suited for complex queries and transactions.

2. **NoSQL Databases:**
   a. These databases are designed to handle large volumes of unstructured, semi-structured, or structured data. One of its types is document store like MongoDB which was our chosen in our application for the following reasons:
      i. **Performance:**
         1. **Efficient Data Retrieval:** Document stores can retrieve entire documents in a single read operation, which can be faster than joining multiple tables in a relational database.

2. **Indexing:** MongoDB supports various indexing techniques, including single field, compound, geospatial, and full-text search indexes, which improve query performance.

ii. **Developer Productivity:**
   1. **Ease of Use:** JSON-like documents (BSON) are intuitive and align well with modern programming practices, particularly in web and mobile development.
   2. **Rich Query Language:** MongoDB provides a powerful query language with support for ad-hoc queries and aggregation making it easy to perform complex data retrieval and analysis.

iii. **Handling Big Data:**
   1. MongoDB is well-suited for managing large volumes of unstructured or semi-structured data, making it ideal for applications involving big data analytics, logging, and content management.

iv. **Support for Various Data Types:**
   1. **Embedded Documents and Arrays:** MongoDB supports storing nested documents and arrays, allowing for rich data structures to be stored and queried natively.
   2. **Geospatial Data:** Built-in support for geospatial queries makes MongoDB a good choice for location-based applications.

v. **Community and Ecosystem:**
   1. **Vibrant Community:** MongoDB has a large and active community, providing extensive documentation, tutorials, and third-party tools.
   2. **Cloud Integration:** MongoDB Atlas, the fully managed cloud version of MongoDB, provides seamless integration with various cloud providers, offering scalability, backup, and monitoring services out-of-the-box.

2.4.5 Authentication Mechanism

Authentication is a critical aspect of any application that ensures secure access and protects sensitive user data. In our backend development, we carefully considered different authentication mechanisms to implement within our application. This section provides an overview of various authentication types and highlights the usage of JSON Web Tokens (JWT) for authentication in our project.

Different Types of Authentications:

Session-based Authentication: Session-based authentication involves generating a unique session identifier for each authenticated user. The server stores the session information, typically in a server-side database or cache, to validate subsequent requests. This approach requires maintaining server-side state and can be susceptible to session hijacking or CSRF attacks.

Token-based Authentication: Token-based authentication uses a token, typically a string of characters, to authenticate requests. Tokens can be issued and verified on each request, eliminating the need for server-side session storage. Two commonly used token-based authentication mechanisms are JSON Web Tokens (JWT) and OAuth.

OAuth-based Authentication: OAuth is an authorization framework that allows third-party applications to access user data from an identity provider without exposing the user's credentials. While OAuth is primarily focused on authorization, it can be used in conjunction with token-based authentication for user authentication.

Usage of JWT Token Authentication

For our application, we implemented JWT token-based authentication. JSON Web Tokens are compact, URL-safe tokens that securely transmit information between parties. Here are some reasons why we chose JWT for authentication:

- JWT tokens are self-contained, eliminating the need for server-side storage or database lookups, allowing for easy scalability and reducing server overhead.
- JWT tokens can be signed and encrypted to ensure data integrity and secure sensitive user information from tampering by malicious actors.
- JWT tokens' ease of exchange between different systems and services makes them an ideal choice for our decentralized application involving communication between frontend, backend, and external services.
- JWT tokens stored on the client-side reduce frequent authentication requests, improving user experience with seamless access to the application without constant re authentication.

By implementing JWT token-based authentication in our application, we ensure secure and efficient access to user-specific resources and API endpoints. The token-based approach, combined with encryption and stateless nature, enhances the overall security and scalability of our backend system.

2.5 Artificial Intelligence

At the heart of our project lies a sophisticated AI system that integrates several state-of-the-art technologies to deliver an intelligent reading experience. The AI components include:

- **Large Language Models (LLMs):** These models, such as OpenAI's GPT-3.5 Turbo, are employed for natural language understanding and generation, enabling the assistant to comprehend user queries and generate coherent responses as well as comprehensive precise summaries of given text, especially from books and user documents.
- **Retrieval-Augmented Generation (RAG):** RAG enhances the LLMs' capabilities by retrieving relevant information from user documents and the web, ensuring context-aware responses.
- **Vector Databases:** By leveraging vector databases, we efficiently store and search embeddings generated from user documents and queries, facilitating quick and relevant information retrieval.
- **Optical Character Recognition (OCR):** Utilizing advanced OCR models, our system can process scanned documents, converting images of text into editable and searchable data.
- **Text-to-Speech (TTS):** TTS models, like VITS, enable the assistant to convert written text into natural-sounding speech, allowing users to listen to their documents on demand in real-time.
- **Deployment using ASGI (FastAPI):** The entire system is deployed using ASGI with FastAPI, ensuring high performance, scalability, and real-time interaction capabilities.

2.6 Deployment

Deployment is a crucial step in making our application available to users. It involves setting up the necessary infrastructure and deploying the application code to a production environment. In this section, we will discuss the deployment process for both the web and mobile components of our application.

2.6.1 Web App Deployment

For deploying the web component of our application, we utilized the following tools and technologies:

Docker: Docker provided us with a containerization platform that allowed us to package our application and its dependencies into portable containers. This ensured consistent deployment across different environments and simplified the deployment process.

Chapter 3: User Interface / User Experience (UI/UX)

3.1 Introduction

Bookipedia is a cutting-edge reading assistant application designed to enhance the reading experience through AI-powered interactions. Users can upload their books, read them within the app, and interact with the text in various ways, including summarizing, finding additional resources, highlighting, and converting text to speech. The app is developed with a user-friendly interface and a sleek dark mode design, making it a perfect tool for avid readers and researchers alike. Bookipedia aims to make reading more engaging and accessible, providing users with tools to better understand and interact with their books.

3.1.1 Branding

Branding refers to the deliberate actions you take to influence people's perception of your product or service so they will choose your brand time and again. Essentially, it is the way your product or service lives in the hearts and minds of your customer.

While it may seem like a simple idea to comprehend, branding is an ever elusive, hotly debated topic that is not easy to define. Why? Because, as we mentioned above, branding is emotive, and subjective and not something that can necessarily be measured or quantified.

In order to fully conceptualize branding, let's go back a step and understand what a brand is. To quote author and entrepreneur Seth Godin, "A brand is the set of expectations, memories, stories and relationships that, taken together, account for a consumer's decision to choose one product or service over another". Meaning branding is everything you do to actively influence those decisions.

Think about it, when consumers need to make a choice, how will they make their decisions? Whatever motivates them, branding is what shapes the perception of your business in their minds and ultimately what converts new customers into loyal buyers.

The branding of Bookipedia is centered around a modern and sophisticated aesthetic, which is reflected in the app's dark mode theme. The primary color, #265d85, is chosen for its calming yet authoritative tone, symbolizing trust and knowledge. This color is consistently used throughout the app to maintain a cohesive look and feel. The logo of Bookipedia is simple yet distinctive, designed to be easily recognizable. Typography choices align with the modern aesthetic, using clean and readable fonts that enhance the user experience. Overall, the branding of Bookipedia aims to convey a sense of innovation, reliability, and ease of use.

3.1.2 User Experience Design

It is the process design teams use to create products that provide meaningful and relevant experiences to users. This involves the design of the entire process of acquiring and integrating the product, including aspects of branding, design, usability and function.

The user experience (UX) design of Bookipedia focuses on simplicity, intuitiveness, and efficiency. The app's layout is designed to be straightforward, ensuring that users can easily navigate through different sections without any confusion. Key UX design principles applied in Bookipedia include:

1. **Consistency:** Consistent use of colors, fonts, and icons helps users understand the interface quickly and reduces the learning curve.
2. **Feedback:** Immediate feedback is provided for user actions, such as highlighting text or converting it to speech, ensuring users are aware that their actions have been registered.
3. **Accessibility:** Features like dark mode and text-to-speech enhance accessibility, making the app usable for a wider audience, including those with visual impairments or reading difficulties.
4. **Efficiency:** The design minimizes the number of steps required to perform key actions, such as uploading a book or accessing the AI chat, making the app efficient to use.

3.1.3 User Interface Design

It is the process designers use to build interfaces in software or computerized devices, focusing on looks or style. Designers aim to create interfaces which users find easy to use and pleasurable. UI design refers to graphical user interfaces and other forms e.g., voice-controlled interfaces.

The user interface (UI) design of Bookipedia is clean, modern, and visually appealing. Key elements of the UI design include:

1. **Dark Mode:** The app's dark mode provides a comfortable reading experience, reducing eye strain, especially in low-light conditions. The primary color, #265d85, is used for highlights and important UI elements, ensuring they stand out without being harsh on the eyes.
2. **Home Page:** The home page displays the most read books and recent books, in a card-based layout, making it easy to scan and select items. Each card is designed with a clear visual hierarchy, using different font sizes and weights to differentiate between titles and additional information.
3. **Library Page:** The library page showcases uploaded books with progress bars indicating the reading percentage of each book. This visual representation helps users quickly understand their reading progress. The use of icons and thumbnails provides visual cues, making it easy to identify books at a glance.
4. **AI Interaction:** The AI chat interface is designed to be intuitive, with a chat bubble layout that users are familiar with from messaging apps. Action buttons for summarizing, highlighting, and other features are prominently placed for easy access.

3.1.4 User Research Mapping

Mapping is one of the most effective ways of presenting your findings and insights to your team and stakeholders. UX mapping is a process that narrows the gap between users' or customers' needs and the product. It's important to capture the whole picture before you start building or even designing.

To ensure that Bookipedia meets the needs of its users, extensive user research was conducted. The user research mapping process involved several stages:

1. **User Interviews:** Interviews with potential users helped identify common pain points in the reading experience and specific needs that Bookipedia could address.
2. **Surveys:** Online surveys gathered quantitative data on user preferences, such as preferred reading modes, common use cases for AI interactions, and feedback on existing reading apps.
3. **Usability Testing:** Prototypes of the app were tested with users to observe their interactions and gather feedback on the design and functionality. This helped identify areas for improvement and refine the UX/UI design.
4. **Persona Development:** Based on the research data, user personas were created to represent different types of users. These personas guided the design process, ensuring that the app addresses the needs of a diverse user base.
5. **Journey Mapping:** User journey maps were developed to visualize the steps users take when interacting with Bookipedia. This helped identify potential friction points and opportunities to enhance the user experience.

3.2 Bookipedia Logo

Logo shown in figure (Figure 3.1)

Figure 3.1: Bookipedia LOGO

3.3 Persona

A persona is a fictional, yet realistic, description of a typical or target user of the product. A persona is an archetype instead of an actual living human, but personas should be described as if they were real people.

Persona: Eman, the Medical Student

Background:

- Age:21
- Occupation: Medical Student
- Education: Pursuing a degree in Medicine
- Technology Proficiency:Proficient with smartphones, tablets, and computers for studying and research.
- Interests: Eman is passionate about medicine, anatomy, and healthcare advancements. She enjoys staying updated with medical research and exploring related topics.

Demographics:

- Gender: Female
- Location:Urban area
- Income:Student
- Family Status: Single, lives in student accommodation

Goals:

- Eman's primary goal is to excel in her medical studies and keep up with the latest developments in the field.
- She enjoys reading medical textbooks, research papers, and articles to deepen her understanding of complex medical concepts.

Challenges:

- Time Constraints: Balancing rigorous academic demands with personal interests and hobbies.
- Information Overload:Accessing and digesting vast amounts of medical literature and research can be overwhelming.
- Need for Efficiency:Eman seeks tools that can help streamline her study sessions and provide quick access to relevant information.
- Needs and Behaviors:
- Digital Reading Preferences:Eman prefers digital formats for their portability and convenience, allowing her to study anytime, anywhere.
- Interactive Learning: Enjoys platforms that offer interactive features such as highlighting, annotating, and real-time information retrieval.
- Personalization:Values tools that offer personalized recommendations based on her medical interests and study habits.
- Tech-Savvy:Comfortable using technology and appreciates innovative tools that

enhance her learning experience.

How Our Project Meets Eman's Needs:

- AI-Enhanced Insights:Our application provides Eman with instant, referenced insights while she reads medical textbooks and research papers, helping her grasp complex medical concepts effectively
- Interactive Features: Eman can highlight text, ask questions via an AI chat interface, and engage in discussions within the platform, facilitating interactive learning.
- Cross-Device Sync:Ensures Emily can seamlessly access her study materials and annotations across her smartphone, tablet, and laptop.
- Personalized Recommendations: Based on her medical interests and study patterns, the platform suggests relevant medical literature and research papers that align with Eman's academic pursuits.

Quotes:

- "Studying medicine involves a lot of reading and learning. A tool that can help me understand complex topics quickly and efficiently would be incredibly valuable."
- "I'm always looking for ways to deepen my knowledge and stay updated with medical advancements. Having access to curated resources and insights would save me a lot of time and effort."

3.4 Information Architecture

3.4.1 Overview

Information architecture (IA) is a science of organizing and structuring content of websites, web and mobile applications, and social media software. An American architect and graphic designer, Richard Saul Wurman, is a founder of the IA field. Today, there are many specialists working on IA development who have established the Information Architecture Institute. According to the IA Experts, information architecture is the practice of deciding how to arrange the parts of something to be understandable.

3.4.2 Wireframing

The overriding goal of wireframes like this (Figure 3.2), (Figure 3.3) is to describe the structure and functionality of a user's flow – functionality being anything that the user interacts with, such as buttons, menus and dropdowns. This is achieved by creating a visual representation that communicates the product's structure, information hierarchy, behavior, navigation and placement of content.

Without touching on the visual details or the UI design, a wireframe demonstrates the product's general layout and states which elements need to be included on each page. There are many available wireframe tools for UX designers to choose from.

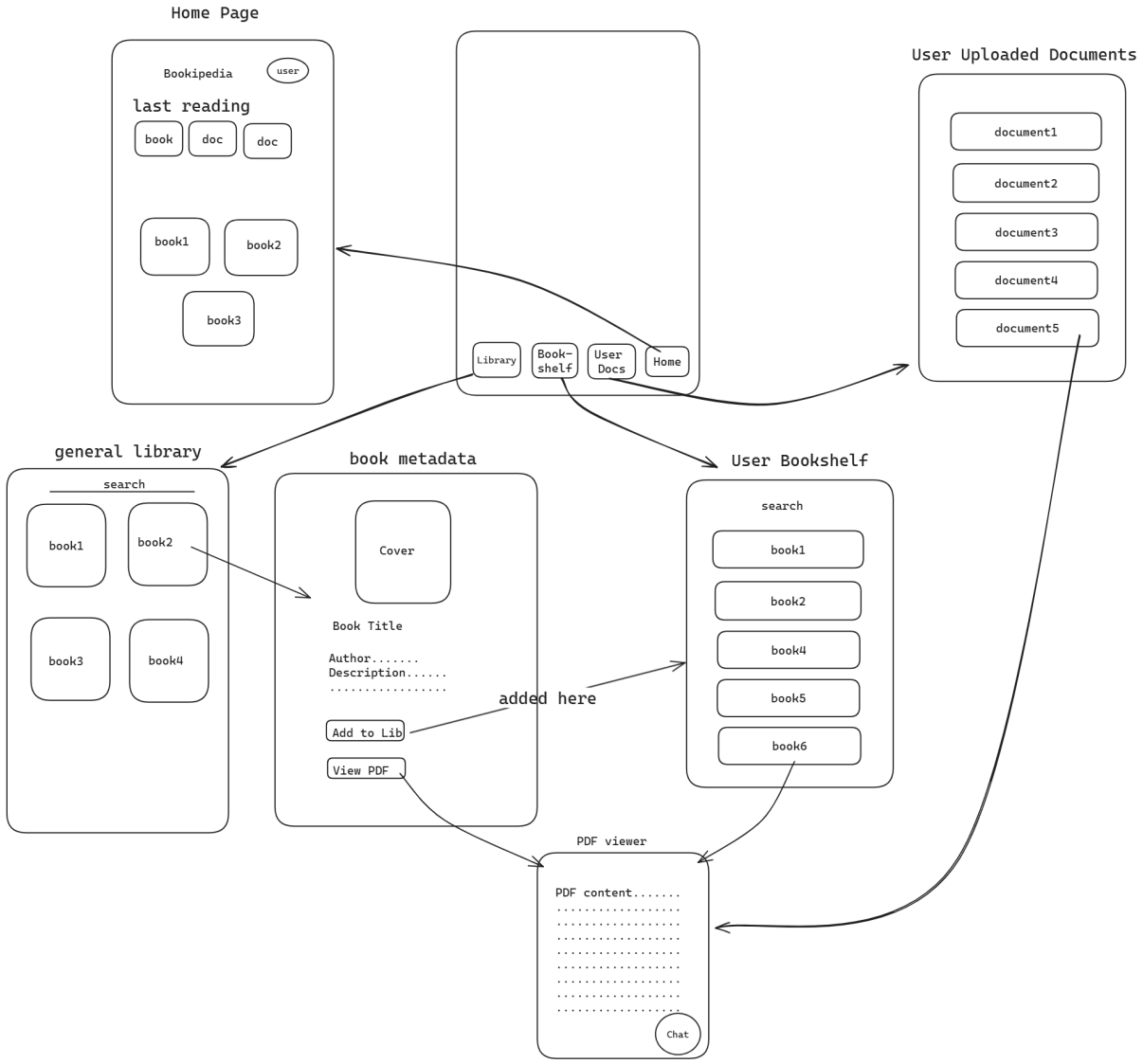

Figure 3.2: Wireframing Interface Pages

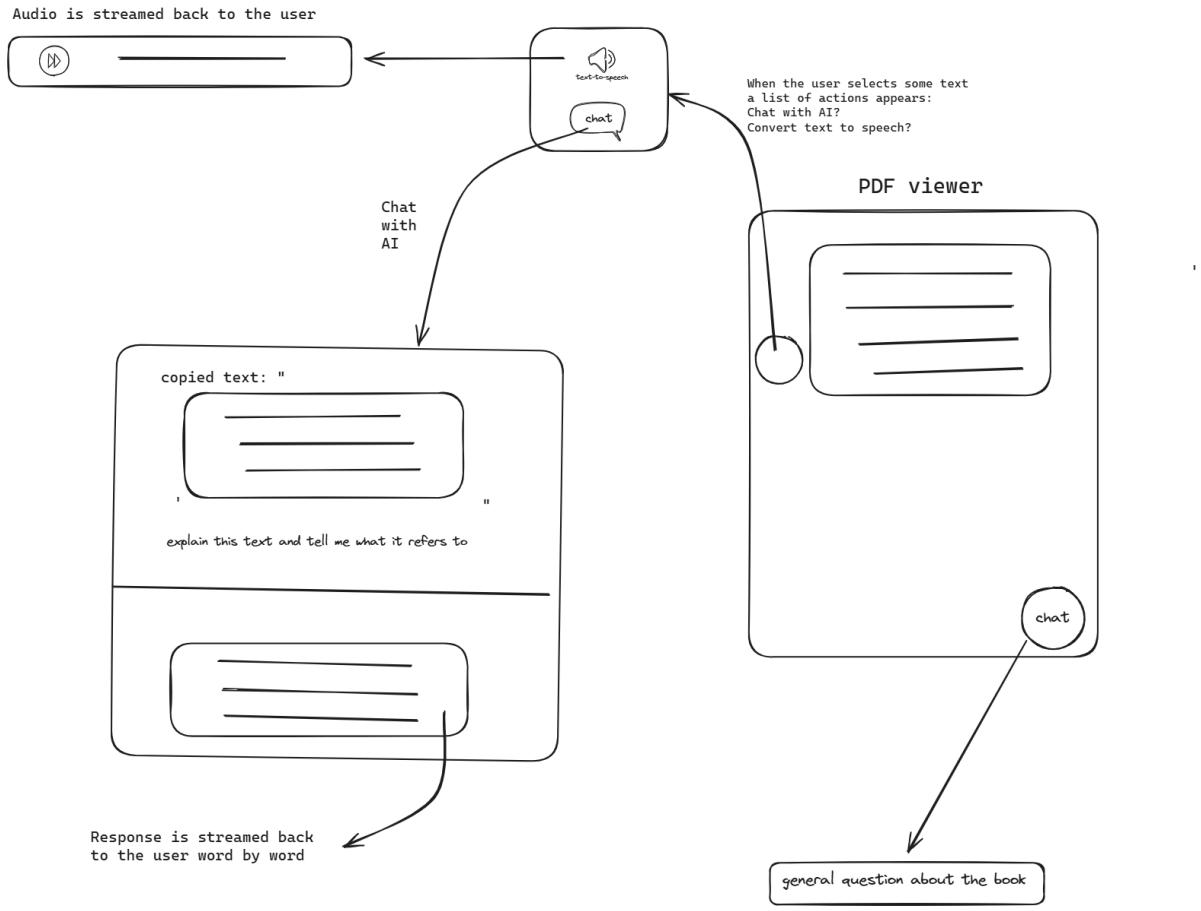

Figure 3.3: Wireframing AI Actions in The Application

3.5 User Flow

A user flow diagram, such as the one illustrated (Figure 3.4), visually maps the user's journey through an application. Starting from the initial "Open App" step, it guides users through essential interactions like logging in, accessing documents, browsing, and saving books. Decision points, such as "Have an account?", direct the flow based on user responses, ensuring a tailored experience. By detailing actions like adding or deleting documents, and reading or saving books, the diagram encapsulates the app’s functionality, enhancing clarity and user understanding.

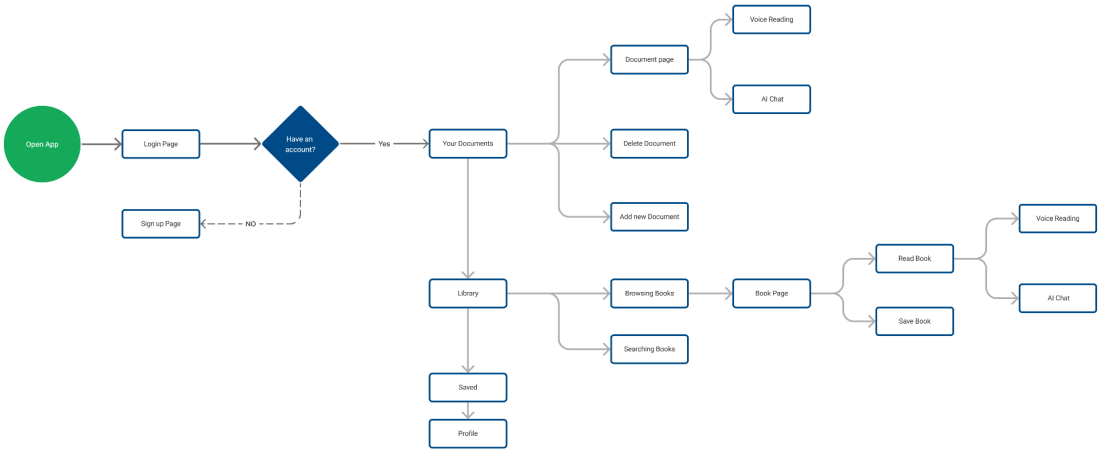

Figure 3.4: User Flow

3.6 Color Palette

WE use this color palette (Figure 3.5) consists of five colors, each with a specific hexadecimal code:

1. **#06283D**: A deep, dark blue that can be used for primary elements, providing a strong and professional look.
2. **#256D85**: A teal shade offering a refreshing and calm vibe, suitable for secondary elements or highlights.
3. **#47B5FF**: A bright, vibrant blue that stands out, perfect for buttons or call-to-action items to draw user attention.
4. **#DFF6FF**: A very light blue that can be used for backgrounds or to create contrast without overwhelming the primary content.
5. **#FFFFFF**: Pure white, an essential color for backgrounds and text to ensure readability and a clean design.

These colors work harmoniously together, providing a modern and appealing look for the application, while ensuring usability and accessibility.

Figure 3.5: Color Pallet

3.7 UI Screens

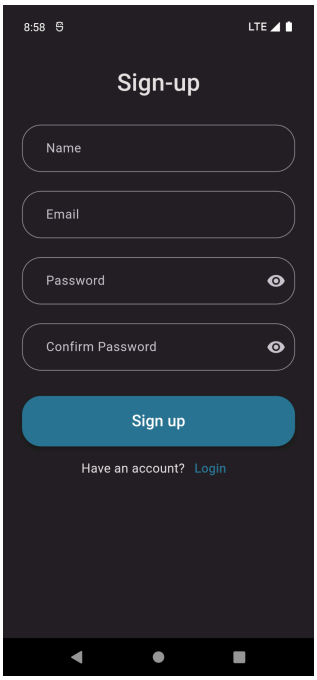

Figure 3.6: Sign-Up

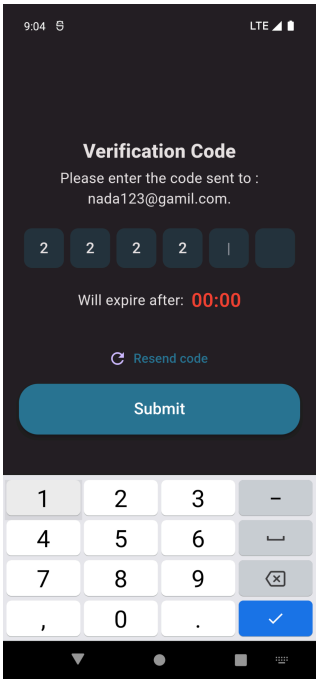

Figure 3.7: Verification

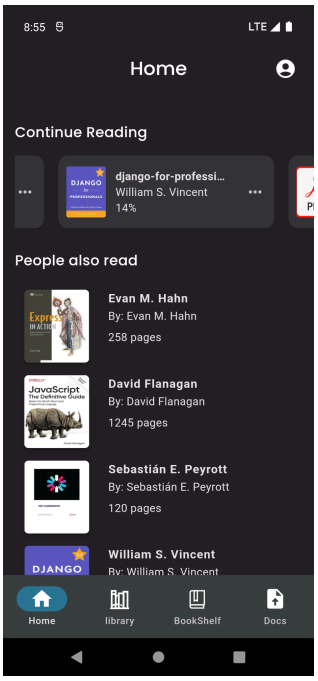

Figure 3.8: Home Page

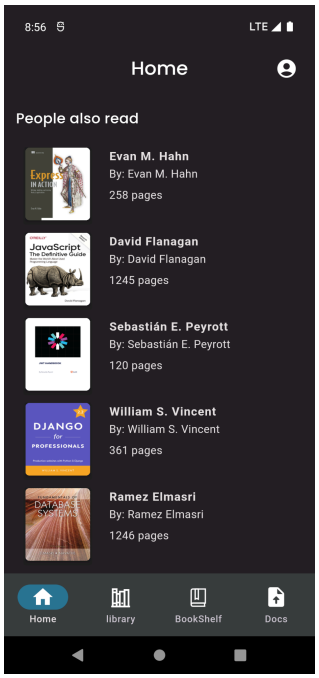

Figure 3.9: Book Recommendations

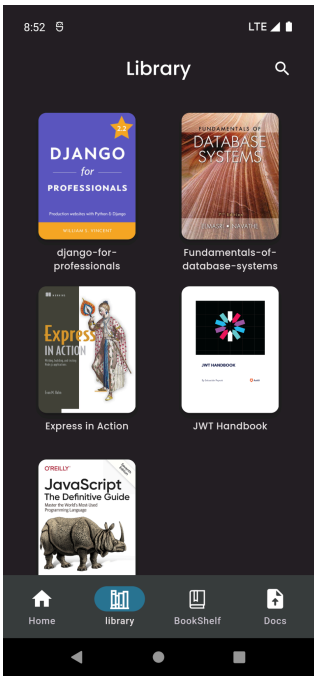

Figure 3.10: Library

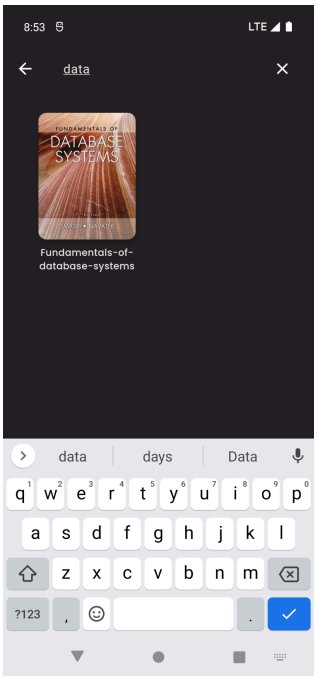

Figure 3.11: Search

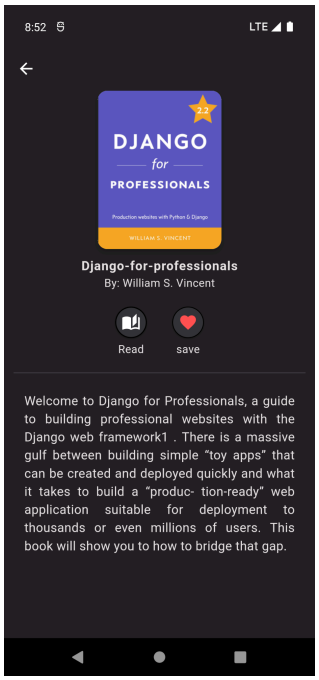

Figure 3.12: Book Metadata

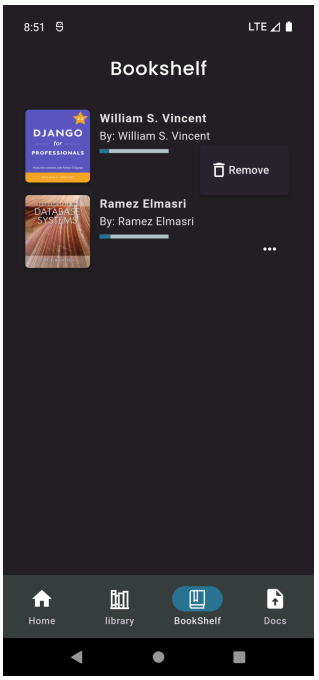

Figure 3.13: Bookshelf

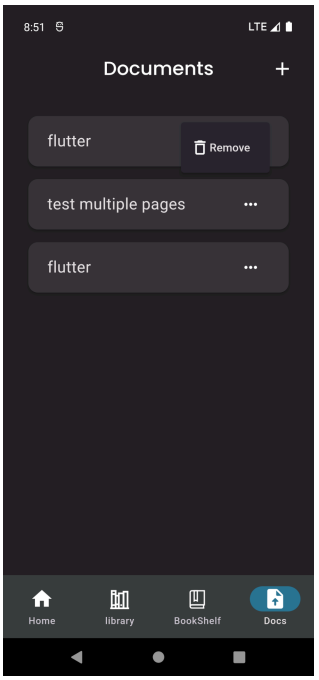

Figure 3.14: User Documents

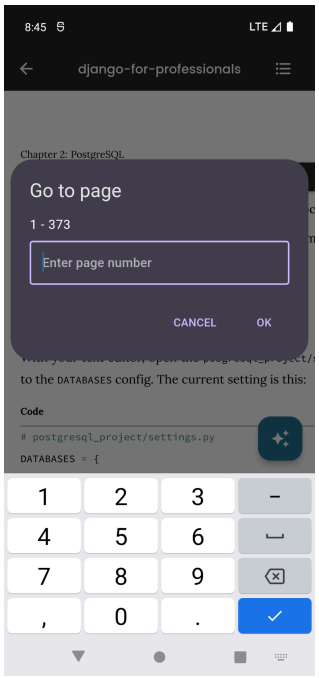

Figure 3.15: Go To PDF Page

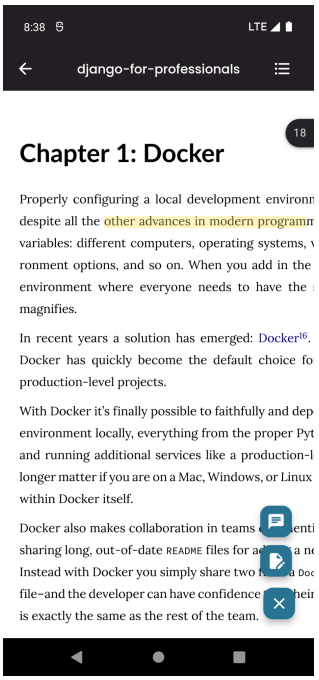

Figure 3.16: Text HIghlight

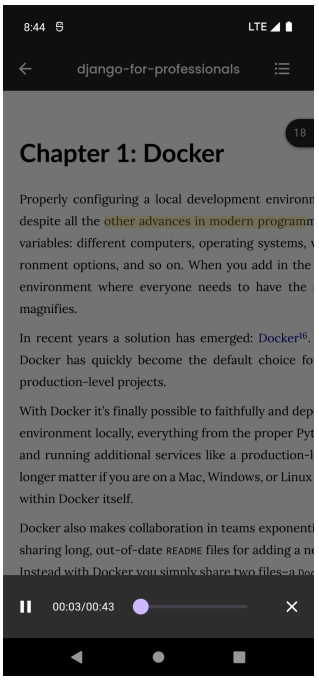

Figure 3.17: Text To Speech

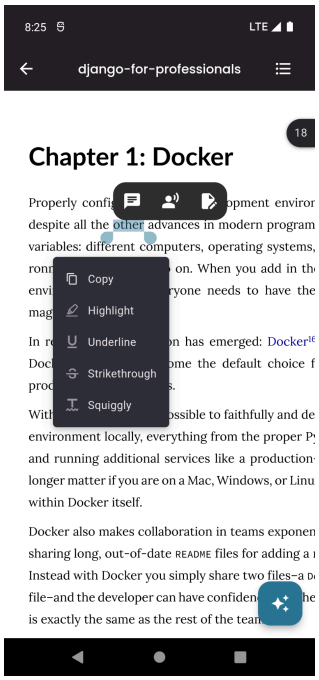

Figure 3.18: Select Text Options

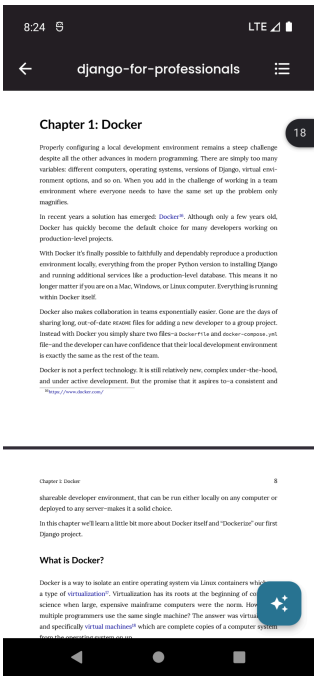

Figure 3.19: AI-Icon Options

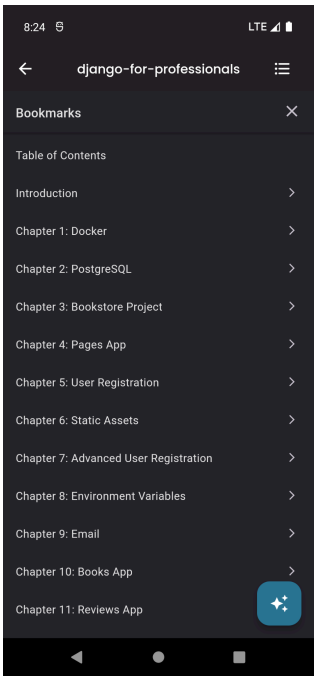

Figure 3.20: PDF Indexes

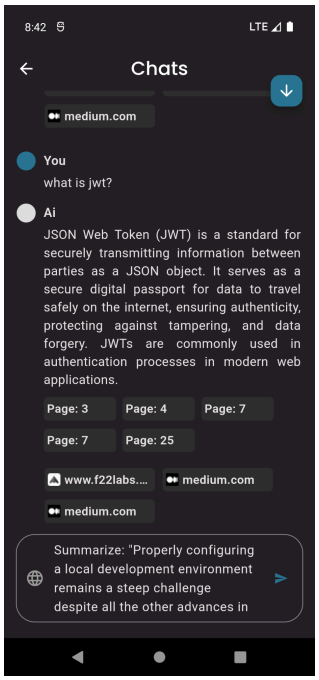

Figure 3.21: Summarize Text

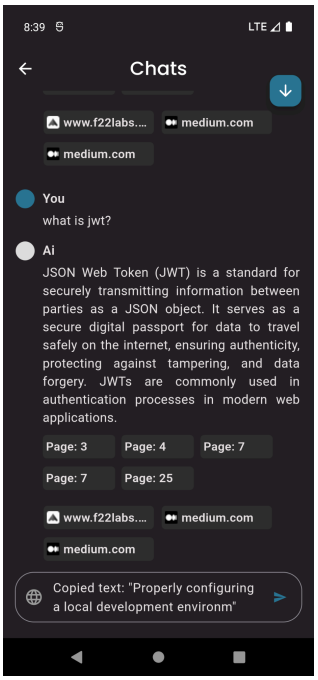

Figure 3.22: Ask about Text in Book

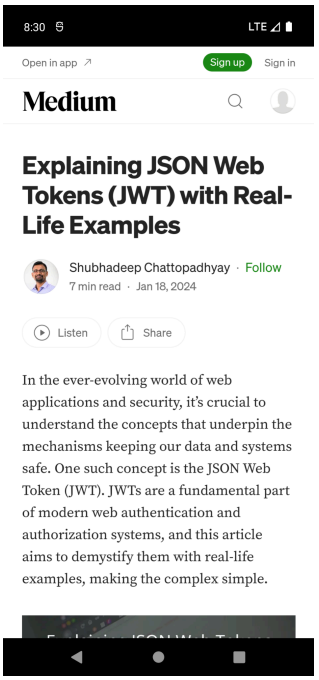

Figure 3.23: Web Sources in Question Answer

Chapter 4: Mobile Application Development

4.1 Introduction

4.1.1 What is Mobile Development?

Mobile development refers to the process of creating software applications that run on mobile devices such as smartphones and tablets. This encompasses a range of activities from designing the user interface to coding, testing, and deploying the application. Mobile development is typically categorized into:

1. Native Development: Creating apps for a specific operating system using the platform's native programming language and development environment.
    a. iOS: Apps are developed using Swift or Objective-C in Xcode.
    b. Android: Apps are developed using Kotlin or Java in Android Studio.
2. Cross-Platform Development: Creating apps that can run on multiple operating systems using a single codebase. This approach uses frameworks like React Native, and Flutter.
3. Hybrid Development: Combining elements of both native and web applications. Hybrid apps are built using web technologies (HTML, CSS, JavaScript) and are wrapped in a native shell, allowing them to run on multiple platforms.

4.1.2 Importance of Mobile Development

1. **Accessibility:** Mobile apps allow users to access services and information anytime, anywhere.
2. **User Engagement:** Mobile apps provide personalized experiences, push notifications, and offline access, enhancing user engagement.
3. **Business Growth:** Mobile apps can be a significant revenue stream through in-app purchases, subscriptions, and ads.
4. **Efficiency:** Apps streamline processes and provide functionalities that are essential for modern businesses.

4.2 Flutter

4.2.1 What is Flutter?

Flutter is an open-source UI software development kit (SDK) created by Google. It is used to develop applications for Android, iOS, Linux, macOS, Windows, and the web from a single codebase. The main components of Flutter are:

1. Dart Platform: Flutter apps are written in the Dart language, which is also developed by Google. Dart is optimized for building fast apps on any platform.
2. Flutter Engine: This is a portable runtime for hosting Flutter applications. It implements the core libraries, including animation and graphics, file and network I/O, accessibility support, plugin architecture, and more.

3. Foundation Library: The foundation library includes the building blocks for Flutter applications. It provides basic classes and functions that are used to construct a Flutter app.
4. Widgets: Flutter is built around the concept of widgets. Everything in Flutter is a widget, from the layout to the elements on the screen. Widgets describe what their view should look like given their current configuration and state.

4.2.2 Why Flutter

1. **Single Codebase for Multiple Platforms**: Flutter allows developers to write one codebase and compile it into the respective machine code for Android, iOS, web, and desktop applications.
2. **Fast Development**: With features like Hot Reload, developers can quickly see the results of their changes without restarting the entire app.
3. **Expressive and Flexible UI:** Flutter's rich set of pre-designed widgets and its powerful, customizable widget system make it easy to build complex, expressive UIs.
4. **High Performance:** Flutter is compiled to native ARM code using Dart, ensuring that it performs as well as natively written apps.
5. **Strong Community and Ecosystem:** The Flutter community is growing rapidly, and there is a vast array of packages and plugins available to extend Flutter's functionality.

4.3 Flutter Architecture

4.3.1 What is Clean Architecture?

Clean Architecture is a design pattern that aims to separate the concerns of an application into distinct layers, making it easier to maintain, test, and scale. This approach is beneficial for large applications where separation of concerns, independence of frameworks, and testability are crucial.

Key Principles of Clean Architecture

1. **Separation of Concerns**: Each layer of the architecture has a specific responsibility, making the code more manageable and understandable.
2. **Dependency Rule:** Dependencies should point inward. Outer layers can depend on inner layers, but inner layers should not depend on outer layers.
3. **Testability:** By isolating business logic from external concerns (like UI or database), it becomes easier to test the core functionality of the application.

Layers in Clean Architecture (Figure 4.1)

1. **Presentation Layer**: Handles the UI and user interactions. In Flutter, this would be your widgets and state management solutions.
2. **Domain Layer:** Contains the business logic and entities of the application. It is independent of any frameworks and can be tested in isolation.
3. **Data Layer:** Manages data sources, such as network requests, database access, and repositories. It implements the repositories defined in the domain layer.

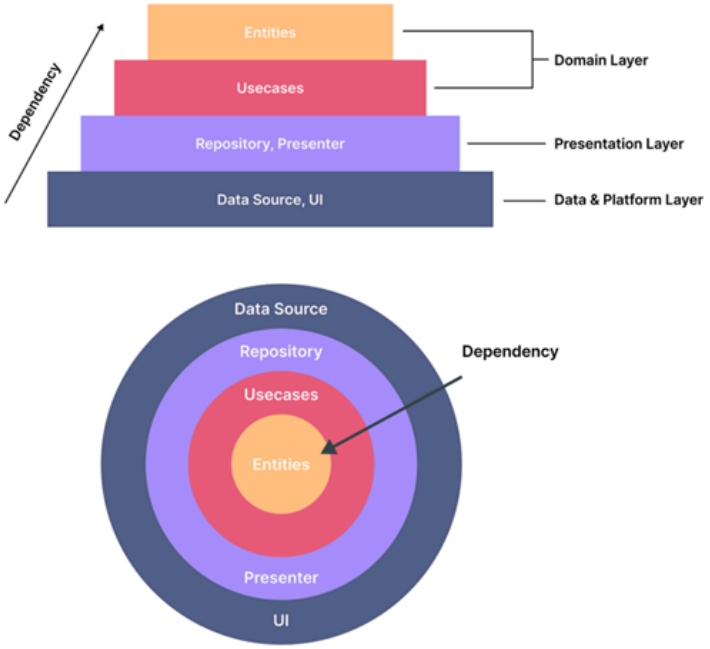

Figure 4.1: Layers in Clean Architecture

4.3.2 State Management

State management is a crucial aspect of Flutter application development, ensuring that the user interface (UI) responds correctly to user input and other events. In Flutter, state refers to any data that can affect the appearance or behavior of your app. There are several approaches to state management in Flutter, each with its own use cases and advantages.

Common State Management Approaches in Flutter

1. StatefulWidget
2. Provider
3. Riverpod
4. Bloc (Business Logic Component)
5. GetX

4.3.3 Bloc Architecture

BLoC stands for Business Logic Component. It is an architectural pattern used in Flutter for managing the flow of data, state, and business logic in an application. It provides a structured way to separate the presentation layer (UI) from the business logic layer. As a result, this helps in developing more maintainable, testable, and scalable Flutter apps. BLoC architecture is often used to create reactive and high-performance applications.

Bloc architecture (Figure 4.2) consists of the following components:

1. **Business Logic:** The core logic and functionality of the Flutter application reside within BLoCs. It contains tasks like data fetching, transformation, validation, and any other operations that aren't directly related to the UI. By isolating business logic, it becomes easier for Flutter developers to reuse and test.
2. **Events:** In BLoC, developers can initiate changes or actions by sending events to the BLoC. Events are essentially user interactions or triggers, such as button clicks, text input, or network requests. These Events represent what should happen next in your app. Thus, it helps in creating intelligent algorithms in the app.
3. **States:** BLoC emits states in response to events. These States represent the different snapshots or states of your application's UI and data. Each state represents a specific view, and changes in state trigger UI updates.

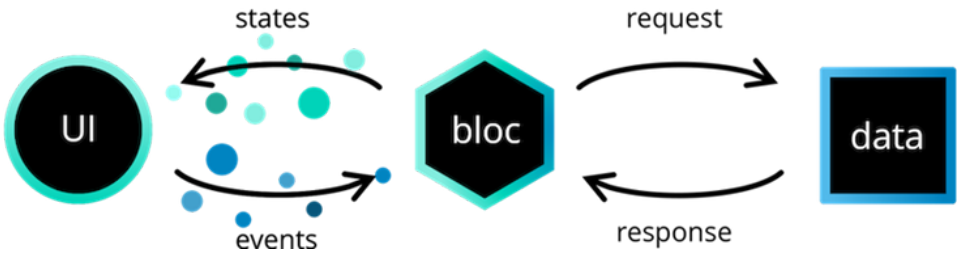

Figure 4.2: BLoC Architecture

4.4 Networking In Flutter

4.4.1 What Is Dio

Dio is a powerful HTTP client for Dart, which is very useful for making network requests in Flutter applications. It provides a more straightforward and flexible way to handle network requests compared to the built-in `http` package.

Key Features of Dio:

1. **Interceptors:** Allows you to intercept requests and responses for logging, modifying, and error handling.
2. **Global Configuration:** Easily configure common options like base URL, headers, timeouts, etc., for all requests.

3. **Form Data:** Simplifies handling form data and file uploads.
4. **Cancellation:** Supports canceling ongoing requests.
5. **Error Handling:** Provides comprehensive error handling mechanisms.

4.5 Implementation

4.5.1 PDF Viewer

Leveraging the Syncfusion PDF Viewer package, we managed to provide a seamless experience for users to view, navigate, and interact with PDF documents on mobile devices.

Key Features:

1. Smooth PDF Rendering:
    a. Utilized Syncfusion PDF Viewer for high-performance and accurate PDF rendering.
    b. Ensured smooth scrolling and page transitions for a better user experience.
2. Intuitive User Interface:
    a. Designed a clean and responsive UI using Flutter's rich set of widgets.
    b. Provided easy navigation through thumbnails, bookmarks, and page jumping.
3. Search and Highlight:
    a. Implemented a powerful search functionality to allow users to find specific text within PDFs.
    b. Enabled text highlighting for better visibility of search results.
4. Zoom and Pan:
    a. Added support for pinch-to-zoom and double-tap-to-zoom for enhanced readability.
    b. Enabled panning to navigate large documents effortlessly.
5. Annotation and Markup:
    a. Included tools for adding annotations, such as text notes, highlights, and drawing.
    b. Allowed users to save and retrieve annotations for future reference.
6. Offline Access:
    a. Enabled downloading and offline viewing of PDF documents for convenience.
    b. Ensured efficient handling of large files with minimal performance impact.

4.5.2 Chat

We can summarize this feature into the following points:

1. The question will be sent through the endpoint to the AI model to be processed.
2. The generated answer will be streamed back to the user and displayed with a proper animation to enhance user experience.
3. The sources of the generated answer will be displayed afterwards enabling the user to navigate to the specific pages in the document that the model based its answer on as well as web sources.
4. The user may select a specific text from a document to be included in the question.
5. The previous messages will be also provided to the user.

4.5.3 Text-to-Speech

1. The raw chunks of audio data are received then converted to a .wav file .
2. Using the audio players package we managed to play , pause , stop and seek a specific part of the audio which provides great flexibility.

4.6 Important Packages

1. **dio:** A powerful HTTP networking package, supports Interceptors, Aborting and canceling a request, Custom adapters, Transformers, etc.
2. **flutter_bloc:** Flutter Widgets that make it easy to implement the BLoC (Business Logic Component) design pattern. Built to be used with the bloc state management package.
3. **bloc:** A predictable state management library that helps implement the BLoC (Business Logic Component) design pattern.
4. **syncfusion_flutter_pdfviewer:** Flutter PDF Viewer library is used to display a PDF document seamlessly and efficiently.
5. **file_picker:** A package that allows you to use a native file explorer to pick single or multiple absolute file paths, with extension filtering support.
6. **animated_text_kit:** provides text animations.
7. **webview_flutter:** A Flutter plugin that provides a WebView widget on Android and iOS.
8. **audioplayers:** A Flutter plugin to play multiple audio files simultaneously.

Chapter 5: Artificial Intelligence

5.1 Introduction

It has been clearly established through our project outline that central to its vision and objectives is the use of cutting-edge Artificial Intelligence techniques and SoTA models. As our project aims to provide a truly intelligent reading assistant, it was imperative to incorporate various paradigms of the AI landscape, such as natural language understanding and generation, vision, as well as speech synthesis.

The core aspect of our project is its ability to understand the context of the user's queries, with respect to the document (or book) they are reading. In order to achieve that we employ a cutting-edge technique, referred to as Retrieval Augmented Generation (RAG) which in turn utilizes Large Language Models (LLMs) and other technologies to achieve a context-aware chat interface. Other features that are central to our vision are the use of Optical Character Recognition (OCR) to further handle scanned documents in addition to the default of text-based ones, and the leveraging of speech synthesis in the form of Text to Speech (TTS) models in order to allow the user to listen to their book being read back to them on demand.

In the following sections, we discuss the core technologies used, the method in which we incorporate them into the overall system, and how they are implemented and deployed to fulfill the front-end client's requests.

5.2 Large Language Models

Large Language Models (LLMs) are advanced deep learning models designed to process and generate natural language text by training on extensive language datasets. These models, typically based on transformer architectures, have achieved significant milestones in natural language processing (NLP) due to their ability to perform a wide range of language tasks with high accuracy.

LLMs are trained on diverse and comprehensive corpora, encompassing text from books, articles, websites, and other sources. This training allows them to produce coherent and contextually relevant responses to various prompts. Additionally, LLMs compress vast amounts of information into their parameters, enabling them to exhibit a form of emergent reasoning and knowledge synthesis across different domains. However, despite their sophisticated capabilities, LLMs exhibit inherent limitations, including:

1. **Lack of Groundedness:** LLMs, like other probabilistic machine learning models, can only generate outputs based on calculated probabilities from the input data. Without access to definitive ground truth, these models are prone to 'hallucinating' a response, particularly when the model has not seen (been trained on) the context being posed by a user's query

2. **Context Limit:** One might argue that grounding the LLM's responses could be achieved by feeding the entire document alongside the input. However, this is impractical. Even if a model with a sufficiently large context window could be built, which is not currently feasible, the LLM would still face a 'needle in a haystack' problem, struggling to focus on the relevant parts of the provided text.
3. **Knowledge Cut-Off:** LLMs are trained on real-world data and generate their responses based only on the data they were trained on. That means that the LLM lacks any sense of current world events or novel knowledge.

To address these limitations, we introduce Retrieval-Augmented Generation (RAG), the foundational framework that powers the core features of our project.

5.3 Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation or RAG for short, is the culmination of emerging techniques and technologies that aim to 'augment' the text generation capabilities of LLMs by adding relevant context to the queries the model is responding to.

The need for Retrieval-Augmented Generation (RAG) arises due to the previously discussed limitations of large language models (LLMs) when used as the foundation for a context-aware reading assistant. RAG seeks to overcome these limitations by enhancing the query context with only the most relevant information from a given source of knowledge. In our project, these sources include the user's books and documents, as well as the web.

A RAG pipeline can be generally described to perform the following steps: Document Preprocessing, Document Indexing, Information Retrieval and Context and Response Generation. They are demonstrated in further detail in the following sections.

5.3.1 Document Preprocessing

In order to address the context limit limitation of the LLM, techniques of splitting text into coherent splits become imperative. In addition, to use unstructured document formats such as PDFs as a knowledge base would prove challenging.

In this step, various methods of text and format extraction, splitting and structuring are employed to produce a data structure of the text splits to be later retrieved based on their relevance to the query.

It is noted that the web can also serve as a knowledge base, with text being extracted from HTML pages and processed in a similar manner.

5.3.2 Information Indexing and Retrieval

To generate the context of a user query based on a knowledge source, it is necessary to retrieve text splits that are relevant to the query. This suggests that the text splits should be searched across according to their meaning rather than syntactic structure. In order to achieve that, the semantic information in textual data needs to be extracted and represented in a searchable space.

The most common model to represent semantic information in text is vector embedding. Hence, text splits produced in the preprocessing step are embedded using a text embedding model. The user query can also be embedded at inference-time and the most similar vector embeddings can be retrieved using vector similarity search.

One can readily infer that for large knowledge bases, the process of comparing the query vector to every vector embedding in the knowledge base would be prohibitively expensive and time-consuming. In addition, vector and text data need to be persisted in a database system.

Vector Databases address this issue by implementing a Hierarchical Navigable Small World (HNSW) model, a vector indexing technique that is significantly more efficient than brute-force vector search. Additionally, they offer traditional database functionalities to support routine CRUD (Create, Read, Update, Delete) operations.

5.3.3 Context and Response Generation

Once relevant text segments are retrieved, the context for the query can be generated. Some basic RAG systems use these segments as they are. However, since embeddings are generated independently of the query, the retrieved segments may not be optimally ranked due to the disconnect between document embeddings and query embeddings. To address this, further post-processing can be performed to produce more coherent contexts. This involves balancing context specificity and extensiveness while maintaining cost-effectiveness.

Various techniques can be employed in this regard, including reranking models and multi-granular retrieval, both of which are leveraged in this project, the latter being a novel algorithm proposed by our team.

As soon as the appropriate context is produced, the LLM can hence be prompted with a prompt template that combines both the generated context and the user query in an effective manner.

5.4 Employing Optical Character Recognition

As our application aims to equip users with all the necessary tools to maximize and broaden their ability to extract and gain knowledge, it became essential to leverage Optical Character Recognition (OCR) technology. OCR enables the conversion of various types of documents, such as scanned paper documents or images captured by a digital camera, into editable and searchable data. By integrating OCR, users can seamlessly process scanned text, thereby expanding the range of accessible information and enhancing the overall functionality of our application.

The state-of-the-art in OCR technology currently employs a two-stage approach. A text detection model first detects and localizes text in images or scanned documents, and subsequently, a text recognition model recognizes the detected text and outputs it as readable text data.

Our project aims to integrate scanned documents seamlessly. The user should be able to use our RAG system with their scanned documents just as they would with any text-based document. To achieve this, the localization data from the detection model can be exploited to enable overlaying the recognized text on top of the source image. This approach allows us to provide the user with an editable, searchable document while preserving its original visual structure, effectively eliminating any compromise in user experience.

5.5 Text to Speech

The final building block in our AI toolset is Text-to-speech models. Text-to-speech (TTS) technology is an important component of our intelligent reading assistant, designed to enhance accessibility and user experience. TTS models convert written text into spoken words, enabling users to listen to their documents or books instead of reading them. This capability is particularly beneficial for individuals with visual impairments, those who prefer auditory learning, or users who wish to engage with content while multitasking.

The landscape of Text-to-Speech (TTS) technology has evolved significantly due to advancements in deep learning and deep generative models, such as transformer models, generative adversarial networks (GANs), and variational autoencoders (VAEs). These developments have enabled the production of highly natural human-like speech.

The latest state-of-the-art TTS models achieve remarkable performance in both synthesis quality and computational efficiency. They combine highly natural speech synthesis with efficient inference, ensuring high-quality, natural speech at very high real-time synthesis factors, thus allowing us to deliver a seamless auditory experience, making our reading assistant more accessible and versatile.

5.6 Model Research and Selection

With a clear understanding of the core technologies required to achieve our project's functionalities, we conducted extensive research to identify SoTA models and solutions that meet the necessary requirements to realize our vision.

In this section, we explore the models we selected for our final implementation of each module in the system, and the thought process behind their selection.

5.6.1 Large Language Models

For the core natural language understanding and generation tasks, we explored several options to integrate LLMs into our system. Initially, we considered a locally running quantized[1] instance of the 13B version of the LLaMA-2[2] model. However, obtaining the resources necessary to run even a quantized version of a large language model proved extremely challenging. Additionally, the significant drawbacks in performance[3] which emerge naturally from using a much smaller model compared to available paid options, led us to explore the latter.

After surveying the available solutions, most notably OpenAI's GPT-3.5 Turbo[4] and GPT-4 Turbo[5], as well as Google's Gemini 1.0 Pro[6], **GPT-3.5 Turbo** proved to be the most cost-effective option, providing better performance in various areas than Gemini 1.0 Pro (Table 5.1) and being significantly less costly than GPT-4 Turbo[7].

Table 5.1: Benchmarking Results of Various Available LLMs[8]

		Model			
Task	**Dataset**	Gemini Pro	GPT 3.5 Turbo	GPT 4 Turbo	Mixtral
Knowledge-based QA	MMLU (5-shot)	65.22	67.75	**80.48**	68.81
	MMLU (CoT)	62.09	70.07	**78.95**	59.57
Reasoning	BIG-Bench-Hard	67.53	71.02	**83.90**	60.76
Mathematics	GSM8K	76.42	78.01	**92.72**	71.65
	SVAMP	81.10	82.30	**92.60**	81.60
	ASDIV	85.31	89.07	**92.75**	83.16
	MAWPS	96.50	98.00	**98.67**	96.00
Code Generation	HumanEval	59.76	74.39	**76.83**	45.12
	ODEX	39.86	**52.62**	45.79	40.55
Machine Translation	FLORES (5-shot) Unblocked	53.31	52.43	**54.00**	40.97
	FLORES (5-shot) All	21.68	40.00	**48.24**	30.27
Web Agents	WebArena	7.12	8.87	**14.90**	1.39

5.6.2 Vector Databases

Vector databases play a crucial role in our system's ability to efficiently handle large-scale semantic searches. They store and manage vector embeddings, which are critical for performing fast and accurate similarity searches. Choosing the optimal vector database for

our system hinged mainly on criteria such as performance, integration, scalability, and support for advanced searching and retrieval techniques.

After evaluating various vector databases such as Weaviate[9], Chroma DB[10], and Pinecone[11] we selected **Weaviate** as our vector database solution for the following reasons:

- **Performance**: Weaviate demonstrated excellent search performance, leveraging HNSW indexing as well as gRPC and GraphQL APIs to ensure fast and accurate retrievals.
- **Scalability**: It is designed to scale efficiently, making it well-suited for handling large datasets and growing user bases.
- **Integration**: Weaviate offers seamless integration with various frameworks and APIs, aligning well with our infrastructure.
- **Advanced Features**: Beyond HNSW indexing, Weaviate supports additional advanced features such as hybrid searching capabilities. This allows it to combine multiple search strategies, such as vector similarity search and keyword-based search, optimizing retrieval accuracy across diverse types of queries.
- **Community and Support**: Weaviate has an active community and strong support, ensuring we can resolve issues promptly and stay updated with the latest advancements.
- **Deployment Options**: Weaviate provides flexible deployment options, including cloud-based, on-premises, and hybrid solutions, allowing us to start with a cost-effective setup and scale up our infrastructure as we grow.

5.6.3 Embedding Models

When selecting an embedding model to generate vector representations for the text splits obtained from user documents, several key considerations were taken into account. The model must produce sufficiently representative vector embeddings to enable accurate retrievals. Additionally, the dimensionality of these embeddings should be balanced to optimize search efficiency, and the embedding generation process should be efficient enough to accommodate embedding entire documents at scale.

Sentence Transformers are Transformer-based models that generate dense vector representations for entire sentences and bodies of text. Built on the transformer architecture, these models excel at capturing nuanced meanings across entire texts due to the transformer's multi-head attention mechanism and are particularly effective in tasks requiring deep semantic understanding and similarity. This makes them exceptionally powerful for retrieval tasks, enabling precise and efficient search and comparison of complex text data.

Our choice of embedding model fell on the **Alibaba-NLP/gte-large-en-v1.5**[12]. This sentence transformer is built upon a BERT-like encoder backbone[13] that incorporates a combination of optimizations most notably:

- Rotary Positional Encodings (RoPE)[14] to overcome context size limitations.

- Substituting conventional activation functions with Gated Linear Units (GLUs), which have been shown to improve transformer performance in language understanding tasks without computational drawbacks.[15]

Our selections of the model is attributed to the following reasons:

- **Performance:** The gte-large-en-v1.5 embedding model achieves SoTA scores on the MTEB[16] benchmark surpassing OpenAI's text-embedding-3-large[17] and various other larger models, especially at retrieval tasks, while maintaining a lower dimensionality of 1024 dimensions as compared to the former's 3072.
- **Model Size:** It is a much smaller model, with only 434 million parameters, it's 16 times smaller than its closest competitors[16], which makes it vastly faster and more efficient.
- **Context Size:** The model supports a context length of up to 8192 tokens, 16 times larger than most models in its size category[16].

These factors collectively make Alibaba-NLP/gte-large-en-v1.5 a superior choice for generating efficient and effective embeddings for our use case.

5.6.4 Reranking Models

While vector search via cosine similarity provides a quick and efficient way to retrieve relevant text splits, it often lacks the depth and nuance required for highly precise matching. This is particularly critical in our application, where understanding the subtleties of user queries and the context within documents is essential for delivering accurate and relevant responses. To address these limitations, we incorporate reranking models into our retrieval pipeline.

Reranking models enhance the initial search results by performing deeper contextual analysis and advanced semantic understanding. They operate at a token-level interaction, allowing for a more granular examination of the text. This additional layer of analysis significantly improves the relevance and precision of the final search results, ensuring that the returned text splits are not only contextually appropriate but also aligned with the user's specific intent.

Table 5.2 below compares the key differences between vector search and reranking, highlighting the strengths and limitations of each approach. This comparison underscores the complementary nature of these techniques and the necessity of reranking models to achieve the desired level of accuracy and relevance in our intelligent reading assistant.

The search for a reranking model was naturally constrained by the same context-limit requirements as the embedding model, since the reranker's inputs are the text splits obtained from documents. Additionally, inference speed was a priority due to the computationally expensive nature of rerankers.

Table 5.2: Comparison of Vector Search and Reranking for Retrieval Tasks

	Model	
Task	**Vector Search via Cosine Similarity**	**Reranker**
Interaction Level	Document-level embeddings	Token-level interactions
Computational Demand	Low	High
Most computation happens at	Offline, i.e. indexing time	Online, i.e. query time
Result	Broad but superficial matching	Highly relevant and precise matching
Strengths	- Fast and efficient - Simple implementation	- Deep contextual understanding - Advanced semantic analysis
Limitations	- Limited by lack of depth and context - May miss nuances of user intent	- Computationally intensive - Requires more sophisticated models
Best For	Provides a quick, efficient first pass	Adds depth, enhancing accuracy and relevance of final search results

We chose the **jinaai/jina-reranker-v1-turbo-en**[18] for several clear reasons. This model achieves SoTA scores on various reranking benchmarks while maintaining a relatively small number of parameters—only 37.8 million[18]. This efficiency is due to the model being a distilled[19] version of a much larger model, which originally had 137 million parameters[18]. Moreover, the model employs a non-absolute positional encoding in its encoder backbone, specifically Attention with Linear Biases (ALiBi)[20], allowing it to handle context windows of up to 8192 tokens, similar to its embedding counterpart.

In summary, this reranking model offers significant efficiency and performance advantages over its competitors, excelling in inference speed, which was our primary requirement, while still maintaining competitive accuracy.

5.6.5 OCR Models

In our quest to identify the most efficient and accurate OCR solution for our intelligent reading assistant project, we initially considered Tesseract[21], an established OCR engine known for its robustness and open-source accessibility. However, Tesseract proved unsuitable for our specific use case involving the processing of scanned documents. Its CPU-bound nature, being largely a rule-based approach, and lack of parallelization capabilities resulted in significant inefficiencies, particularly when handling large volumes of documents. Additionally, Tesseract struggled with the accuracy required for high-quality OCR in complex and varied document layouts, particularly scanned documents.
Given these limitations, we pivoted to a two-stage approach using deep learning models, which are inherently more efficient and can be parallelized on modern hardware, significantly enhancing performance. After evaluating several options, we selected a combination of DBNet[22] for text detection and CRNN[23] (Convolutional Recurrent Neural Network) for text

recognition. This combination leverages the strengths of both models to provide superior accuracy and efficiency.

DBNet for Text Detection: DBNet (Differentiable Binarization Network) is an advanced text detection model designed to localize text in images. It excels in detecting text in complex backgrounds and various orientations, making it ideal for scanned documents. To further enhance performance, we chose to use the MobileNetV3[24] backbone for DBNet. MobileNetV3 is a state-of-the-art lightweight neural network that balances accuracy and computational efficiency, making it particularly suited for deployment on resource-constrained environments.

CRNN for Text Recognition: For text recognition, we employed CRNN, which combines the strengths of convolutional layers for feature extraction and recurrent layers for sequence modeling. This hybrid approach allows CRNN to accurately recognize text, even in cases where characters are closely spaced or slightly distorted. Similar to our text detection model, we integrated the MobileNetV3 backbone into CRNN to boost performance and efficiency. The result is a highly capable text recognition system that can handle a wide variety of fonts and handwriting styles.

The final models we deployed are:

- **db-mobilenet-v3-large** for text detection
- **crnn-mobilenet-v3-large** for text recognition

This approach offers several benefits:

1. **Performance:** The combination of DBNet and CRNN with MobileNetV3 backbones achieves performance comparable to state-of-the-art (SoTA) cloud-based OCR solutions[25]. The models are able to accurately detect and recognize text in complex, scanned documents.
2. **Efficiency:** Utilizing MobileNetV3, both models maintain high accuracy while being computationally efficient, processing roughly up to 7 pages of scanned documents per second in our testing[26]. This efficiency is crucial for preprocessing large volumes of scanned documents at scale.
3. **Scalability:** The deep learning-based approach, with its ability to be parallelized, ensures that our OCR pipeline can scale effectively as the volume of documents increases.
4. **Cost Reduction:** By deploying these models locally rather than relying on cloud-based solutions, we significantly cut down operational costs. This local deployment also offers greater control over data privacy and security.

In summary, our choice of db-mobilenet-v3-large and crnn-mobilenet-v3-large for the OCR pipeline strikes a balance between high performance and efficiency, making it a robust and scalable solution for processing scanned documents in our intelligent reading assistant project.

5.6.6 Text-to-Speech Models

In the development of our intelligent reading assistant, it was important to select a TTS model that balances natural speech with efficient real-time synthesis. After extensive research and evaluation of various TTS models, we chose the Variational Inference with Adversarial Learning for End-to-End Text-to-Speech (VITS) model[27].

VITS stands out in the TTS landscape due to its ability to produce highly natural and human-like speech while delivering high real-time synthesis factors due its parallel architecture[27]. Unlike traditional two-stage TTS systems, which separate text-to-spectrogram and spectrogram-to-waveform generation, VITS integrates these stages into a single, end-to-end framework. This integration not only simplifies the training process but also enhances the coherence and quality of the generated speech.

Key features that influenced our selection of VITS include:

- **High Performance**: VITS leverages Conditional Variational Autoencoders (VAE), normalizing flows, and adversarial training to generate realistic and expressive speech. This combination allows VITS to capture the natural variability in human speech, such as intonation and rhythm, resulting in high-quality audio output.
- **Efficiency**: The parallel generation process of VITS significantly reduces inference time compared to autoregressive models. This efficiency is crucial for real-time applications, ensuring that our reading assistant can deliver prompt and seamless auditory feedback to users.
- **Expressiveness and Naturalness**: The use of a stochastic duration predictor and sophisticated modeling of speech variations enables VITS to produce speech that is not only intelligible but also expressive. This feature enhances the listening experience, making the synthesized voice sound more lifelike and engaging.

By integrating the VITS model into our intelligent reading assistant, we are able to offer a TTS solution that combines superior speech quality with efficient, real-time processing.

5.7 Pipeline Description

5.7.1 New Document Processing Pipeline and Uploading to Weaviate Vector Database

The pipeline is illustrated in Figure 5.1 below:

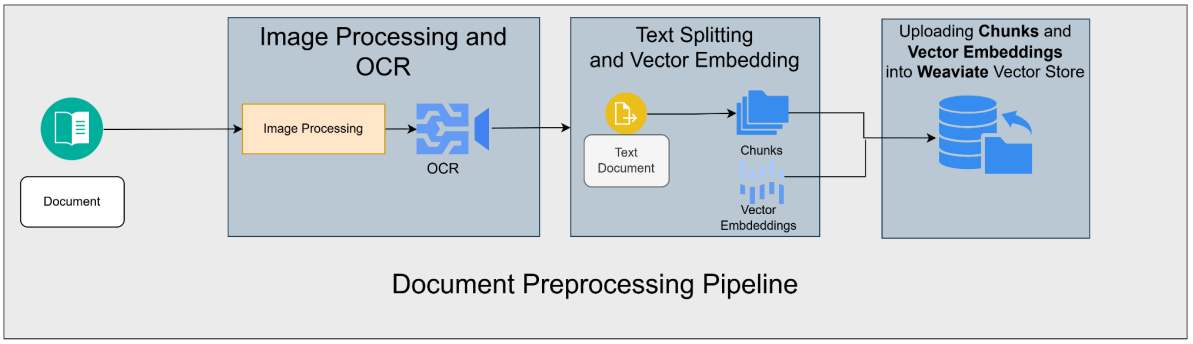

Figure 5.1: Document Preprocessing Pipeline

1. **OCR (Optical Character Recognition):** Convert non-textual documents (e.g., PDFs of scanned images) into a selectable textual format. This is achieved through image processing and using an OCR tool called docTR.
2. **Splitting Document Text into Chunks**: Split large text documents into smaller and more manageable chunks for context generation.
3. **Generating Embedding Vectors**: Create vector embeddings for each text chunk to facilitate semantic search. The output consists of vector embedding for each chunk.
4. **Persisting Document Chunks with Vector Embeddings**: Efficiently store text chunks and their corresponding vector embeddings.

5.7.2 Question Answering Pipeline

The Pipeline is illustrated in Figure 5.2.

1. **LLM Decision for Retrieving Chunks**: Determine whether chunk retrieval is necessary to answer the user's question.
2. **Retrieving Chunks Pipeline**: Retrieve relevant chunks from the vector database and the web.
    a. **Generating Retrieval Query**: Generate a precise and informative retrieval query based on the user's question and chat history summary.
    b. **Document Chunk Retrieval from Vector Database**: Retrieve relevant document chunks stored in the vector database.
    c. **Web Retrieval**: Use DuckDuckGo as the search engine for web retrieval.
    d. **Merging Document and Web Chunks**: Concatenate and format the retrieved chunks to create a coherent context, including metadata for reference.

3. **Question Answering**: Generate a response to the user's question using the retrieved context.
    a. **Input**: Combine the user's question, chat history, and merged chunks from both document and web retrieval.
    b. **Processing**: Pass the prepared input to the LLM to generate an answer.
    c. **Output**: Stream the generated response to ensure a seamless user experience.

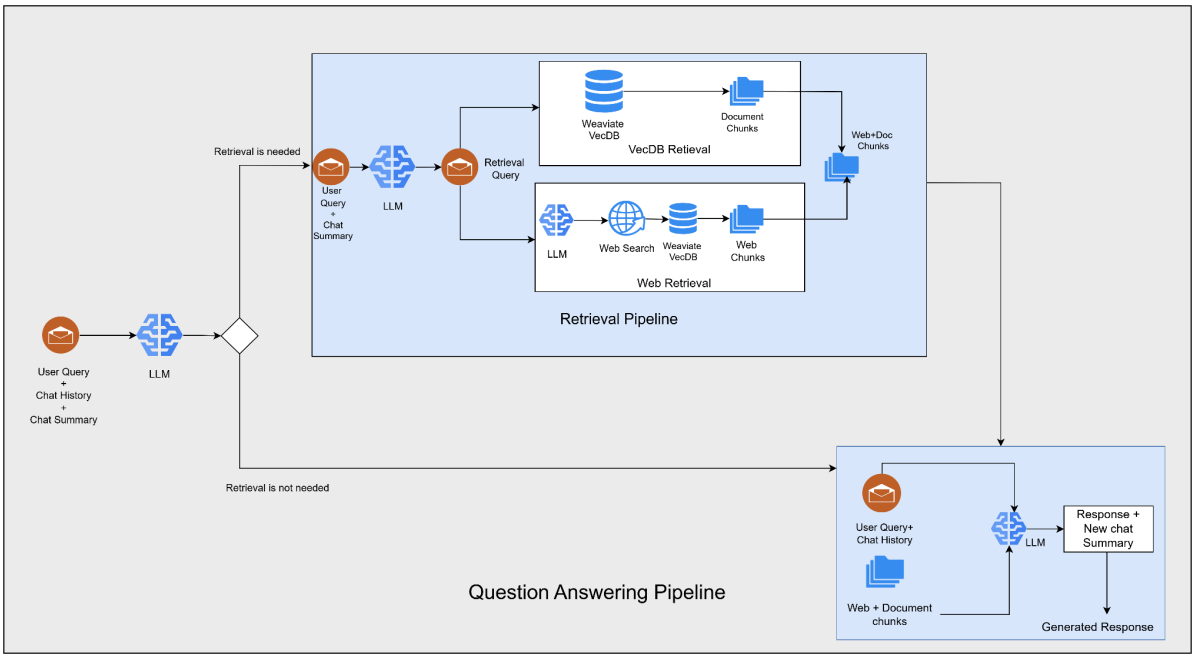

Figure 5.2: Question Answering Pipeline

5.8 Implementation

5.8.1 Preprocessing Pipeline (OCR and Embedding)

The goal of the preprocessing step is to generate searchable text chunks that persist in a Vector Database for later retrieval during query time. This is achieved through the following four steps:

1. OCR (Optical Character Recognition):

A. **Objective:** Convert documents into a selectable textual format, particularly those in formats such as PDFs or scanned images. This step is conducted only if the document is not already in a textual format.
B. **Tool Used:** docTR, a system enabling seamless use of Text Detection and Recognition models.
C. **Steps Involved:-**

a. **Image Preprocessing:**
    i. **Noise Reduction:** Remove noise to enhance text clarity (e.g., using filters).
    ii. **Normalisation**: Adjust the image contrast and brightness for uniformity.
    iii. **Adaptive Thresholding:** Use a computational method that adaptively thresholds the image, ensuring text is black and background is white.
    iv. **Deskewing:** Correct any tilt in scanned images to align text horizontally.
b. **Text Detection:** Use the db-mobilenet-v3-large model to detect and localise text.
c. **Text Recognition:** Use the crnn-mobilenet-v3-large model to predict text.
d. **Text Overlay:** Overlay text on the original document image using the hOCR file generated by the OCR process.

2. Splitting Document Text into Chunks:

A. **Objective:** Divide the document text into manageable chunks for context generation during query retrieval.
B. **Tool Used:** A LangChain class designed for text splitting.
C. **Method:** The text splitter splits large text documents into smaller chunks based on a list of characters. It tries to keep paragraphs, sentences, and words together as long as possible, splitting only when necessary.

3. Generating Embedding Vectors:

A. **Objective:** Create vector embeddings for each text chunk to facilitate semantic search.
B. **Tool Used:** Alibaba-NLP/gte-large-en-v1.5 text embedding model.
C. **Output:** Vector embeddings for each chunk, which will be used for context generation during user query retrieval.

4. Persisting Document Chunks with Vector Embeddings:

A. **Objective:** Store the text chunks and their corresponding vector embeddings efficiently.
B. **Tool Used:** Weaviate Vector Database.
C. **Function:** Persist and retrieve document chunks and their embeddings efficiently.

5.8.2 Retrieval

In this step, relevant context related to the user query is generated from document chunks stored in the vector database and, optionally, from web retrieval/searching. This process involves four steps:

1. **Decision-Making by Agent (LLM):** A large language model (LLM) determines if retrieval is needed to answer the user's specific question. It also decides whether web

retrieval is necessary or if the retrieving from document chunks in the vector database are sufficient.

2. **Generating Retrieval Query with LLM:** The LLM generates a retrieval query based on the user's question and the chat summary history. This query should be informative and clear to facilitate the efficient retrieval of relevant chunks.

3. **Retrieving Chunks:**
    a. **Document Chunks Retrieval:** Document chunks are retrieved from those stored in the vector database. A hybrid search approach is used, considering both semantic and keyword matching.
    b. **Web Retrieval:** Web searches are conducted using the DuckDuckGo browser to obtain relevant HTML files. These web pages are chunked, preprocessed, and stored in the vector database for the retrieval process. Relevant chunks are then retrieved and filtered with the reranker model.

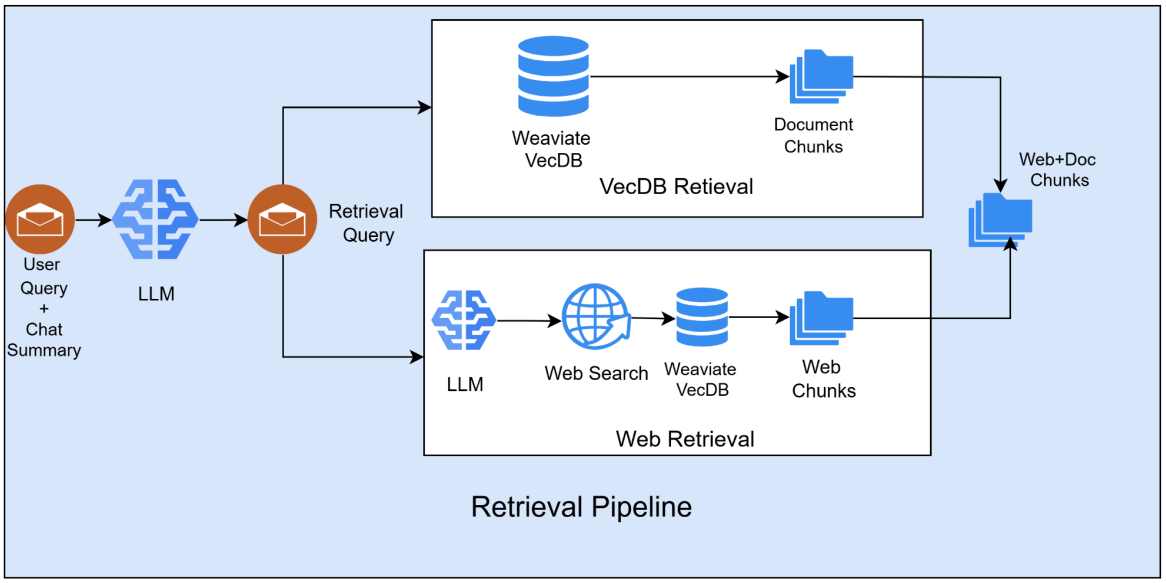

Figure 5.3: Retrieval Pipeline

5.8.3 Context and Response Generation

Auto-Merging Retrieval:

Fine-grained chunking of documents can enhance the relevance of retrieved content by omitting irrelevant information, and thus reducing the cost of large language model (LLM) prompts due to a smaller context size. However, this approach can also lead to information dispersion, where important contextual information becomes fragmented across separate chunks, reducing the completeness of the retrieved context.

To address this issue, we propose an auto-merging retrieval algorithm. This algorithm mitigates the reduced completeness caused by granular chunking by referencing smaller, related chunks (children) to their larger, aggregated form (parent) at multiple levels and combining multiple retrieved sibling chunks into a coherent context on retrieval.

In this approach, when the number of child chunks related to a specific parent chunk exceeds a defined threshold, all child chunks are merged to form the parent chunk, as illustrated in Figure 5.4. This merging process, combined with reranking (discussed in the following subsection), has demonstrated in our testing to significantly enhance answer groundedness and context relevance, while also reducing the cost of the LLM prompt compared to basic retrieval methods.

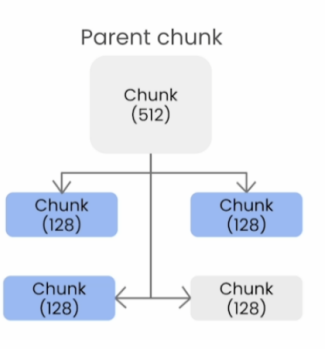

Figure 5.4: Auto-Merging

Reranking

The retrieved chunks from both the document and the web are filtered using a reranker which ranks and subsequently selects only the most relevant chunks for the user's specific question. These selected chunks are then concatenated to form the context needed to answer the user's query. Additionally, metadata is included with the text chunks as a reference.

Response Generation

Given the user's question, chat history, and generated context, the Language Model (LLM) generates a response. This answer is streamed to ensure a seamless user experience.

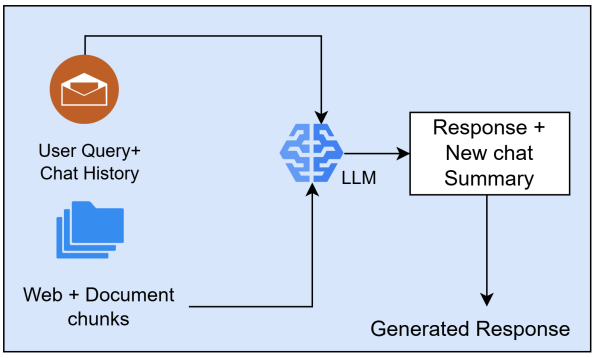

Figure 5.5: Response Generation

5.8.4 Large Text Summarization

When summarizing a large amount of text spanning several pages, the text often exceeds the context window limit of the LLM. To address this, we utilized an efficient technique known as map-reduce (Figure 5.6). In this technique, the text is split according to the LLM's context window size and summarized independently, generating multiple pre-summaries. These pre-summaries are then grouped and summarized into one final summary. This process can recur if the combined size of the pre-summaries still exceeds the context window limit.

To optimize this process, several considerations were made. After obtaining the text from all pages, it is split according to the LLM's context window limit, minimizing the number of pre-summary LLM calls while balancing the size of each call. All pre-summary calls are executed concurrently to reduce overhead. Finally, the final summary is streamed to ensure a seamless user experience.

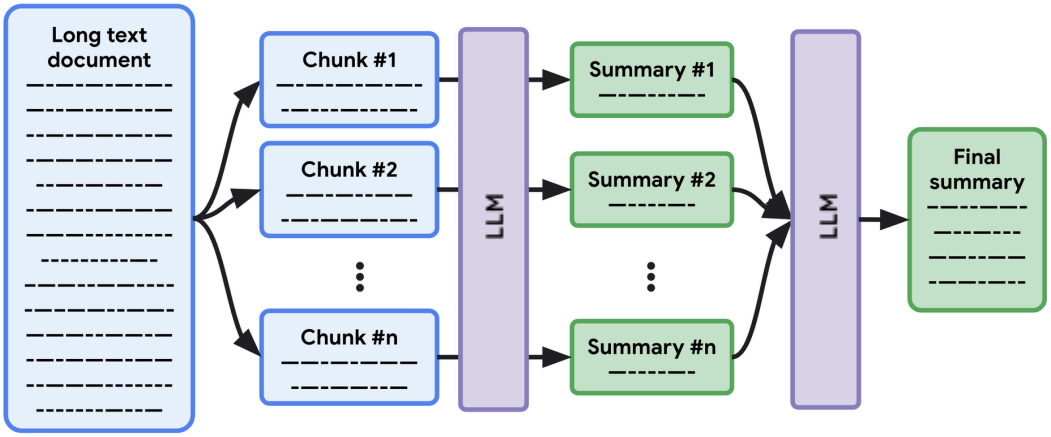

Figure 5.6: Map-Reduce Summarisation

5.8.5 TTS

As discussed in the previous chapter, TTS stands for Text-to-Speech, which refers to machine learning models that convert text into an audio format. The tool utilized for this purpose is Piper, a fast, local neural text-to-speech system that provides a straightforward interface for configuring and using multiple TTS models and voice options.

```
voice = PiperVoice.load(PIPER_MODEL_PATH, PIPER_CONFIG_PATH, use_cuda=False)
```

The primary features required for TTS are synthesizing voice from a given text or specific pages within a document. These tasks are inherently similar, as the latter involves extracting the text string by iterating over the necessary document pages. The process of generating an audio byte stream from a text string is demonstrated in the following code snippet within the synthesize_audio function. This function splits the string into lines, iterates over these lines, and synthesizes a stream for each line, as shown below:

```
def synthesize_audio(text: str):
     lines = text.split('\n')
     for line in lines:
       audio_stream = voice.synthesize_stream_raw(line, length_scale=1/speed)
           for audio_bytes in audio_stream:
               yield audio_bytes
```

5.9 Deployment

The AI models used in this project were deployed using FastAPI, a modern, fast (high-performance) web framework for building APIs with Python. FastAPI allows for easy and efficient deployment of machine learning models, providing asynchronous request handling. One of the key features that enhance the performance and scalability of our API is the use of background tasks.

5.9.1 API Structure

Multiple endpoints were implemented to provide different functionalities. Below is a brief overview of each endpoint and its purpose:

- **Add Document Endpoint:** Processes a document by either pre-processing it if it's text-based or applying OCR and then pre-processing it. The endpoint then generates embeddings and adds them to the vector database.
- **Delete Document Endpoint:** Deletes a document from the Weaviate Vector-Store.
- **Chat Response Endpoint:** Generates a new response for a user's question in the chat.
- **Text Summarization Endpoint:** Summarizes specified pages of the document.
- **Text to Speech Endpoint:** Generates an audio stream from a given text or specified pages of a document.

5.9.2 Background Tasks

FastAPI's support for asynchronous processing and background tasks plays a crucial role in our architecture. Background tasks offload intensive operations to run concurrently with other processes, optimizing resource utilization and enhancing API responsiveness. The process_document function handles document preprocessing, including text extraction and OCR, without blocking the main request thread. Similarly, the summarize_chat function updates chat summaries asynchronously, ensuring prompt user responses while additional computations are performed in the background. This design improves user experience by reducing latency and allows the server to manage multiple tasks simultaneously, increasing throughput and maintaining scalability. Integrating background tasks ensures complex workflows are handled efficiently, making the system robust and responsive even under heavy load.

5.9.2 Integration

To integrate our AI server with the rest of the system, careful design choices were made in the main back-end API. The main back-end server acts as an intermediary between the front-end client and the AI server, essentially serving as a gateway to its services. This design provides a consistent level of abstraction to the front-end client for all remote services it may call, including those of the AI server.

Additionally, this design abstracts the back-end server from the AI server, eliminating the need for the AI server to directly call back-end APIs. This separation of concerns allows the AI server to focus on its core functionality while relying on the main back-end server to handle client interactions and data management.

By acting as a gateway, the main back-end server simplifies the integration of the AI server into the greater system. This ensures seamless interaction between the different system components. Further details on the main back-end server's role in this integration can be found in Chapter 6: Back-End.

Chapter 6: Back-End

6.1 Introduction

The backend is a critical component of any software application. It refers to the part of the application that runs on the server, communicates with the database, and processes requests from the client-side. The importance of using a backend lies in its ability to handle complex business logic, data storage, and security aspects of an application. Another important aspect of using a backend is security. By storing sensitive data on the server side, the risk of data breaches and other security threats is minimized. Additionally, the backend can implement various security measures such as encryption, user authentication, and access control to protect against unauthorized access. In the following sections we will discuss the backend design and concepts including the database.

6.2 Database Design

The first step after understanding the flow of the project and all the desired features is to design the database schema accordingly.

6.2.1 What Is Database Design

Database design is the process of creating a detailed data model for a database. This involves identifying the data to be stored in the database, organizing it into tables and defining the relationships between these tables. The goal of database design is to create a structure that efficiently and accurately stores and retrieves the required data. It also involves defining the rules for how data can be accessed, updated, and deleted from the database.

6.2.2 Steps of Database Design

ERD Design: ERD design (Figure 6.1) is a visual representation of how data is organized in a database. It shows entities, attributes, and the relationships between them. Its purpose is to create a blueprint for building a database that meets specific requirements.

Document Data Model
Data is stored in documents, typically in JSON or BSON format.
Collections: Documents are grouped into collections.
Schema Flexibility: No strict schema; each document can have a different structure.

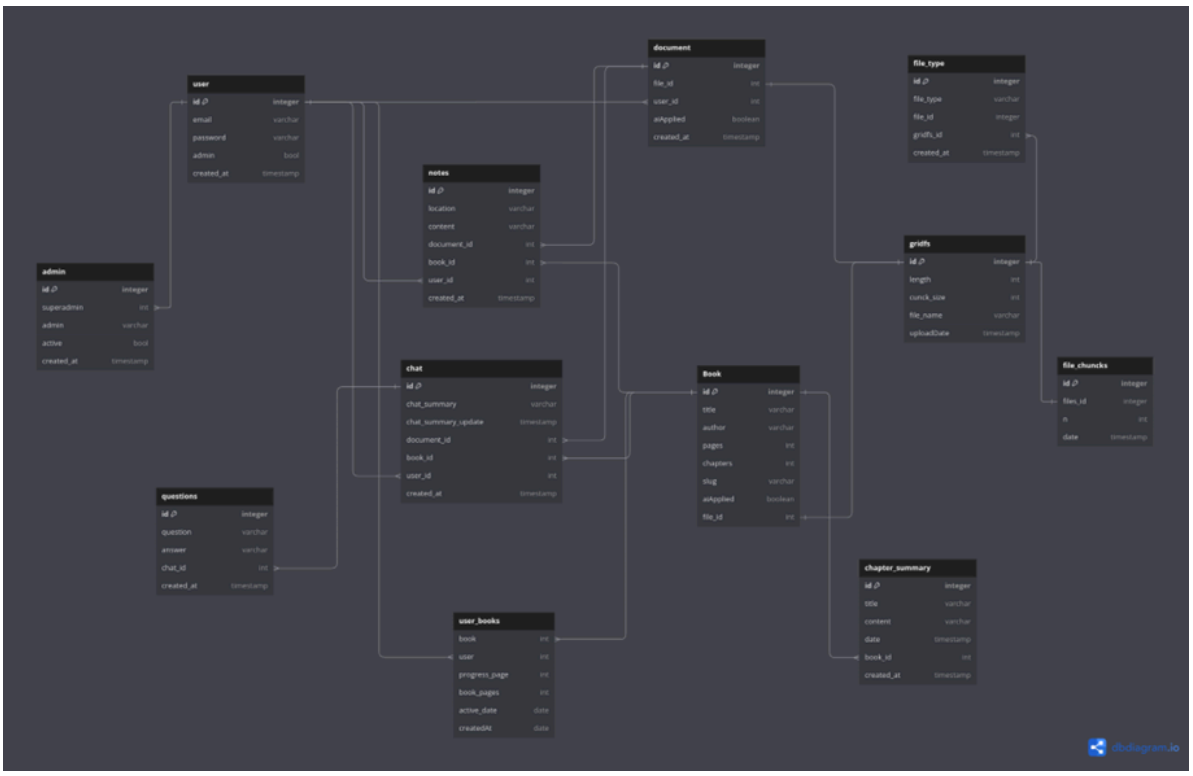

Figure 6.1: Database Design Schema

6.3 Application Design

6.3.1 RESTful Api

In our project we have adopted a RESTful API which is a type of web API that uses HTTP requests to interact with web resources. It is based on the principles of REST, a software architectural style that defines a set of constraints for creating scalable, distributed systems.

A RESTful API typically exposes resources as URLs, and supports a set of operations (called CRUD operations) for interacting with those resources:

1. Create (POST): Create a new resource.
2. Read (GET): Retrieve an existing resource.
3. Update (PUT/PATCH): Update an existing resource.
4. Delete (DELETE): Remove an existing resource.

6.3.2 Building a RESTful Api with Express.js

1. Install and set up Express.js
    a. In the terminal we have run the command “npm install express”.
2. Define routes
    a. Using the Express.js Router to define routes for our APIs, specifying the HTTP methods (GET, POST, PUT, DELETE), the route URLs, and the callback functions that handle the requests

In our project we have separated routers and controllers to ensure that routing logic is kept separate from the business logic, making the application easier to understand and manage.

6.3.3 Authentication - Authorization

We have provided a robust, secure and flexible Auth mechanism in our project through JWT and a lot of endpoints.
After signing up, logging in, resting password, resting email and updating password, a new jwt is sent to the client as a json response with expiration after 90 days.

Endpoints:

1. **User Registration: /auth/signup**
    a. An endpoint to allow users to create an account by providing their credentials.
    b. After the user request, an otp with expiration 5 minutes will be sent to his email to confirm his registration.
    c. Finally, the user must take otp and send it at the URL **/auth/confirm-your-account** to complete the registration process.
    d. If the otp becomes invalid, the user can make a request at the URL **/auth/resend-verification-email**. If an unconfirmed user tries to login, the app will ask the user to resend verification email at the previous URL.
2. **User login: /auth/login**
    a. User can login into the app with his email and password.
    b. After a login request it returns with the token which the flutter developer must save to indicate that it is an authorized user.
3. **Update user data : /auth/update-user-data**
    a. Users can update his data except for password and email.
4. **Update password: /auth/update-password**
    a. User provide his old password, email, new password to update his password
5. **Forget password : /auth/forget-password**
    a. When a user cannot remember his password during logging in, he can make a request to forget the old password then reset password to a new one.
    b. Then user takes the otp sent to his email to the URL /auth/reset-password
6. **Reset password: /auth/reset-password**
    a. When the user claims that he forgot the password, an email with OTP Token sent to the user to use it to reset the password, by entering the new password.

6.3.4 Security Consideration

- We have used the **node.bcrypt.js library** to hash user password before saving it in the database.
- For checking password we have used the **bcrypt.compare()** function.
- We have used the **crypto-js library** to save otp hashed database.
- We built a **hashOtp** function that used to save the otp sent to the user hashed in the database and hash the otp taken from the user to compare with the saved hashed otp.

6.3.5 Third party services used

We have used **Brevo** in the production environment and **mailtrap** in the development environment for sending email service to users.

6.3.6 Keep our database clean from unverified or virtual users

Previously we have said that the user must complete the process of registration to be able to use our application by requesting resend verification email, but If the user was not a real or he forgot about completing the process , We must save our database from unneeded data through running a cron job (It is a task running periodically) at 2 am to delete unverified users stayed in the database more than a half hour.

6.3.7 File Upload

We have used **Multer Middleware** to upload files from users.

Multer accepts an options object to specify the storage , file limit and file filter.

According to our business logic, we have specified that the uploaded file is an image in case it is a book cover, or a PDF in case it is a user document or a book. A document file will be uploaded by the user, with a size limit of 5 MB, and it will be available only to that user (no one else can access the document). On the other hand, a book file will be uploaded by the application admins. It is expected to have a larger size and will be available to all users of the application.After file is uploaded, it will be saved in the disk storage of the backend server then will be deleted after saving it in the database .

6.3.8 Storing Files

For storing PDF files in mongodb we have used **GridFS**. GridFS is a specification for storing and retrieving files that exceed the BSON-document size limit of 16 MB (Figure 6.2).

GridFS divides the file into parts, or chunks, and stores each chunk as a separate document. GridFS uses two collections to store files. One collection stores the file chunks, and the other stores file metadata. When you query.

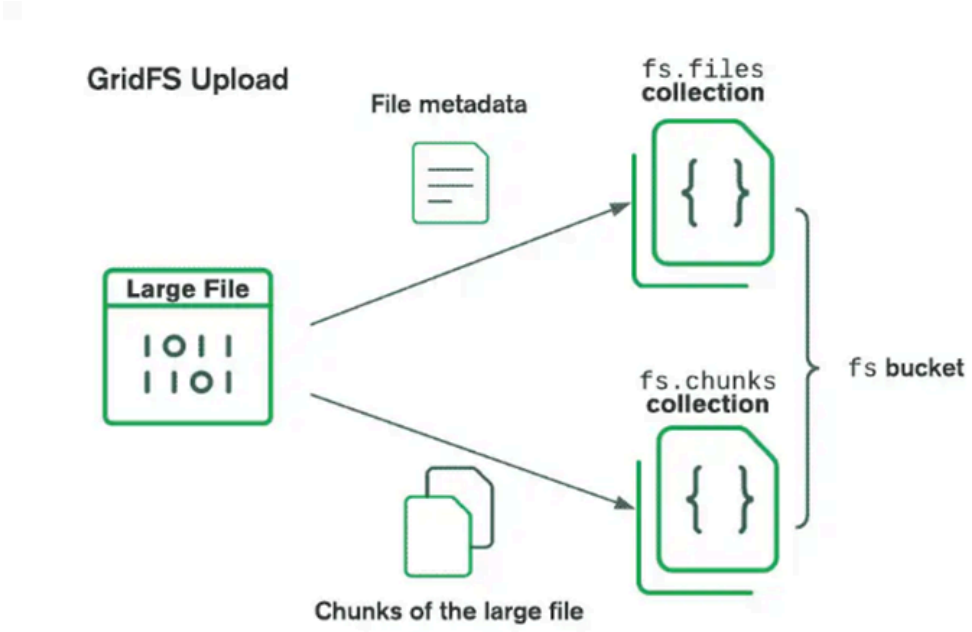

Figure 6.2: GridFS Collections and Chunks

GridFs Configuration:

- Create a bucket or get a reference to an existing one to begin storing or retrieving files from GridFS.
- **Upload File**
  - Use the openUploadStream() method from GridFSBucket to create an upload stream for a given file name.
- **Download File**
  - Use the openDownloadStream() method from GridFSBucket to create a download stream.
- **Delete File**
  - WE will delete fs.files, fs.chunks collections to delete the hall file.

Store Images:

We have used a third party service to store images called **Cloudinary**.

6.4 Backend Integration with AI

How do we manage ai-requests (chat, text-to-speech, files upload/update)?
How do we sync data with ai-api?

6.4.1 When a user uploads a document, how do we notify ai-api for that?

Upload document process shown in (Figure 6.3)

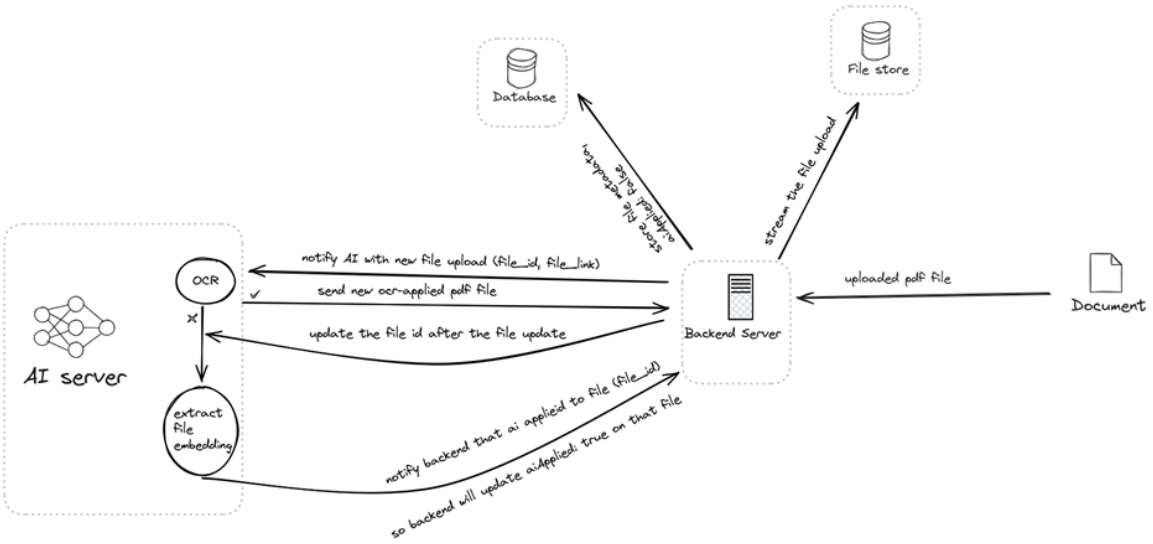

Figure 6.3: Upload Document Process

User ⇒ upload a document file, or admin: upload an official book.

Backend ⇒ save the uploaded file, marked that aiApplied: false, extract the file_id, if the file is document the lib_doc: false, if the file is book lib_doc: true to mark if the file may need ocr or not to ai, then send the request with file_id, lib_doc, file_link to AI.

AI ⇒ they take the file information, get the pdf file, scan if needed ocr? post the new ocr file to backend to update the file, in the backend response is the new file_id, and if not need ocr or after apply ocr, the file then go to the next phase to generate the word embeddings to the file, and after they complete their work. finally they post an acknowledgement to the backend that the file with that id, ai applied to it. in the backend we take the file_id, and from file_type table we search what is this file_id actually is a book or document, and then in the file type entity we marked aiApplied: true, so in flutter they open the option to apply ai operations in that file.

6.4.2 When a user asks a question, what data is being sent to ai-api and how do we manage that?

Question Answering Cycle shown in (Figure 6.4)

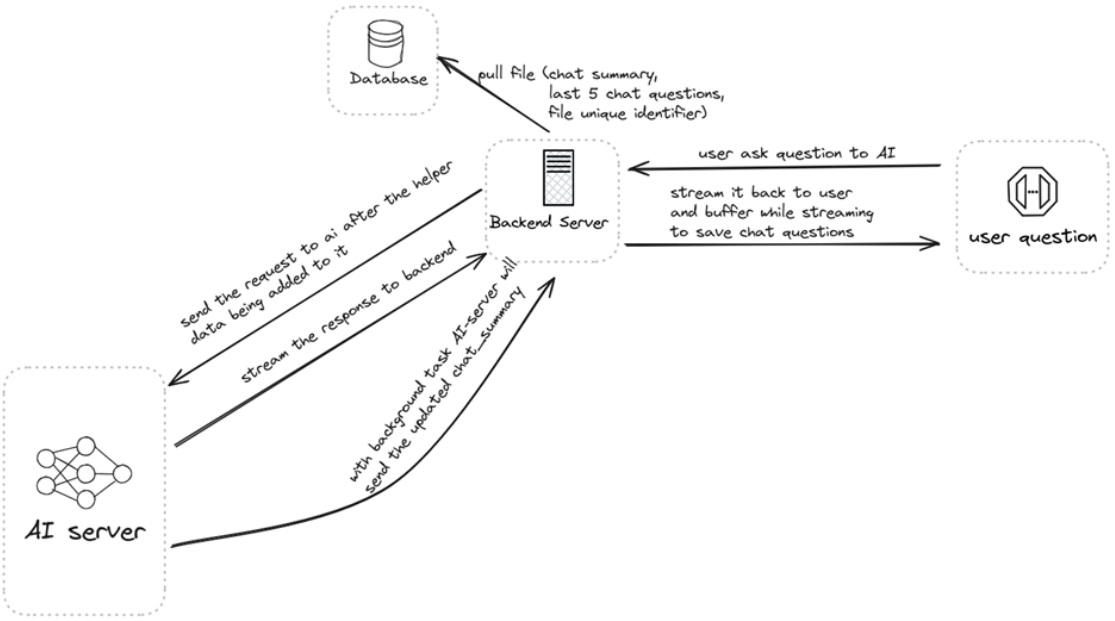

Figure 6.4: User Ask Question Process with AI

User ⇒ ask question in chat from the book/document.

Backend ⇒ take that question and id search with file type to the actual file_id with question asked in, and search in chat model where the file_id is what user asked for, and extract the chat_summary, then we send a request to ai with: question, file_id, chat_summary.

AI ⇒ they generate a response to the question and send it as a response in stream, and then in the background task they update the chat summary and post it to the backend to be updated.

6.4.3 Text To Speech

Text To Speech process shown in (Figure 6.5)

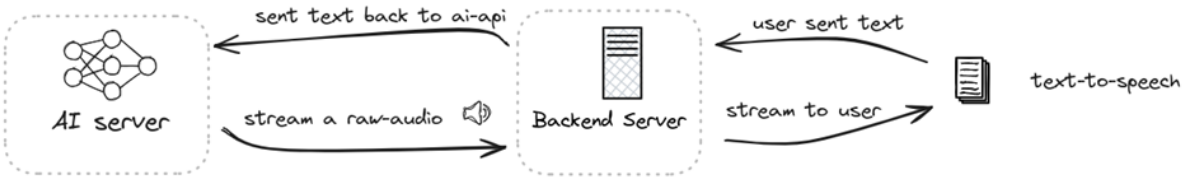

Figure 6.5: Text To Speech Pipeline

User ⇒ user send a text.

Backend ⇒ sent the text to AI.

AI ⇒ convert to speech, in row-audio format, stream it to backend, and backend the stream it back to flutter.

6.5 Admin Functionality

Admin Login shown in (Figure 6.6)

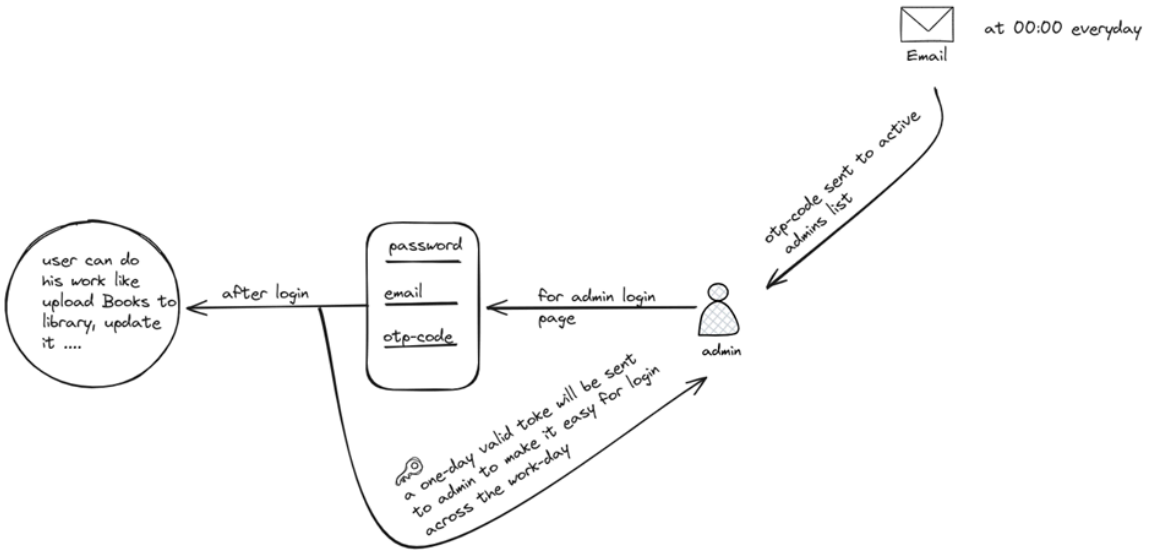

Figure 6.6: Admin Login

The superuser will add the admins emails list, but the state for each admin is not active.
For **Signup** ⇒ When an admin needs to sign up, it will activate his admin state, and his jwt token will be only a one-day valid token.

For **Login** ⇒ everyday there is an otp-code sent to active admins emails to let them use this code for login to apply two-factor authentication, to control admin activity, so if admin does not work one day we easily not send in login email to him.
After the Login ⇒ there is a one-day valid jwt token sent back to the user to make the login operation easy the next time.

Chapter 7: Deployment

7.1 Introduction

Deployment is a crucial step in making our application available to users. It involves setting up the necessary infrastructure and deploying the application code to a production environment. This section will discuss the deployment process for the application's backend (Node.JS Server).

7.2 Docker

We containerized our app using Docker to sustain the compatibility of the codebase across different machines. A simple Dockerfile would save much effort handling the configuration for each deployed instance.

Dockerfile:

```
FROM node:lts-alpine
ENV NODE_ENV=production
WORKDIR /usr/src/app
COPY ["package.json", "package-lock.json*",
"npm-shrinkwrap.json*", "./"]
RUN npm install --production --silent && mv node_modules
../
COPY . .
EXPOSE 3000
RUN chown -R node /usr/src/app
USER node
CMD ["npm", "start"]
```

Above is a simple Dockerfile that we used to get our app up and running as a container for easier deployment and a unified development environment.

7.3 Docker Compose

Docker Compose would be suitable for further development in case we need to add a load balancer, it would be easy to add a new docker container within our existing Docker Compose configuration file (shown below).

Docker-compose.yml:

```yaml
version: '3.4'

services:
  bookipedia:
    container_name: server
    image: bookipedia
    build:
      context: .
      dockerfile: ./Dockerfile
    environment:
      NODE_ENV: production
    ports:
      - 3000:3000
    env_file:
      - .env
```

Docker and Docker Compose made our development much easier and compatible across all our machines and on the remote server, where we deployed the server.

7.4 Deployment Options

We rented and configured a Linux VPS server from scratch to meet our needs. We used Docker and Docker Compose as our deployment technologies because of their modernity and rich features that enable easier development and deployment.

7.4.1 VPS Hosting

We approached Codescalers and they agreed to rent us a VPS Linux server with 1 CPU, 2GB RAM, and 25GB storage which is fairly enough for our app.

VPS server info is shown in Figure 7.1.

We installed the required packages, especially Docker and Docker Compose as they represent the backbone of the deployment process.

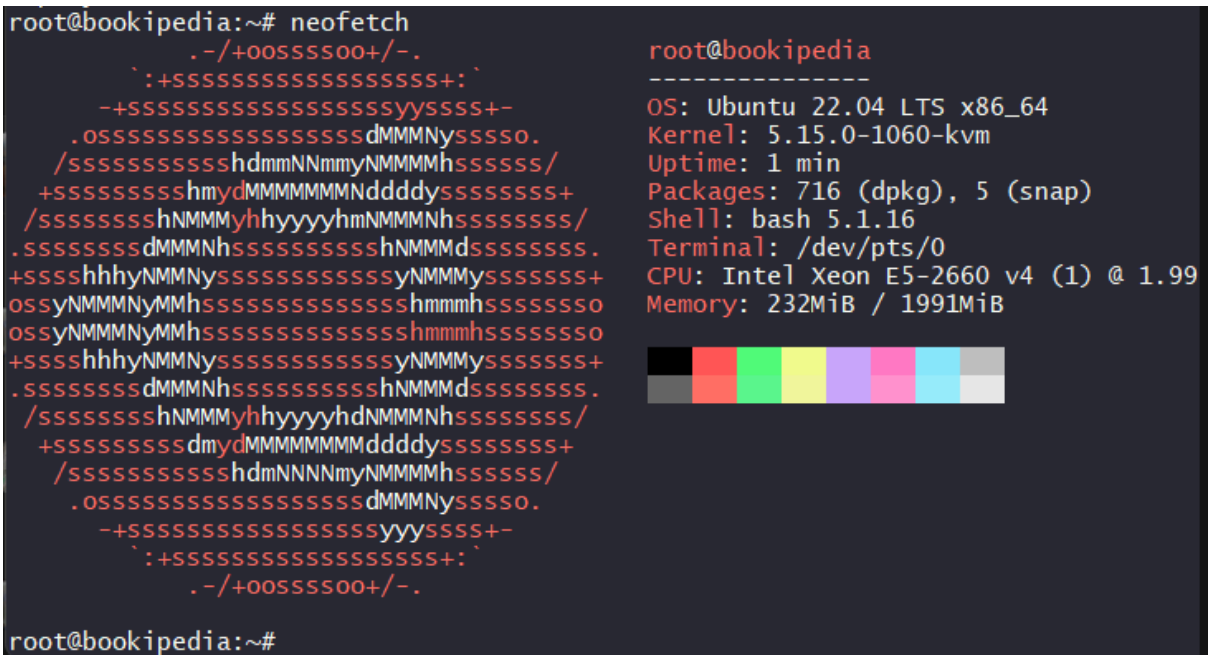

Figure 7.1: VPS Server Info

We created a bash script to allow easier deployment from our GitHub Repo to the server. It implements the following approach:

- Removing the old repository folder.
- Cloning the latest commit from the GitHub repo.
- Copying the environment variables (existing on the server) to the new repo folder.
- Stopping all the currently running containers.
- Removing all current docker containers.
- Removing all existing docker images.
- Navigate to the new repo folder.
- Build the Docker Compose and make it up and running in the background.
- Echo “deployed”.

Deployment Script:

```
#!/bin/bash

cd ~/app
rm -rf bookipedia

git clone
https://github.com/mhmadalaa/bookipedia

cp .env bookipedia/

containers=$(docker ps -a -q)
```

```
if [ -n "$containers" ]; then
    docker stop $containers
fi

docker rm $(docker ps -a -q)

docker rmi $(docker images -q)

containers=$(docker ps -a -q)

if [ -n "$containers" ]; then
    docker stop $containers
    docker rm $containers
fi

images=$(docker images -q)

if [ -n "$images" ]; then
    docker rmi $images
fi

cd bookipedia
docker-compose build && docker-compose up -d

echo "deployed"

# ./deploy.sh
```

Server usage is shown in Figure 7.2.

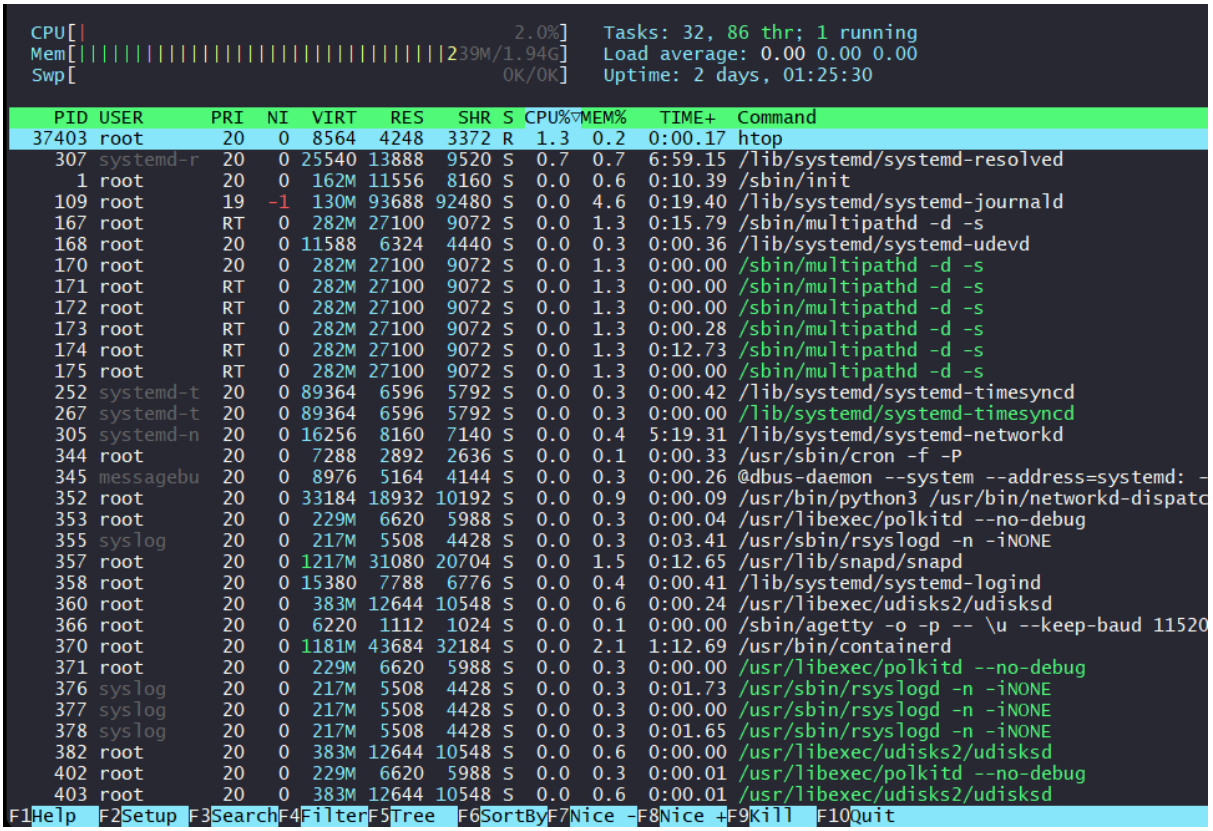

Figure 7.2: Server Usage

Conclusion

As we reach the culmination of this graduation book, we reflect on the profound journey of discovery and innovation that it encapsulates. This project represents not just a collection of technical achievements, but a testament to the dedication, hard work, and collaborative spirit of all those involved.

Throughout this book, we have delved into the cutting-edge techniques and state-of-the-art models that underpin the intelligent reading assistant. From the integration of large language models (LLMs) and retrieval-augmented generation (RAG) techniques to the utilization of optical character recognition (OCR) and text-to-speech (TTS) technologies, each chapter has unfolded a new layer of complexity and sophistication. These advanced methodologies have enabled the creation of a context-aware, interactive system capable of enhancing the user's reading experience in unprecedented ways.

The journey has not been without its challenges. We navigated the inherent limitations of LLMs, such as their lack of groundedness and context limits, by developing innovative solutions like the auto-merging retrieval algorithm and efficient large text summarization techniques. These efforts have significantly improved the relevance and coherence of the system's responses, ensuring a seamless user experience.

This project is a significant step forward in the realm of artificial intelligence and its application to real-world problems. It showcases the potential of AI to transform everyday tasks, making them more intuitive and accessible. As we look to the future, the advancements documented in this book lay a solid foundation for further exploration and development.

We extend our heartfelt gratitude to everyone who contributed to this project. Your expertise, passion, and perseverance have been the driving force behind this achievement. As we conclude this chapter, we remain excited about the endless possibilities that lie ahead and committed to pushing the boundaries of what technology can achieve.

Thank you for joining us on this remarkable journey.

References

[1] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, Dmitry Kalenichenko: “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference”, 2017;arXiv:1712.05877.

[2] Llama 2: Open Foundation and Fine-Tuned Chat Models | Research - AI at Meta

[3] meta-llama/Llama-2-13b · Hugging Face

[4] GPT 3.5 Turbo - Models - OpenAI API

[5] GPT 4 Turbo - Models - OpenAI API

[6] Gemini Pro - Google DeepMind

[7] Pricing | OpenAI

[8] Syeda Nahida Akter, Zichun Yu, Aashiq Muhamed, Tianyue Ou, Alex Bäuerle, Ángel Alexander Cabrera, Krish Dholakia, Chenyan Xiong, Graham Neubig: “An In-depth Look at Gemini's Language Abilities”, 2023; arXiv:2312.11444.

[9] Weaviate - Vector Database

[10] Chroma Docs

[11] Pinecone Docs

[12] Alibaba-NLP/gte-large-en-v1.5 · Hugging Face

[13] Alibaba-NLP/transformer++ · Hugging Face

[14] Su, Jianlin, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. "Roformer: Enhanced transformer with rotary position embedding.", 2024; arXiv:2104.09864.

[15] Shazeer, Noam. "Glu variants improve transformer.", 2020; arXiv:2002.05202.

[16] MTEB Leaderboard

[17] Embeddings - Models - OpenAI API

[18] jinaai/jina-reranker-v1-turbo-en · Hugging Face

[19] Nathan Brown, Ashton Williamson, Tahj Anderson, Logan Lawrence: “Efficient Transformer Knowledge Distillation: A Performance Review”, 2023; arXiv:2311.13657.

[20] Ofir Press, Noah A. Smith, Mike Lewis: “Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation”, 2021; arXiv:2108.12409.

[21] tesseract-ocr/tesseract: Tesseract Open Source OCR Engine

[22] Minghui Liao, Zhaoyi Wan, Cong Yao, Kai Chen, Xiang Bai: “Real-time Scene Text Detection with Differentiable Binarization”, 2019; arXiv:1911.08947.

[23] Baoguang Shi, Xiang Bai, Cong Yao: “An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition”, 2015; arXiv:1507.05717.

[24] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, Hartwig Adam: “Searching for MobileNetV3”, 2019; arXiv:1905.02244.

[25] docTR documentation

[26] All testing was done on an Intel Core i7-11850H CPU and an Nvidia RTX 3060-6GB (mobile) GPU at 16 GBs of RAM.

[27] Jaehyeon Kim, Jungil Kong, Juhee Son: “Conditional Variational Autoencoder with Adversarial Learning for End-to-End Text-to-Speech”, 2021; arXiv:2106.06103.