SWE 411 - Software Design Document Template

This template for this software design document (SDD) is adopted from the *IEEE Software Engineering Standards Collection*, IEEE Press and other SDD templates.

Title Page
- Name of document (refer to product, e.g., “Software Design Document for the …”)
- Date
- Team member names

Table of Contents
- Give page numbers for each section

1. Introduction

This Software Design Description (SDD) document presents a comprehensive design overview for the Muqin mobile application, a cutting-edge reading platform leveraging artificial intelligence to enhance the experience of book consumption. It serves as a blueprint for developers, providing detailed guidance on the app's architectural design, components, interfaces, and data management. This document captures the design decisions for the implementation phase, aligning with the previously outlined Software Requirements Specification (SRS).

The scope of the software extends to creating a seamless, intuitive, and accessible mobile application that caters to the modern reader, emphasizing the Arabic-speaking market. Muqin's design integrates innovative AI features such as book summarization, visual content generation, and text-to-speech capabilities, defining a novel trajectory for digital reading platforms.

1.1. Purpose

The purpose of this SDD is to guide the development team through a structured design approach, ensuring that the Muqin app is developed in line with the outlined requirements and best practices. This document will serve as a cornerstone for developers, architects, and designers who are tasked with the app's development. It will also be of interest to project stakeholders, such as project managers and testers, who will utilize it to understand the design logic and for developing test plans. Furthermore, it will help the documentation team in creating accurate support and technical materials for end-users.

1.2. Scope

The Muqin app, to be developed for both Android and iOS platforms, is designed as a comprehensive digital reading application. Its core functionalities are rooted in providing an enriched reading experience through features like AI-generated summaries, audio narration, and visual content generation.

The proposed software will:

- Facilitate reading by providing AI-powered book summaries.
- Offer an immersive reading experience with synthesized voiceovers.
- Enhance user engagement through AI-generated visual aids.
- Include a personalized user experience with reading history tracking and recommendations.

The software will not:

- Support printing functionalities.
- Allow for direct author-reader interactions within the app.
- Include features for live reading sessions or book clubs.

- Facilitate e-book conversion services.

The primary goal is to enhance accessibility and foster a reading culture within the Arabic-speaking community by harnessing AI capabilities. The success of the Muqin app will be measured by user adoption rates, satisfaction scores, and its contribution to promoting literacy.

1.3. References

- Software Requirements Specification for Muqin App, [Document ID], [Version Number].
- [Any additional books, articles, or documents that were used in the creation of the SDD should be listed here, with complete citations.]

1.4. Overview

The rest of the SDD is organized into several sections that delve into the different aspects of the software's design:

- **System Architecture:** Outlines the high-level structure of the app and the relationships between its major components.
- **Data Design:** Describes the data management strategy, database schema, and data flow within the application.
- **Interface Design:** Details the user interface, external interfaces, and the interactions between the app and its environment.
- **Component Design:** Breaks down the software into individual components and modules, providing a detailed specification for each.
- **Security Design:** Specifies the security measures and protocols implemented to protect user data and ensure privacy.

Each section includes diagrams, models, and narratives that collectively describe how Muqin will be constructed and how its parts will interact to fulfill the requirements as specified in the SRS.

1.5. Constraints

The design of the Muqin app operates under several constraints that are instrumental in shaping its architecture:

- **Language and Localization:** Designing the app to support Arabic, with the possibility of future extensions to other languages, imposes specific localization considerations.
- **Technological:** The reliance on AI and cloud services dictates the choice of technologies and platforms that must be used, impacting scalability and performance.
- **Content Licensing:** Agreements with publishing partners require adherence to legal and contractual stipulations, influencing content availability and distribution mechanisms.
- **Accessibility:** Ensuring that the app is usable by readers with disabilities mandates compliance with international accessibility standards.
- **Regulatory Compliance:** The app will adhere to data privacy laws applicable to the regions it operates in, potentially limiting certain features or functionalities.

These constraints are fundamental to the design process and will be revisited throughout the development to ensure compliance and effectiveness of the final product.

2. **System Overview** - Briefly introduce the system context and design and discuss the background to the project. Also add revised [Use case diagram]

This section offers an overview of the project's background by going into the Muqin mobile application's design and larger context. The goal of what follows is to provide a quick overview of the system's design, function, and operating environment.

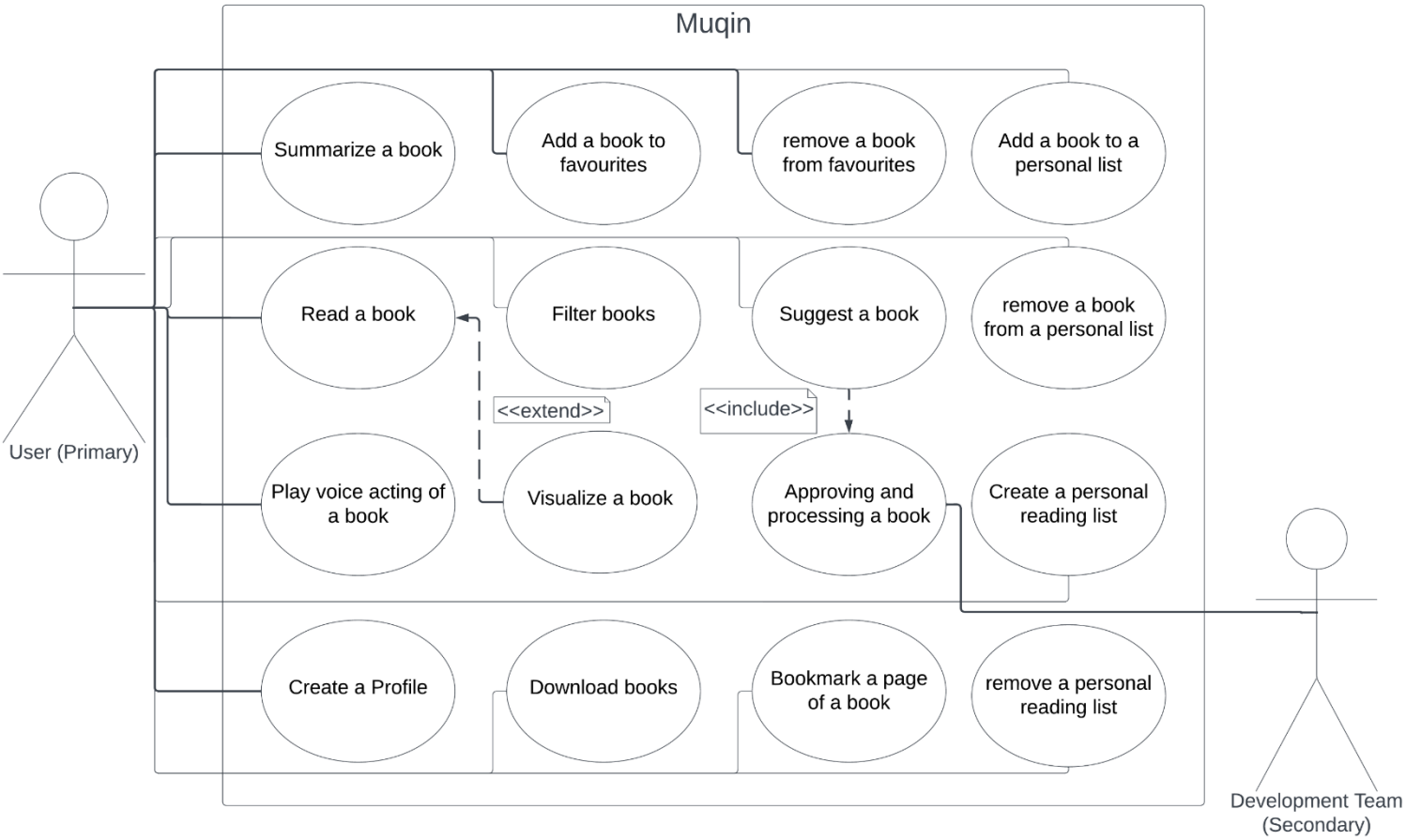
2.1 Overview

The Muqin mobile application is a reaction to the changing digital reading environment, especially in the Arabic-speaking community. Muqin aims to reinvent the reading experience through including innovative artificial intelligence into its fundamental features, with an increasing focus on technological innovation.

2.2 System Context

Supporting users on both the iOS and Android platforms, the Muqin app functions within the ever-changing mobile application environment. It integrates with a number of additional components, including cloud services for AI functions and data storage, to give readers a smooth, interactive reading experience.

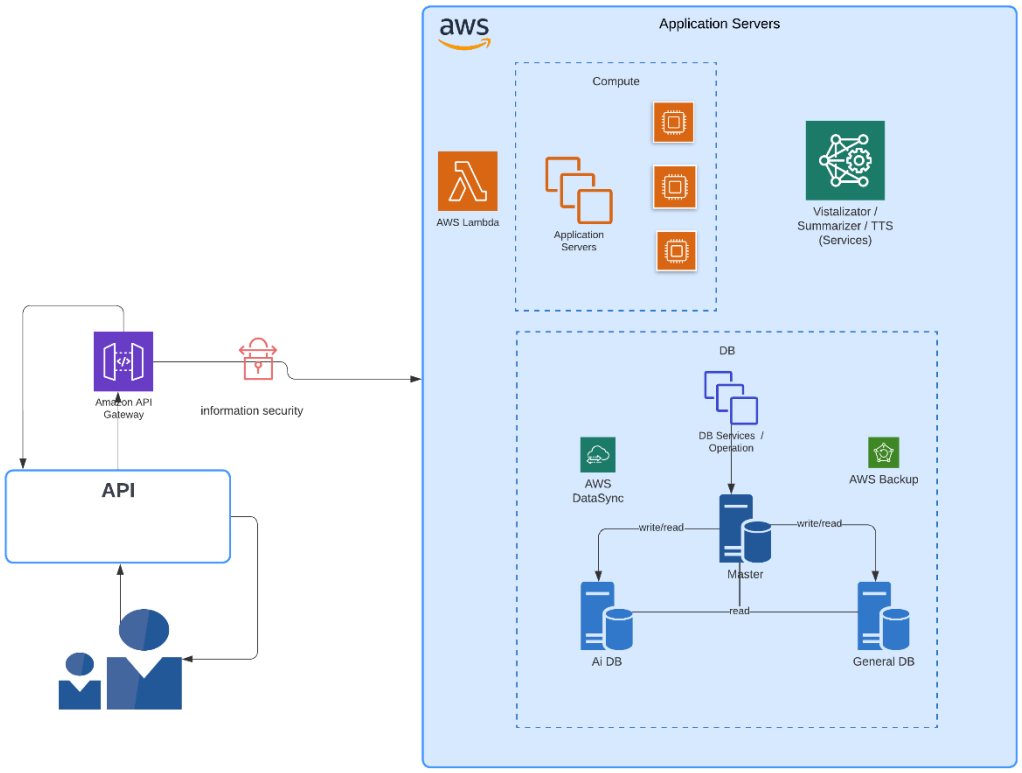
2.3 Revised use case diagram:



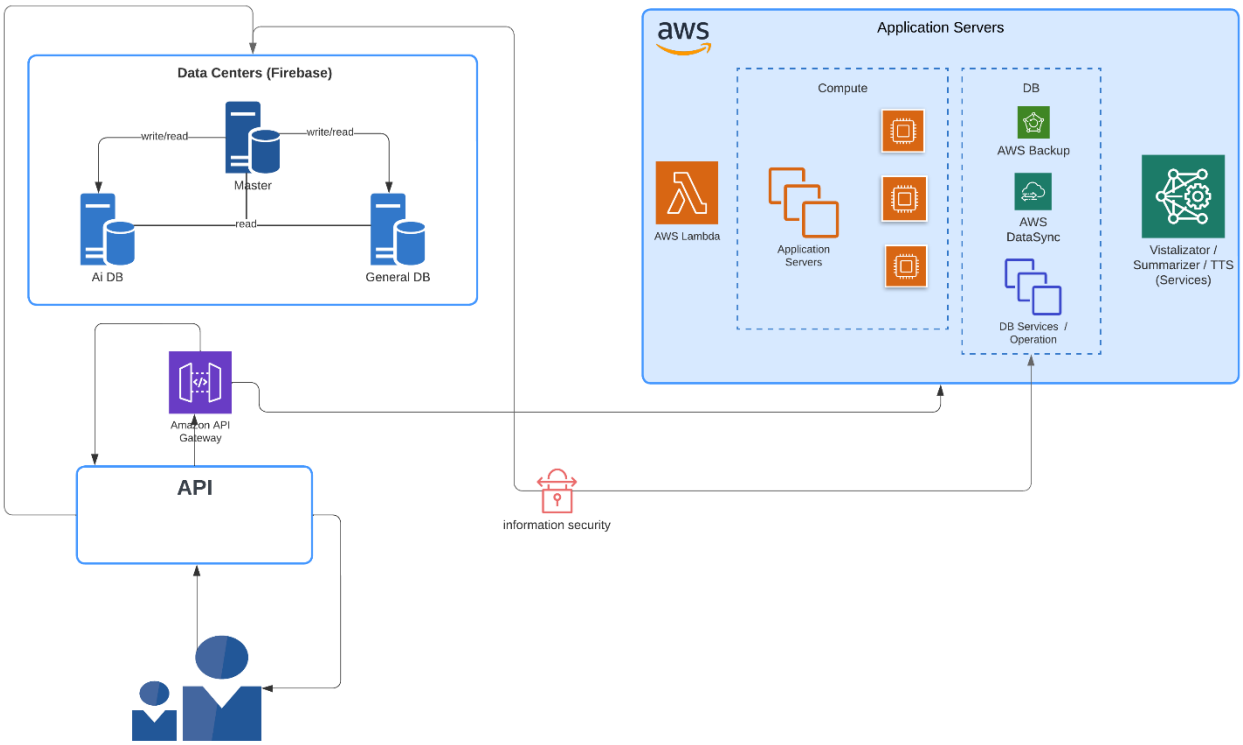
3. **System Architecture**

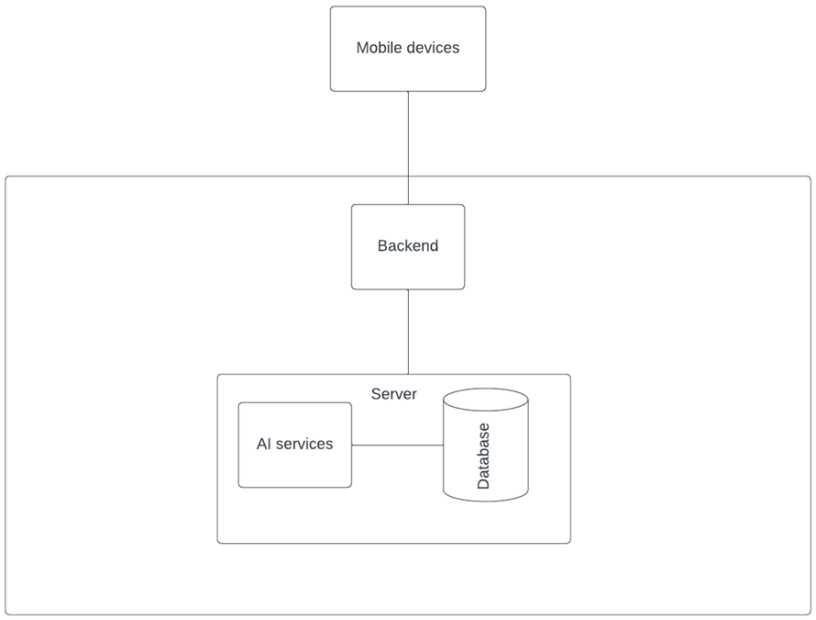
3.1. Deployment Architectural Description – [deployment architecture] – Also include a block diagram (for the deployment architecture) showing major subsystems and the interconnections among them.

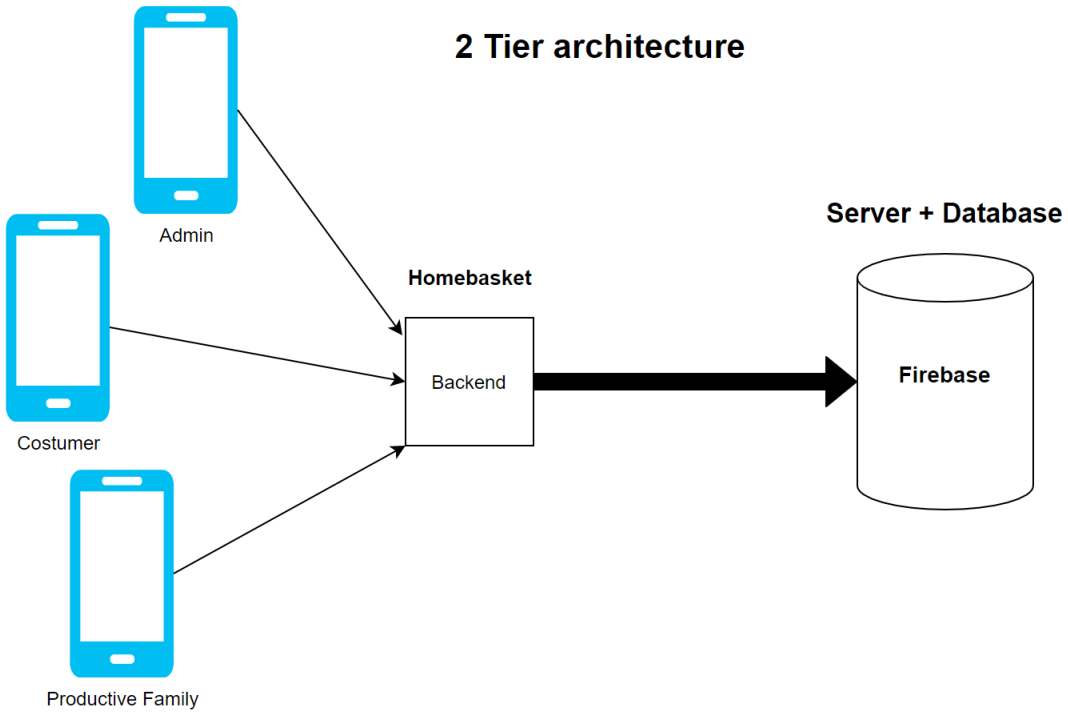
Deployment architecture provides a basis for understanding the physical distribution of the system across a set of processing nodes



3.2. Alternative Deployment Architectural Design Description (at least one) – [alternative deployment architecture] – Also include a block diagram (for the deployment architecture) showing major subsystems and the interconnections among them for an alternative architecture(s).







3.3. Design Rationale – Discuss and evaluate the differences between the designs. Discuss the rationale for selecting the architecture described in 3.1, including the critical issues and trade/offs that were considered.

Description of each component in both designs:

1. **User (Mobile Devices):** These represent the end-users' smartphones and tablets, which will run the Muqin app.
2. **API (Backend)**: The Muqin app running on users' devices, responsible for delivering the core functionalities of the app, including access to AI-generated book summaries, audio narration, and visual content generation through Amazon API Gateway.
3. **Application Servers:** These servers host the application's logic and services, including user tracking, recommendation algorithms, and AI services.
   - AI Services: These services are responsible for generating book summaries, providing voice synthesis for audio narration, and creating AI-generated visual aids.
   - AWS Lambda: Computing platform (event-driven, serverless)
   - Database management system (AWS Backup, AWS DataSync , DB Services / Operation)
4. **Data Centers (Firebase)**: responsible for storing books, some user information, cashs, and other relevant information.

Alternative design:

Pros:

1. Modularity: Each component can be optimized and scaled independently.
2. Improved Performance: Separation allows for fine-tuning of AI services and databases.
3. Scalability: Easier to scale individual components based on demand.
4. High Availability: Redundancy can be implemented separately for critical services.

Cons:

1. Increased Infrastructure Cost: Operating and maintaining multiple servers can be expensive.
2. More Complex Management: Requires separate configuration and maintenance for each component.
3. Latency: Data transfer between components may introduce latency.

Main Design:

Pros:

1. Cost Efficiency: Reduces infrastructure and operational costs by consolidating components.
2. Simplified Management: Fewer components to manage and maintain.
3. Reduced Latency: With fewer inter-component data transfers, there may be reduced latency.

Cons:

1. Limited Scalability: Combining components may limit the ability to scale specific services independently.
2. Reduced Modularity: Difficulty in optimizing and scaling individual components.
3. Single Point of Failure: If the combined server experiences issues, it affects multiple functionalities.

Comparison:

1. Scalability:

Alternative Architecture: Provides better scalability options for individual components.
Main Architecture: Scalability may be limited due to the combined nature of components.

2. Cost Efficiency:

Alternative Architecture: Can be more expensive due to the need for multiple servers.
Main Architecture: Offers cost savings through consolidation.

3. Management Complexity:

Alternative Architecture: More complex with separate components to configure and maintain.

Main Architecture: Simplifies management but introduces a single point of failure.

4. Latency:

Alternative Architecture: May have increased latency due to data transfer between components.

Main Architecture: Offers reduced latency by minimizing data transfer.

5. Modularity:

Alternative Architecture: Provides a high degree of modularity for optimizing and scaling components independently.

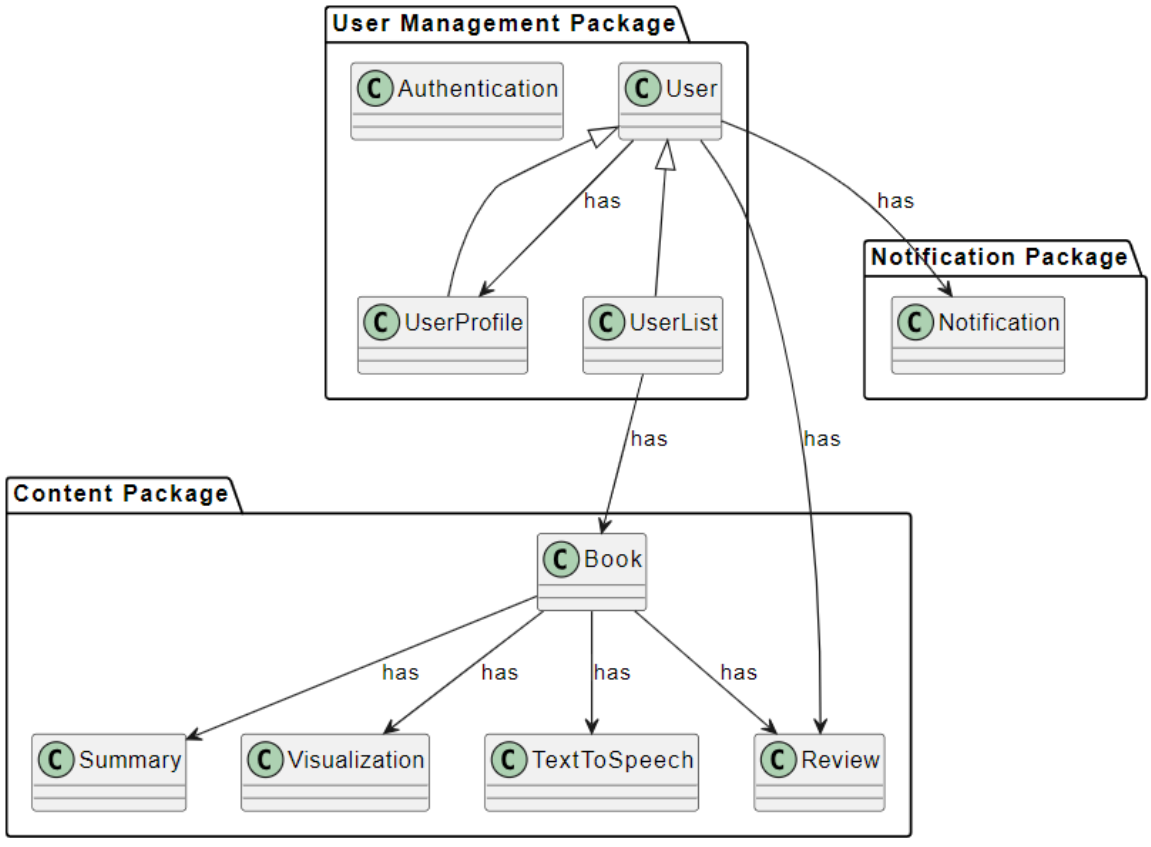
Main Architecture: Sacrifices modularity for cost savings.

Rationale:

We chose the main design for its low cost and easier implementation as the first design is a practical choice when the primary focus is on cost savings, simplicity, and efficiency. It's well-suited for projects with budget constraints, straightforward use cases, and those that prioritize faster development and reduced management complexity. However, it's important to acknowledge the limitations and trade-offs in terms of scalability and modularity when opting for this approach.

3.4. Component Decomposition Description [package diagram] – A decomposition of the subsystems that summarizes the software components. Describe your reason for this packaging (consider packaging principals, coupling and cohesion, etc.). Show all the classes in each package.

(High cohesion within modules and low coupling between modules are often regarded as related to high quality in OO programming languages.)



1. **User Management Package**:

- **Cohesion**: All classes in this package are centered around user-related functionalities. **User**, **UserProfile**, and **UserList** are closely related in terms of their functionality and data they manage. **Authentication** is crucial for user identity and access control, making it a natural fit in this package.
- **Loose Coupling**: By isolating user management functionalities in a separate package, changes in user management won't directly impact other unrelated functionalities like content management or notifications.

2. **Content Package**:
   - **Cohesion**: This package is dedicated to the core content of the application, i.e., books and their related features. **Book**, **Review**, **Summary**, **Visualization**, and **TextToSpeech** are all directly related to the content that users will interact with. Grouping them together ensures that any changes or enhancements to the content can be managed cohesively.
   - **Loose Coupling**: Keeping content-related classes separate from user management or notifications ensures that changes in one domain don't inadvertently affect the other.
3. **Notification Package**:
   - **Cohesion**: The sole class in this package, **Notification**, is distinct in its functionality. Notifications might be triggered by various events across the application, but their management and delivery are unique processes.
   - **Loose Coupling**: By keeping notifications separate, the system can easily integrate with different notification delivery mechanisms in the future without affecting other parts of the application.

Overall Design Principles:

- **High Cohesion within Modules**: Each package is designed to have high internal cohesion. This means that classes within a package have closely related functionalities and work together to achieve a specific subset of the application's overall functionality.
- **Low Coupling between Modules**: The packages are designed to be loosely coupled with each other. This ensures that changes or enhancements in one package have minimal impact on others, promoting maintainability and scalability.
- **Modularity**: The clear division into packages allows for better modularity, making it easier to manage, develop, test, and maintain the application. Each module can be developed and tested in isolation before being integrated into the larger system.
- **Scalability and Maintainability**: With this packaging, as the application grows, new classes can be easily added to the relevant packages without disrupting the existing structure. It also makes it easier to identify and fix issues since they're likely contained within a specific package.

4. Technology selection

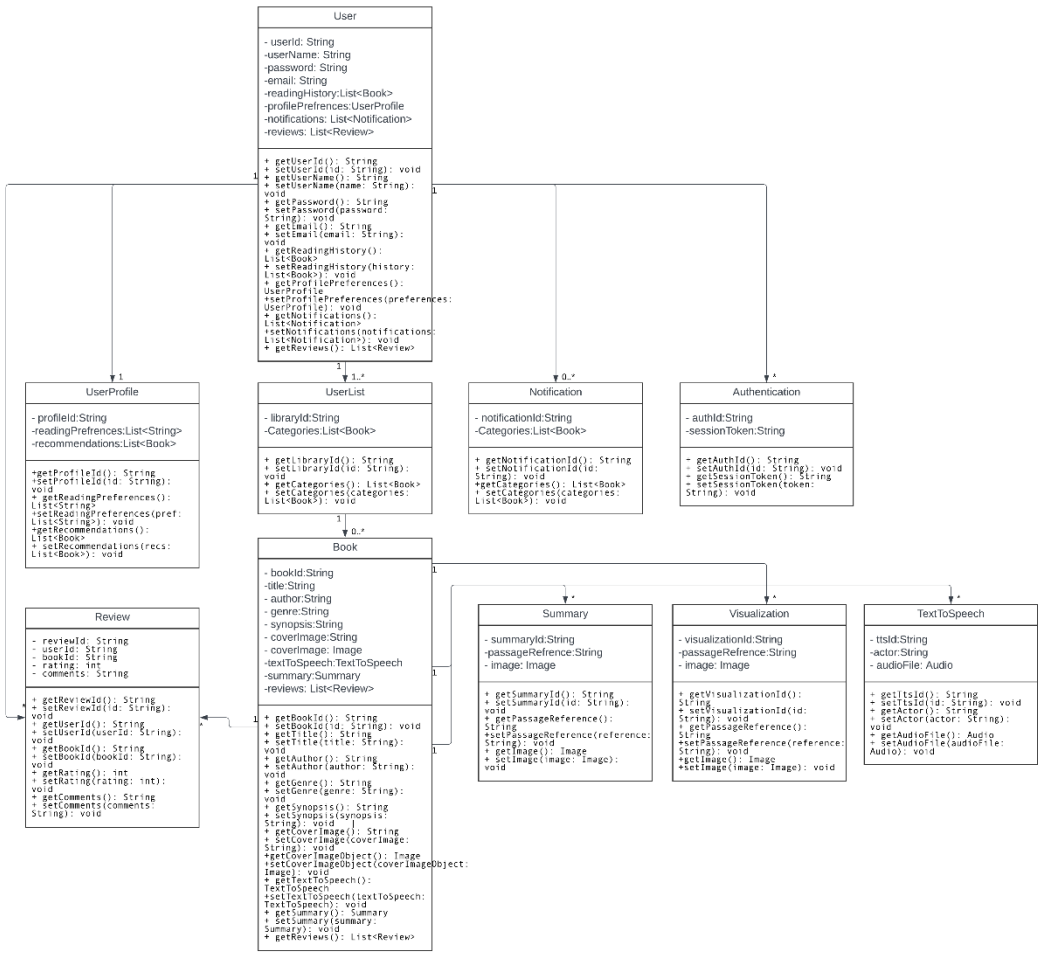
4.1. Provide detailed comparison of the available technology options such as programming language, database, hardware, etc. and justify your selection.

CROSS-PLATFORM MOBILE APPLICATION	For building the cross-platform mobile application, Flutter was selected as the development framework. Flutter uses the Dart programming language and allows creating native iOS and Android apps from a single codebase. Compared to alternatives like React Native or Xamarin, Flutter provides: ▪ Native performance and UI components rather than web or hybrid approaches ▪ Fast development cycle with hot reload support ▪ Excellent documentation and wide community support ▪ Options for native code integration when needed ▪ Growing ecosystem of packages and plugins
UX/UI DESIGN	For UX/UI design, Figma will be used given its real-time collaboration capabilities and smooth integration with Flutter via plugins to bring designs into code.
BACKEND	For the backend, Firebase was chosen to provide several services: ▪ Firebase ML for hosting and serving the AI/ML models built using TensorFlow. This leverages Google's advanced infrastructure for machine learning. ▪ Realtime Database for structured user data like profiles, libraries, analytics etc. It provides synchronization and offline support. ▪ Cloud Firestore as a flexible NoSQL database for unstructured content like books, summaries, multimedia. It scales easily. ▪ Cloud Storage for storing binary files like images, videos, audios. Integrates well with CDNs for fast delivery. Firebase was preferred over alternatives like AWS, Azure or custom servers because of its tight integration with Flutter, extensive documentation, generous free tier for projects of this scale.
AWS (AS SCALING UP)	For the initial launch and MVP of Muqin, Firebase provides an ideal backend solution. It allows quick development and iteration without managing server infrastructure. However, as the application usage and data grows exponentially, we plan to migrate our primary database to AWS RDS. Firebase Realtime Database and Cloud Firestore costs increase rapidly with scale due to their pay-as-you-go model.
	For natural language processing capabilities like summarization, we will use: ▪ OpenAI ChatGPT API - Generative language model fine-tuned on books to generate summaries.

AI FEATURES	For image generation and content visualization: ▪ OpenAI DALL-E 3 - Leading text-to-image AI system to create illustrations from book descriptions. ▪ RunwayML (alternative) - APIs for generating graphics if needed.
	For text-to-speech conversion: • 11ElevenLabs TTS API - Provides natural sounding speech in Arabic language.
	Note : We might fine-tune some of these pretrained models on Arabic data to better adapt them.
VERSION CONTROL	For version control, code hosting, and project management, GitHub and Git were chosen as the de facto standards that are well-supported across countless tools and integrate nicely with Flutter.

5. **Component Design**

5.1. System Components: [Detailed Class Diagram] - A top-down description of the design components. Describe the good design principles that have been used when designing the classes (coupling and cohesion, etc.). Include all attributes, methods and relationships etc.



1. **User:**
   - Represents a system user with attributes for user identification, account settings, and personal details.
   - Methods allow for actions such as logging in, setting preferences, managing passwords, reading user history, managing notifications, and reading reviews.
   - Relationships with UserProfile, UserList, Notification, and Review suggest that a user has associated profiles, lists, notifications, and reviews.
2. **UserProfile:**
   - Represents the profile of a user.
   - Methods allow for the management of profile data such as preferences, lists of books, and reading history.

- It has a relationship with Book indicating that a user's profile can contain a list of books.

3. **UserList:**
    - Represents a list or collection of books a user might have (e.g., a reading list).
    - Provides methods to get or update the list of books.
4. **Book:**
    - Represents a book with associated metadata and content.
    - Methods support actions like reading, borrowing, returning, rating, and adding to a reading list.
5. **Review:**
    - Represents a user's review of a book.
    - Provides methods to write, read, edit, and delete reviews.
6. **Notification:**
    - Represents notifications that can be received by a user.
    - Methods allow for the management of notifications.
7. **Authentication:**
    - Represents the system's authentication mechanisms.
    - Methods facilitate logging in, logging out, and validating session tokens.
8. **Summary:**
    - Represents a summary of the book.
    - Provides methods to manage and view summaries.
9. **Visualization:**
    - Represents a visual representation of the book.
    - Methods allow for visual content to be managed and viewed.
10. **TextToSpeech:**

- Represents an audio representation of the book.
- Provides methods for managing and playing back audio.

Design Principles:

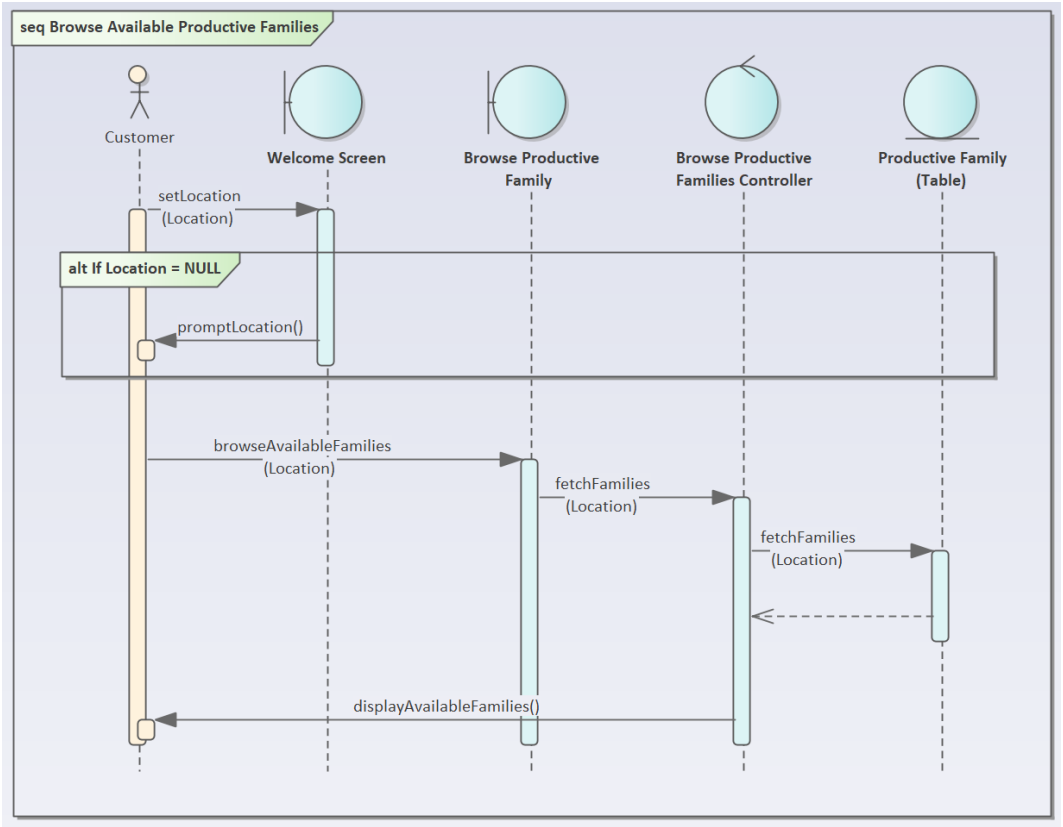
1. **Cohesion:**
    - Each class has a clear, singular responsibility. For instance, the User class is responsible for user-related functions, the Book class manages book data, and the Review class handles reviews.
    - This means each class has high cohesion, which is a sign of a well-designed system.
2. **Coupling:**
    - The diagram shows relationships between classes, which can indicate coupling. However, the design seems to be using association relationships, which is a looser form of coupling.
    - For example, the User has relationships with UserProfile, UserList, Notification, and Review, but these relationships appear to be associations rather than tighter forms of coupling like aggregation or composition.
3. **Encapsulation:**
    - Each class has private attributes and provides public methods to access and manipulate those attributes. This ensures data is protected and only accessible in controlled ways.
4. **Inheritance & Polymorphism:**
    - While not explicitly shown in the diagram, if there are subclasses of Book, Review, or other classes, then inheritance and polymorphism principles could be applied for a more flexible design.
5. **Dependency Inversion:**

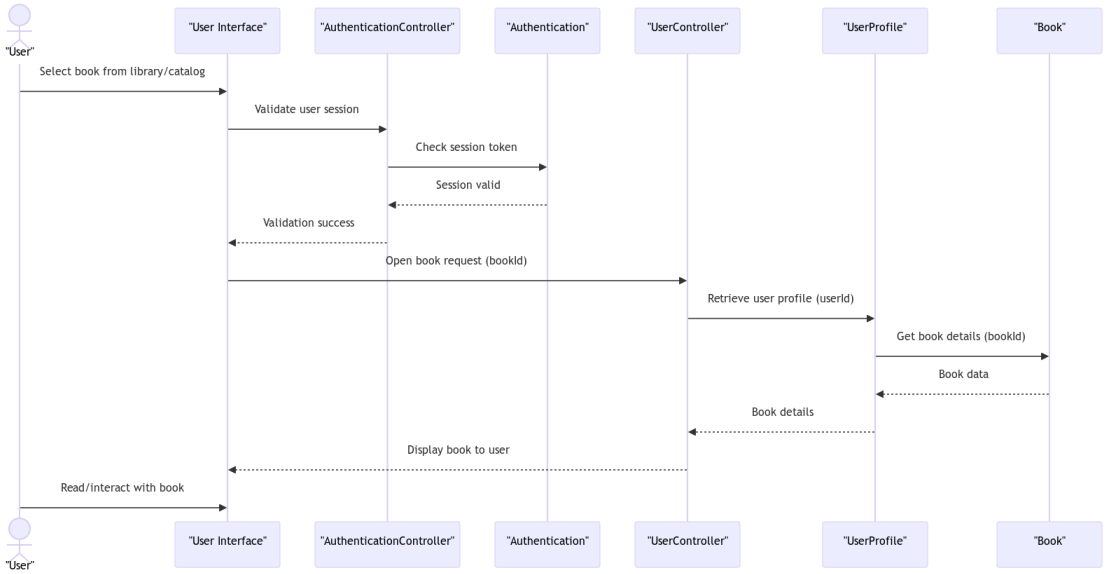
- Higher-level modules (like User or Book) do not appear to depend directly on lower-level modules. Instead, they rely on abstractions like methods.

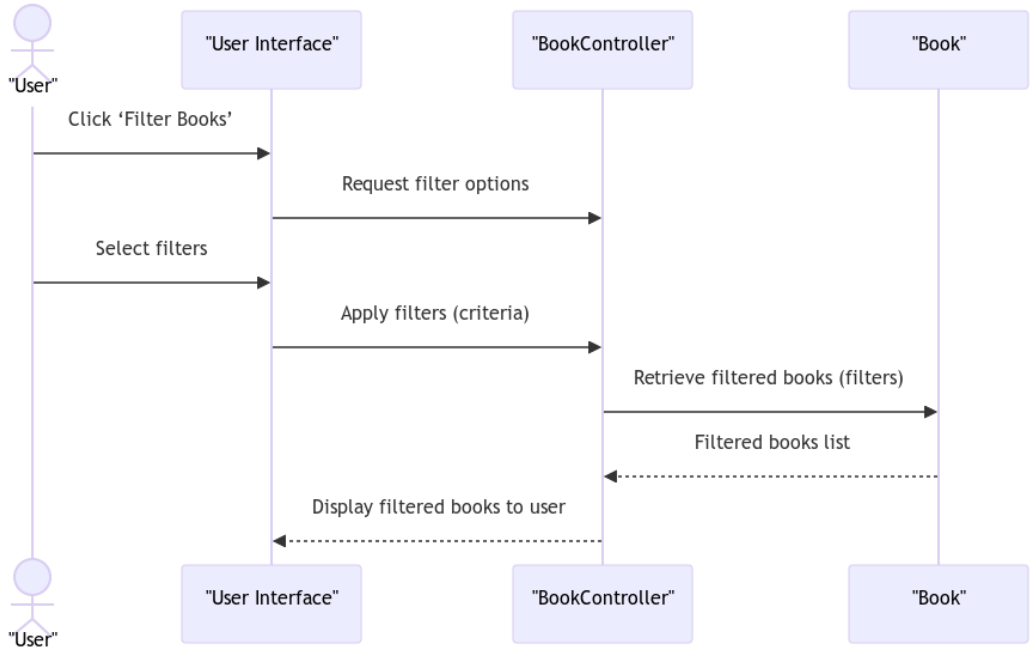
6. **Separation of Concerns:**
   - Classes like Authentication, Visualization, and TextToSpeech are good examples of separating concerns. Instead of lumping all functionalities into the User or Book classes, specialized classes handle specific tasks.

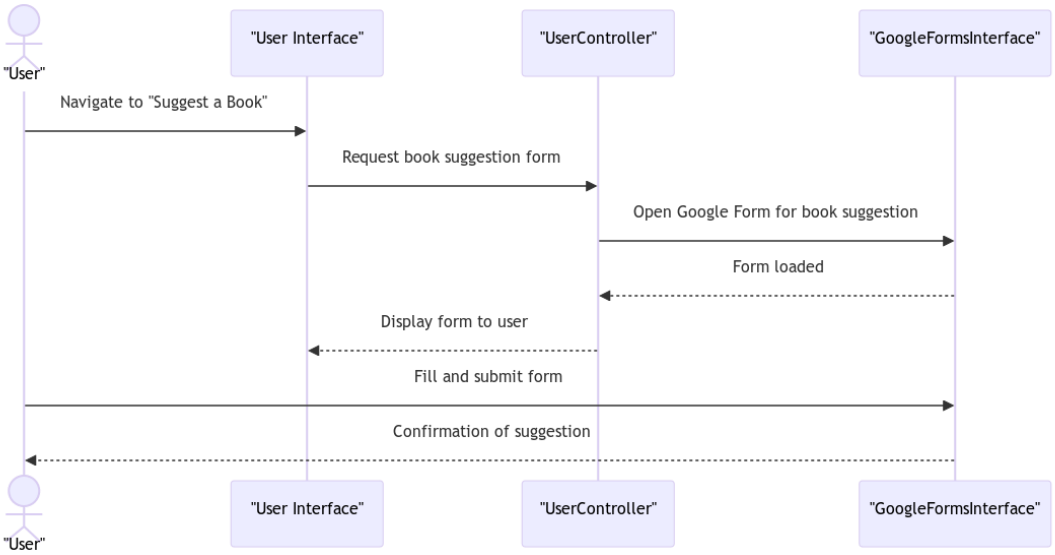
In conclusion, the diagram seems to follow many good design principles, ensuring a system that is maintainable, scalable, and flexible. Proper application of OOP principles like cohesion, coupling, and encapsulation further makes the design robust and efficient.

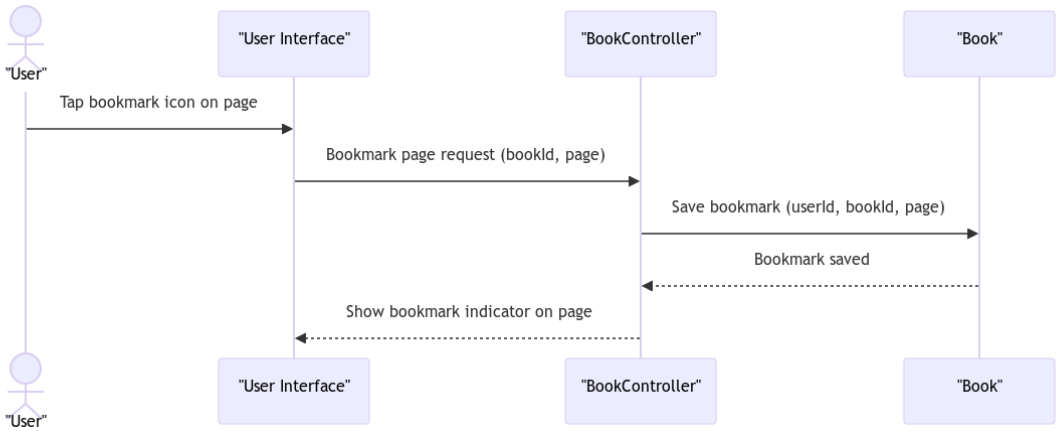
5.2. Function - What does the component do? Description of its processing. [sequence diagram]. Complete sequence diagram with interfaces, controllers and entities etc

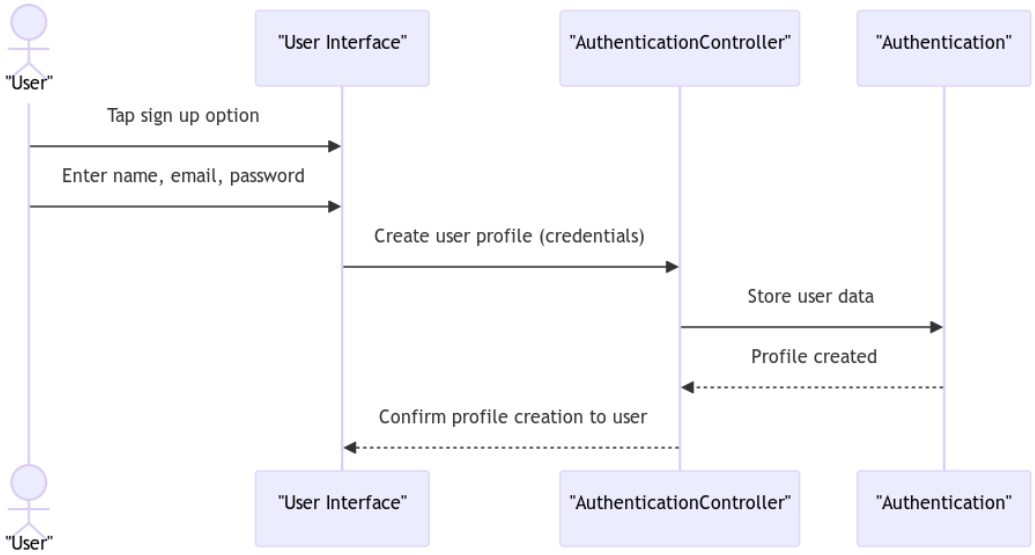


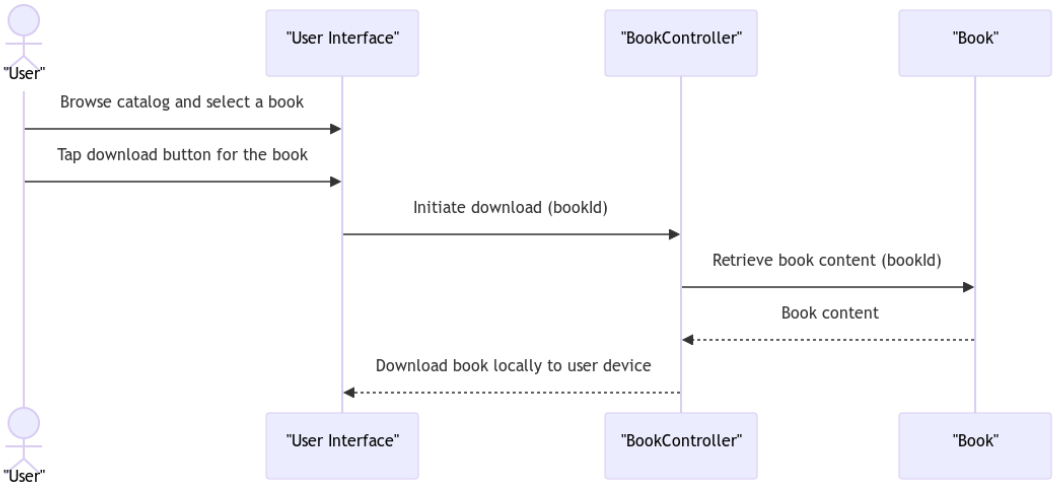

"User"
"User Interface"
"AuthenticationController"
"Authentication"
"UserController"
"UserProfile"
"Book"
Select book from library/catalog
Validate user session
Check session token
Session valid
Validation success
Open book request (bookId)
Retrieve user profile (userId)
Get book details (bookId)
Book data
Book details
Display book to user
Read/interact with book
"User"
"User Interface"
"AuthenticationController"
"Authentication"
"UserController"
"UserProfile"
"Book"

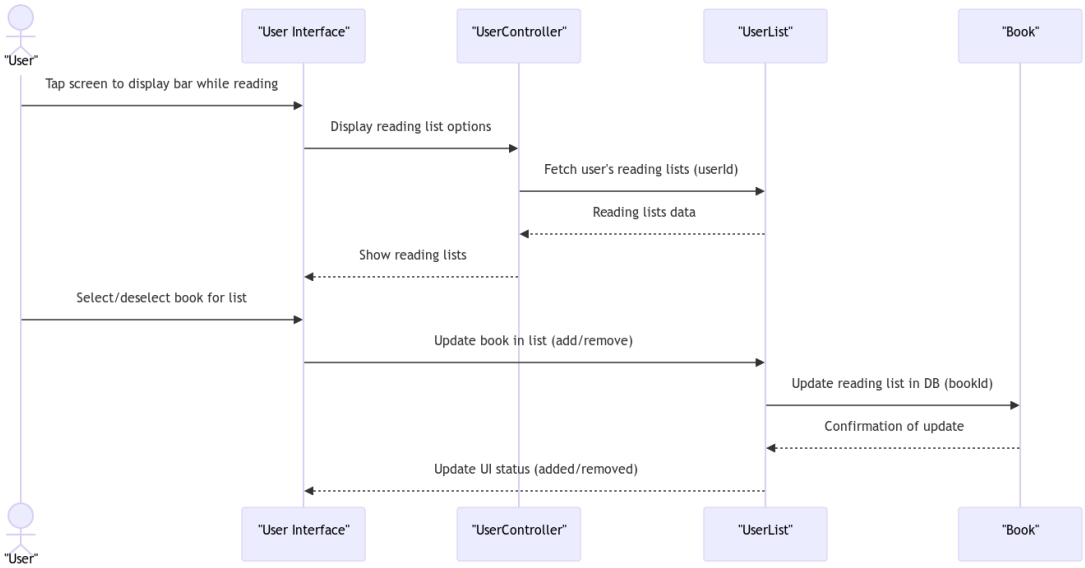

"User"
"User Interface"
"BookController"
"Book"
Click ‘Filter Books’
Request filter options
Select filters
Apply filters (criteria)
Retrieve filtered books (filters)
Filtered books list
Display filtered books to user
"User"
"User Interface"
"BookController"
"Book"

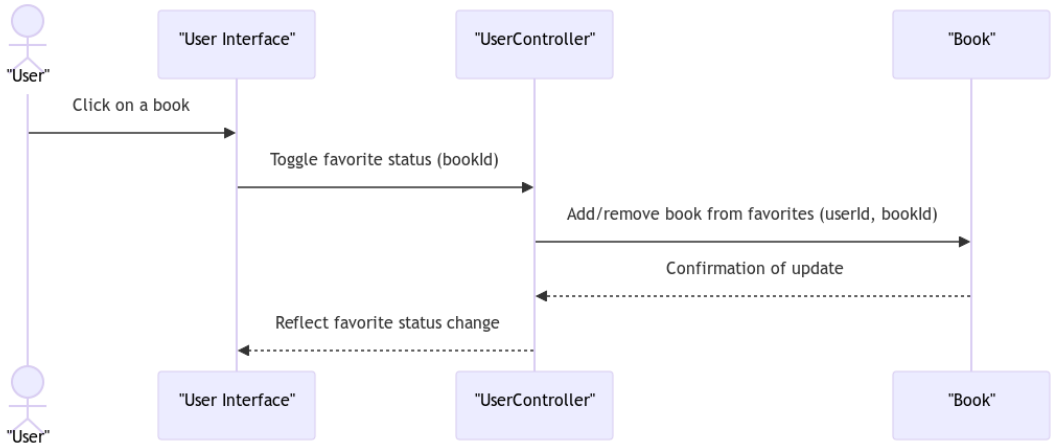

"User"
"User Interface"
"UserController"
"GoogleFormsInterface"
Navigate to "Suggest a Book"
Request book suggestion form
Open Google Form for book suggestion
Form loaded
Display form to user
Fill and submit form
Confirmation of suggestion
"User"
"User Interface"
"UserController"
"GoogleFormsInterface"

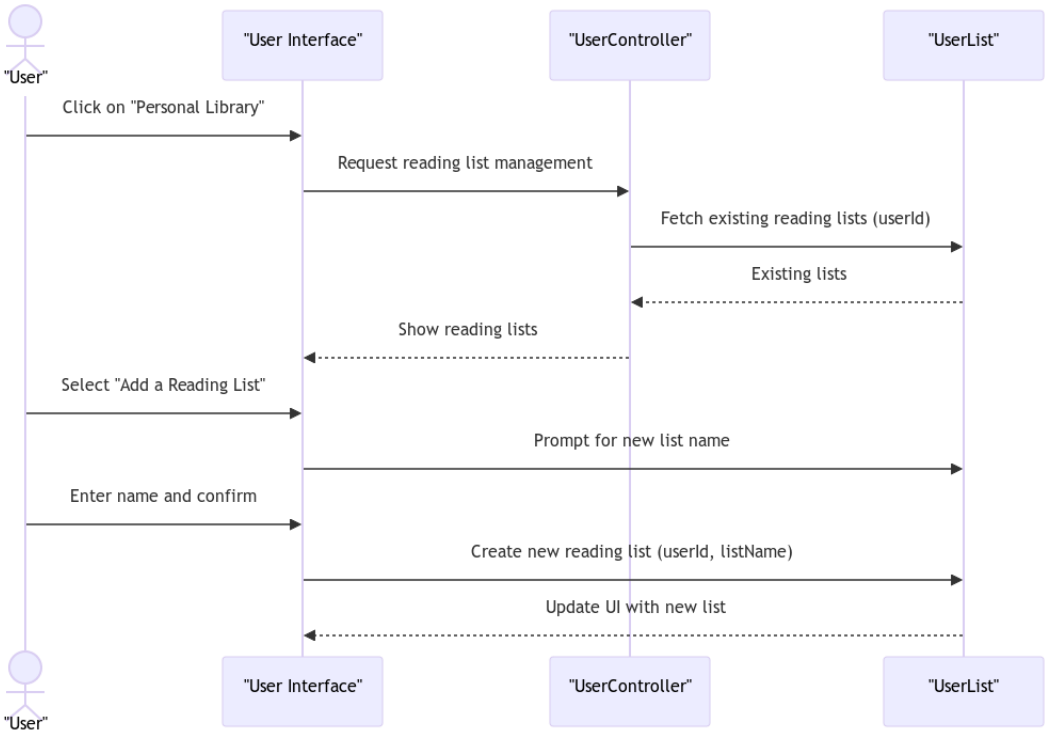

"User"
"User Interface"
"BookController"
"Book"
Tap bookmark icon on page
Bookmark page request (bookId, page)
Save bookmark (userId, bookId, page)
Bookmark saved
Show bookmark indicator on page
"User"
"User Interface"
"BookController"
"Book"

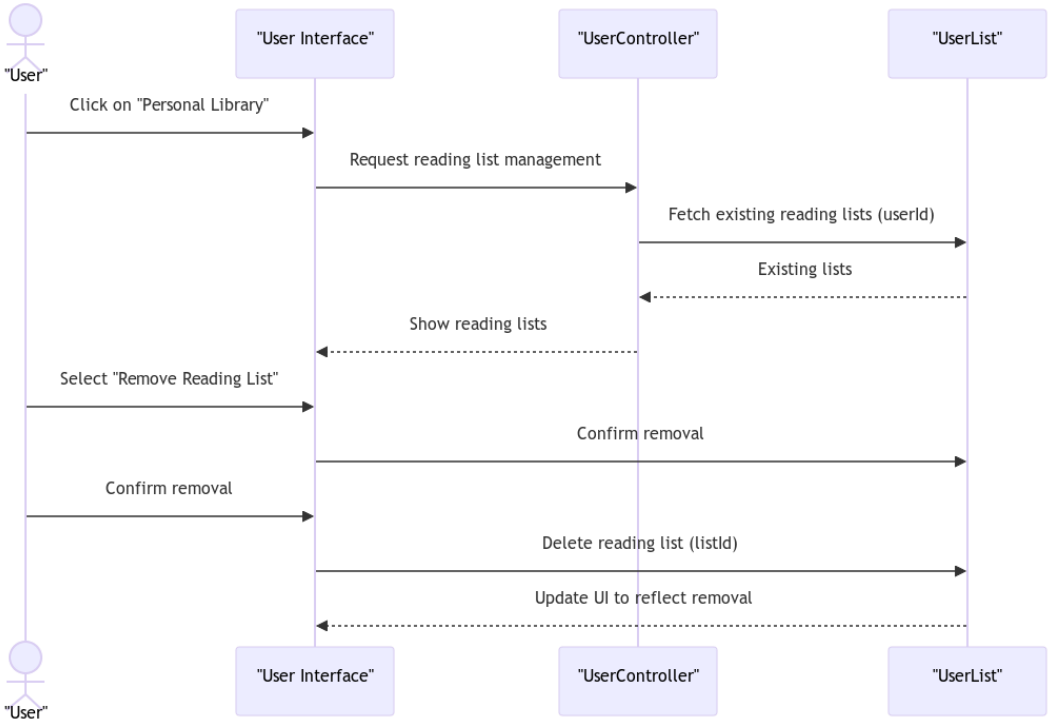

"User"
"User Interface"
"AuthenticationController"
"Authentication"
Tap sign up option
Enter name, email, password
Create user profile (credentials)
Store user data
Profile created
Confirm profile creation to user
"User"
"User Interface"
"AuthenticationController"
"Authentication"

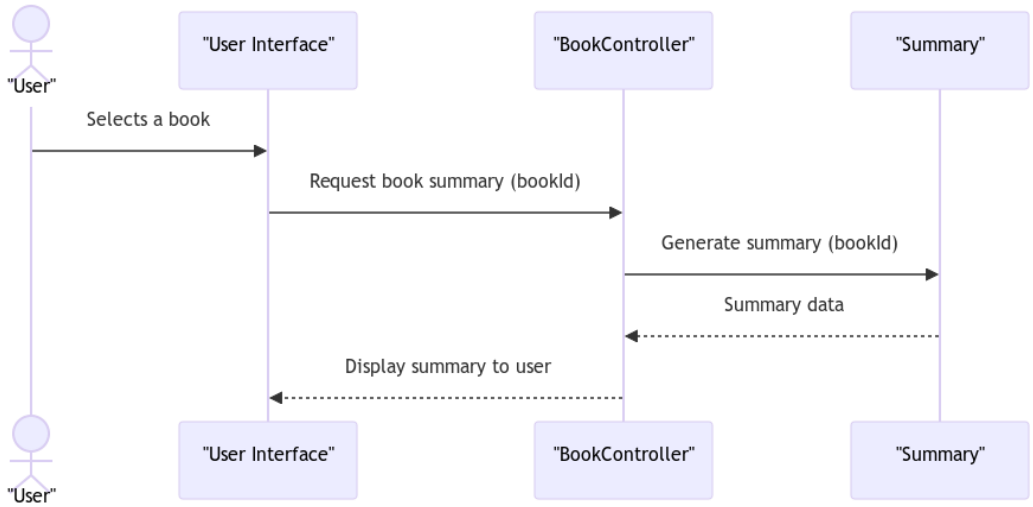

"User"
"User Interface"
"BookController"
"Book"
Browse catalog and select a book
Tap download button for the book
Initiate download (bookId)
Retrieve book content (bookId)
Book content
Download book locally to user device
"User"
"User Interface"
"BookController"
"Book"

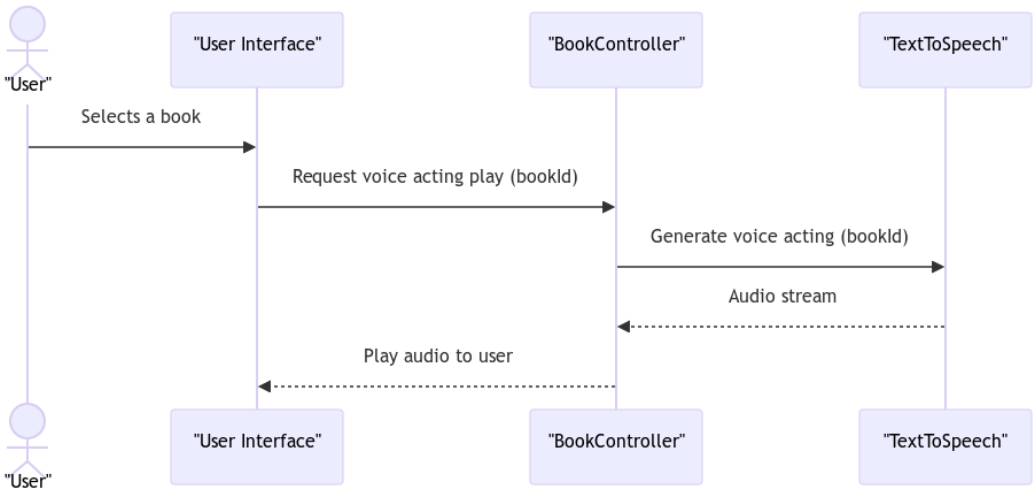

"User"
"User Interface"
"UserController"
"UserList"
"Book"
Tap screen to display bar while reading
Display reading list options
Fetch user's reading lists (userId)
Reading lists data
Show reading lists
Select/deselect book for list
Update book in list (add/remove)
Update reading list in DB (bookId)
Confirmation of update
Update UI status (added/removed)
"User"
"User Interface"
"UserController"
"UserList"
"Book"

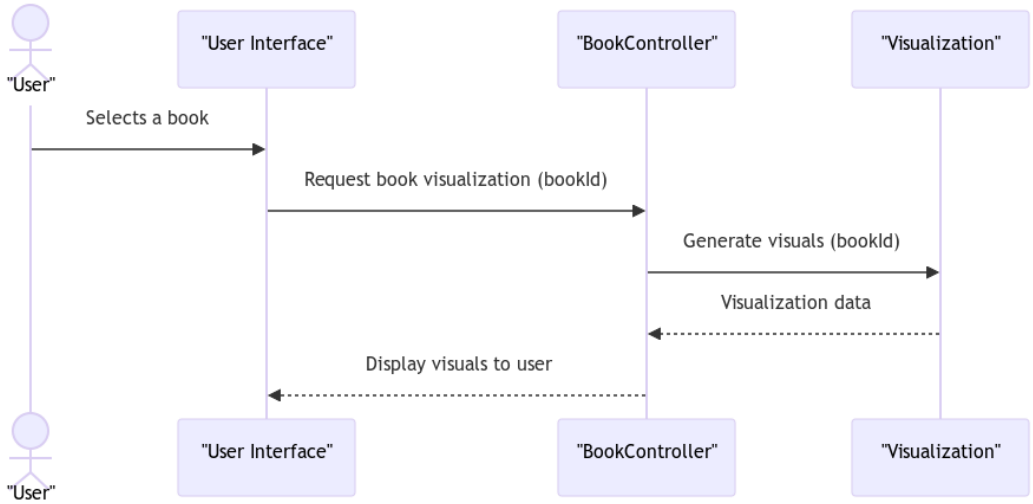

"User"
"User Interface"
"UserController"
"Book"
Click on a book
Toggle favorite status (bookId)
Add/remove book from favorites (userId, bookId)
Confirmation of update
Reflect favorite status change
"User"
"User Interface"
"UserController"
"Book"

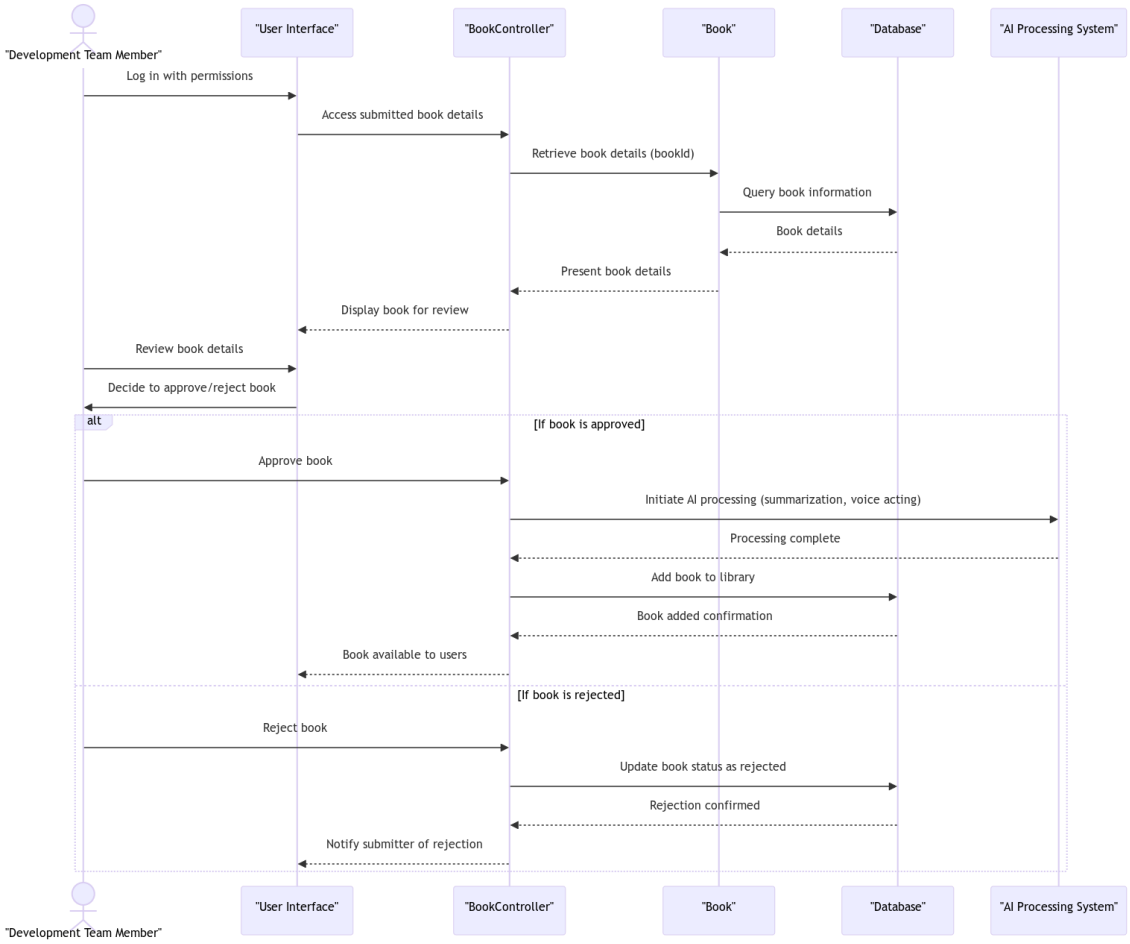

"User"
"User Interface"
"UserController"
"UserList"
Click on "Personal Library"
Request reading list management
Fetch existing reading lists (userId)
Existing lists
Show reading lists
Select "Add a Reading List"
Prompt for new list name
Enter name and confirm
Create new reading list (userId, listName)
Update UI with new list
"User"
"User Interface"
"UserController"
"UserList"

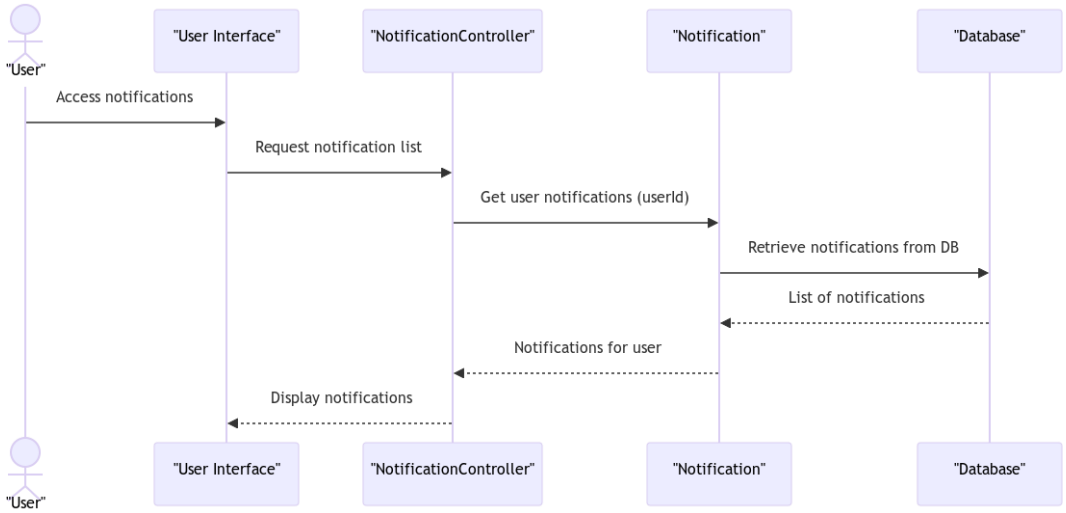

"User"
"User Interface"
"UserController"
"UserList"
Click on "Personal Library"
Request reading list management
Fetch existing reading lists (userId)
Existing lists
Show reading lists
Select "Remove Reading List"
Confirm removal
Confirm removal
Delete reading list (listId)
Update UI to reflect removal
"User"
"User Interface"
"UserController"
"UserList"


"User"
"User Interface"
"BookController"
"Summary"
Selects a book
Request book summary (bookId)
Generate summary (bookId)
Summary data
Display summary to user
"User"
"User Interface"
"BookController"
"Summary"

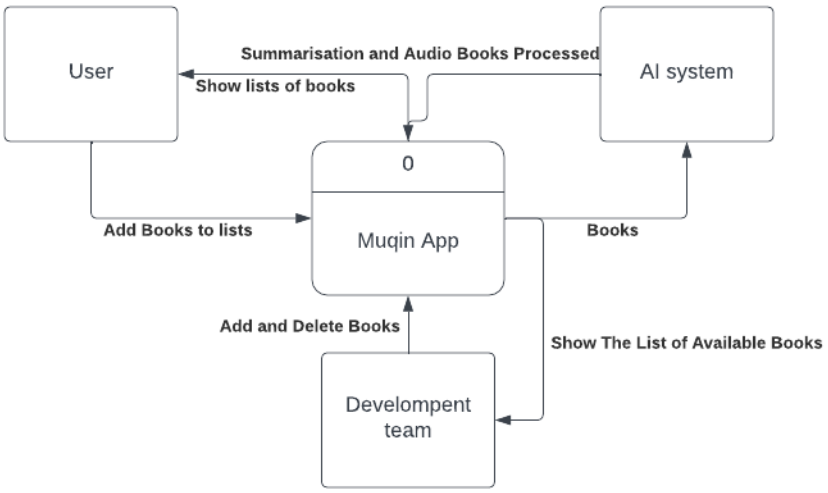

"User"
"User Interface"
"BookController"
"TextToSpeech"
Selects a book
Request voice acting play (bookId)
Generate voice acting (bookId)
Audio stream
Play audio to user
"User"
"User Interface"
"BookController"
"TextToSpeech"

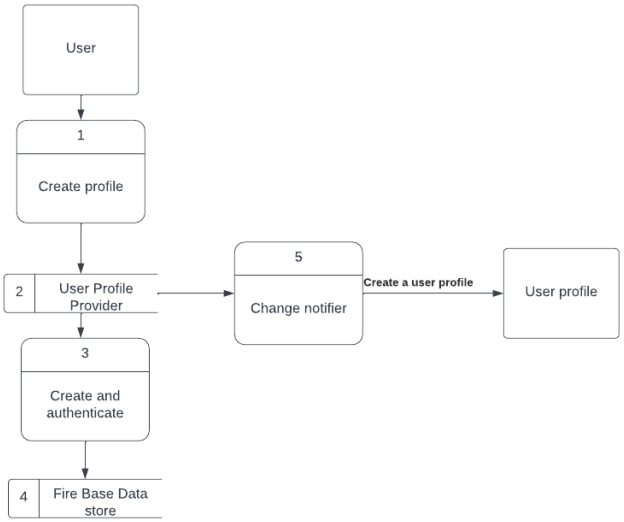

"User"
"User Interface"
"BookController"
"Visualization"
Selects a book
Request book visualization (bookId)
Generate visuals (bookId)
Visualization data
Display visuals to user
"User"
"User Interface"
"BookController"
"Visualization"

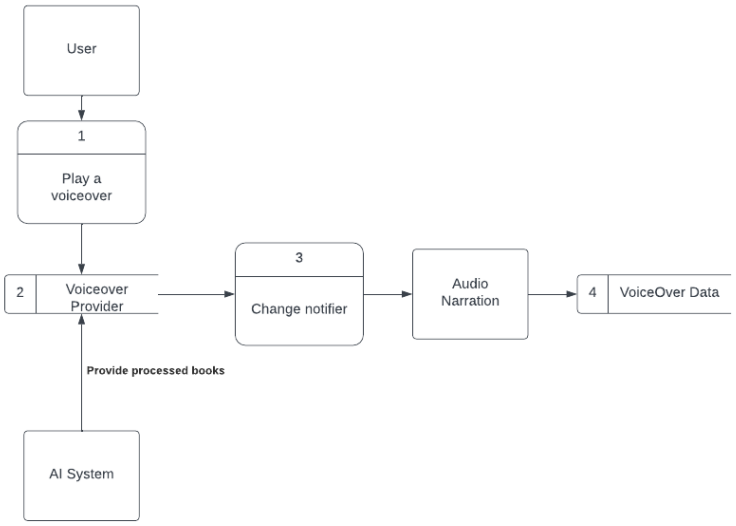

"Development Team Member"
"User Interface"
"BookController"
"Book"
"Database"
"AI Processing System"
Log in with permissions
Access submitted book details
Retrieve book details (bookId)
Query book information
Book details
Present book details
Display book for review
Review book details
Decide to approve/reject book
alt
[If book is approved]
Approve book
Initiate AI processing (summarization, voice acting)
Processing complete
Add book to library
Book added confirmation
Book available to users
[If book is rejected]
Reject book
Update book status as rejected
Rejection confirmed
Notify submitter of rejection
"Development Team Member"
"User Interface"
"BookController"
"Book"
"Database"
"AI Processing System"

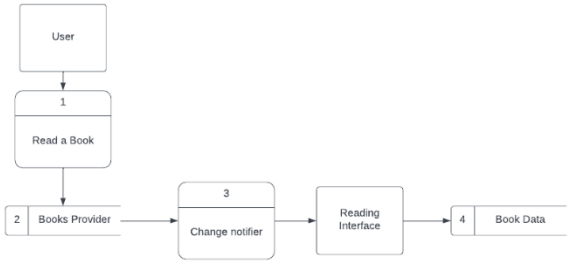

"User"
"User Interface"
"NotificationController"
"Notification"
"Database"
Access notifications
Request notification list
Get user notifications (userId)
Retrieve notifications from DB
List of notifications
Notifications for user
Display notifications
"User"
"User Interface"
"NotificationController"
"Notification"
"Database"

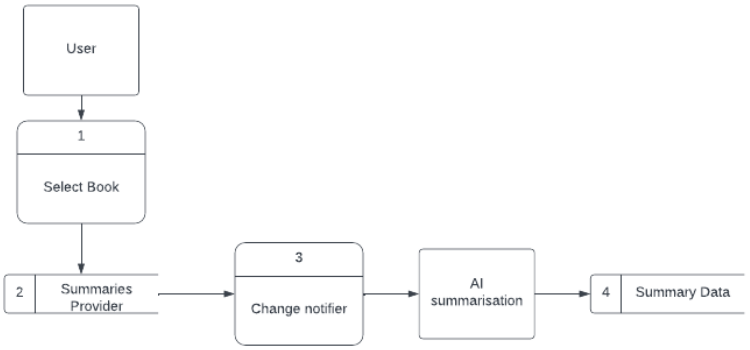
5.3.Interfaces - Control and data flow into and out of the component. [Dataflow diagrams]. Draw DFD 0 for the whole system and DFD 1 for each feature.



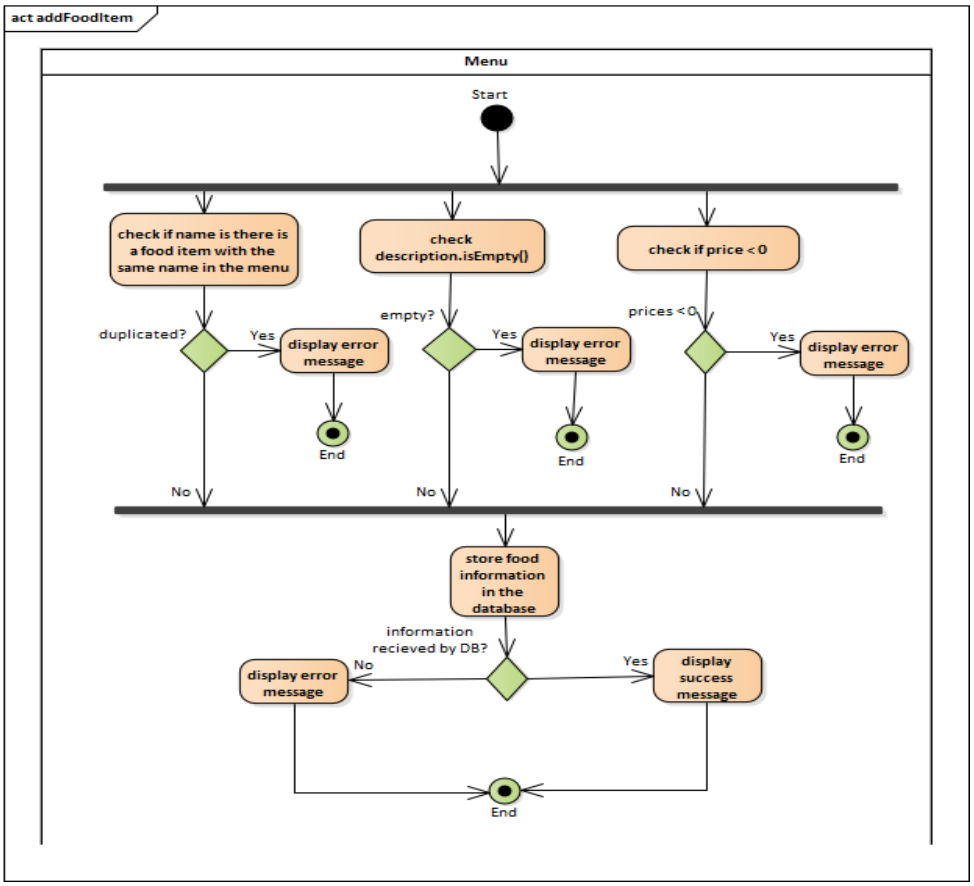



User
1
Play a voiceover
2
Voiceover Provider
3
Change notifier
Audio Narration
4
VoiceOver Data
Provide processed books
AI System


User
1
Read a Book
2
Books Provider
3
Change notifier
Reading Interface
4
Book Data


User
1
Select Book
2
Summaries Provider
3
Change notifier
AI summarisation
4
Summary Data

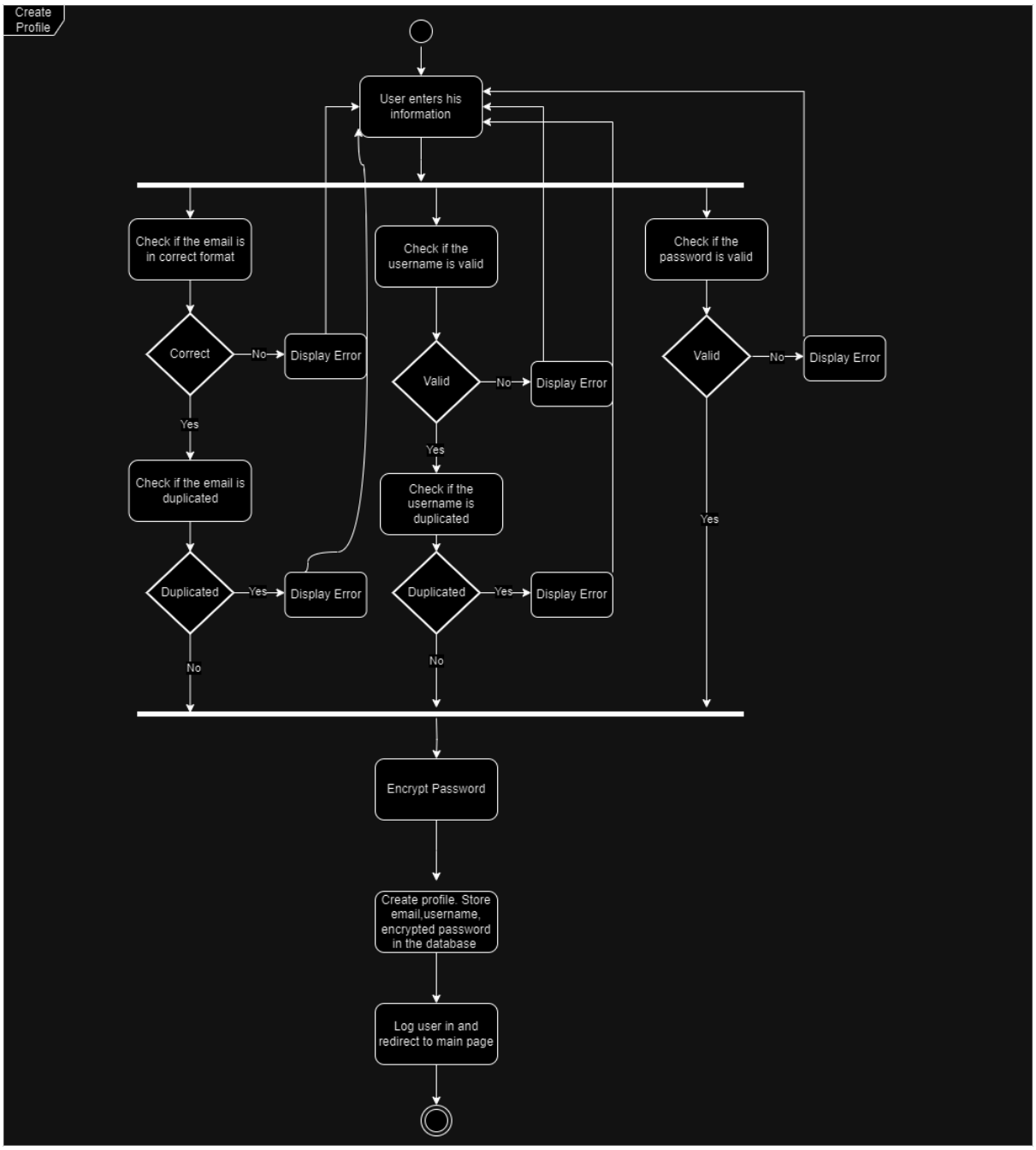
5.4.[Pseudo code] and [Activity Diagram]. Detailed pseudo code and activity diagram for the non-trivial methods; {Set, Get, default constructors, etc.} are considered trivial methods where pseudo code is not required.




```
addFoodItem(name, description, price):

IF price <= 0 THEN
        display('Wrong price input.')
        return
IF description.isEmpty() THEN
        display('Please fill-in the description')
        return
IF databaseHandler.nameExists(name) THEN
        display(name + ' is already in your menu')
        return

databaseHandler.createFoodItem(name, description, price)

IF databaseHandler.error THEN
        display('An error has occurred, please try again.')
ELSE
        display(name + ' has been added successfully')
```

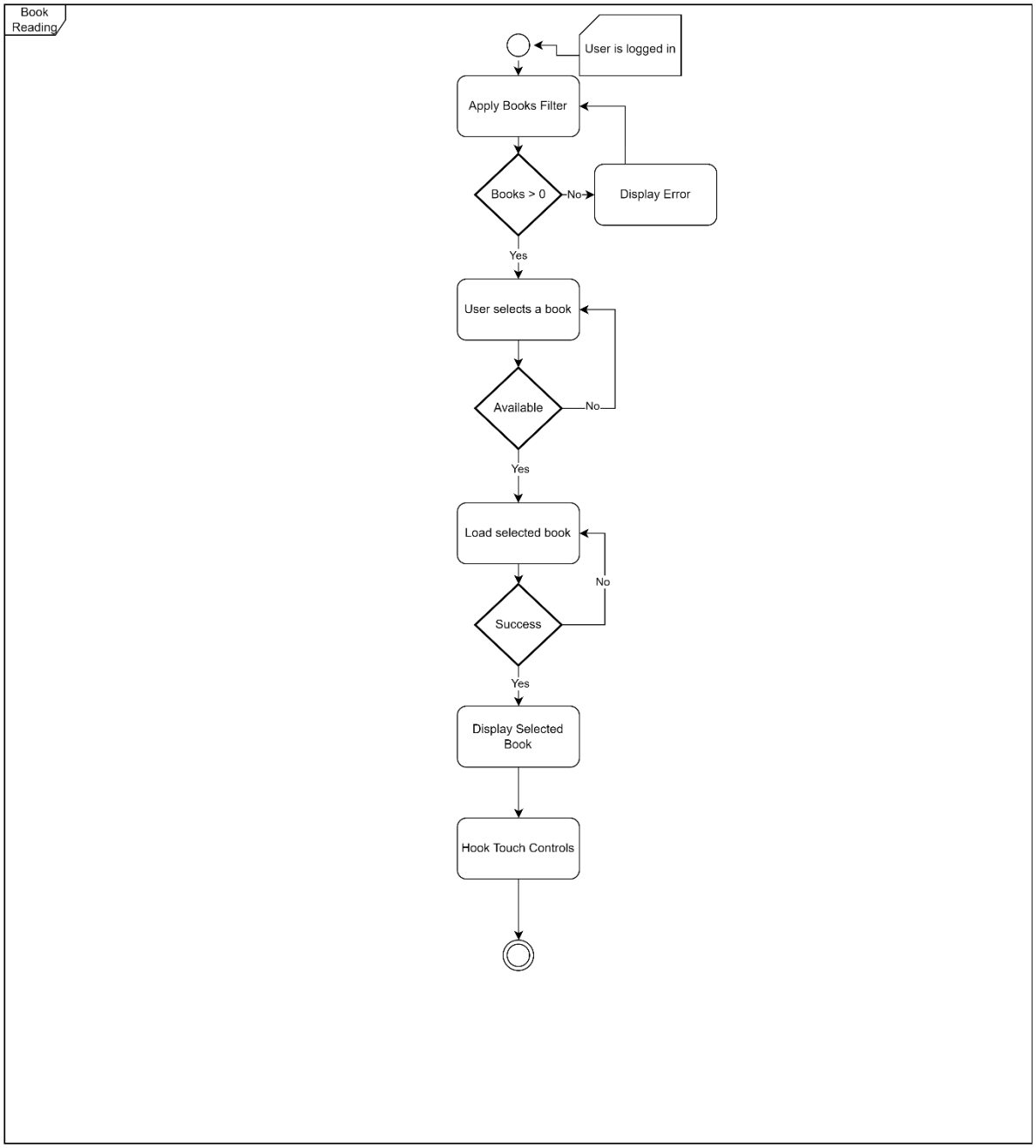
1. Sign Up

```
1  username,email,password = input()
2  if(!valid(email)):
3      display("email is invaild")
4      jump to line 1
5  if(duplicate(email)):
6      display("This email already exists")
7      jump to line 1
8  if(!valid(username)):
9      display("Username is invalid")
10     jump to line 1
11 if(duplicate(username)):
12     display("Username already exists")
13     jump to line 1
14 if(!valid(password)):
15     display("Password is invalid")
16     jump to line 1
17
18 encryptedPass  = encrypt(password)
19 User validUser = new User(username,email,encryptedPass)
20 DB.storeUser(validUser)
21 Login(validUser)
22 redirect(validUser,mainPage)
```


Create Profile
User enters his information
Check if the email is in correct format
Check if the username is valid
Check if the password is valid
Correct
No
Display Error
Valid
No
Display Error
Valid
No
Display Error
Yes
Yes
Yes
Check if the email is duplicated
Check if the username is duplicated
Duplicated
Yes
Display Error
Duplicated
Yes
Display Error
No
No
Encrypt Password
Create profile. Store email,username, encrypted password in the database
Log user in and redirect to main page

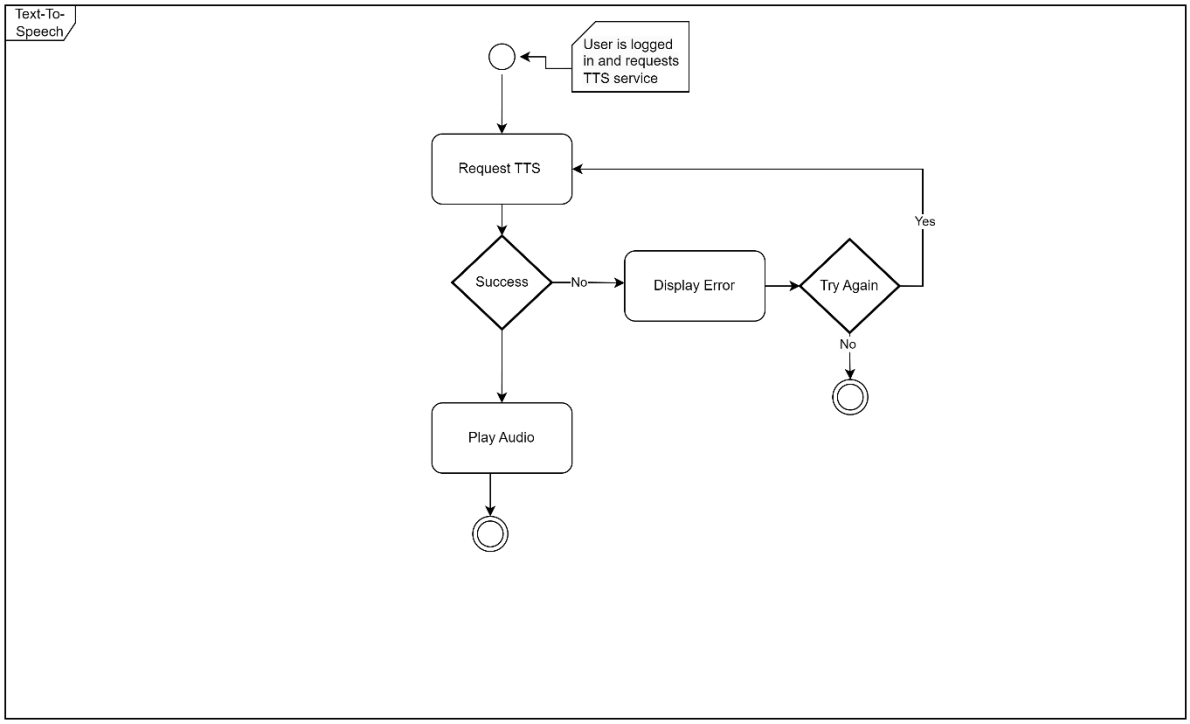
2. Book Reading

```
FilterBooks(Book[] allBooks,Filter f):
    return filter(allBooks,f)


Book_Read(Book SelectedBook):
    DB = ConnectDB()
    if(ConnectDB.connection is 'connected'):
        if(!DB.get_Book(SelectedBook).error):
            if(userinput(try_again)):
            Book_Read(SelectedBook)
        Book sBook = DB.get_Book(SelectedBook)
        DisplayBook(sBook)
        HookTouch(screen,sBook)
    else:
        if(userinput(try_again)):
            Book_Read(SelectedBook)
    return
Start_Book_Reading(Book[] allBooks):
    filter f = getSelectedFilters()
    FilterBooks(filter f,allBooks)
    Book SelectedBook = getSelectedBook()
    Book_Read(SelectedBook)
```



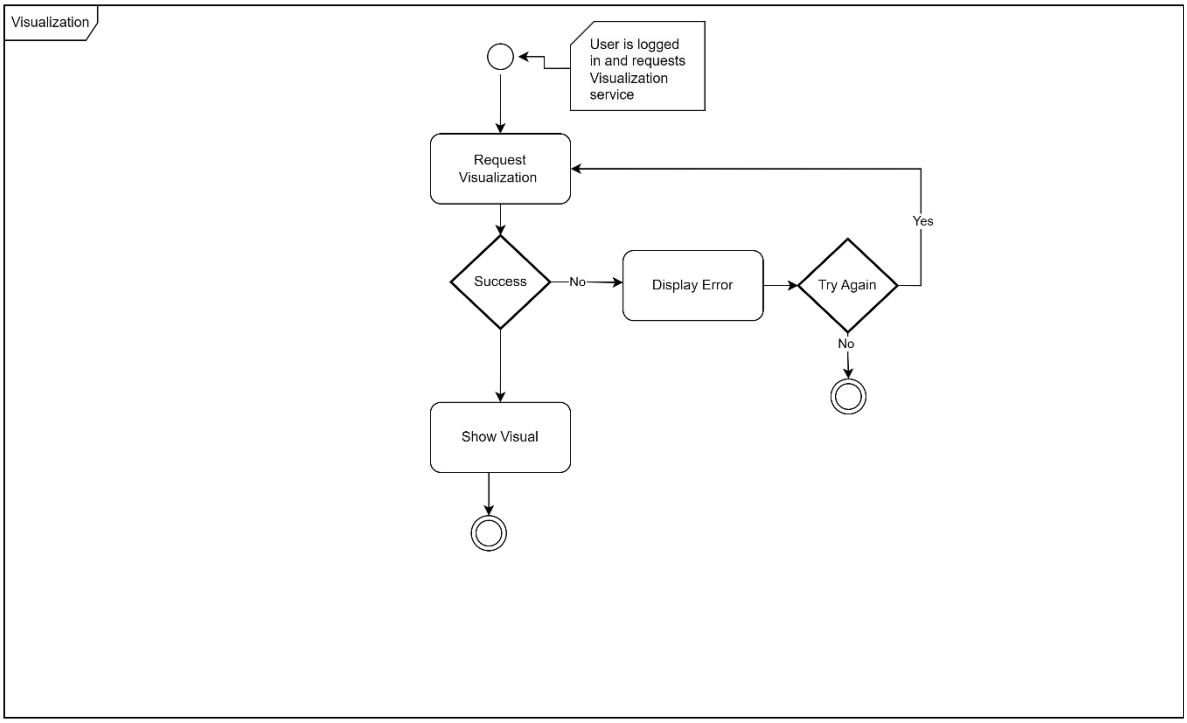
3. Text-To-Speech

```
TTS(Book book):
    TextToSpeech ttsBook = AI.TTS_Service.getTTS(Book book)
    if(!TTS_Service.getTTS(Book book)):
        display("Error in generating the audio book, please try again")
        if(userinput(try_again)):
            TTS(book)
        else:
            return
    PlayAudio(ttsBook)
```



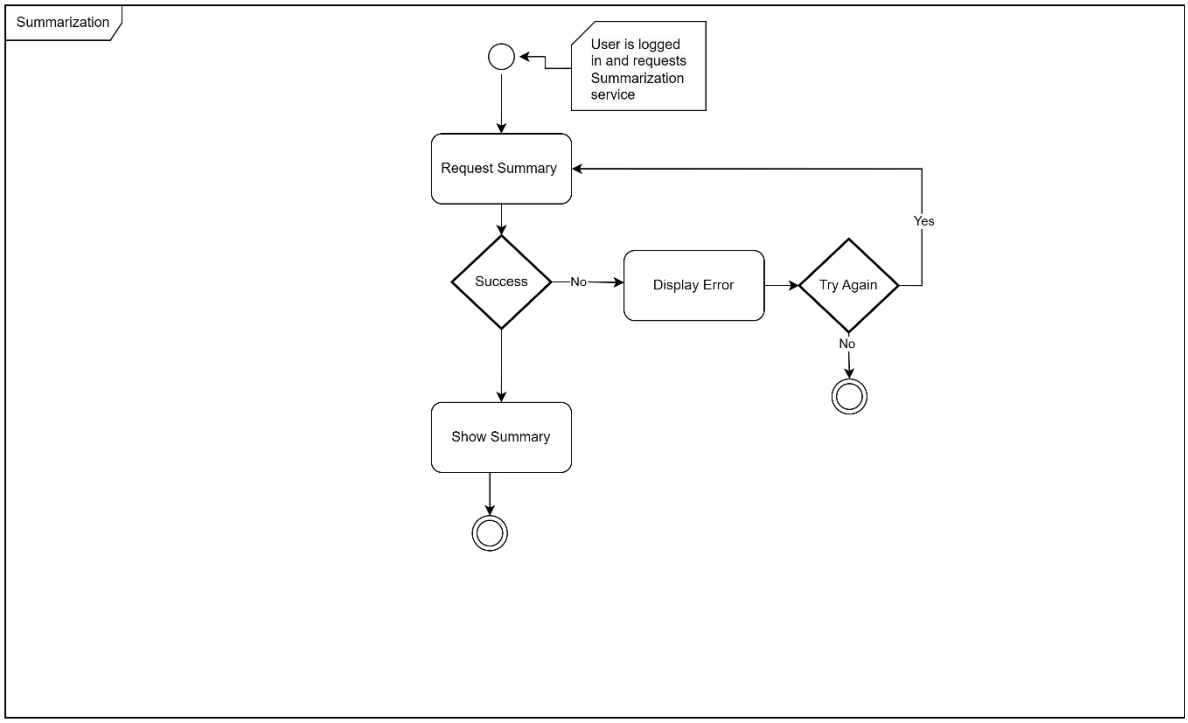
4. Visualization

```
("4.Visualization")
VisualBook(Book book):
Visualization img = AI.Visual_Service.getVisualization(Book book)
    if(!Visual_Service.getVisualization(Book book)):
        display("Error in generating the image, please try again")
        if(userinput(try_again)):
            VisualBook(book)
        else:
            return
    DisplayVisual(img)
```

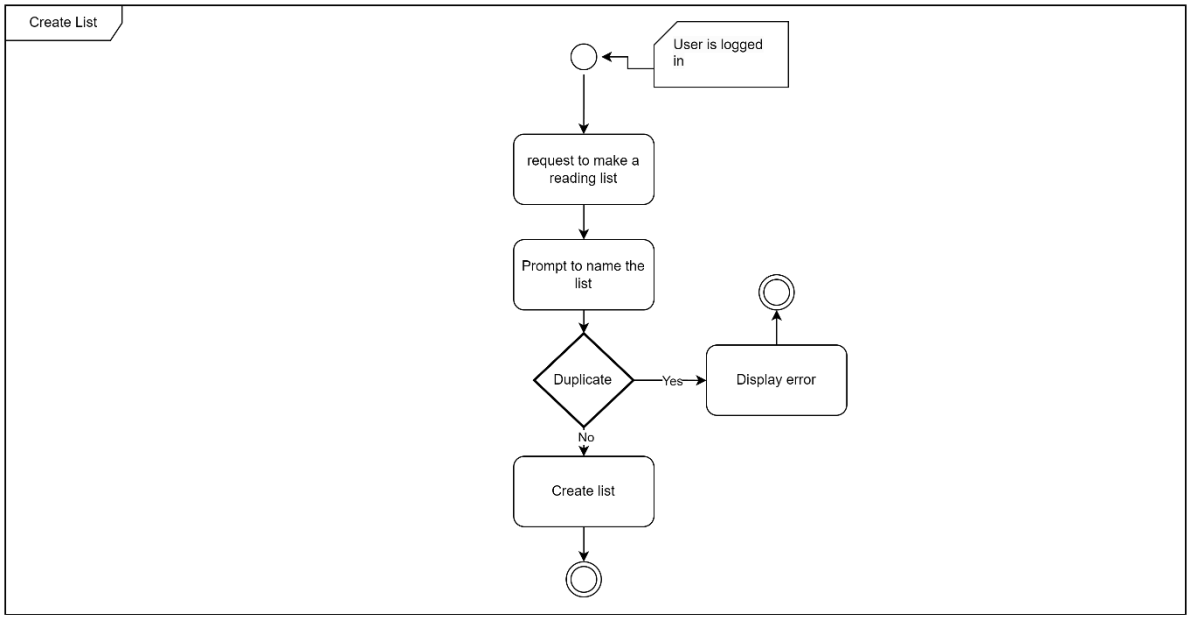


5. Summarization

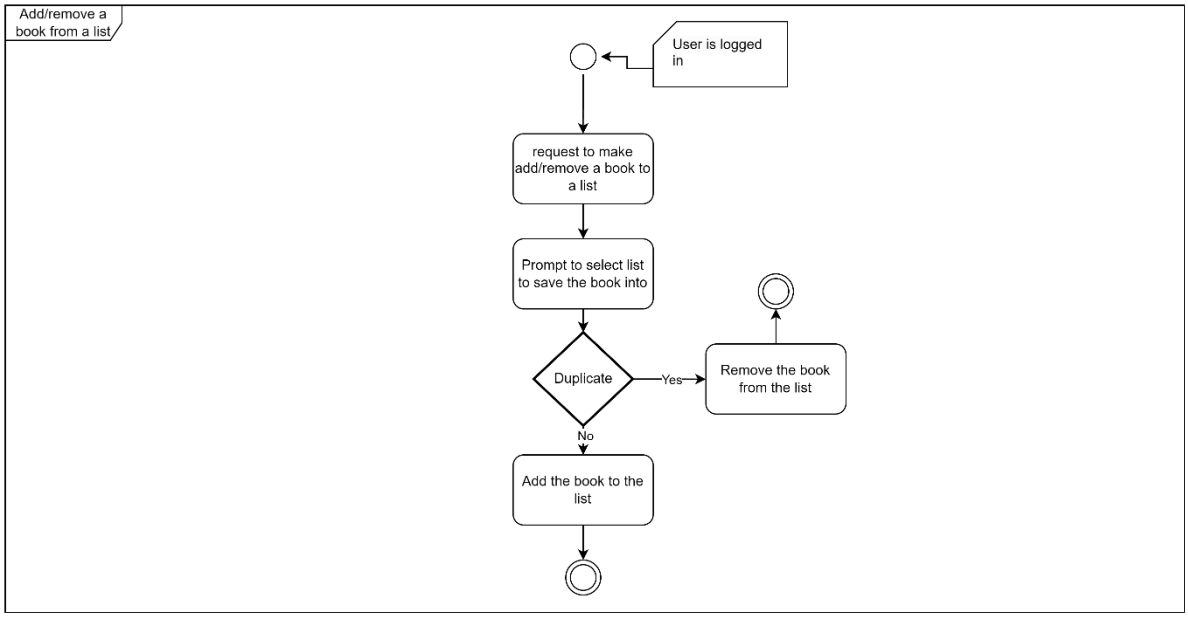
```
("5.Summarization")
SumBook(Book book):
Summary sumText = AI.Summarization_Service.getSummary(Book book)
    if(!Summarization_Service.getSummary(Book book)):
        display("Error in generating the image, please try again")
        if(userinput(try_again)):
            SumBook(book)
        else:
            return
    Display_Summary(SumText)
```



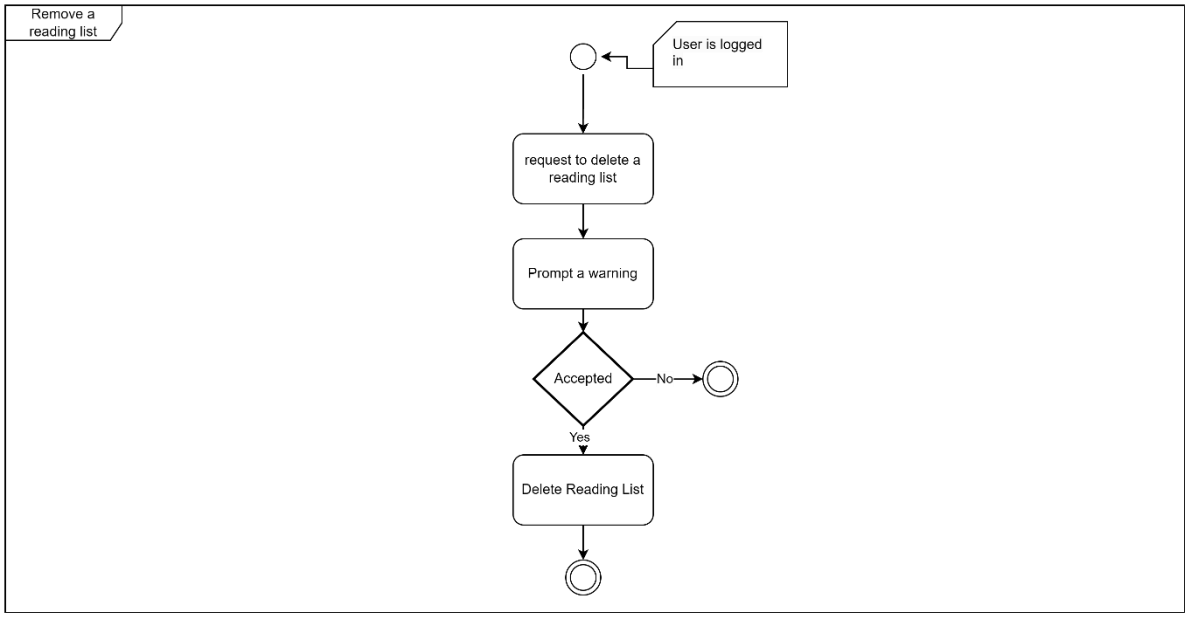
6. Library Management




```
CreateListRequest():
    String name = prompt("Please Enter The name of the list")
    if(name exists in User.ReadingLists:
        Display("Error while creating the list "+name+" Already exists")
        return
    List<Book> newList = new List<Book>(name)
    User.ReadingLists.add(newList)
    return
```



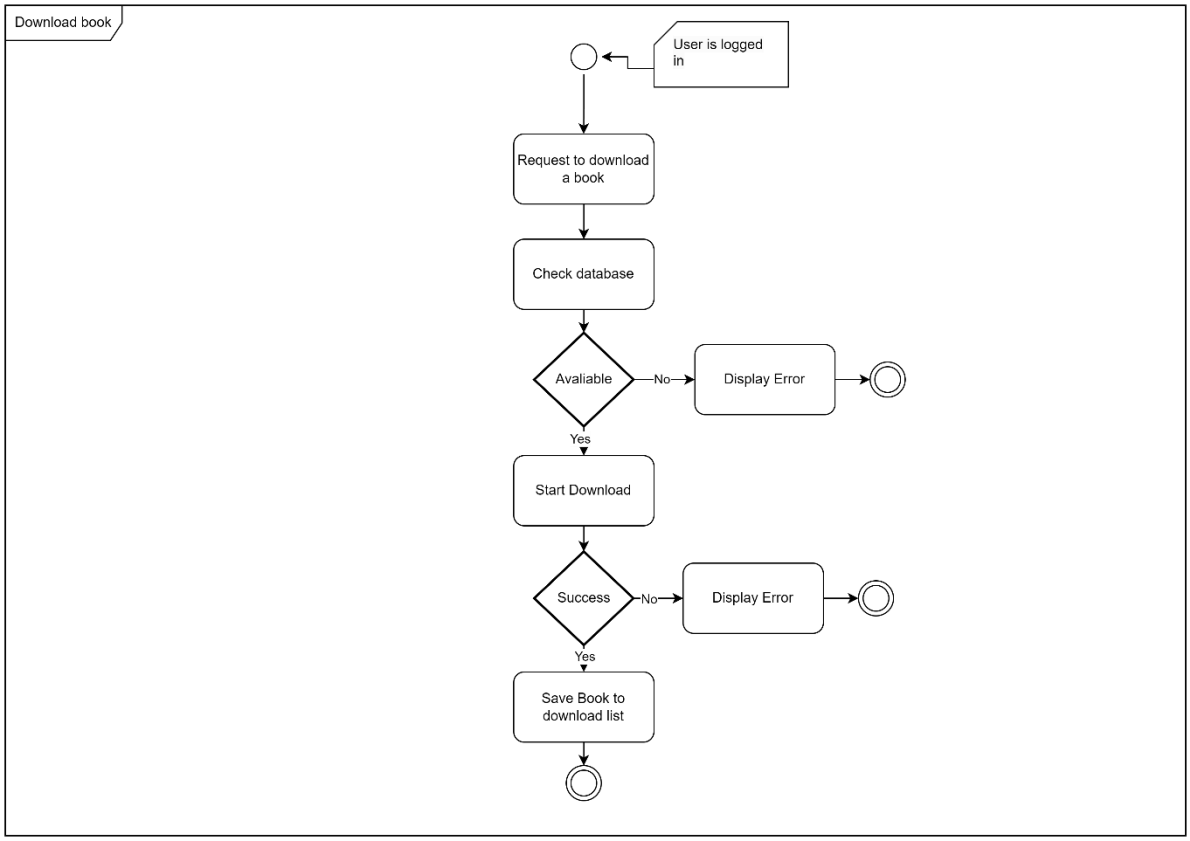

```
addRemoveBookFromList(Book book):
    List<Book> selectedList = prompt("Select which list do you want to modify")
    if(Book in selectedList):
        selectedList.remove(book)
    else:
        selectedList.add(book)
    return
```




```
DeleteList(List<Book> list):
    result = prompt("Are you sure you want to remove the list "+list.name+" from your reading lists")
    if(result):
        User.ReadingLists.Remove(list)
    return
```

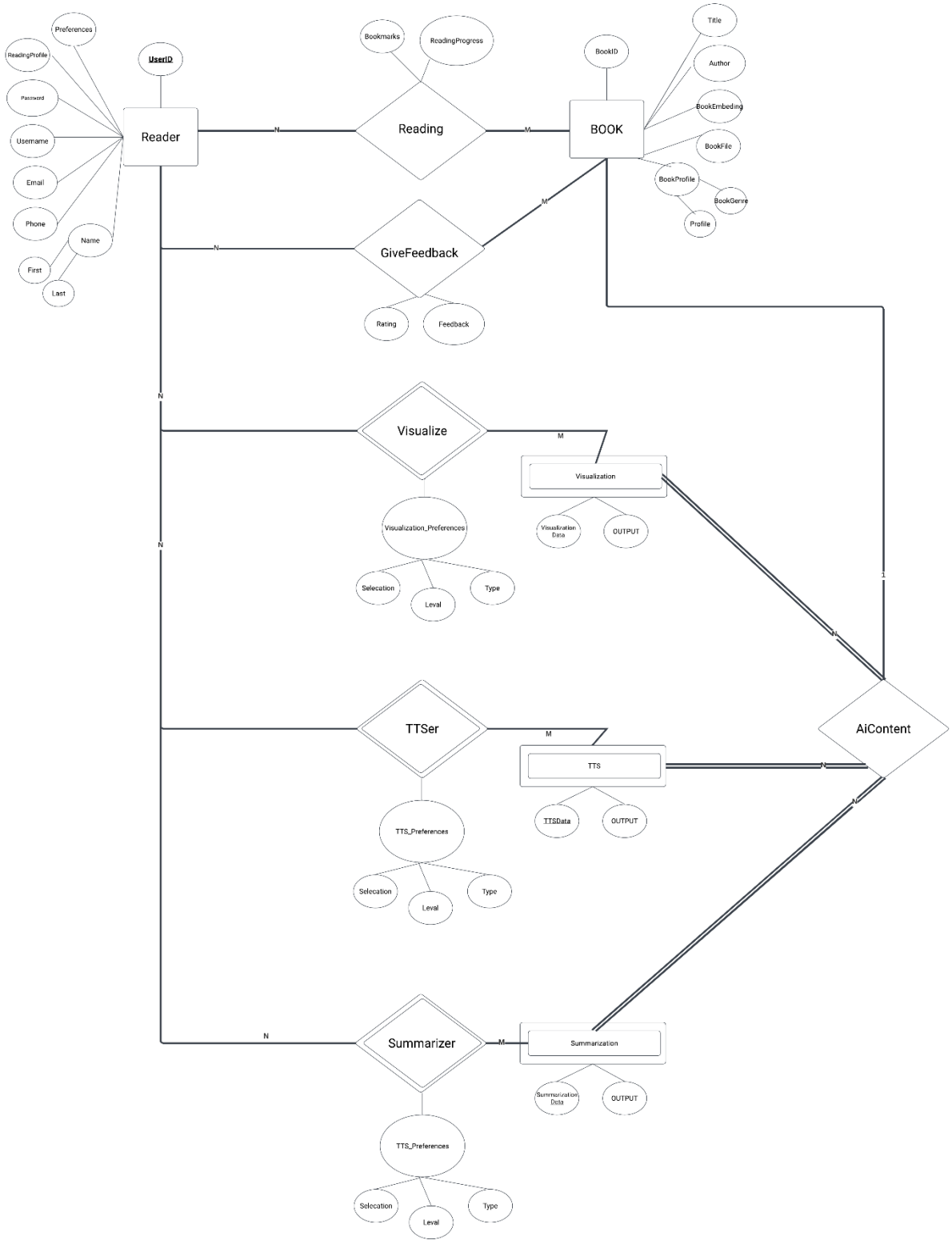
7. Download Book

```
DownloadBook(Book book):
    DB = connectDB();
    if(DB.connectionError):
        Display("Error while making a connection to the database")
        return
    Bookdownload = DB.getBook(book).download()
    if(Bookdownload.status.isfail):
        Display("Error while downloading the book to your device !")
        return
    device.save(Bookdownload.result,"./downloadedbooks")
    userLists['downloads'].add(book)
    return
```



6. Data Design

6.1. Database Description [Database Design/ ER Diagram] – Describe the persistent data structures (i.e., the Database structure), state the primary keys, foreign keys, and alternative keys, integrity rules and constraints



6.2.Data Dictionary – An alphabetic list of names used by the system (entities, types, services, relations, attributes). Include a description of the named entity.

Customers

Attributes	Relations (PK, etc.)	Types	Services (Size)	Description of the Attributes
Reader Entity				
UserID	PK	Integer	10	Unique identifier for each reader.
Username	Unique	String	10	Name chosen by the reader for identification.
Email	Unique	String	70	The reader's email address.
Phone		String	20	Contact phone number of the reader.
Password		String	100	A secure password for account access.
ReadingProfile		JSON/String	Variable	Information about the reader's reading habits.
Preferences		JSON/String	N/A	General preferences in books and reading.
Name (First, Last)		String	25	The reader's first and last names.
Book Entity				
BookID	PK	Integer	10	Unique identifier for each book.
BookFile		Binary/File	N/A	The digital file of the book.
BookProfile (Genre, Profile)		JSON/String	Variable	Genre and specific profile of the book.
BookEmbeding		Binary/File	Variable	Data used for AI analysis or recommendations.
Title		String	30	The title of the book.
Author		String	25	The author of the book.
TTS Week Entity				
Data	Partial Key	String	Variable	Selected text for TTS processing.
OUTPUT (Voice)		Audio File	Variable	The voice output generated from TTS processing.
Summarization Week Entity				
Data	Partial Key	String	Variable	Selected text data for summarization.
OUTPUT (Text)		Text File	Variable	The summarized text output.
Visualization Week Entity				
Data	Partial Key	String	Variable	Selected text for visualization.

OUTPUT (Images)		Image File	Variable	Visual representations or images produced.
Reading Relationship				
Bookmarks		Integer/Array	Variable	Marks specific places in the book.
ReadingProgress		Float/Percentage	10	The reader's progress in the book.
GiveFeedback Relationship				
Rating		Integer	Variable	The reader's rating of the book.
Feedback		Text/String	Variable	Written feedback or review of the book.
TTSer Week Relationship				
TTS_Preferences		JSON/String	50	Reader's preferences for TTS output.
Visualize Week Relationship				
Visualization_Preferences		JSON/String	50	Preferences for visualizations related to reading material.
Summarizer Week Relationship				
Summarization_Preferences		JSON/String	50	Preferences for how text is summarized.

7. **Human Interface Design** [Screen layout]
   7.1. Screen Images – Screenshots showing (high fidelity prototype) the complete interface from the user's perspective. https://www.figma.com/proto/0nBVeo1uix5tnZUgQ8E2qM/Muqin
   7.2. Report Formats – a description of major reports provided by the system.

8. **Requirements Matrix** [traceability matrix]– provide a cross reference that traces components and data structures to the requirements.

Classes Responsible / Requirements Code	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	Security	Reliability	Availability	Maintainability	Portability
User	X		X	X	X					X	X	X	X	X	X
User Controller	X		X	X	X					X	X	X	X	X	X
Book		X		X	X										
Book Controller		X		X	X										
Authentication	X										X	X	X	X	X
Profile			X												

Library Management					X									
Text-To-Speech						X								
Visualization							X							
Summarization								X						
Notification									X					
Database	X	X	X	X	X						X	X	X	X	X

9. **Resource Estimates** - A summary of computer resource estimates required for operating the software (e.g., RAM, Storage, Bandwidth etc).

- **User Device requirements**
  1. RAM
     Android:
     - Minimum: 2GB
     - Recommended: 4GB or higher

     IOS:
     - Minimum: 1GB
     - Recommended: 4GB or higher
  2. CPU
     Android:
     - Minimum: Quad-core processor (e.g., Qualcomm Snapdragon 400 series)
     - Recommended: Octa-core processor (e.g., Qualcomm Snapdragon 600 series or equivalent) or higher

     IOS:
     - Minimum: Dual-core processor (e.g., Apple A9)
     - Recommended: Hexa-core or higher (e.g., Apple A12 Bionic, or newer)
  3. GPU
     - Any integrated GPU with the minimum or recommended CPU types.
  4. Storage
     Android:
     - 100MB or higher

     IOS:
     - 80MB or higher
  5. Bandwidth
     - Minimum: 500 Kbps
     - Recommended: 10 Mbps

- **Server Requirements**
  1. RAM
     - Minimum: 16GB
     - Recommended: 64GB or higher
  2. CPU
     - Minimum: 6-core processor (e.g., Intel Xeon E5-1660 or Equivalent)
     - Recommended: 8-core processor (e.g., Intel Xeon E5-2687W or better)
  3. GPU

        - Nvidia RTX 2060 Series or higher
    4. Storage
        - 500 GB or higher
    6. Bandwidth
        - Minimum: 5 Mbps
        - Recommended: 20 Mbps

10. **Definitions**, Acronyms, and Abbreviations - provide definitions of all terms, acronyms and abbreviations needed for the SDD.
    - GPU: Graphics Processing Unit
    - CPU: Central Processing Unit
    - RAM: Random Access Memory