TED UNIVERSITY

CMPE 492 / SENG 492 Senior Project

<<BooTunes>>

Low Level Design Report

Spring 2025

Team Members:

Mehlika Eroğlu, 58717180232, Software Engineering

Melih Aydın, 28180576276, Computer Engineering

Elif Alptekin, 31376233198, Computer Engineering

Batuhan Dalkılıç, 11491297756, Computer Engineering

Supervisor: Venera Adanova

Jury Members:

Gökçe Nur Yılmaz

Eren Ulu

Firat AKBA

Contents

1. Introduction

1.1. Object Design Trade-offs

In the object-oriented design process of the BooTunes project, some important decisions were made in terms of the performance, flexibility, reusability and ease of maintenance of the system, and in certain cases, compromises were made consciously. In this section, the basic choices, analyses and compromises encountered in the object design process of the project are explained in detail.

One major decision involved selecting an NLP model that offered an ideal balance between precision and performance. More accurate transformer-based models like BERT offered high accuracy but at the cost of higher computational overhead. Conversely, lighter models provided faster processing times but at the expense of nuanced emotional analysis. Ultimately, a hybrid solution was chosen—deploying a lighter NLP model for real-time analysis with a secondary, more complex model running asynchronously for detailed analysis, thereby balancing performance with accuracy.

Additionally, trade-offs in data storage and management strategies were carefully considered. MongoDB was selected for its schema-less flexibility, providing significant advantages for rapidly evolving data structures at the expense of relational integrity, necessitating careful handling to ensure data consistency.

- **Balance Between Performance and Flexibility**

The main goal of our project is to provide music and visual suggestions that are appropriate for the mood of the page as a result of the page-based sentiment analysis of the book content. While achieving this goal, performance has been a critical element due to the necessity of real-time analysis and media matching. However, a modular structure was also targeted to make the system flexible and expandable.
A modular and object-oriented structure was preferred. Thus, subsystems such as sentiment analysis, music recommendation engine, visual recommendation engine, user management have been designing as independent objects and classes.
Some flexibility has been compromised to improve real-time performance:
For example, in the selection of music and visuals, instead of dynamically pulling data from the

cloud, a decision was made to use a pre-classified and labeled local dataset.
Although design patterns such as Factory Pattern and Strategy Pattern are suggested for flexibility, in some cases, more direct (hard-coded) solutions were preferred due to performance concerns.

- **The Balance Between Reusability and Simplicity**

Creating reusable components is an important goal in the project. In particular, the music recommendation algorithm and sentiment analysis engine should be usable in different applications (e.g. books in different languages, for different content) in the future. For this, a flexible infrastructure was designed using abstract classes and interfaces. For example, the EmotionAnalyzer and MusicRecommender classes were defined to allow different analysis and recommendation algorithms. However, due to time and resource constraints, concrete applications covering only the targeted scenarios were developed instead of the abstract structure open to all possibilities in the first version.

The choices made during the object-oriented design of the BooTunes project were shaped by making balanced and justified decisions between performance, flexibility and user experience. While preserving its modular and extensible structure, some abstractions and flexibility were consciously compromised in order to increase performance. Design patterns and existing libraries are used in the design to ensure compliance with engineering standards.

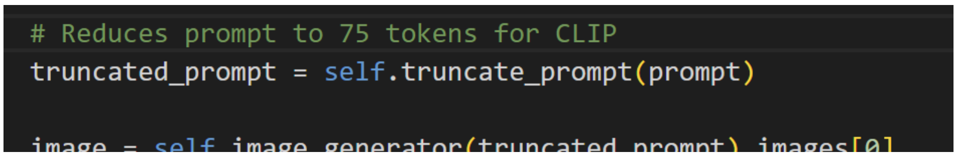
1.2. Interface Documentation Guidelines

In order to ensure that each component developed in the BooTunes project is understandable, sustainable and extensible, certain rules and standards have been adopted for the design of interfaces and the documentation of these interfaces. Since interfaces are the basic elements that manage the communication between system components, these documents have created a common language among software developers during the project process and have enabled the independent development of different components of the software.

First of all, the responsibilities of each class and interface have been clearly defined. Each interface document explains in detail which functions the relevant class undertakes and for which tasks it will be used. Thus, each member of the project team has been able to easily understand which interface provides which function, and a modular development process has been provided by keeping the dependencies between components to a minimum, and processes such as model selection have been shared among the team members. Each team member took responsibility for a different subsystem and carried out the independent parts interconnected until the merging part.

While writing the interfaces, the principles of independence and low coupling were emphasized. Each interface was designed to focus only on its own task area and was structured to work with minimum dependency on other components. For example, the "EmotionAnalyzer" interface is only responsible for performing emotion analysis; functions related to music or visual selection are separated into other interfaces. In this way, the possibility of changes to be made in one component affecting other components is minimized.

Inline comments are also accepted as an integral part of the interface documentation. Information such as what the method does, what parameters it takes, what output it gives in which situations, and error conditions are added as explanations to each method.



```
# Reduces prompt to 75 tokens for CLIP
truncated_prompt = self.truncate_prompt(prompt)

image = self.image_generator(truncated_prompt).images[0]
```

As a result, the interface documentation processes in the BooTunes project were comprehensively planned and implemented to create understandable, clear, maintainable and independent components. This facilitates communication within the team and ensures that the system is manageable and expandable in the long term.

1.3. Engineering Standards BooTunes

While developing the BooTunes project, we took into account some important standards and principles accepted in the field of software engineering. We used UML, IEEE, Object-Oriented Design (OOD) and SOLID principles to make the design and development processes of the project more organized, understandable and sustainable.

First, we used UML (Unified Modeling Language) diagrams to show both the general architecture and detailed design of our project, which were included in our previous reports. We drew use case diagrams to understand user scenarios, class diagrams to show the relationships between classes in the system, and sequence and activity diagrams to show how the system works. In this way, we ensured that everyone in the team could clearly see the structure of the project and developed a common understanding of how all modules would work.

We followed Object-Oriented Design (OOD) approaches during coding. We designed our classes especially by considering SOLID principles. For example, we paid attention to each class having a single responsibility, not breaking classes when adding new features, and making sure that dependencies are flexible. Thanks to these principles, we are trying to establish a modular and flexible structure.

Finally, we used Git to ensure version control of the code. To avoid conflicts within the team, we implemented a feature branch, development branch, and main branch structure. Thus, each new feature was developed in its own branch and only the checked-in code was included in the main project.

1.4. Definitions, Acronyms, and Abbreviations

- API (Application Programming Interface): An interface that allows software components to exchange data.
- UML (Unified Modeling Language): A language used to visually model software systems.
- OOD (Object-Oriented Design): An object-oriented, modular and reusable software design approach.
- SOLID: Five fundamental object-oriented design principles for flexible and sustainable software development.
- IEEE: An organization that determines internationally accepted standards in software engineering processes.
- PEP8: An official code writing and style guide accepted for the Python language.
- React: A JavaScript library for developing modern and dynamic web interfaces.
- Emotion Analysis: An artificial intelligence process that analyzes the emotions of texts on book pages.
- Genre: Classifications to which music belongs (e.g. happy, sad).

- RESTful API: An architecture that provides standard data exchange over HTTP.
- JSON: A lightweight data format used in API data transmission.
- Branch: Side development branches opened for different features during software development.
- Main Branch: The main development branch containing approved and working codes.
- Feature Branch: A temporary development branch created for new features.
- Pull Request (PR): A request made to add the codes developed in the branch to the main project.
- JWT (JSON Web Token): A token structure used for user authentication and secure session management.

2. Packages

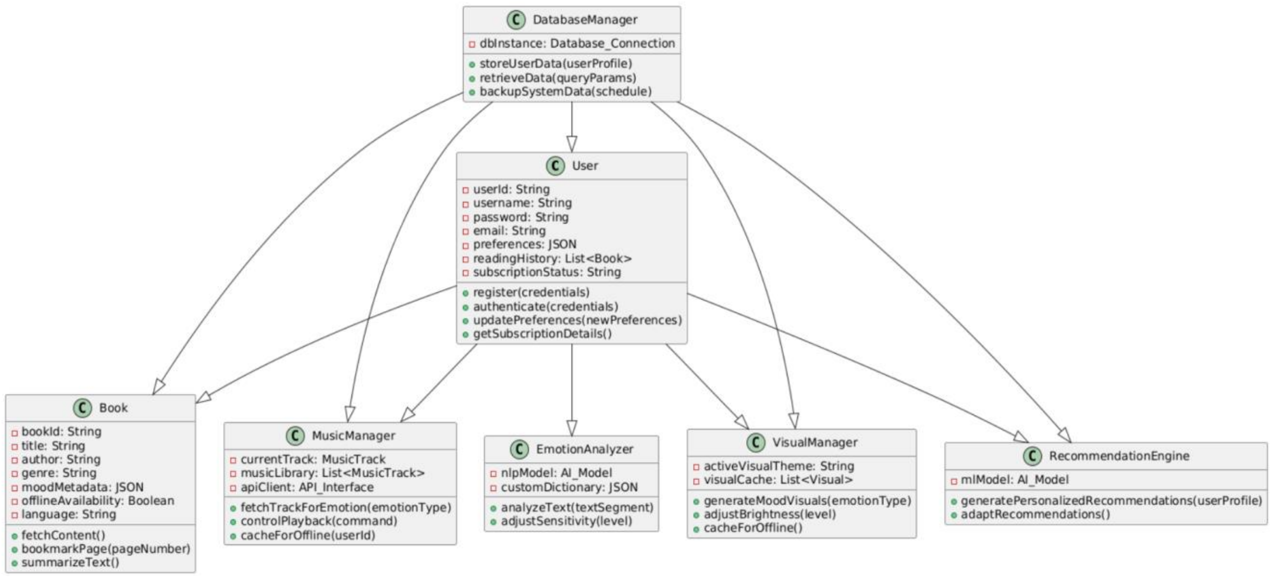
The BooTunes software system is structured into multiple well-defined packages to ensure modularity, maintainability, scalability, and ease of development. Each package encapsulates a distinct aspect of the system, allowing independent development and integration. This section provides a deep dive into each package and its components, detailing the attributes, methods, and responsibilities of each class.

2.1. Server Packages

The server-side implementation of BooTunes follows a **Model-Controller-Service** architecture, with a strong emphasis on API-driven communication, microservices, and asynchronous data processing. The system is designed to handle high-throughput operations with minimal latency, ensuring a seamless reading and listening experience for users.

2.1.1. Model Package

The Model Package contains all core data entities, business logic, and schema definitions used within the BooTunes backend. It is responsible for defining how data is structured, retrieved, and manipulated across the system.



➢ <u>**User Side**</u>

Attributes:

- `userId`: Unique identifier for the user.
- `username`: Display name of the user.
- `password`: Securely hashed password for authentication.
- `email`: User's email address for account recovery.
- `preferences`: JSON object storing music, visual, and reading preferences.
- `readingHistory`: List of books read and emotional responses logged.
- `subscriptionStatus`: Indicates free or premium access level.

Methods:

- `register(credentials)`: Creates a new user account.
- `authenticate(credentials)`: Validates user login credentials.
- `updatePreferences(newPreferences)`: Updates personalized settings.
- `getSubscriptionDetails()`: Retrieves active subscription details.

➢ <u>**Book Side**</u>

Attributes:

- `bookId`: Unique identifier for the book.
- `title`: Book title.

- `author`: Author’s name.
- `genre`: Categorization of the book.
- `moodMetadata`: Stores emotional metadata detected in text.
- `offlineAvailability`: Boolean indicating offline access.
- `language`: Specifies the book's language.

Methods:

- `fetchContent()`: Retrieves and loads book content.
- `bookmarkPage(pageNumber)`: Saves current reading position.
- `summarizeText()`: Generates a summary based on past reading activity.

- **EmotionAnalyzer**

Attributes:

- `nlpModel`: Pretrained deep learning model for sentiment detection.
- `customDictionary`: User-trained emotional dictionary.

Methods:

- `analyzeText(textSegment)`: Processes text and extracts dominant emotional themes.
- `adjustSensitivity(level)`: Modifies analysis depth for user preference.

- **MusicManager**

Attributes:

- `currentTrack`: Stores currently playing track.
- `musicLibrary`: Collection of available emotion-tagged tracks.
- `apiClient`: Interface to interact with external music APIs.

Methods:

- `fetchTrackForEmotion(emotionType)`: Retrieves an appropriate track.
- `controlPlayback(command)`: Handles play, pause, skip actions.
- `cacheForOffline(userId)`: Stores user-specific music preferences for offline playback.

- **<u>VisualManager</u>**

Attributes:

- `activeVisualTheme`: Stores currently selected visual mode.
- `visualCache`: Preloaded assets for rapid retrieval.

Methods:

- `generateMoodVisuals(emotionType)`: Generates AI-generated images or abstract effects.
- `adjustBrightness(level)`: Modifies visual intensity based on preference.
- `cacheForOffline()`: Ensures offline visuals remain accessible.

- **<u>RecommendationEngine</u>**

Attributes:

- `mlModel`: Machine learning model for personalized suggestions.

Methods:

- `generatePersonalizedRecommendations(userProfile)`: Suggests books and music tailored to the user.
- `adaptRecommendations()`: Continuously fine-tunes suggestions based on feedback.

- **<u>DatabaseManager</u>**

Attributes:

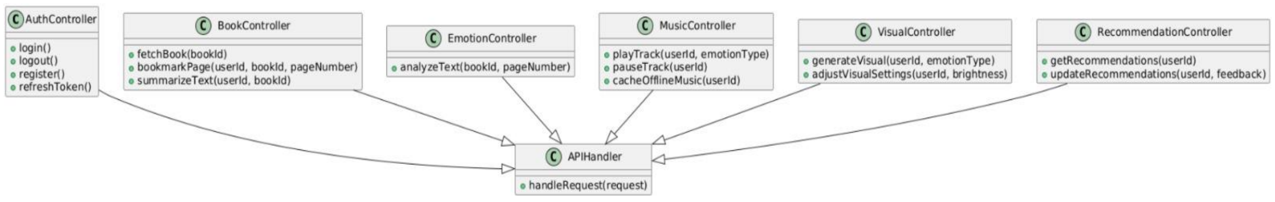
- `dbInstance`: Connection instance to MongoDB.

Methods:

- `storeUserData(userProfile)`: Securely saves user preferences and history.
- `retrieveData(queryParams)`: Fetches relevant data objects.
- `backupSystemData(schedule)`: Ensures data is periodically backed up for recovery.

2.1.2. Controller Package

The Controller Package is responsible for managing API requests, business logic, and communication between the frontend and backend components. Each controller acts as an interface layer to ensure structured and secure interactions.

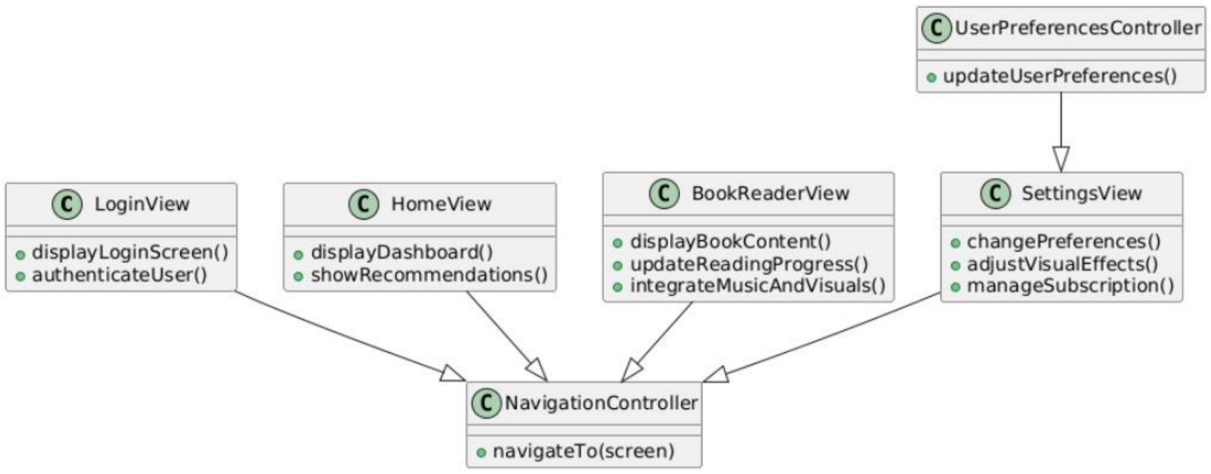

AuthController
login()
logout()
register()
refreshToken()
BookController
fetchBook(bookId)
bookmarkPage(userId, bookId, pageNumber)
summarizeText(userId, bookId)
EmotionController
analyzeText(bookId, pageNumber)
MusicController
playTrack(userId, emotionType)
pauseTrack(userId)
cacheOfflineMusic(userId)
VisualController
generateVisual(userId, emotionType)
adjustVisualSettings(userId, brightness)
RecommendationController
getRecommendations(userId)
updateRecommendations(userId, feedback)
APIHandler
handleRequest(request)

- AuthController
  - Manages login, logout, and registration functionality.
  - Handles JWT-based authentication and token refresh mechanisms.
- BookController
  - Facilitates book retrieval, bookmarking, and offline storage management.
  - Provides endpoints for accessing book summaries and NLP analysis.

- EmotionController
  - Processes real-time text emotion analysis.
  - Communicates with EmotionAnalyzer for AI-based emotion prediction.
- MusicController
  - Adjusts music selection based on real-time emotional analysis.
  - Handles streaming and offline music storage.
- VisualController
  - Generates corresponding mood visuals dynamically.
  - Adjusts visual intensity based on user-defined settings.

2.2. Client Packages

The Client Layer is developed using React.js for web-based access and React Native for mobile devices. This layer manages user interactions, displays AI-enhanced reading elements, and provides intuitive UI controls.



- View Package
  - LoginView: Manages user authentication screens.
  - HomeView: Displays primary dashboard and curated recommendations.
  - BookReaderView: Handles e-book display with AI-driven music and visuals.
  - SettingsView: Enables UI customization and preference updates.
- FrontendController Package
  - NavigationController: Handles routing and state management across UI views.
  - UserPreferencesController: Manages real-time customization settings for users.

2.3. Summary of Package Responsibilities

3. **Package**	**Responsibility**
Model	Defines data structures and interactions with persistent storage.
Controller	Manages API requests, business logic, and authentication.
View	Provides UI rendering and user interaction components.

FrontendController	Manages client-side state and navigation.
Database	Handles data persistence, indexing, and backups.
EmotionAnalyzer	Performs AI-based sentiment analysis on book content.
MusicManager	Matches music with detected emotions and controls playback.
VisualManager	Generates dynamic visuals based on reading mood.
RecommendationEngine	Suggests books and music based on user activity.

This modular approach ensures BooTunes remains highly scalable, adaptable, and easily extensible, allowing future updates and AI-driven enhancements without disrupting the core functionality.

3. Class Interfaces

This section provides detailed explanations of the class interfaces in BooTunes, including attributes and methods for each class. The class structure follows a Model-Controller-View architecture, ensuring modularity, maintainability, and clarity.

3.1. Model Classes

- **User**
  - userId: String - A unique identifier assigned to each user.
  - username: String - The display name chosen by the user.
  - password: String - The user’s securely hashed password.
  - email: String - The registered email address for communication and authentication.

- preferences: JSON - A JSON object storing the user's preferences, such as preferred music genres and visual themes.
- readingHistory: List<Book> - A list tracking books read and associated emotional responses.
- subscriptionStatus: String - Defines whether the user has a free or premium account.
- register(credentials: UserCredentials): User - Creates a new user account.
- authenticate(credentials: UserCredentials): Boolean - Validates the provided credentials.
- updatePreferences(newPreferences: JSON): void - Modifies the user’s stored preferences.
- getSubscriptionDetails(): SubscriptionData - Retrieves the user’s subscription status.

➢ **Book**

- bookId: String - A unique identifier for each book.
- title: String - The title of the book.
- author: String - The name of the author.
- genre: String - The genre classification (e.g., thriller, romance, fiction).
- moodMetadata: JSON - Stores emotional metadata extracted from the book’s content.
- offlineAvailability: Boolean - Indicates whether the book can be accessed offline.
- language: String - Specifies the language of the book.
- fetchContent(): String - Loads the book’s content for reading.
- bookmarkPage(pageNumber: Int): void - Saves the user's current reading position.
- summarizeText(): String - Generates a summary using AI models.

➢ **EmotionAnalyzer**

- nlpModel: AI_Model - The pre-trained NLP model used for emotion detection.
- customDictionary: JSON - A customizable dictionary of emotional keywords.

- Methods:
    - analyzeText(textSegment: String): EmotionData - Processes book text to detect emotional tones.
    - adjustSensitivity(level: Int): void - Modifies the depth of emotion detection based on user settings.

➢ **MusicManager**

- currentTrack: MusicTrack - The currently playing track.
- musicLibrary: List<MusicTrack> - A collection of available music tracks.
- apiClient: API_Interface - An interface for fetching music from external APIs (e.g., Spotify).
- fetchTrackForEmotion(emotionType: String): MusicTrack - Selects a track that matches the detected mood.
- controlPlayback(command: String): void - Controls music playback (play, pause, stop).
- cacheForOffline(userId: String): void - Stores music tracks locally for offline playback.

➢ **VisualManager**

- activeVisualTheme: String - Stores the currently applied visual theme.
- visualCache: List<Visual> - Preloaded visual effects for quick retrieval.
- generateMoodVisuals(emotionType: String): Visual - Creates a visual representation based on the detected mood.
- adjustBrightness(level: Int): void - Adjusts brightness for user preference.
- cacheForOffline(): void - Ensures visuals are available offline.

➢ **RecommendationEngine**

- mlModel: AI_Model - The machine learning model responsible for recommendations.
- generatePersonalizedRecommendations(userProfile: User): List<Book> - Suggests books and music tailored to user preferences.
- adaptRecommendations(): void - Continuously refines suggestions based on user behavior.

- **DatabaseManager**
    - dbInstance: Database_Connection - The connection instance to MongoDB.
    - storeUserData(userProfile: User): void - Saves user data securely.
    - retrieveData(queryParams: Query): DataObject - Fetches stored data based on queries.
    - backupSystemData(schedule: Schedule): void - Handles data backup and recovery processes.

3.2. Controller Classes

The Controller Classes act as intermediaries between the Model and View components, handling user requests, processing business logic, and returning appropriate responses. Each controller manages a specific aspect of the system, ensuring modularity and maintainability.

- **AuthController**

Manages user authentication, registration, and session handling.

Attributes:
- sessionManager: SessionManager - Handles active user sessions.
- tokenService: TokenService - Generates and verifies authentication tokens.

Methods:
- login(credentials: UserCredentials): AuthToken - Verifies user credentials and returns an authentication token if successful.
- logout(userId: String): void - Ends the user's active session and invalidates the authentication token.
- register(userData: User): User - Creates a new user account and stores it in the database.
- refreshToken(oldToken: String): AuthToken - Generates a new authentication token based on a valid expired token.

- **BookController**

Handles book-related operations such as fetching content, bookmarking, and generating summaries.

Attributes:
- bookService: BookService - Handles book retrieval and processing.

Methods:
- fetchBook(bookId: String): Book - Retrieves the book details based on the provided ID.
- bookmarkPage(userId: String, bookId: String, pageNumber: Int): void - Saves the current reading position for a specific user.

- summarizeText(userId: String, bookId: String): String - Generates a concise AI-powered summary of the book’s content.

- **EmotionController**

Manages real-time text analysis to detect emotions within book content.

Attributes:

- emotionService: EmotionAnalyzer - NLP-based service to detect emotional cues in text.

Methods:

- analyzeText(bookId: String, pageNumber: Int): EmotionData - Analyzes the text on a given page and returns detected emotions.

- **MusicController**

Controls the selection, playback, and caching of emotion-driven music tracks.

Attributes:

- musicService: MusicManager - Manages track selection and playback.

Methods:

- playTrack(userId: String, emotionType: String): void - Selects and plays a music track matching the detected emotion.
- pauseTrack(userId: String): void - Pauses the currently playing track for the specified user.
- cacheOfflineMusic(userId: String): void - Preloads and stores frequently played tracks for offline access.

- **VisualController**

Handles the generation and adjustment of visuals based on detected emotions.

Attributes:

- visualService: VisualManager - Manages the selection and rendering of visual effects.

Methods:

- generateVisual(userId: String, emotionType: String): Visual - Creates a visual representation that matches the book’s emotional tone.
- adjustVisualSettings(userId: String, brightness: Int): void - Modifies brightness and visual intensity settings for user customization.

- **RecommendationController**

Provides book and music recommendations tailored to user preferences.

Attributes:

- recommendationService: RecommendationEngine - AI-based service for personalized suggestions.

Methods:

- getRecommendations(userId: String): List<Book> - Returns a list of recommended books based on the user's history and preferences.
- updateRecommendations(userId: String, feedback: FeedbackData): void - Adjusts recommendations dynamically based on user feedback.

- **APIHandler**

Processes all incoming API requests and routes them to the appropriate controllers.

Attributes:

- requestQueue: Queue<APIRequest> - Stores incoming API requests for processing.

Methods:

- handleRequest(request: APIRequest): APIResponse - Parses the request, invokes the appropriate controller, and returns a response.

This Controller Layer ensures smooth communication between the Model and View components, enabling dynamic content delivery and efficient user interactions.

3.3. Client Classes

The Client Classes define the user interface (UI) components of BooTunes, handling user interactions and rendering data from the Controller Layer. These classes ensure a seamless and engaging user experience by integrating real-time data with UI elements.

- **LoginView**

Handles user authentication, login, and session management.

Attributes:

- authService: AuthController - Handles authentication requests.
- errorMessage: String - Stores any login error messages.

Methods:

- displayLoginScreen(): void - Renders the login page with input fields.
- authenticateUser(username: String, password: String): Boolean - Sends credentials to the server and validates authentication.
- showErrorMessage(message: String): void - Displays error messages in case of failed login attempts.

- **HomeView**

Provides the main dashboard displaying user recommendations and quick access to books.

Attributes:

- recommendationService: RecommendationController - Fetches personalized book and music recommendations.
- userPreferences: JSON - Stores user settings for UI customization.

Methods:

- displayDashboard(): void - Shows recommended books, recent reads, and featured content.

- showRecommendations(): List<Book> - Fetches and displays AI-generated recommendations.

- **BookReaderView**

Renders book content and integrates emotion-based enhancements.

Attributes:
- bookService: BookController - Manages book content retrieval.
- musicService: MusicController - Controls mood-based music playback.
- visualService: VisualController - Handles emotion-based visual effects.
- currentPage: Int - Stores the user's current reading position.

Methods:
- displayBookContent(bookId: String): void - Loads and displays book text.
- updateReadingProgress(pageNumber: Int): void - Saves the user's reading position.
- integrateMusicAndVisuals(): void - Syncs book content with AI-generated visuals and music.
- adjustTextSize(size: Int): void - Allows users to modify text size for accessibility.

- **SettingsView**

Provides an interface for users to customize their experience.

Attributes:
- userService: UserPreferencesController - Fetches and updates user preferences.
- darkModeEnabled: Boolean - Stores theme preferences.
- selectedMusicGenre: String - Stores preferred music genre.

Methods:
- changePreferences(newPreferences: JSON): void - Updates user settings for music, visuals, and notifications.
- adjustVisualEffects(brightness: Int, contrast: Int): void - Modifies visual effects settings.
- manageSubscription(action: String): void - Handles subscription upgrades or cancellations.

- **NavigationController**

Manages screen transitions and navigation between views.

Attributes:
- currentScreen: String - Stores the currently displayed screen.
- historyStack: List<String> - Tracks navigation history.

Methods:
- navigateTo(screen: String): void - Transitions to a specified screen.
- goBack(): void - Navigates back to the previous screen.
- resetNavigation(): void - Clears the navigation history and returns to the home screen.

- **UserPreferencesController**

Handles the storage and modification of user preferences.

Attributes:

- preferencesData: JSON - Stores user-specific settings.
- settingsCache: Map<String, String> - Temporarily caches settings changes before saving.

Methods:

- updateUserPreferences(newPreferences: JSON): void - Saves the latest preference changes to the database.
- fetchUserPreferences(userId: String): JSON - Retrieves stored user settings from the server.
- resetToDefault(): void - Restores default preferences.

This Client Layer ensures an interactive and dynamic user experience in BooTunes, allowing seamless access to personalized content and real-time AI enhancements.

4. Glossary

- API: Interface that allows software components to communicate.
- AI: Intelligent systems that perform human-like tasks.
- Backend: Server side of the application.
- Book Page Emotion Analysis: Analyze the emotion on the book page.
- CRUD: Add, read, update, delete data operations.
- JWT: Secure authentication method.
- Machine Learning: An artificial intelligence system that learns from data.
- Music Genre: A type of music (e.g. happy, sad).
- OOD: Object-oriented software design.
- React: Web interface development library.
- RESTful API: Data transmission over HTTP.
- SOLID Principles: Object-oriented software design principles.
- Token: Digital authentication data.
- UI: User interface.
- Version Control: A system for tracking code changes.

5. References

- ✓ ACM Code of Ethics: https://www.acm.org/code-of-ethics
- ✓ Hugging Face Transformers Documentation: https://huggingface.co/docs/transformers
- ✓ MongoDB Documentation: https://www.mongodb.com/docs/
- ✓ React Documentation: https://reactjs.org/docs/getting-started.html
- ✓ React Native Documentation: https://reactnative.dev/docs/getting-started
- ✓ Spotify API Documentation: https://developer.spotify.com/documentation/web-api
- ✓ IEEE Code of Ethics: https://www.ieee.org/about/corporate/governance/p7-8.html