Software Design Document (SDD)
Group 7

MuseAI

Lyrics and Chord Generation System

Software Engineering Project

METCS 673

BOSTON UNIVERSITY

Tridev Rapeti
Shreni Singh
Lingjie Yuan
Aayush Raghuvanshi

29th October 2024

1 Introduction

1.1 Purpose

This Software Design Document (SDD) aims to give a thorough overview of the MuseAI system's architecture, design, and functioning. Based on user-specified criteria such musical themes, genres, and styles, MuseAI is an artificial intelligence-powered song generating tool that generates creative musical compositions, complete with chords and lyrics. This document accomplishes several goals:

- To outline the system architecture and the relationships between various components of MuseAI.
- To specify each module's needs and specs, guaranteeing the development team's understanding and cooperation.
- to give developers, designers, and stakeholders a clear, organized grasp of the system's features and design choices in order to promote communication.
- To serve as a reference document throughout the development process, providing guidance for implementation and testing.

By articulating the purpose of this SDD, we aim to ensure that all stakeholders are on the same page regarding the project's objectives, allowing for efficient collaboration and successful project outcomes.

1.2 Scope

The scope of the MuseAI system encompasses the following components and functionalities:

- **User Interface (UI):** Users will be able to enter musical themes and genres in an easy-to-use and intuitive environment thanks to the user interface. It will show the lyrics and chords of the generated song, guaranteeing a smooth interaction experience.
- **Backend Services:** All application logic, user session management, and communication between the database, music generating engine, and user interface will be handled by the backend. Additionally, it will oversee the permission and authentication procedures for user access.
- **Music Generation Engine:** In order to assess user inputs and produce creative music material in real time, this core module will make use of sophisticated algorithms, possibly including deep learning techniques. It will guarantee that the produced music complies with the supplied parameters and user preferences.
- **Database Management System (DBMS):** User information, preferences, and generated music material will all be stored and retrieved by the DBMS. In order to manage an increasing volume of users and information, it will be built with scalable operations in mind.

- **Deployment Environment:** In order to guarantee high availability, scalability, and accessibility, the MuseAI system will be implemented in a cloud environment. Because of the thin-client architecture, users will be able to access the application from a variety of devices with little need for local resources.

The marketing plan for the MuseAI application and post-launch support and maintenance protocols will be covered separately as the project develops, and neither will be covered in this document. The technical components of the system's design and implementation are the only emphasis of the SDD.

2 References

The main sources that influenced the MuseAI system's design and development are listed in this section. These sources include classic works on design patterns, music production methods, and software engineering ideas. The resources listed below are crucial for comprehending the technology and methods used in this project:

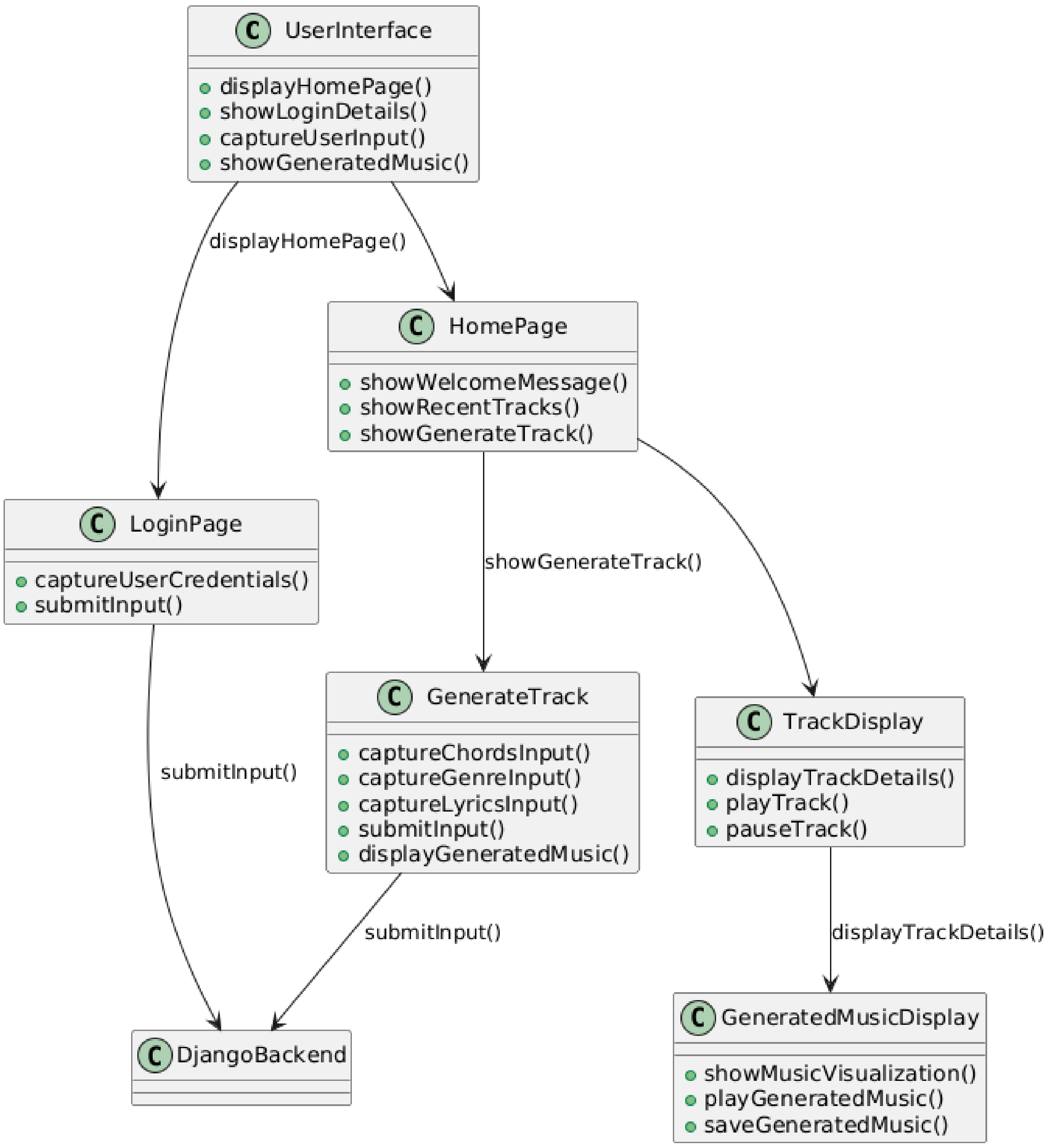
- Pressman, R. S. (2014). *Software Engineering: A Practitioner's Approach.* McGraw-Hill Education.
- Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley.
- Boulanger, A. (2012). *Deep Learning for Music Generation: A Survey.* IEEE Transactions on Neural Networks and Learning Systems.
- Official Documentation for Django Framework. *Django Project.* Retrieved from https://www.djangoproject.com/.
- Research articles on music generation algorithms, including:
  - Huang, A., Yang, Y. (2020). *Music Generation with Transformers: A Review of Recent Approaches.* Journal of New Music Research.
  - Dong, H. W., Yang, Y. (2021). *AI-Generated Music: A Review of Recent Advances and Challenges.* IEEE Access.

3 Decomposition Description

3.1 Module Decomposition

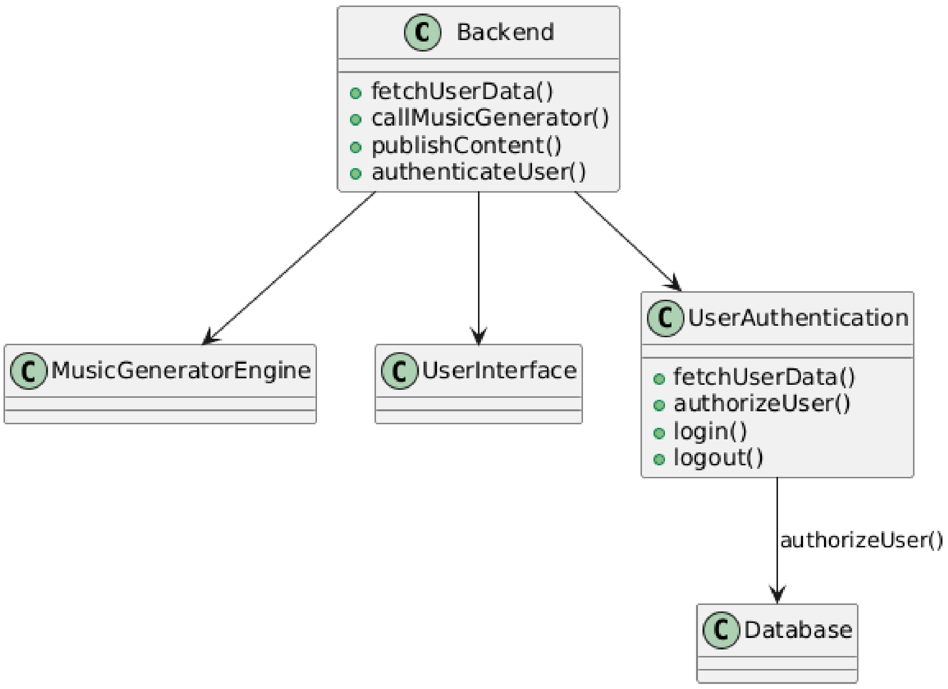
3.1.1 User Interface (UI)

- Manages user interactions and displays results.
- Class Diagram:



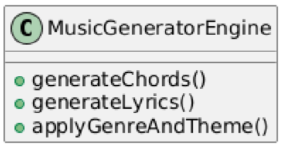
3.1.2 Backend

- Receives input from the user interface and calls Music Generation Engine and database in necessary events.
- Class Diagram:



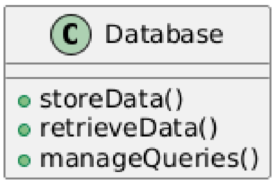
3.1.3 Music Generator Engine

- Receives call with parameters from Backend and runs complex LLM Algorithms to generate desired content and send the data back to Backend
- Class Diagram:



3.1.4 Database

- Receives call with parameters from Backend for user data, fetches user data, and sends them back to Backend.
- Class Diagram:

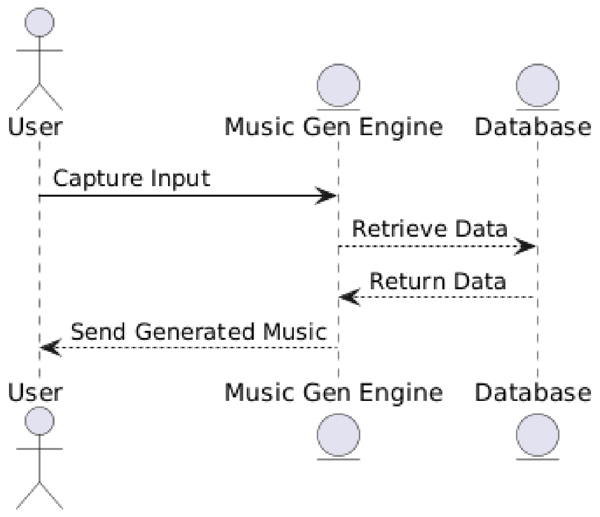


3.2 Concurrent Process Decomposition

3.2.1 User Interface (UI) Process

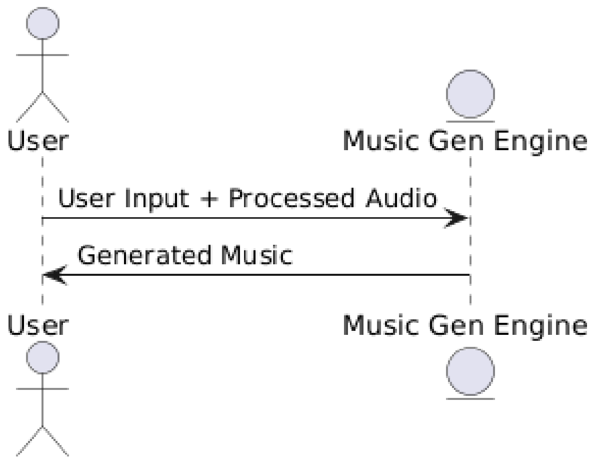
- Manages real-time user interactions.

- Flow Diagram:



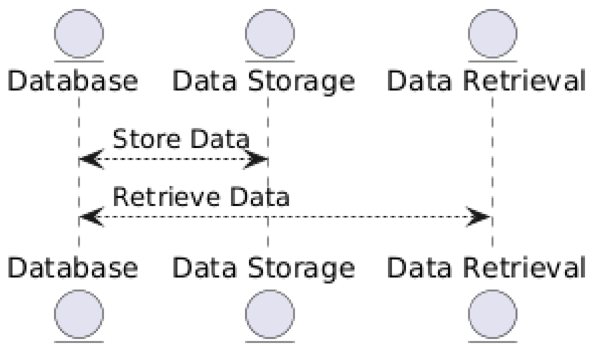
3.2.2 Music Generation Process

- Runs LLM algorithm and generates chords and lyrics parallelly while UI shows some graphics.
- Flow Diagram:



3.2.3 Database Operations Process

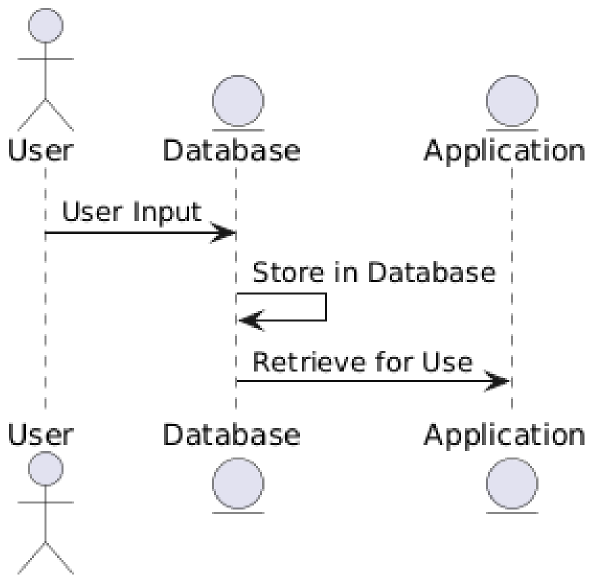
- Handles data storage and retrieval in parallel.
- Flow Diagram:



3.3 Data Decomposition

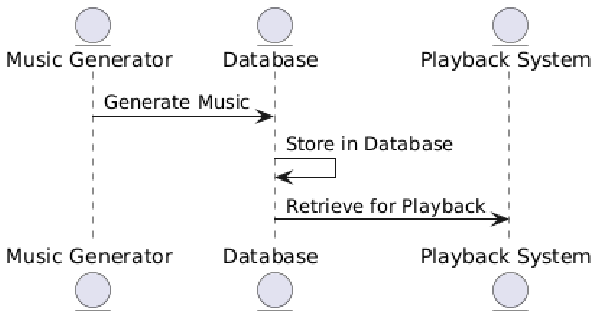
3.3.1 User Data

- Includes profiles, preferences, and authentication details.
- Flow Diagram:



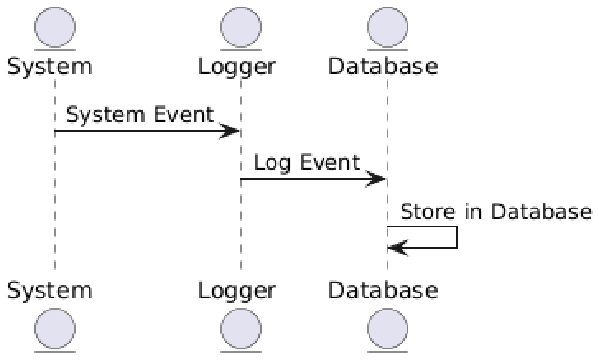
3.3.2 Generated Music

- Stores output of the Music Generation Engine.
- Flow Diagram:



3.3.3 System Logs

- Logs of interactions, performance, and errors.
- Flow Diagram:



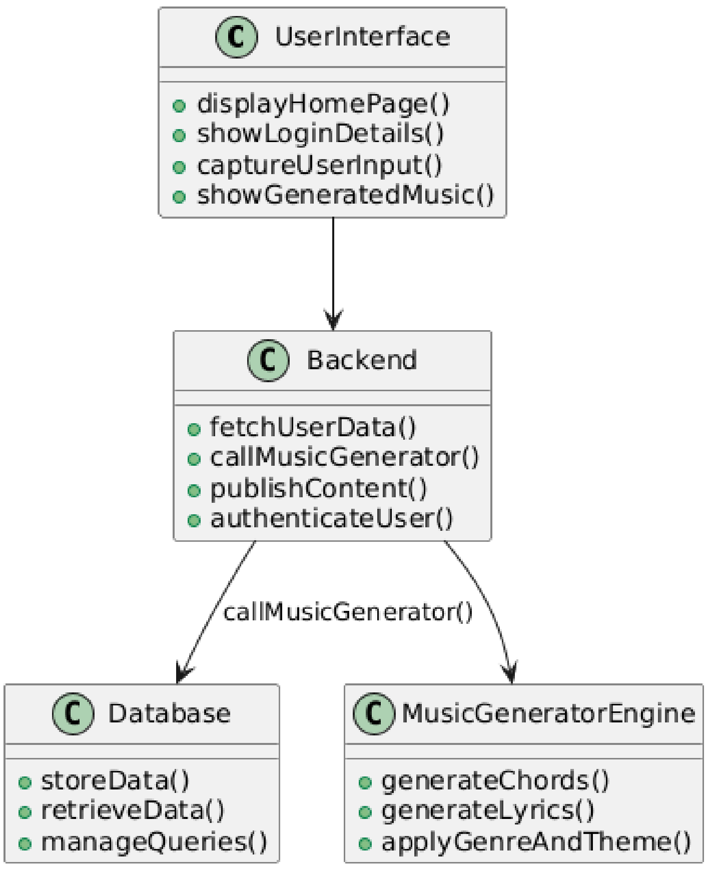
4 Dependency Description

4.1 Intermodule Description

We have four modules in total:

- **User Interface(UI):** Captures user input, displays generated chords and lyrics.
- **Backend:** Manages logic and data flow between UI, Music Generator Engine, and Database.
- **Music Generator Engine:** Generates chords and lyrics based on genre and theme inputs.
- **Database:** Stores user data and generated content.

4.1.1 Class Diagram



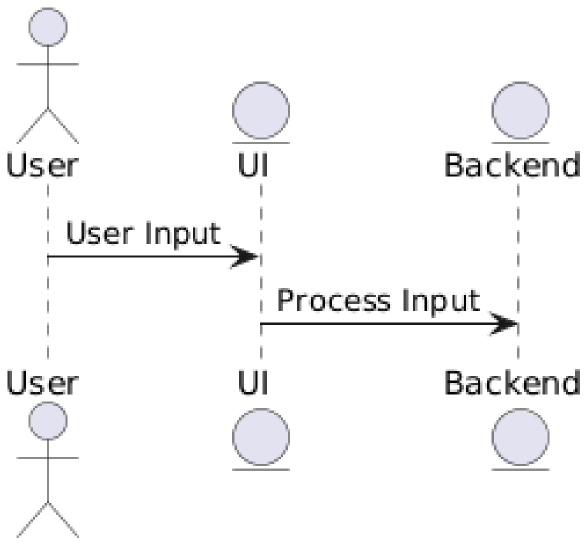
4.2 Interprocess Description

Processes:

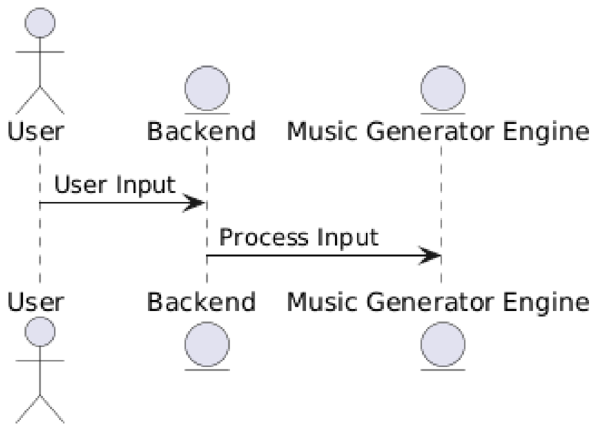
- **User Input Handling:** Manages real-time user inputs.
- **Backend Processing:** Handles logic and data communication.
- **Chord and Lyrics Generation:** Generates content using AI algorithms.
- **Data Management:** Stores and retrieves data.
- **Authentication Process:** Manages user sessions.

4.2.1 Flow Diagram

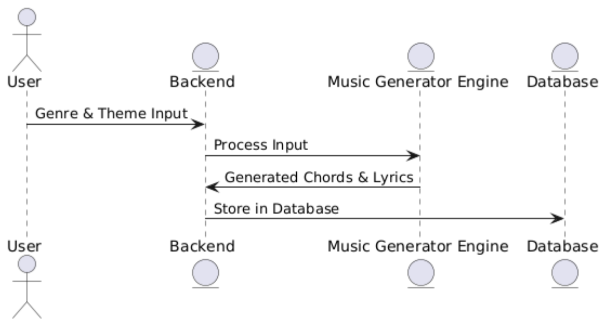
- **User Input Handling**



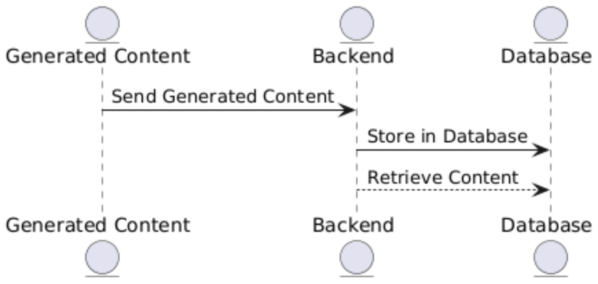
- **Backend Processing**



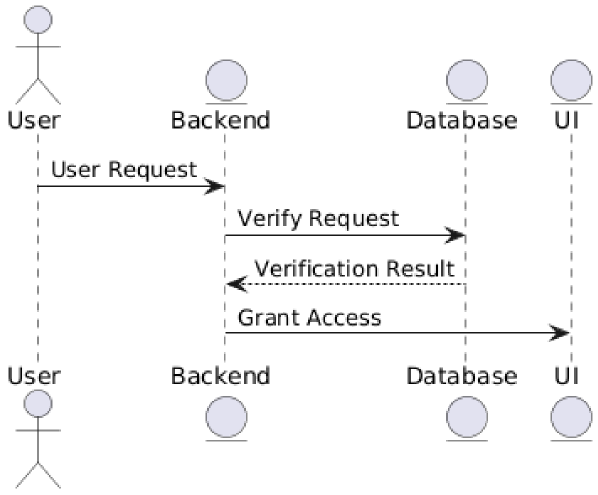
- **Chord and Lyrics Generation**



- **Data Management**



- **Authentication Process**



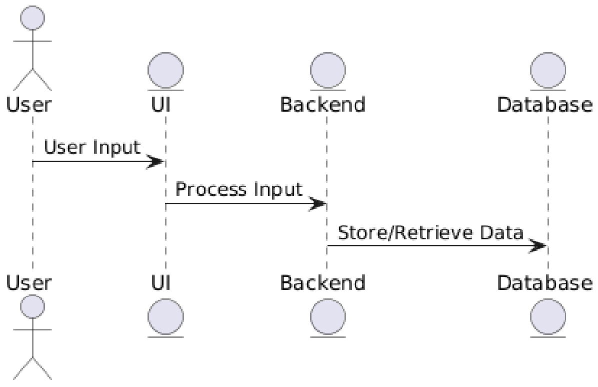
4.3 Data Dependencies

Data Components

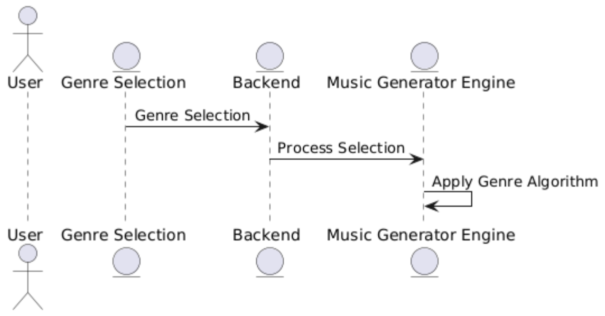
- **User Data:** Profiles, preferences, and authentication details.
- **Genre Data:** Information and algorithms related to different genres.
- **Lyrics Theme Data:** Data related to various lyrics themes.
- **Generated Content:** Chords and lyrics generated by the system.
- **System Logs:** Logs of interactions and performance.

4.3.1 Flow Diagram

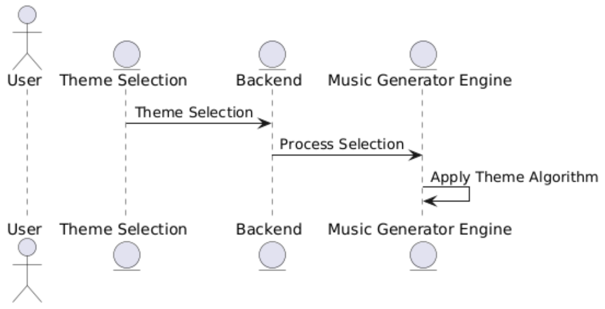
- **User Data**



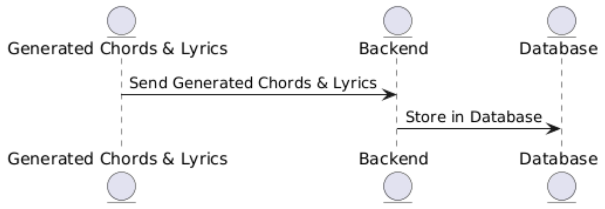
- **Genre Data**



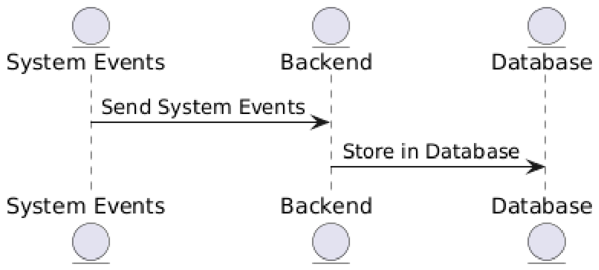
- **Lyrics Theme Data**



- **Generated Content**



- **System Logs**



5 Interface Description

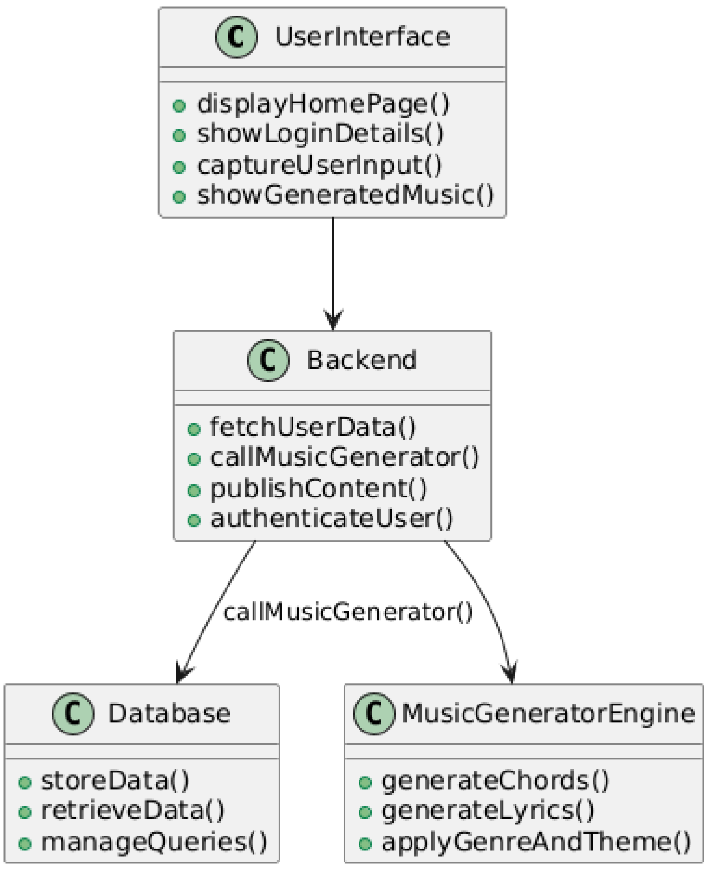
5.1 Module Interface

This section describes the interaction between different modules within the system.

5.1.1 Modules

- User Interface (UI)
- Backend
- MusicGeneratorEngine
- Database

5.1.2 Class Diagram



5.1.3 Interactions

- **UI to Backend:**
  - **Functions:** *displayHomePage(), showLoginDetails(), showGeneratedMusic()*
  - **Dataflow:** HTML Templates are rendered from backend, user inputs are captured and sent to backend, and generated content and user login details are received.
- **Backend to MusicGeneratorEngine:**
  - **Functions:** *callMusicGeneratorEngine()*
  - **Dataflow:** User inputs for genre and lyrics themes are collected and sent to Music Generator Engine and receive generated content in return.

- **Backend to Database:**
  - **Functions:** *fetchUserData()*
  - **Dataflow:** User data is requested from database based on a primary key which the user provides and receives user data from the database.

5.2 Process interface

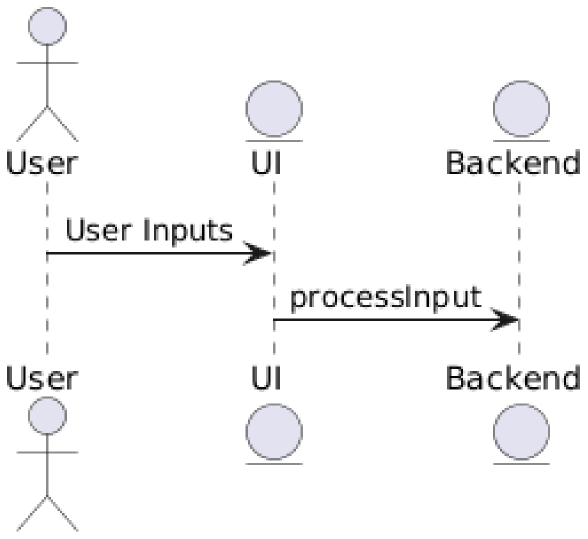
This section describes the interaction between different processes within the system.

5.2.1 Processes

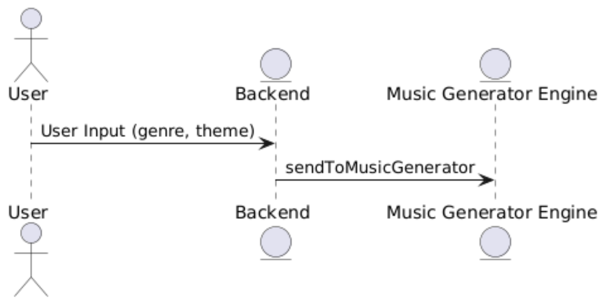
- User Input Handling
- Backend Processing
- Chord and Lyrics Generation
- Data Management
- Authentication Process

5.2.2 Flow Diagram

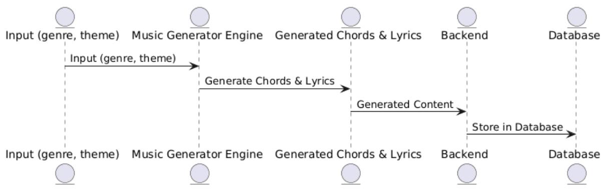
- **User Input Handling**



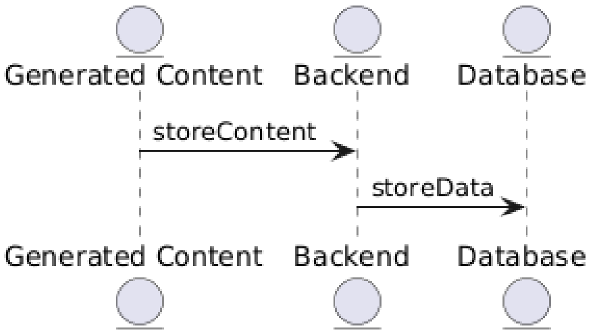
- **Backend Processing**



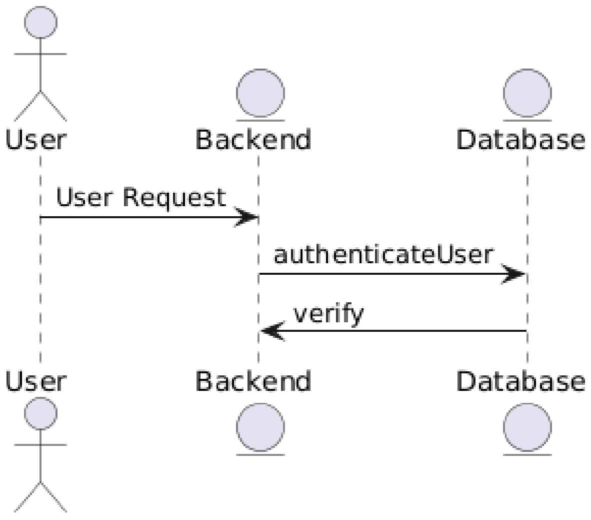
- **Chord and Lyrics Generation**



- **Data Management**



- **Authentication Process**



6 Detailed Design

6.1 Module Detailed Design

Each component of the MuseAI system is described in the module detailed design section, along with its functions, interfaces, and relationships with other modules. Every module is made to work independently while enhancing the functionality and user experience of the system as a whole.

6.1.1 User Interface (UI)

The User Interface (UI) module is the primary point of interaction for users. It is designed to be intuitive and responsive, allowing users to easily input their preferences and view generated music content.

- **Responsibilities:**
  - Capture user inputs for music generation (genre, theme, etc.).
  - Display generated chords and lyrics in a user-friendly format.
  - Provide feedback to users (e.g., loading indicators during music generation).
- **Interfaces:**
  - Interacts with the Backend module via RESTful API calls to send user input and receive generated content.
  - Utilizes front-end technologies (HTML, CSS, JavaScript, and possibly frameworks like React or Vue.js) to enhance user interaction.
- **Design Considerations:**
  - Ensure responsiveness across different devices (mobile, tablet, desktop).
  - Implement accessibility features to cater to users with disabilities.
  - Provide clear navigation and user guidance to facilitate a seamless user experience.

6.1.2 Backend Services

The Backend module acts as the intermediary between the UI and the other components of the system. It manages application logic, data flow, and user authentication.

- **Responsibilities:**
  - Process user requests and manage data flow between the UI, Music Generator Engine, and Database.
  - Handle user authentication and authorization.
  - Implement business logic to validate user inputs and manage session states.
- **Interfaces:**
  - Communicates with the UI through REST APIs to send and receive data.
  - Interfaces with the Music Generator Engine to initiate music generation processes.
  - Connects to the Database for user data storage and retrieval.
- **Design Considerations:**
  - Ensure scalability to handle multiple user requests simultaneously.
  - Implement security measures (e.g., input validation, encryption) to protect user data.
  - Utilize efficient algorithms for data processing and management.

6.1.3 Music Generator Engine

The Music Generator Engine is the core of the MuseAI system, responsible for generating chords and lyrics based on user-defined parameters.

- **Responsibilities:**
  - Generate music content (chords and lyrics) based on inputs from the Backend.
  - Apply relevant algorithms and models to produce high-quality music outputs.
- **Interfaces:**
  - Receives genre and theme inputs from the Backend.
  - Sends generated music data back to the Backend for storage and retrieval.
- **Design Considerations:**
  - Utilize state-of-the-art machine learning techniques for music generation.
  - Ensure the generated content adheres to the user's selected genre and theme.
  - Implement performance optimizations to reduce generation time.

6.1.4 Database

The Database module is responsible for managing the storage and retrieval of user data, preferences, and generated music content.

- **Responsibilities:**
  - Store user profiles, preferences, and authentication details.
  - Archive generated music content for future access and playback.
  - Manage data integrity and security.
- **Interfaces:**
  - Communicates with the Backend to store and retrieve data as needed.
- **Design Considerations:**
  - Choose a suitable database system (SQL or NoSQL) based on data requirements and access patterns.
  - Implement backup and recovery mechanisms to protect user data.
  - Ensure fast data retrieval and storage to support real-time interactions.

6.2 Data Detailed Design

This section outlines the data structures and storage methods used in the MuseAI system. Effective data management is crucial for the application's performance and reliability.

6.2.1 User Data

User data is essential for personalizing the user experience and managing user interactions with the system.

- **Data Structure:**
  - `UserProfile`
    - `userID`: Unique identifier for the user (Primary Key).
    - `username`: User's display name.
    - `passwordHash`: Hashed password for authentication.
    - `preferences`: JSON object storing user preferences (e.g., favorite genres, themes).
    - `createdAt`: Timestamp of when the user profile was created.
- **Storage:**
  - Stored in the Database under the `Users` collection/table.
  - Indexes created on `userID` and `username` for fast lookups.

6.2.2 Generated Music Data

This data includes the output of the music generation process, which needs to be efficiently stored and retrievable.

- **Data Structure:**
  - `GeneratedMusic`
    - `musicID`: Unique identifier for the generated music (Primary Key).
    - `userID`: Foreign key linking to the user who generated the music.
    - `chords`: Text or array of chords generated.
    - `lyrics`: Text of the generated lyrics.
    - `genre`: Genre applied to the music.
    - `theme`: Theme applied to the music.
    - `createdAt`: Timestamp of when the music was generated.
- **Storage:**
  - Stored in the Database under the `GeneratedMusic` collection/table.
  - Indexes created on `musicID` and `userID` for efficient data retrieval.

6.2.3 System Logs

System logs track application performance, user interactions, and errors for monitoring and debugging purposes.

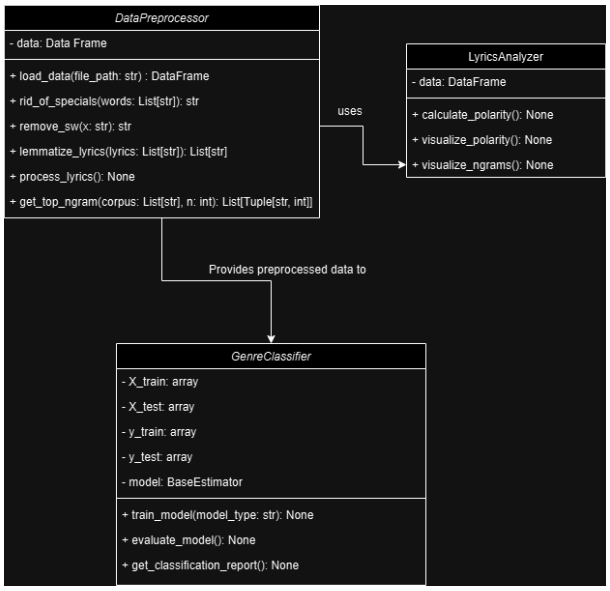
- **Data Structure:**
  - `SystemLog`

        * `logID`: Unique identifier for the log entry (Primary Key).
        * `eventType`: Type of event (e.g., user action, system error).
        * `description`: Description of the event.
        * `timestamp`: Timestamp of when the event occurred.

- **Storage:**
    - Stored in the Database under the `Logs` collection/table.
    - Regular archiving of old logs to maintain performance and manage storage efficiently.

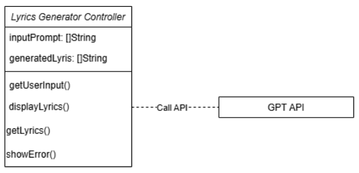
7 Additional Class Diagrams

7.1 Genre Classifier class diagram



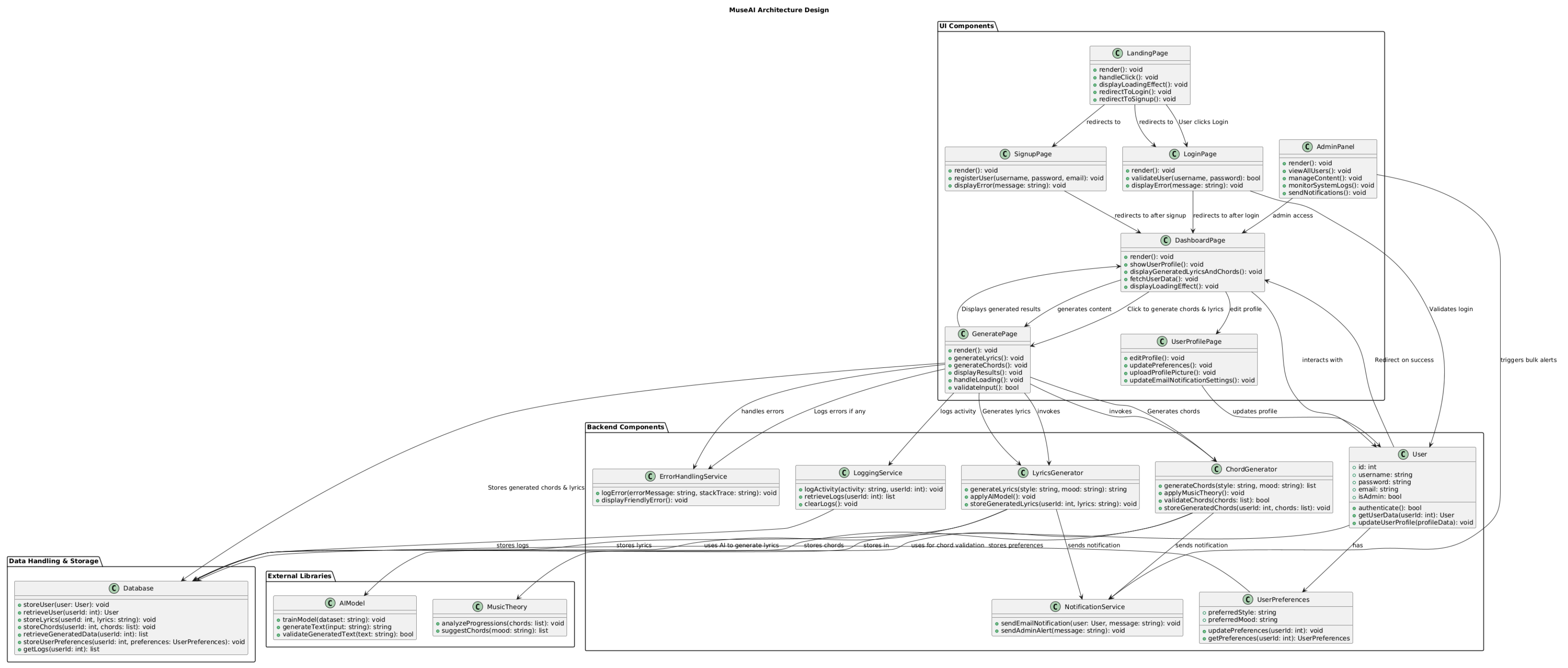
DataPreprocessor
- data: Data Frame
+ load_data(file_path: str) : DataFrame
+ rid_of_specials(words: List[str]): str
+ remove_sw(x: str): str
+ lemmatize_lyrics(lyrics: List[str]): List[str]
+ process_lyrics(): None
+ get_top_ngram(corpus: List[str], n: int): List[Tuple[str, int]]
uses
LyricsAnalyzer
- data: DataFrame
+ calculate_polarity(): None
+ visualize_polarity(): None
+ visualize_ngrams(): None
Provides preprocessed data to
GenreClassifier
- X_train: array
- X_test: array
- y_train: array
- y_test: array
- model: BaseEstimator
+ train_model(model_type: str): None
+ evaluate_model(): None
+ get_classification_report(): None

7.2 Lyrics Generator class diagram



For the Lyrics Generator part, we only have 1 class in our project, the Lyrics Generator Controller. It helps the website to call the GPT API, pass the input keywords, and get the generated lyrics. We also add a showError function here for a better user and developer experience.

7.3 Architecture Overview


MuseAI Architecture Design
UI Components
LandingPage
+ render(): void
+ handleClick(): void
+ displayLoadingEffect(): void
+ redirectToLogin(): void
+ redirectToSignup(): void
redirects to
redirects to
User clicks Login
SignupPage
+ render(): void
+ registerUser(username, password, email): void
+ displayError(message: string): void
LoginPage
+ render(): void
+ validateUser(username, password): bool
+ displayError(message: string): void
AdminPanel
+ render(): void
+ viewAllUsers(): void
+ manageContent(): void
+ monitorSystemLogs(): void
+ sendNotifications(): void
redirects to after signup
redirects to after login
admin access
DashboardPage
+ render(): void
+ showUserProfile(): void
+ displayGeneratedLyricsAndChords(): void
+ fetchUserData(): void
+ displayLoadingEffect(): void
Displays generated results
generates content
Click to generate chords & lyrics
edit profile
Validates login
GeneratePage
+ render(): void
+ generateLyrics(): void
+ generateChords(): void
+ displayResults(): void
+ handleLoading(): void
+ validateInput(): bool
UserProfilePage
+ editProfile(): void
+ updatePreferences(): void
+ uploadProfilePicture(): void
+ updateEmailNotificationSettings(): void
interacts with
Redirect on success
triggers bulk alerts
handles errors
Logs errors if any
logs activity
Generates lyrics
invokes
invokes
Generates chords
updates profile
Backend Components
ErrorHandlingService
+ logError(errorMessage: string, stackTrace: string): void
+ displayFriendlyError(): void
LoggingService
+ logActivity(activity: string, userId: int): void
+ retrieveLogs(userId: int): list
+ clearLogs(): void
LyricsGenerator
+ generateLyrics(style: string, mood: string): string
+ applyAIModel(): void
+ storeGeneratedLyrics(userId: int, lyrics: string): void
ChordGenerator
+ generateChords(style: string, mood: string): list
+ applyMusicTheory(): void
+ validateChords(chords: list): bool
+ storeGeneratedChords(userId: int, chords: list): void
User
+ id: int
+ username: string
+ password: string
+ email: string
+ isAdmin: bool
+ authenticate(): bool
+ getUserData(userId: int): User
+ updateUserProfile(profileData): void
Stores generated chords & lyrics
stores logs
stores lyrics
uses AI to generate lyrics
stores chords
stores in
uses for chord validation
stores preferences
sends notification
sends notification
has
NotificationService
+ sendEmailNotification(user: User, message: string): void
+ sendAdminAlert(message: string): void
UserPreferences
+ preferredStyle: string
+ preferredMood: string
+ updatePreferences(userId: int): void
+ getPreferences(userId: int): UserPreferences
Data Handling & Storage
Database
+ storeUser(user: User): void
+ retrieveUser(userId: int): User
+ storeLyrics(userId: int, lyrics: string): void
+ storeChords(userId: int, chords: list): void
+ retrieveGeneratedData(userId: int): list
+ storeUserPreferences(userId: int, preferences: UserPreferences): void
+ getLogs(userId: int): list
External Libraries
AIModel
+ trainModel(dataset: string): void
+ generateText(input: string): string
+ validateGeneratedText(text: string): bool
MusicTheory
+ analyzeProgressions(chords: list): void
+ suggestChords(mood: string): list