
Software Design Specification

for

MuseScribe

Version 1.0

Prepared by Osniel T, Alexander J, Victoria M, Jorge M

MDC

02/16/2025

Copyright © 1999 by Karl E. Wiegers. Permission is granted to use, modify, and distribute this document.(SRS sections)

Copyright © 1994-1997 by Bradford D. Appleton. Permission is hereby granted to make and distribute verbatim copies of this document provided the copyright notice and this permission notice are preserved on all copies.(SDS sections)

Revision History	ii
1. Introduction.....	1
1.1 Purpose	1
1.2 Document Conventions	1
1.3 Intended Audience and Reading Suggestions	1
1.4 Product Scope.....	1
1.5 References	2
2. Overall Description	2
2.1 Product Perspective	2
2.2 Product Functions.....	2
2.3 User Classes and Characteristics.....	2
2.4 Operating Environment.....	2
2.5 Design and Implementation Constraints	3
2.6 User Documentation.....	3
2.7 Assumptions and Dependencies.....	3
3. External Interface Requirements	3
3.1 User Interfaces Overview	3
3.2 Hardware Interfaces	4
3.3 Software Interfaces.....	4
4. Design Considerations	4
4.1 Assumptions and Dependencies.....	5
4.2 General Constraints	5
4.3 Development Methods	5
5. System Architecture.....	5
5.1 Architectural Strategies.....	5
5.2 High level Overview of System Architecture	6
6. Human Interface Design.....	6
6.1 Screen Images	6
6.2 Screen Objects and Actions.....	9
7. Detailed System Design.....	9
7.1 Data Structures	9
7.2 Frontend Application (Vanilla JavaScript with ABC.js)	9
7.3 AI Processing Service	13
7.4 Backend API (Java Spring Boot)	14
Appendix A: Glossary.....	17

Revision History

Name	Date	Reason For Changes	Version
Alexander J	02/12/25		
Osniel T	03/05/25		

Osniel T	03/30/25	Update 7 sections to align with codebase updates	
----------	----------	--	--

1. Introduction

1.1 Purpose

This SDS specifies the design of MuseScribe a music notation application that transcribes microphone and MIDI input audio into sheet music. It covers the detailed system architecture, module design, data structures, and interfaces necessary for implementation. The document targets software developers, system integrators, and quality assurance personnel, ensuring that every design decision aligns with project goals of accuracy, scalability, and usability.

1.2 Document Conventions

The documentation provided follows the standard IEEE formatting. Headings, subheadings, and bullet lists are used consistently to delineate sections and highlight key points. All priority requirements are clearly marked, and any assumptions or dependencies are stated explicitly.

1.3 Intended Audience and Reading Suggestions

- **Developers & Engineers:** To implement modules based on detailed design specifications.
- **Project Managers & System Architects:** For oversight on design decisions and architectural strategies.
- **Testers & QA Specialists:** To understand component interactions and integration points. Readers should begin with the overall description in Section 2, then proceed to detailed design sections (Sections 4–7).

1.4 Product Scope

1.4.1 Purpose

This software is intended to be used by music writers to simplify the process of music notation. This software will identify pitches and durations of notes played on an audio input device, such as a MIDI or microphone, and transcribe it to the music staff.

1.4.2 High Level Flow

- **Step 1:** The user either uploads an audio file or connects a MIDI instrument.
- **Step 2:** Input is preprocessed on the client side.
- **Step 3:** Audio/MIDI data is transmitted to the backend for transcription using the MT3 model.
- **Step 4:** transcribed midi data is converted to abc notation and returned to the client
- **Step 5:** The frontend retrieves the data and renders it as interactive sheet music via ABC.js.
- **Step 6:** Users can edit, playback, and export the final sheet music.

1.5 References

- **MT3 Documentation:** <https://github.com/magenta/mt3>
- **ABC.js Documentation:** <http://abcjs.net/>
- **Spring Boot Documentation:** <https://spring.io/projects/spring-boot>
- **Additional References:** IEEE SDS templates and Java/Python development best practices.

2. Overall Description

2.1 Product Perspective

MuseScribe is a self-contained, web-based product developed from scratch to integrate advanced AI transcription with a user-friendly interface. It is not an extension of an existing product line but rather a modern solution designed to replace manual transcription methods with automated, efficient processing.

2.2 Product Functions

- **Audio-to-MIDI Transcription:** Using the MT3 model to process audio inputs.
- **Real-Time MIDI Processing:** Capturing live MIDI signals for immediate transcription.
- **Sheet Music Rendering:** Converting MIDI data into editable sheet music via ABC.js.
- **Real-Time Microphone Processing:** Converting live audio input from microphone into transcription.
- **User Dashboard:** Allowing users to manage, search, and export transcriptions.
- **Notebook Management:** Allows users to create/list/delete notebooks that group together sheet music.
- **Interactive Playback:** Enabling users to listen to and edit the transcriptions.
- **Google Login:** Allow users to login via their Google accounts.

2.3 User Classes and Characteristics

- **Musicians & Composers:** Need high transcription accuracy and intuitive editing features.
- **Students & Educators:** Require a clear interface for learning and demonstrating music theory.
- **Developers & Integrators:** Demand modular, well-documented APIs for system extension or integration.
- **General Enthusiasts:** Seek a straightforward tool to create and share musical scores.

2.4 Operating Environment

- **Backend:** Java Spring Boot hosted on Railway or AWS Elastic Beanstalk, interfacing with MongoDB.
- **Frontend:** Web application built using JSP pages and styled with Tailwind CSS.
- **AI Processing:** Python FastAPI service running TensorFlow (MT3 model) on a GPU-enabled AWS EC2 instance.

- **User Devices:** Modern desktop and tablet browsers; MIDI devices via USB/Bluetooth; microphones for audio input.

2.5 Design and Implementation Constraints

- **Performance:** Real-time transcription must maintain low latency.
- **Scalability:** The system architecture must support horizontal scaling, especially for AI processing and data storage.
- **Interoperability:** Must adhere to MIDI standards and REST API protocols.
- **Security:** Data must be transmitted securely, and user privacy maintained.
- **Deployment:** Separate servers for backend and AI services to optimize load distribution.

2.6 User Documentation

- **User Manuals:** Detailed guides for musicians and composers.
- **API Documentation:** Technical details for developers on REST endpoints and module interactions.
- **Tutorials and FAQs:** Online videos and help articles for troubleshooting common issues.

2.7 Assumptions and Dependencies

- The MT3 model operates on a dedicated GPU server to ensure efficient transcription.
- All modules communicate via secure, well-defined REST APIs.
- MIDI devices used by end users conform to standard protocols for compatibility.
- Deployment assumes stable internet connectivity and modern browser environments.

3. External Interface Requirements

3.1 User Interfaces Overview

MuseScribe's user interface provides an intuitive graphical user interface that will allow all users to interact with the system through a web browser. Major core functions that are exposed for users include the following:

- **File upload interface**
 - Users are able to drag and drop .wav and .mp3 files or can use the built-in file picker to upload locally stored recordings
 - Includes input validation for users: file type, size, and status indicators
- **Recording interface**
 - Buttons for recording, pause, resume, and stop
 - Displays to users real-time animated sound waveforms
- **Sheet Music Viewer**
 - Displays user's interactive sheet music based on returned ABC notation
 - Users can use playback controls to play, pause and loop and save
 - Allows for users to edit the notation in a text editor view
- **Saved Transcription Display**

- Retrieves a list of saved music sheets from the backend
 - Allows for the deletion of any individual entries
 - Includes metadata for each file created: title, creation date, last modified timestamp
- **Login Interface**
 - Dynamic login with Google button redirects users to Google log in screen
 - After login, the user is redirected to the main page.

3.2 Hardware Interfaces

- **MIDI Devices:**
 - Supports USB and Bluetooth connections via the Web MIDI API.
- **Audio Devices:**
 - Utilizes built-in or external microphones for audio capture.
- **Display:**
 - Optimized for high-resolution screens on desktops and tablets.
- **Speakers:**
 - Required for audio playback of the rendered music sheet

3.3 Software Interfaces

- **Spring Boot REST API:**
 - Manages user requests, authentication, and routing between subsystems.
- **MongoDB Database:**
 - Stores transcription records, user data, and system logs.
- **Python FastAPI (MT3 Service):**
 - Provides endpoints for processing audio/MIDI data and returning transcription results.
- **ABC.js:**
 - Renders MIDI data as interactive sheet music on the frontend.
- **Third-Party Libraries:**
 - Includes Tailwind CSS for styling and JSTL tag libraries for UI components.
- **OAuth2:**
 - Handles secure user authentication, retrieves the users data after successful login.

4. Design Considerations

- **Hardware:**
 - A GPU-enabled server is available for AI model processing.
- **Software:**
 - Dependencies include TensorFlow for AI processing, Spring Boot for backend, and MongoDB for storage.
- **User Environment:**
 - End users have access to modern web browsers and necessary audio/MIDI devices.
- **Network:**
 - Reliable, secure communication channels exist between system components.

4.1 General Constraints

- **Resource Management:**
 - Efficient use of memory, CPU, and network resources is critical for handling concurrent sessions.
- **Standards Compliance:**
 - Must follow MIDI, REST, and web security standards.
- **Maintainability:**
 - A modular design supports independent updates and scalability.
- **Error Handling:**
 - Comprehensive mechanisms are in place to log and recover from errors gracefully.
- **Latency:**
 - Real-time processing must meet strict performance criteria.
- **Resource Management:**
 - Efficient use of memory, CPU, and network resources is critical for handling concurrent sessions.
- **Standards Compliance:**
 - Must follow MIDI, REST, and web security standards.
- **Maintainability:**
 - A modular design supports independent updates and scalability.
- **Error Handling:**
 - Comprehensive mechanisms are in place to log and recover from errors gracefully.

4.2 Development Methods

- **Agile Methodology:**
 - The project is developed in sprint-based cycles for iterative progress and continuous feedback.
- **Microservices Architecture:**
 - Separates core backend logic (Java Spring Boot) from AI processing (Python FastAPI), ensuring modularity and independent scalability.
- **Component Reuse:**
 - Leveraging established libraries (ABC.js, MongoDB, Tailwind CSS) accelerates development and enhances system stability.
- **Code Reviews and Documentation:**
 - Regular reviews ensure adherence to design standards and thorough documentation of code changes.

5. System Architecture

5.1 Architectural Strategies

MuseScribe's design emphasizes modularity, scalability, and performance:

- **Backend (Java Spring Boot):**

- Implements a robust REST API for handling user requests, authentication, validation and business logic.
 - Uses dependency injection and middleware for error handling, logging, and security.
- **Data Storage (MongoDB):**
 - Provides flexible, schema-less storage to accommodate diverse transcription data.
 - Supports horizontal scaling as data volumes increase.
- **AI Processing (Python + FastAPI):**
 - Dedicated microservice hosting the MT3 model, optimized for GPU-accelerated inference.
 - Isolated to ensure that AI processing does not interfere with core API performance.
- **Trade-Offs:**
 - The microservices model adds communication complexity but improves maintainability and future-proofing.
 - Separation of concerns allows independent updates to each subsystem.

5.2 High level Overview of System Architecture

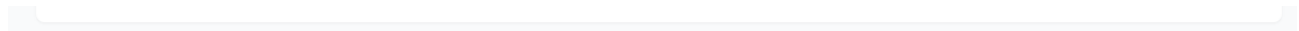
The system is divided into four primary subsystems:

- **Frontend:**
 - Developed using static HTML incorporating Tailwind CSS, JSTL Core, JSTL Formatting, and ABC.js libraries. This module provides the user interface for file uploads, MIDI connections, and interactive sheet music rendering.
- **Backend API:**
 - Built on Java Spring Boot, it orchestrates data flow between the frontend, AI service, and database.
- **AI Processing Service:**
 - A Python FastAPI microservice responsible for running the MT3 model to transcribe audio/MIDI input into MIDI data.
- **Data Storage:**
 - MongoDB stores transcription records, metadata, and user session information.

6. Human Interface Design

6.1 Screen Images

Dashboard Initial View



Music Sheet Edit View

Notebook Select View

Login Page



MuseScribe, Music Transcription
Made Easy.

Sign In

Email:

Password:

Login



Google Login

[Forgot Password?](#)

Don't have an account already?

[Sign up now!](#)

Registration

Registration Validation

6.2 Screen Objects and Actions

- The upload audio button allows users to upload their .mp3 or .wav files.
- The Drag and Drop detects file drop from user and will validate format.
- Record button allows users to begin recording using their microphone.
- The Pause/Stop buttons will pause or end a recording session.
- There is a Waveform animation that displays real-time audio recording
- ABC viewer shows the users an editable music sheet transcription and the ability to playback.
- Edit notation button allows users to edit their notation.
- Export button lets users download their music sheets in the standard music notation format.

7. Detailed System Design

7.1 Data Structures

This section identifies and describes the primary data structures utilized within the application, emphasizing their roles and importance in managing and organizing data efficiently:

User Data Structure

The application uses a structured JSON format to manage user-related information:

Example:

Purpose: Stores user identity and account information, along with references to associated music sheets.

Music Sheet Data Structure

Music sheets are represented and managed using structured JSON objects:

[REDACTED]

Purpose: Represents transcribed musical data linked to specific users, enabling efficient retrieval and storage.

Transcription Response Structure

Responses from the external MT3 transcription service adhere to a defined JSON structure:

[REDACTED]

Purpose: Clearly communicates transcription results or errors back to the frontend, facilitating proper UI handling and feedback.

These structured data types collectively ensure data integrity, enhance application performance, and streamline user interactions.

7.2 Frontend Application (JavaScript with jQuery and ABC.js)

7.2.1 Responsibilities

The frontend application provides an intuitive, responsive, and interactive user experience. It utilizes JavaScript, jQuery, and ABC.js to effectively manage user interactions, handle file uploads, visualize music transcriptions, and offer playback functionalities. Key responsibilities include:

- Facilitating audio file uploads through a user-friendly modal interface with enhanced drag-and-drop functionality.
- Directly uploading audio files to an external MT3 AI transcription service.
- Clearly indicating transcription progress with dynamic loading visuals.
- Rendering interactive music sheets using ABC.js, including advanced playback and editing options.
- Providing real-time visual feedback with animated sound waves during audio recording.
- Allowing users to save, retrieve, and review previously saved music sheets.

7.2.2 Uses and Interactions

User

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]

External MT3 Transcription Service

- Audio files uploaded from the frontend are directly sent to the external MT3 AI transcription service.
- Transcription results (ABC notation) are dynamically retrieved and displayed.

Backend API (Java Spring Boot)

- Utilizes AJAX requests (via jQuery) to securely store and retrieve music sheets and manage user-related data.

Available endpoints:

- DELETE /api/music-sheets/{sheetId}: Deletes a specified music sheet.
- POST /api/users: Creates a new user.
- GET /api/users/{id}: Retrieves details of a user.
- PUT /api/users/{id}: Updates user details.
- DELETE /api/users/{id}: Deletes a user.
- POST /api/notebooks/{id}: Create notebook.
- DELETE /api/notebooks/{notebookId}: Delete a notebook.
- GET /api/notebooks/{userId}: List notebooks for user.
- GET /api/notebooks/{notebookId}/sheets: List sheets in notebook.
- POST /api/music-sheets/{notebookId}: Create sheet in notebook.
- GET /api/music-sheets/{notebookId}: List sheets in notebook.
- PUT /api/music-sheets/{sheetId}: Updates sheet
- DELETE /api/music-sheets/{sheetId}: Deletes sheet

ABC.js

- Converts ABC notation into interactive, visually appealing sheet music.
- Supports playback functionality, real-time editing, and responsive design adjustments.
- Recorder.js
- Enables robust, browser-based audio recording.
- Provides real-time sound wave visualizations to enhance user feedback during recordings.
- Exports recorded audio in WAV format for immediate transcription.

7.2.3 Constraints

- Ensures compatibility with major modern browsers, including Chrome, Firefox, Edge, and Safari.
- Provides seamless responsiveness across multiple device types (desktop, tablet, and mobile).
- Handles asynchronous operations effectively to prevent delays or blocking.
- Incorporates stringent security measures to validate file types and sizes during uploads.

7.2.4 7.2.4 Composition

UI Components

- **Audio Upload Modal:**
 - Enhanced with intuitive drag-and-drop capabilities using jQuery.
 - Performs immediate validation of file types (.wav, .mp3) and provides instant feedback regarding success or errors.
 - Offers an intuitive and accessible interface for selecting and uploading files.
- **Recording Interface (Recorder.js):**
 - Facilitates audio recording using browser-supported audio contexts.
 - Displays real-time sound wave animations via canvas, significantly enhancing user interaction.
 - Supports pause, resume, and direct export of recorded audio for transcription purposes.

- **ABC.js Sheet Music Viewer:**
 - Dynamically generates interactive sheet music from ABC notation.
 - Includes playback controls (play, pause, loop) and editing capabilities.
 - Adjusts responsively to different screen sizes and device orientations.
- **Saved Sheets Display:**
 - Retrieves and displays previously stored music sheets via secure backend integration.

State Management and Event Handling

- Efficiently manages event binding, DOM manipulation, AJAX requests, and user interface interactions using jQuery.
- Controls dynamic UI states such as loading indicators, transcription statuses, and content updates.

7.2.5 Resources

Libraries and Frameworks

- jQuery for streamlined UI interactions, event handling, and AJAX communications.
- ABC.js for rendering and playback of interactive music notation.
- Recorder.js for efficient audio recording and visual representation via sound waves.

Dependencies

- External MT3 AI transcription service accessed through a secure ngrok endpoint.
- Java Spring Boot backend API for secure management of transcription data and user information.

7.2.6 Processing Workflow

File Upload and Transcription

- Users upload audio files through the improved drag-and-drop modal.
- Files are directly uploaded to the external MT3 transcription service using the fetch API.
- Visual indicators clearly communicate transcription progress until ABC notation results are received.

Audio Recording

- Users initiate and control audio recordings directly within the browser.
- Real-time sound wave visuals provide continuous user feedback during recordings.
- Recorded audio is exported as WAV files and immediately submitted for MT3 transcription.

Saving and Displaying Music Sheets

- Transcribed ABC notation can be manually saved through AJAX calls to the backend API (POST /api/music-sheets/{notebookId}).
- Users can securely retrieve and review previously saved music sheets (GET /api/notebooks/{notebookId}/sheets) and manage them via provided endpoints.

Error Handling

- Comprehensive error handling ensures users receive immediate and clear notifications of issues related to file uploads, input validation, and transcription errors, promoting smooth and reliable interactions.

7.3 AI Processing Service

This is a cloud-based transcription service that leverages Google Colab, FastAPI, and MT3 (Music Transcription Transformer) to convert uploaded audio files into MIDI and ABC notation.

7.3.1 Responsibilities

- Accepts MP3/WAV audio files and processes them using MT3.
- Provides a FastAPI endpoint for receiving and processing files.
- Converts transcribed MIDI files into ABC notation using EasyABC.
- Facilitates integration with the Spring Boot backend.
- Manages real-time transcription requests within Google Colab.
- Ensures secure, public API exposure via ngrok.
- Implements a frontend loading indicator during transcription.

7.3.2 Uses/Interactions



- **Frontend (JavaScript + ABC.js)**
 - Sends file upload requests to MT3 server.
 - Displays transcribed music notation.
 - Shows a loading animation while processing occurs.
- **Backend (Spring Boot + MongoDB)**
 - Handles API calls and forwards audio files to FastAPI.
 - Stores transcription data and notifies the frontend upon completion
- **Google Colab (FastAPI + MT3)**
 - Executes audio processing and model inference.
 - Converts output into MIDI JSON format.
 - Uses EasyABC to transform MIDI into ABC notation.
- **ngrok**
 - Exposes FastAPI to external services.
 - Provides temporary secure public API endpoints.

7.3.3 Constraints

- **GPU Requirement:** Requires Colab Pro for high-performance inference.
- **Session Duration:** Google Colab sessions may disconnect, requiring manual restarts.
- **Latency Considerations:**
 - Response times depend on file size and model complexity.
 - ngrok introduces minor network overhead.
- **File Format Restrictions:**
 - Only supports .mp3 and .wav files.
- **Concurrency Limitations:**
 - Only one request can be processed at a time per Colab instance.

7.3.4 Composition

- 1) **upload_audio(file: UploadFile) → JSON**
 - a) Receives an uploaded audio file via a FastAPI endpoint.

- b) Validates the file format to ensure it is MP3 or WAV.
 - c) Saves the uploaded file temporarily in Google Colab's file system.
 - d) Calls `transcribe_audio()` to process the file using the MT3 model.
 - e) Returns a structured JSON response containing both MIDI and ABC notation.
 - f) Handles errors such as unsupported file types and large file sizes.
- 2) `transcribe_audio(file_path: str) → Dict`**
- a) Loads the MT3 model from Google Magenta's repository.
 - b) Uses Librosa to preprocess the audio:
 - i) Loads the audio file as a waveform.
 - ii) Converts stereo to mono if necessary.
 - iii) Normalizes the amplitude to prevent clipping.
 - iv) Resamples the audio to 16 kHz, ensuring compatibility with MT3.
 - c) Runs inference using TensorFlow and JAX to extract note sequences from the waveform.
 - d) Filters out low-confidence predictions and adjusts timing alignment.
 - e) Calls `convert_midi_to_abc()` to transform the generated MIDI into ABC notation.
 - f) Returns a structured JSON response containing processed musical data.
 - g) Handles edge cases, such as excessive background noise, low-quality recordings, or silent input.
- 3) `convert_midi_to_abc(midi_path: str) → str`**
- a) Uses EasyABC to transform MIDI into ABC notation.
 - b) Extracts MIDI events, including note-on, note-off, pitch, velocity, and timing.
 - c) Maps MIDI pitch values to corresponding ABC note names based on octave and key signature.
 - d) Adjusts note duration based on MIDI tick values, ensuring rhythmic accuracy.
 - e) Formats the ABC notation output with metadata headers:
 - i) T: Title of the transcription.
 - ii) M: Time signature (default: 4/4 or detected from MIDI metadata).
 - iii) L: Default note length.
 - iv) K: Key signature inferred from MIDI events.
 - f) Returns a fully formatted ABC string, ready for rendering in ABC.js.
 - g) Handles complex MIDI sequences, including chords and multi-track data, ensuring accurate transcription.
- 4) `test_data() → JSON`**
- a) Returns a predefined example MIDI transcription converted into ABC notation.
 - b) Includes common melodic and rhythmic patterns for testing.
 - c) Used for debugging and system validation without requiring an actual audio file upload.
 - d) Allows frontend developers to integrate and test music visualization without relying on live transcriptions.

7.3.5 Resources

- **Google Colab GPU (NVIDIA T4 or better) for model execution.**
- **Python Libraries:**
 - TensorFlow, JAX, NumPy, Librosa (for MT3 inference and preprocessing).
 - FastAPI, Uvicorn (for API hosting and handling requests).
 - EasyABC (for converting MIDI to ABC notation).
 - ngrok (for exposing API endpoints to external services securely).
- **External Dependencies:**
 - Spring Boot backend (to manage and forward transcription requests).
 - Frontend TailwindCSS/ABC.js (for rendering and displaying transcriptions visually).
 - MongoDB (for storing and retrieving transcribed results).

7.3.6 Processing

1) Audio Upload

- a) The frontend submits an MP3/WAV file.
- b) The backend forwards the request to FastAPI on Colab.
- c) The loading animation is displayed on the UI.

2) MT3 Transcription

- a) The FastAPI service saves the file.
- b) The MT3 model processes the audio into MIDI format.
- c) The MIDI file is converted into ABC notation.

3) Response Handling

- a) FastAPI returns transcription results as JSON.
- b) The backend stores the result and notifies the frontend.
- c) The loading animation disappears.
- d) The frontend renders the music sheet using ABC.js.

4) Error Handling

- a) Invalid File Type → Returns 400 Bad Request.
- b) Processing Timeout → Returns 504 Gateway Timeout.
- c) Google Colab Disconnection → Requires manual restart.

7.4 Backend API (Java Spring Boot)

7.4.1 Responsibilities

The Java Spring Boot Backend API acts as a central management system responsible for handling user-related operations and managing transcribed music sheets. Its primary tasks include:




- Managing user data (creation, retrieval, updating, deletion).
- Storing and retrieving transcribed music sheets received from the frontend.
- Associating stored music sheets with corresponding user accounts.
- Providing secure RESTful API endpoints for interactions with frontend clients.
- Ensuring robust error handling, request validation, and data integrity.

The backend does not directly handle audio file uploads or audio transcription processing.

7.4.2 Uses/Interactions



Frontend Application:

- Receives requests to store, retrieve, or delete transcribed music sheets.
- Create Sequence 
- Retrieve Sequence 
- Delete Sequence 

- Manages user account operations, such as creating new accounts and fetching existing user details.

MongoDB:

- Performs persistent data storage and management for users and music sheets.

7.4.3 Constraints

- API responses must be efficient, ensuring minimal latency during database operations.
- Secure access must be enforced for all endpoints, with proper validation and sanitization of inputs to avoid security vulnerabilities.
- Scalable database interactions to handle concurrent requests effectively.

7.4.4 Composition**Controllers:**

- MusicSheetController (/api/music-sheets): Handles CRUD operations for music sheet data.
 - POST /api/music-sheets/{notebookId} - Stores a new music sheet associated with a user in notebook.
 - GET /api/music-sheets/{notebookId} - Retrieves all music sheets associated with a user that are found in notebook.
 - PUT /api/music-sheets/{sheetId} - Update music sheets.
 - DELETE /api/music-sheets/{sheetId} - Deletes a specified music sheet.
- UserController (/api/users): Handles CRUD operations for user data.
 - POST /api/users - Creates a new user.
 - GET /api/users/{id} - Retrieves details of a specific user.
 - PUT /api/users/{id} - Updates information of a specified user.
 - DELETE /api/users/{id} - Deletes a specified user.
- NotebookController (/api/notebooks): Handles CRUD operations for notebooks.
 - POST /api/notebooks/{userId} - Create notebooks.
 - GET /api/notebooks/{userId} - List user notebooks.
 - GET /api/notebooks/{notebookId}/sheets - Lists sheets inside of notebook.
 - DELETE /api/notebooks/{notebookId} - Delete a notebook.

Service Layer:

- MusicSheetService: Implements business logic to handle music sheet-related operations, including associating sheets with users.
- UserService: Implements logic for managing user data and interactions.

Repositories:

- MusicSheetRepository: Manages direct database interactions for music sheet data.
- UserRepository: Manages direct database interactions for user data.

7.4.5 Resources**Frameworks**

- Spring Boot for REST API development.
- Spring Data MongoDB for effective database management.

Database

- MongoDB for persistent storage of user accounts and music sheet data.

7.4.6 Processing**Storing Transcription Data:**

- Users manually initiate the save operation from the frontend after successful audio transcription by the external MT3 AI service.
- Transcribed ABC notation, along with associated metadata (such as user ID and title), is sent via the frontend to the backend API endpoint (POST /api/music-sheets/{notebookId}).
- The backend stores the data in MongoDB and associates it with the user account.

User Data Operations:

- Backend provides endpoints to perform standard CRUD operations securely and efficiently.

Error Handling and Logging:

- Implements structured exception handling to effectively manage and log database errors, invalid inputs, and network-related issues, ensuring the reliability and robustness of backend operations.

Appendix A: Glossary

- API (Application Programming Interface): A set of routines and protocols enabling communication between software components.
- MIDI (Musical Instrument Digital Interface): A standard protocol for transmitting musical performance data.
- MT3 Model: An AI model developed for converting audio inputs into MIDI data.
- REST (Representational State Transfer): An architectural style for designing networked applications using HTTP methods.
- JSON (JavaScript Object Notation): A lightweight data-interchange format.
- AWS (Amazon Web Services): Cloud computing services used for hosting and deployment.
- GPU (Graphics Processing Unit): A specialized processor used for accelerated computing tasks.
- ABC.js: A JavaScript library for rendering music notation in web browsers.
- Spring Boot: A Java-based framework for building robust RESTful web applications.
- Fast API: A Python micro-framework used for developing lightweight web applications.
- JSTL: Java Server Pages Standard Tag Library used in JSP-based UI rendering.
- Tailwind CSS: CSS framework used to design responsive UI
- Ngrok: Secure tunneling service to expose local servers to the public internet.