Software Requirement Specification

for

MuseScribe

Version 1.0

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

MDC Group 5

02/04/25

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This document specifies the software requirements for MuseScribe, a Java-based music transcription application. MuseScribe converts recorded instrument performances into sheet music by integrating AI-based audio-to-MIDI transcription, MIDI input processing, to a canvas render using ABC Music Notation. This specification will serve as a guideline for developers, testers, and evaluators, ensuring the product meets the desired functionalities and maintain standards and quality.

1.2 Document Conventions

This document follows the IEEE SRS template conventions. 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: For understanding architectural decisions, APIs, and component interactions.
- Musicians & Testers: For insight into user functionalities and performance expectations.
- Professors & Evaluators: For reviewing the system's architecture, design rationale, and compliance with requirements.

1.4 Product Scope

1.4.1 Purpose

MuseScribe is designed to automate the transcription of musical performances into sheet music. By handling both pre-recorded audio files and real-time MIDI inputs, it provides:

- Support multiple instruments (Guitar, Flute, Piano).
- Provide sheet music in Front-End leveraging ABC.js library to display sheet music notes from midi data.
- Enable uploaded or recorded MIDI transcription for any users.

1.4.2 High Level Flow

- The user first logs into the Musescribe web app user Manually Created or Google SSO
- The user uploads MP3/WAV/MIDI file or connects to recording device such as professional microphone or MIDI device.
- Front-End send recorded audio/midi data via HTTPS to MT3 server.
- The audio is transcribed into ABC Notation and returned to the client in JSON format
- The ABC Notation is rendered in Music Notation on the using html canvas element

- The user can then modify transcription details (Tempo, Note Length, Key, etc.) as well as save the music to MongoDB and an object with collection id, title, other information relating to music sheet is returned
- Alert message displays provide indication of Successful or Error response.

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)
- IEEE SRS Standard: (https://ieeexplore.ieee.org/document/278253)

2. Overall Description

2.1 Product Perspective

MuseScribe is a cloud-hosted web application that converts raw audio or MIDI into clean, downloadable sheet music. Although it presents a single, seamless interface to the user, the product is built from three loosely coupled services:

- Spring Boot web layer that manages sign-in, uploads, and the user's personal library.
- GPU-powered AI service that performs MT3 transcription.
- MongoDB cluster that stores accounts, notebooks, and scores.

Users can upload, transcribe, view, edit, and download their scores directly from the browser. No external software or plug-ins required, while the system itself remains fully self-contained and independent of any third-party platforms for sharing.

2.2 Product Functions

User & Access
- Google OAuth Sign-In / Sign-Out – one-click login
- Profile & Session Management – create, retrieve, and terminate sessions; handle basic profile edits.

Capture Layer
- Audio Upload – accept WAV/MP3/MIDI and delete audio files when transcription complete.
- Live MIDI Capture – accept real-time or file-based MIDI input.

Core Processing
- AI Transcription (MT3) – convert audio to note events, return ABC + MIDI.
- Direct MIDI → ABC Conversion – bypass AI when source is already MIDI.

Notation & Playback
- Sheet Rendering (ABC.js) – display notation; note-select, note re-positioning.
- Interactive Playback – start/stop, scrub, adjust tempo.

Organization & Search
- Notebooks – create, view, rename, and delete collections of sheets.

Export & Sharing
- Download MIDI data.

2.3 User Classes and Characteristics

- **Musicians & Composers:** Require a precise, intuitive interface for quick transcription and editing to support performance and composition.
- **Students & Educators:** Use the application as a learning tool to explore music theory and transcription techniques.
- **General Enthusiasts:** Users with basic technical skills who seek to generate and share sheet music from recordings.

2.4 Operating Environment

- **Backend Environment:**
  - Java Spring Boot application running on Railway or AWS Elastic Beanstalk
  - MongoDB is run by a cloud provider and acts as the primary data store.
- **Frontend Environment:**
  - Web application built using JSP and styled with Tailwind CSS.
- **AI Processing Environment:**
  - Python-based service using TensorFlow and the MT3 model, hosted on an AWS EC2 g4dn.xlarge instance.
- **User Environment:**
  - Modern web browsers on desktops, laptops, and tablets.
  - MIDI devices connected via USB or Bluetooth; microphones for audio input.

2.5 Design and Implementation Constraints

The system must adhere to a microservices model: The Java Spring Boot, the MT3 AI transcription service, and the MongoDB service are isolated deployable that communicate only over HTTP/JSON, with no shared runtimes, schemas, or session state. Upload-to-ABC latency should not exceed 2 minutes excluding the AI round-trip, and any synchronous cross-service transactions are forbidden. Authentication flows strictly through Google OAuth2, the AI server never touches the database directly, and all uploaded audio/MIDI files are deleted after ABC Notation data is returned to the client. High performing GPU service is a must as CPU fallbacks are out of scope, and anything breaching these constraints stalls the release pipeline until resolved.

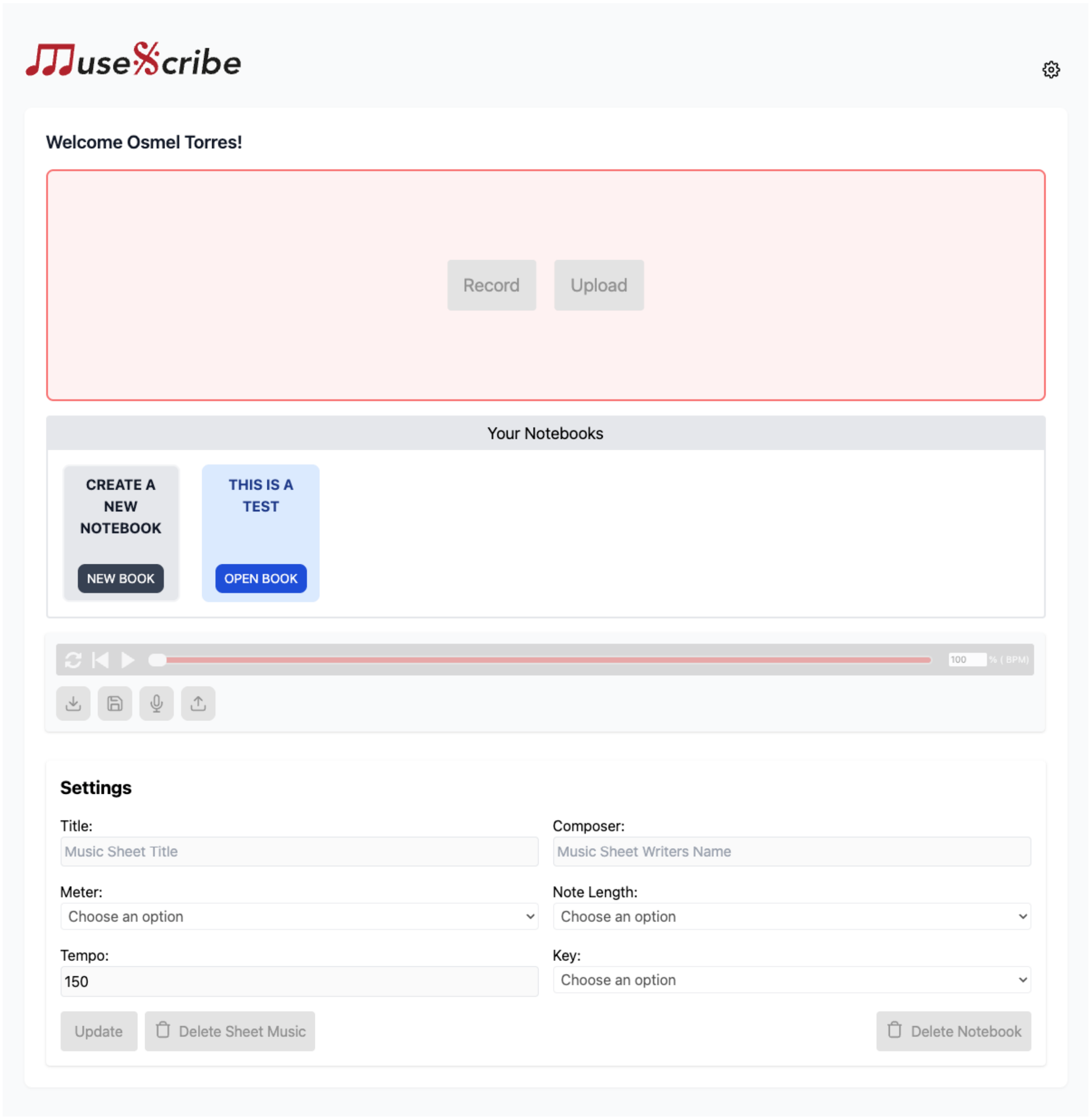
2.6 Assumptions and Dependencies

MuseScribe assumes users have consistent access to internet connection, mouse, keyboard to interact with the application as well as audio capture devices such professional microphones or midi capture devices.

3. External Interface Requirements

3.1 User Interfaces Overview

- **Dashboard**



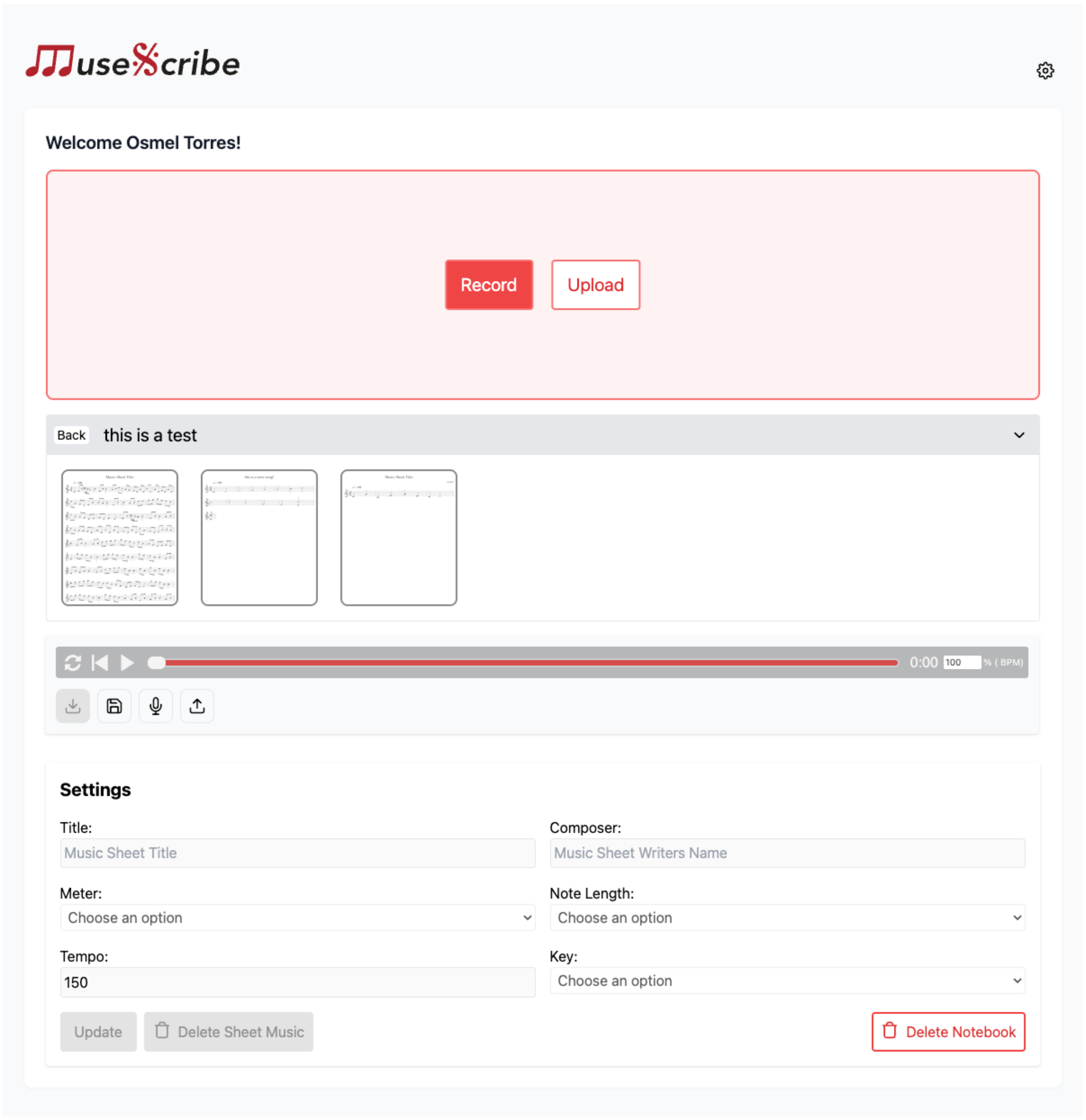
Welcome Osmel Torres!

Settings

Title: Music Sheet Title
Composer: Music Sheet Writers Name
Meter: Choose an option
Note Length: Choose an option
Tempo: 150
Key: Choose an option

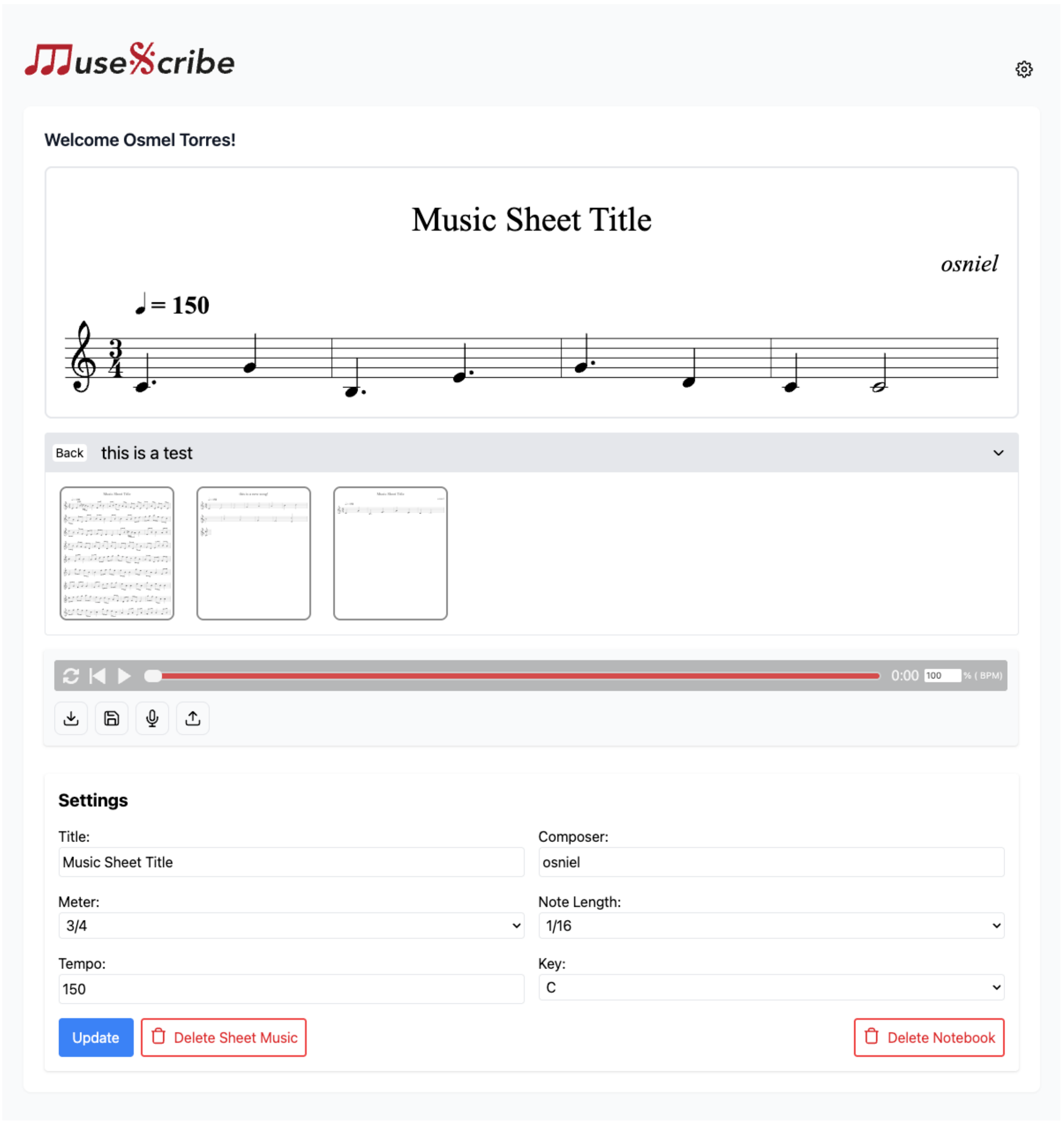
- In this view user can Create a new notebook or select an existing one

- **Pre-Select (Upload/Record Music to create new music sheet)**



In this view, the user can choose to record a new music sheet, upload a pre-recorded file to create a new one, or select an existing one.

- **Edit View**



Welcome Osmel Torres!

Music Sheet Title

osniel

Settings

Title: Music Sheet Title	Composer: osniel
Meter: 3/4	Note Length: 1/16
Tempo: 150	Key: C

Update | Delete Sheet Music | Delete Notebook

In this view user can edit the selected or new music sheet. They can update information such as Title, Composer, Tempo, etc. As well not position and playback music notation.

3.2 Hardware Interfaces

- **MIDI Devices**
    - Supports USB and Bluetooth MIDI devices via the Web MIDI API.
- **Audio Input**

    - Utilizes built-in or external microphones for recording audio. However Professional studio microphones are preferred.
- **Display**
    - Optimized for standard desktop and tablet screens.
- **Internet**
    - Stable internet connection with standard bandwidth (average 25mbps)

3.3 Software Interfaces

- **Spring Boot REST API:**
    - Acts as the intermediary between the frontend and other backend services.
    - Handles authentication, session management, and routing of transcription requests.

- **MongoDB Database:**
    - Stores Notebooks, Music Sheet Transcriptions, and User data.
- **MT3 AI Model Server (Fast API):**
    - Processes audio files and MIDI data conversion, returning structured JSON data.
    - Uses Fast API a modern, high-performance, framework for building APIs with Python based on standard Python type hints.

- **ABC.js:**
    - A JavaScript library used on the frontend to render MIDI data as sheet music.

- **Third-Party Services:**
    - Cloud hosting services (AWS) and libraries:
        - Data Visualization:
            - ABC.js - Visually render ABC notation data on the web app canvas
        - Helper:
            - Recorder.js - A plugin for recording/exporting the output of Web Audio API nodes

4. Design Considerations

4.1 Assumptions and Dependencies

- Average User with general knowledge in Music and Audio Engineering
- The MT3 AI model requires a GPU-enabled server for optimal performance.
- Frontend and backend systems operate on different servers, communicating over secure REST APIs.
- MIDI devices must conform to standard protocols for consistent performance.
- User has access to stable internet connect
- Users have reliable recording hardware such as Microphone and Midi capture devices.

4.2 General Constraints

- **Latency:** Live MIDI input and AI processing must operate with minimal delay.
- **Scalability:** The system must support an increasing number of users and data without performance degradation.
- **Security:** Data in transit between subsystems is encrypted; access to user data is strictly controlled.

4.3 Development Methods

- **Agile Methodology:**
    - The project follows a sprint-based development cycle, promoting rapid prototyping, regular feedback, and continuous integration.
- **Microservices Architecture:**
    - Separates the core business logic (Java Spring Boot) from AI processing (Python Fast API) and MongoDB storing music and user data to allow for independent development and scaling.
- **Component Reuse:**
    - Leverages proven libraries and frameworks (ABC.js, Spring Boot, MongoDB) to accelerate development and ensure system robustness.
- **Iterative Testing and Integration:**
    - Continuous testing, including unit, integration, and user acceptance testing, is embedded throughout the development lifecycle.
- **Documentation and Code Reviews:**
    - Regular documentation updates and code reviews ensure quality and maintainability.

5. System Architecture

5.1 Architectural Strategies

MuseScribe is designed with modularity, scalability, and performance in mind:

- **Backend Services (Java Spring Boot):**
    - Provides a robust REST API for handling user requests, session management, and data processing.
    - Employs dependency injection and middleware for logging, security, and error handling.
- **Data Storage (MongoDB):**
    - Offers a schema-less design that adapts to the diverse and evolving nature of transcription data.
    - Supports horizontal scaling to accommodate increasing data volumes.
- **AI Processing (Python Fast API):**
    - Isolated as a microservice to host the MT3 model, ensuring that AI processing does not impact core API performance.
    - Uses TensorFlow and optimized Python libraries for rapid inference.
- **Trade-offs:**

- The microservices approach introduces additional complexity between interface service communication but offers enhanced scalability and maintainability.
- Separating AI processing ensures that future model upgrades or changes can be implemented without overhauling the entire system.

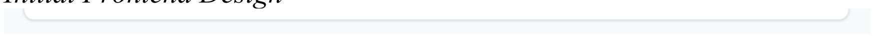
5.2 High level Overview of System Architecture

The MuseScribe system is divided into the following major subsystems:

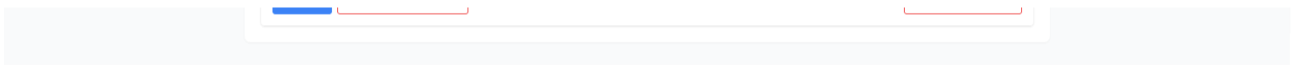
- Frontend (JSP & Tailwind CSS):
  - Provides an interactive user interface for uploading files, initiating live MIDI sessions, and viewing/editing transcriptions.
  - Communicates with backend services via REST API calls.
- Backend API (Java Spring Boot):
  - Acts as the central coordinator, managing business logic, session control, and data routing between the frontend, AI processing service, and MongoDB.
  - Implements secure REST endpoints for user authentication and transcription management.
- AI Processing (Python Flask):
  - Dedicated to running the MT3 model, converting audio to MIDI data.Operates independently to allow isolated scaling and updates.
- Data Storage (MongoDB):
  - Stores transcription data, user details, and metadata.
  - Provides flexible, scalable storage that adapts to evolving data requirements.

6. Human Interface Design

Initial Frontend Design



6.1 Screen Images



6.2 Screen Objects and Actions

- **Audio Control Bar**
  - **Contains controls to Loop, Reset, Play, Timestamp, Tempo**
- **Download Button**
  - Downloads Music Notation in Midi format
- **Save Button**
  - Save Current music sheet with settings details
- **Record Button**
  - Displays Option to select between Midi or Microphone Device
- **Upload Button**

  - Updates Music Sheet with updated settings fields
- Delete Sheet Music Button:
  - Will prompt the user to confirm they would like to permanently delete the selected sheet music
- Delete Notebook
  - Will prompt the user to confirm they would like to permanently delete the notebook containing all music sheets.
- **Settings**
  - Displays fields such as Title, Composer, Meter, etc. To adjust music sheet settings/properties.

7. Detailed System Design

7.1 Data Structures

Key data structures include:

- **MIDI Data Objects:**
  - JSON structures representing note sequences, timestamps, and instrument details.
- **Transcription Records:**
  - MongoDB documents that store metadata (e.g. timestamp, instrument type) and the corresponding MIDI data.
- **ArrayList:**
  - Java ArrayList is a part of the collection's framework, it provides us with dynamic-sized arrays in Java.

7.2 Component 1: Backend API (Java Spring Boot)

7.2.1 Responsibilities

- Handle all incoming API requests from the frontend.
- Validate input data and manage user authentication.
- Coordinate requests between the Front-End, AI processing service, and MongoDB.

7.2.2 Uses/Interactions

- Interacts with the MongoDB database for storing and retrieving transcription and user data.
- Communicates with the Python Flask API to process audio and MIDI conversions.

7.2.3 Constraints

- Must maintain low latency to support real-time transcription scenarios.
- Ensure thread safety and efficient resource allocation during high request volumes.

7.2.4 Composition

- **Controllers:**
  Manage HTTP requests and responses.

- **Service Layer:**
  Contains business logic and coordination between components.
- **Data Access Objects (DAOs):**
  Handle interactions with MongoDB.

7.2.5 Resources

- Utilizes Java libraries and frameworks (e.g., Spring Boot) to manage memory and processing threads.
- Interfaces with external services (e.g., Python Flask API) using HTTP clients.

7.2.6 Processing

- Incoming requests are authenticated, validated, and processed using asynchronous service calls.
- Error handling routines capture and log exceptions, providing meaningful feedback to users.

7.3 Component 2: AI Processing Service (Fast API)

7.3.1 Responsibilities

- Process audio files using the MT3 model to generate MIDI data.
- Provide a RESTful API endpoint for the backend to trigger transcription processes.

7.3.2 Uses/Interactions

- Receives audio data from the backend, processes it using TensorFlow and MT3, and returns MIDI results.
- Runs as an independent microservice, allowing for scalable updates independent of the backend.

7.3.3 Constraints

- Requires GPU resources for efficient processing.
- Must ensure timely response to maintain overall system performance.

7.3.4 Composition

- **API Endpoints:**
  Expose functionality for audio-to-MIDI conversion.
- **Processing Pipeline:**
  Includes preprocessing, model inference, and postprocessing of transcription data.

7.3.5 Resources

- Managed on a dedicated AWS EC2 instance with GPU capabilities.
- Uses Python libraries such as TensorFlow and Flask.

7.3.6 Processing

- Audio input is preprocessed such as normalization before being passed to the MT3 model.
- The model's output is formatted as MIDI data within and sent back to the backend via JSON.

7.4 Component 3: Frontend (Tailwind with ABC.js)

7.4.1 Responsibilities

- Provide an interactive user interface for uploading audio files and connecting MIDI devices.
- Render sheet music using ABC.js based on MIDI data retrieved from the backend.

7.4.2 Uses/Interactions

- Sends REST API calls to the backend for transcription requests.
- Retrieves and displays transcription data in an editable, visual format.

7.4.3 Constraints

- Must be compatible with modern web browsers and support responsive design.
- Ensure minimal latency in rendering updates during user interactions.

7.4.4 Composition

- **Components:**
  User input forms, file upload modules, and a dynamic sheet music viewer.

7.4.5 Resources

- Uses external libraries (ABC.js for notation, Tailwind CSS for styling) to enhance the user experience.
- Interfaces with the backend via secure API endpoints.

7.4.6 Processing

- User inputs trigger asynchronous API calls.
- Retrieved MIDI data is processed and rendered into sheet music notation in real time.

7.5 Appendix A: Glossary

- **API (Application Programming Interface):** A set of routines, protocols, and tools for building software applications.
- **MIDI (Musical Instrument Digital Interface):** A technical standard that describes a protocol, digital interface, and connectors for communication between musical instruments, computers, and other devices.
- **MT3 Model:** An AI-based model used for converting audio into MIDI data.
- **REST (Representational State Transfer):** An architectural style for designing networked applications.
- **JSON (JavaScript Object Notation):** A lightweight data-interchange format used for data exchange.
- **AWS (Amazon Web Services):** Cloud services used for hosting and deployment.
- **GPU (Graphics Processing Unit):** A specialized processor designed to accelerate graphics rendering and compute-intensive tasks.