Application for Transcribing Grand Piano Recordings to Sheet Music in PDF

Michał Jagoda

May 28, 2024

Contents

1 Topic Analysis

1.1 Overview

This project involves creating a desktop application designed to transcribe grand piano recordings into sheet music and export the results as a PDF. This involves the use of audio processing, music transcription, and PDF generation technologies, all performed locally without the need for an internet connection.

1.2 Objectives

- To accurately transcribe piano recordings into readable sheet music.
- To provide a user-friendly interface for uploading audio files and exporting sheet music.
- To offer an offline and local application for accessibility and privacy reasons.

1.3 Key Components

- **Audio Processing:** Capturing and processing the audio signals from the grand piano recordings using the `audio_processor.py`.
- **AI Model:** Generating an initial transcription output from the uploaded MP3 file using the `transcription_worker.py`.
- **Postprocessing Algorithm:** Creating a MIDI file from the AI model's output using the `Sheet_music_generation.py`.
- **PDF Generation:** Using the MuseScore API to generate a PDF from the generated MIDI file with `generate_pdf.py`.
- **User Interface:** Allowing users to interact with the application, upload audio files, view PDF previews, listen to MIDI and MP3 files, and download the resulting sheet music and MIDI files via `window.py`.

2 Technologies Used

- **Python:** The primary programming language used for developing the application.
- **Librosa:** A Python library for audio and music analysis, used for audio processing and feature extraction in `audio_processor.py`.
- **Piano Transcription Inference:** A library for transcribing piano music, used in `transcription_worker.py`.
- **Music21:** A toolkit for computer-aided musicology, used in `Sheet_music_generation.py` to create and manipulate music streams.
- **MuseScore:** A software for creating, playing, and printing sheet music, used in `generate_pdf.py` to convert MIDI files to PDF.

- **PySide6:** A set of Python bindings for Qt libraries, used for creating the graphical user interface in `window.py`.
- **VLC:** A media player used for playing audio and MIDI files within the application.
- **Pdf2image:** A library for converting PDF files to images, used in the GUI to display PDF previews.
- **Tempfile:** A Python module used for creating temporary files, ensuring that the application handles file operations efficiently and securely.
- **WSL2 (Windows Subsystem for Linux):** Used to run some Python libraries that were not available on Windows. As WSL2 lacks support for audio and GUI components, server bridges were implemented to facilitate communication between the audio processing and GUI components running in different environments.

3 Functional Requirements

3.1 Core Functionalities

- **Audio File Upload:**
  - Users can upload MP3 audio files.
- **AI Model:**
  - The system processes the uploaded audio using an AI model to generate an initial transcription.
- **Postprocessing Algorithm:**
  - The application converts the AI model's output into a MIDI file.
- **PDF Generation:**
  - The MuseScore API is used to convert the generated MIDI file into a PDF document.
- **PDF Preview and Download:**
  - Users can preview the PDF of the sheet music.
  - Users can download the PDF and the MIDI file.
- **Playback Feature:**
  - Users can listen to the transcription by playing the MIDI file.
  - Users can also listen to the original MP3 file.

4 Non-Functional Requirements

4.1 Performance Requirements

- **Speed:** The transcription process should be completed within a reasonable time frame (under 15 minutes for a 5-minute recording).
- **Accuracy:** The transcription should accurately reflect the notes, rhythms, and dynamics of the original recording with a high degree of precision (over 90% accuracy).

4.2 Usability Requirements

- **User Interface:** The interface should be intuitive and easy to navigate, even for users with limited technical skills.

4.3 Compatibility Requirements

- **File Format Support:** The application should support common audio file formats and produce PDFs that are compatible with standard PDF readers.

4.4 Maintainability Requirements

- **Modularity:** The application code should be modular to facilitate easy updates and maintenance.
- **Documentation:** Comprehensive documentation should be provided for developers to understand the codebase and contribute to its development.

4.5 Scalability Requirements

- **Future Expansion:** The architecture should allow for easy addition of new features and functionalities.

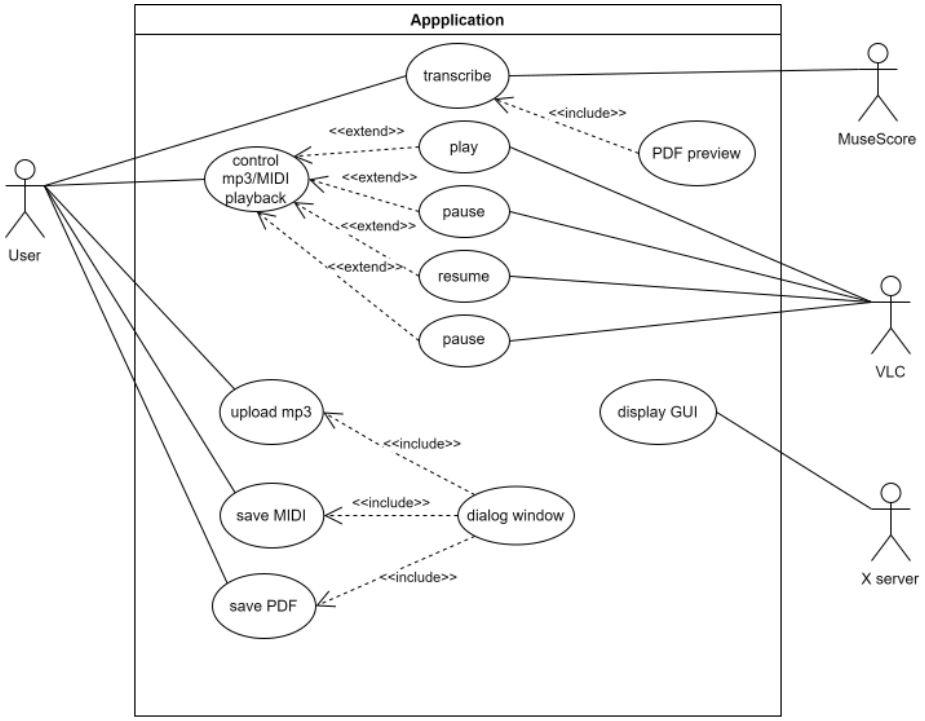

Figure 1: Use Case Diagram

Figure 2: System Architecture Flow Chart

5 Use Case Diagram

6 System Architecture Flow Chart

7 Class Diagram

8 Sequence Diagrams

9 AI Model Architecture

10 Comparison of Original and Generated Sheet Music

11 Conclusion

By addressing both the functional and non-functional requirements outlined above, this application can provide a robust, user-friendly solution for transcribing grand piano recordings into sheet music.

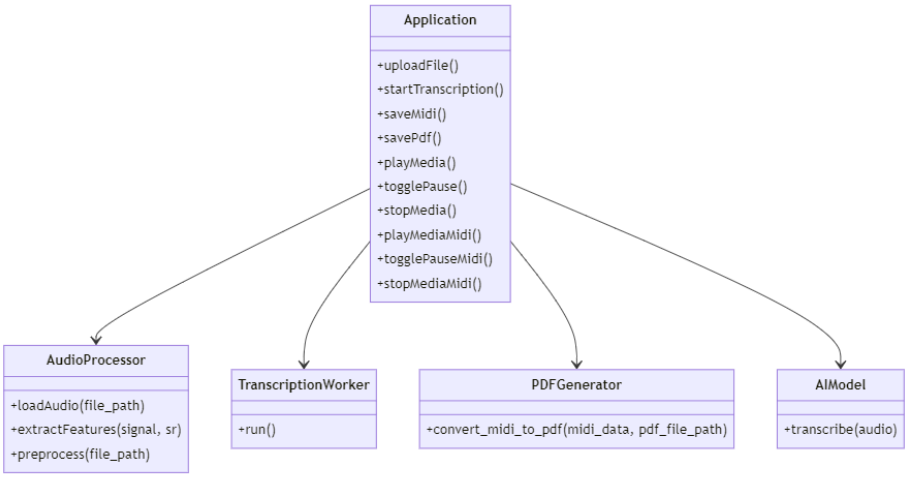

Figure 3: Class Diagram

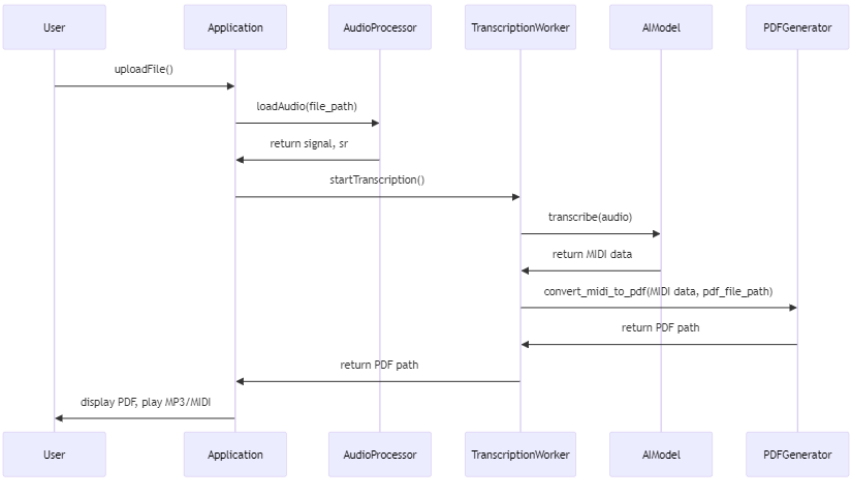

Figure 4: Transcription process sequence diagram

This will not only meet the immediate needs of users but also ensure the application's longevity and adaptability to future demands.

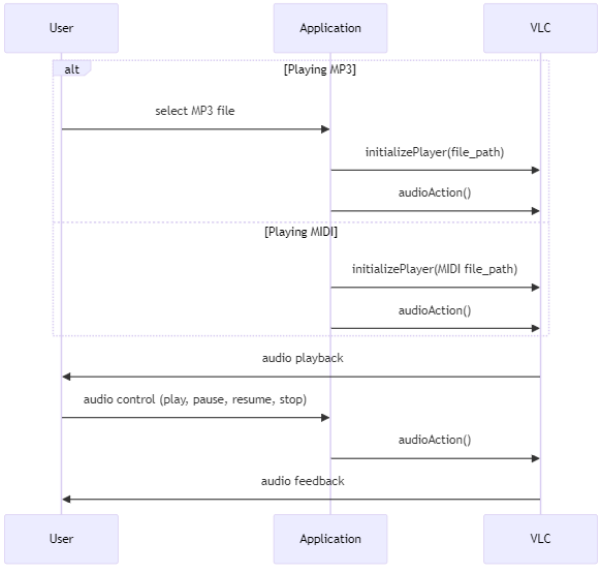

Figure 5: Audio sequence diagram

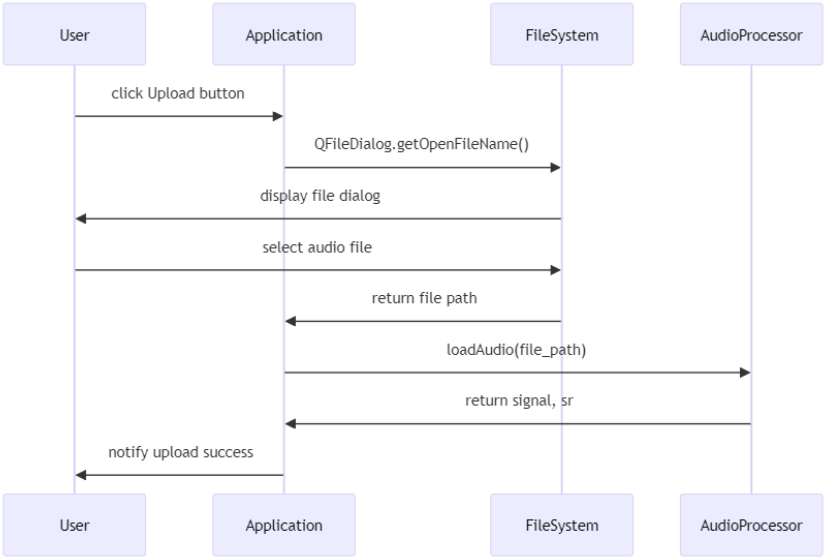

Figure 6: Load file sequence diagram

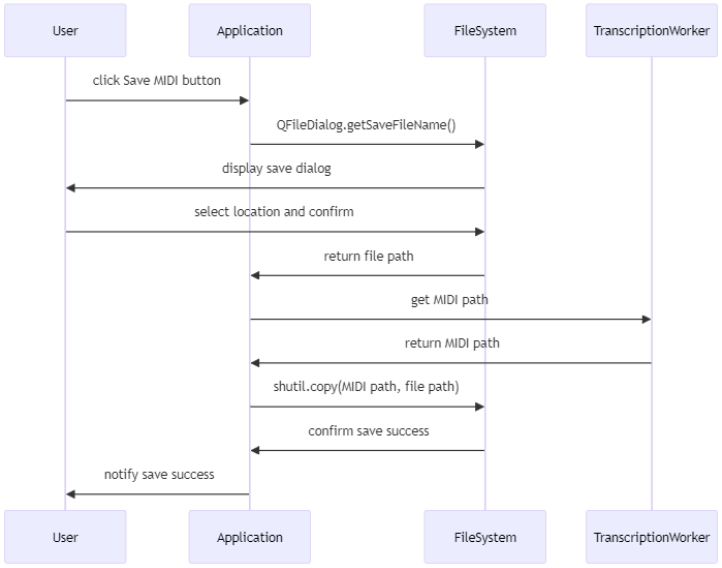

Figure 7: Save MIDI sequence diagram

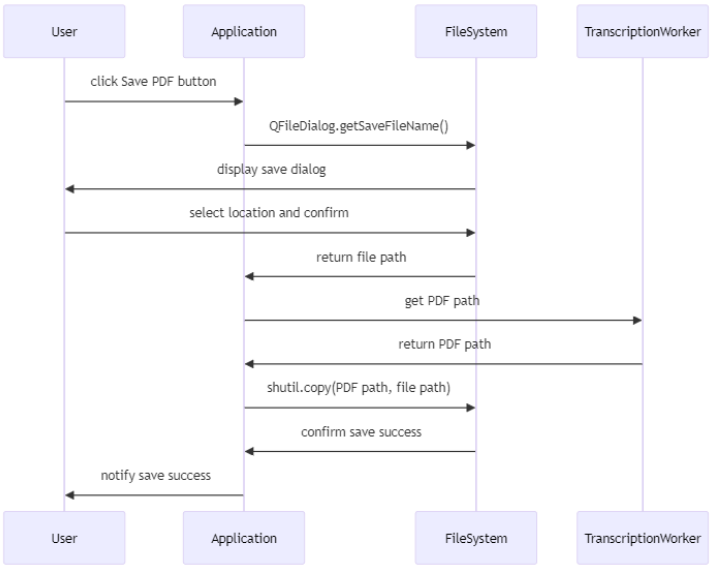

Figure 8: Save PDF sequence diagram

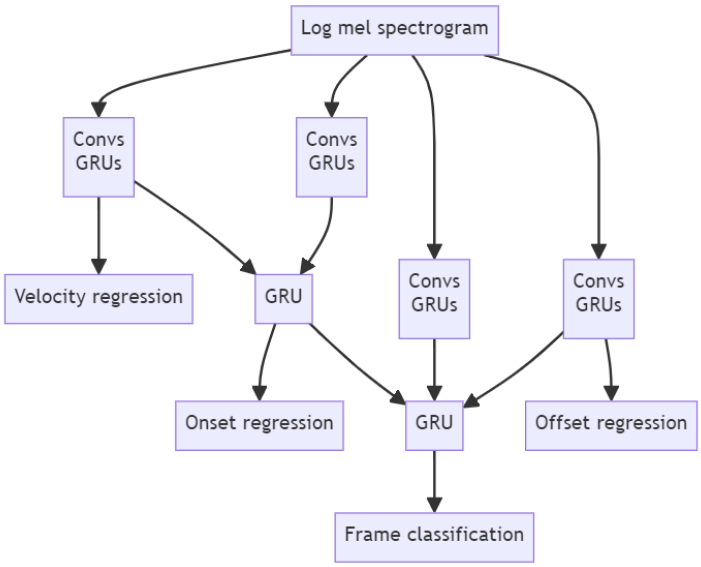

Figure 9: AI Model Architecture

Figure 10: Comparison of Original and Generated Sheet Music