

Aplikacja webowa wspomagająca naukę gry na pianinie

(Web application supporting
piano learning)

Adrian Walczak

Praca inżynierska

Promotor: dr Leszek Grocholski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

27 sierpnia 2025

Streszczenie

Celem pracy było zaprojektowanie, implementacja oraz walidacja systemu wspomagającego naukę gry na pianinie. Główną funkcjonalnością aplikacji jest transkrypcja dowolnych utworów muzycznych do formatu MIDI z wykorzystaniem metod sztucznej inteligencji (m.in. **Transkun**[2] oraz **Basic Pitch** [1]). Użytkownik, posiadając jedynie plik audio lub link do nagrania dostępnego online (np. w serwisie *YouTube*) utworu muzycznego, może wygenerować animację przedstawiającą klawiaturę pianina podczas gry, zmienić tonację, a także wyeksportować go w formatach MIDI, **MusicXML** lub jako zapis nutowy w formacie PDF. Aplikacja została zrealizowana w architekturze klient–serwer: backend w języku Python z wykorzystaniem frameworka **Flask** [4], a frontend w HTML, CSS i JavaScript. W projekcie wykorzystano m.in. biblioteki **synthviz** [9], **music21** [6] oraz **partitura** [7].

The aim of this thesis was to design, implement, and validate a system supporting piano learning. The main functionality of the application is the transcription of arbitrary musical pieces into the MIDI format using artificial intelligence methods (including **Transkun** [2] and **Basic Pitch**[1]). A user, having only an audio file or a link to an online recording (e.g., from *YouTube*), can generate an animation visualizing a piano keyboard during playback, change the key, and export the piece in MIDI, **MusicXML**, or as sheet music in PDF format. The application was developed in a client–server architecture: the backend is implemented in Python with the **Flask** framework [4], while the frontend is built with HTML/CSS/JavaScript. The project makes use of several libraries, including **synthviz**[9], **music21**[6], and **partitura**[7].

Spis treści

1. Wprowadzenie	7
1.1. Cel i zakres pracy	7
1.2. Wymagania funkcjonalne	8
1.3. Wymagania нефunkcjonalne	9
1.4. Przegląd literatury i istniejących rozwiązań	10
2. Implementacja	11
2.1. Strategia rozwoju projektu	11
2.2. Przegląd kluczowych technologii i narzędzi	12
2.3. Dane	16
2.4. Frontend	17
2.5. Funkcjonalności	18
2.5.1. Dodawanie nowego utworu i jego transkrypcja	18
2.5.2. Generowanie wideo treningowego	20
2.5.3. Galeria utworów, zmiana tonacji oraz generowanie partytury	23
2.5.4. Rozpoznawanie nut z mikrofonu w czasie rzeczywistym	25
2.5.5. Prosta gra 3D do nauki nut	26
3. Podręcznik użytkownika	29
3.1. Funkcjonalności	29
3.1.1. Dodawanie nowego utworu i jego transkrypcja	29
3.1.2. Generowanie wideo treningowego	30
3.1.3. Galeria utworów, zmiana tonacji oraz generowanie partytury	31

3.1.4. Rozpoznawanie nut z mikrofonu w czasie rzeczywistym	31
3.1.5. Prosta gra 3D do nauki nut	33
3.2. Sposób uruchomienia	34
3.2.1. Sposób 1: ręczna instalacja i konfiguracja środowiska	35
3.2.2. Sposób 2: Docker	36
4. Podsumowanie	37
4.1. Konkluzja	37
4.2. Dalszy kierunek rozwoju projektu	37
Bibliografia	39

Rozdział 1.

Wprowadzenie

Podczas nauki gry na pianinie często pojawia się problem ograniczonej dostępności zapisów nutowych do mniej popularnych utworów. Dotyczy to zwłaszcza utworów z niszowych gier i filmów, jednocześnie łatwo dostępnych w postaci nagrań wideo lub audio. Dla osoby początkującej, bardzo trudne jest odczytanie nut całego utworu na podstawie dźwięku. Bez pomocy z zewnątrz nie ma zatem możliwości nauki gry takiego utworu.

1.1. Cel i zakres pracy

Przedstawiony wyżej problem zainicjował pomysł na zbudowanie narzędzia opisanego w tej pracy, którego podstawowym celem jest automatyczna transkrypcja, eksport zapisu nutowego oraz wizualizacja gry na pianinie dla dowolnego utworu muzycznego na podstawie jego nagrania audio.

System, korzystając z metod sztucznej inteligencji, analizuje przesłane nagrania i generuje odpowiadające im pliki w formacie MIDI. Dla przesłanych utworów możliwe jest również wygenerowanie animacji przedstawiających klawiaturę pianina podczas gry danego utworu. Tego typu wizualizacje cieszą się dużą popularnością wśród początkujących, ponieważ pozwalają uczyć się utworów ze słuchu i wzroku, bez znajomości zapisu nutowego.

Aplikacja oferuje też możliwość eksportu utworów jako tradycyjny zapis nutowy w formacie PDF, co pozwala na naukę w klasyczny sposób. Możliwy jest również eksport w formatach MusicXML lub MIDI, które są obsługiwane przez większość narzędzi i programów muzycznych, dzięki czemu użytkownik ma możliwość dalszej pracy z utworem - jego edycję, czy wykorzystanie w innym narzędziu wspierającym naukę gry.

Kod źródłowy aplikacji jest dostępny publicznie w repozytorium GitHub: <https://github.com/et1141/piano-audio-to-midi>.

Wersja demonstracyjna online znajduje się pod adresem: <https://et1141>.

github.io/Piano-Learning-Tool/.

1.2. Wymagania funkcjonalne

Aplikacja powinna dostarczać następujące funkcjonalności:

1. Dodawanie i transkrypcja utworów

- Umożliwia dodanie utworu na podstawie pliku audio (**mp3**) lub poprzez podanie linku do nagrania w serwisie **textit**.
- Dokonuje automatycznej transkrypcji nagrania do formatu **MIDI**.
- Umożliwia konwersję zarówno nagrań zagranych w całości na pianinie, jak i takich, gdzie występuje wiele instrumentów.

2. Generowanie animacji treningowej

- Umożliwia wygenerowanie i wyświetlenie animacji przedstawiającej klawiaturę pianina podczas gry danego utworu. Kolejne nuty są symbolizowane jako prostokąty opadające na odpowiednie klawisze, a ich długość odpowiada czasowi trwania dźwięku. W dalszej części pracy będą one określane jako *animacje treningowe*.
- Pozwala na przełączanie się pomiędzy animacjami dodanych wcześniej utworów.

3. Galeria utworów i zarządzanie nimi

- Prezentuje listę dostępnych utworów w formie galerii.
- Umożliwia zmianę tonacji wybranego utworu.
- Pozwala zapisywać zmodyfikowane wersje utworów jako nowe pozycje w galerii (np. po zmianie tonacji).
- Umożliwia edycję informacji o utworach (tytuł, opis, miniatura).

4. Eksport zapisu nutowego i formatów muzycznych

- Umożliwia eksport utworu w formie tradycyjnego zapisu nutowego **PDF**, a także w formatach **MusicXML** oraz **MIDI**.

5. Rozpoznawanie nut w czasie rzeczywistym

- Umożliwia nasłuchiwanie dźwięków z mikrofonu.
- Rozpoznaje zagrane lub zaśpiewane nuty w czasie rzeczywistym.

6. Gra edukacyjna 3D

- Udostępnia grę edukacyjną, której celem jest nauka rozpoznawania i odgrywania nut.
- Generuje na pięciolinii losowe nuty do zagrania w ograniczonym czasie.

1.3. Wymagania niefunkcjonalne

System powinien spełniać następujące wymagania:

1. Wydajność

- Przełączanie między elementami interfejsu odbywa się natychmiastowo.
- Czas trwania złożonych obliczeniowo operacji jest akceptowalnie krótki. Dla utworu trwającego kilka minut czas zajmuje:
 - transkrypcja utworów: do kilkudziesięciu sekund.
 - generowania animacji treningowej: poniżej minuty.
 - generowania zapisów nutowych (MusicXML, PDF): do 7 sekund.
- Przełączanie między wygenerowanymi wcześniej animacjami odbywa się natychmiastowo.
- Pobieranie wygenerowanych wcześniej zapisów utworu rozpoczyna się po czasie 1–2 s od wysłania żądania.

2. Niezawodność i stabilność

- W przypadku błędów wyświetlane są stosowne komunikaty w interfejsie (np. brak pliku, wysłanie zbyt wielu żądań w oknie czasowym).

3. Bezpieczeństwo

- Wdrożony jest zestaw mechanizmów zapobiegającym atakom mającym na celu nadmierne obciążenie serwera (szczególnie istotne ze względu na obsługę złożonych obliczeniowo żądań).

4. Użyteczność

- Interfejs użytkownika jest prosty i minimalistyczny, utrzymany w odcieniach czerni i bieli.
- System jest dostępny zarówno na komputerach stacjonarnych, jak i na urządzeniach mobilnych.

5. Łatwość wdrożenia i konfiguracji

- Dostarczony zostaje podręcznik użytkownika zawierający opis aplikacji i jej funkcjonalności.
- Aplikację można uruchomić zarówno w kontenerze Docker, jak i z użyciem wirtualnego środowiska Python; podręcznik zawiera instrukcję jak to zrobić.
- Dostarczona jest demonstracyjna wersja online.

1.4. Przegląd literatury i istniejących rozwiązań

Dwa pierwsze projekty opisane poniżej nie są narzędziami do nauki gry na pianinie, a jedynie systemami automatycznej transkrypcji muzyki. Odpowiadają one wyłącznie za konwersję nagrania audio do formatu MIDI.

Basic Pitch

Basic Pitch to system automatycznej transkrypcji muzyki od *Spotify*. Ze względu na złożoność tego zadania, najlepsze rezultaty osiągają systemy zaprojektowane z myślą o jednym, konkretnym instrumencie. Jak pokazują wyniki z benchmarków, model od *Spotify*, osiąga dobre lub bardzo dobre wyniki dla większości instrumentów [1]. Co bardzo ważne, dobrze sobie także z nagraniami zawierającymi wiele instrumentów jednocześnie. Nie jest jednak wyspecjalizowanym rozwiązaniem dla pojedynczych instrumentów i w przypadku pianina nie osiąga rezultatów *state-of-the-art*.

Transkun

Jest to system automatycznej transkrypcji muzyki zaprojektowany z myślą o fortepianie. W przeciwieństwie do rozwiązań uniwersalnych, jak omówiony wyżej model od *Spotify*, jego architektura została dostosowana tak, by dawać jak najlepsze rezultaty dla nagrań fortepianowych. **Transkun** uzyskał najwyższe dotychczas wyniki dokładności transkrypcji fortepianu dla zbioru danych **MAESTRO** [2], będącego jednym z najczęściej wykorzystywanych benchmarków w tej dziedzinie [3].

Piano Marvel

Piano Marvel [17] to narzędzie wspierający naukę gry na pianinie. Umożliwia walidację wykonywanych przez użytkownika utworów. Jest bardzo popularne w szkołach muzycznych oraz przez prywatnych nauczycieli. Ma jednak ograniczoną bazę dostępnych utworów. Co prawda pozwala na dodawanie własnych utworów, lecz wymagane jest dostarczenie pliku w formacie MIDI lub MusicXML. To co odróżnia prezentowaną w tej pracy aplikację, to wsparcie nauki dla dowolnego utworu, bez konieczności posiadania jego zapisu nutowego.

Rozdział 2.

Implementacja

W poniższym rozdziale omówione zostaną szczegóły implementacyjne. Sekcje 2.5.1. - 2.5.5. odpowiadają sekcjom 3.1.1. - 3.1.5. podręcznika użytkownika, które zawierają wysokopoziomowy opis poszczególnych funkcjonalności. Warto przeczytać je przed zapoznaniem się z poniższym rozdziałem.

Kod źródłowy dostępny jest w publicznym repozytorium na serwisie `GitHub` ¹.

2.1. Strategia rozwoju projektu

Projekt rozwijany był w sposób iteracyjny. Na początku w pełni skupiłem się na zaimplementowaniu wszystkich zaplanowanych funkcjonalności, często świadomie ignorując ich niedoskonałości, a dopiero później na poprawkach oraz dokładnym opisie ich działania. Zdecydowałem się na takie podejście, ponieważ uznałem, że pisanie dokumentacji równoległe z programowaniem byłoby rozpraszające i mało efektywne. Jednocześnie zależało mi na tym, by w momencie pisania części pisemnej mieć kod „na świeżo” w pamięci. Pozostawienie poprawek miało zmusić mnie do ponownego kontaktu z kodem przy pisaniu części pisemnej, a jednocześnie pozwoliło oszczędzić czas przy pierwszej iteracji implementacji.

Kolejne etapy pracy nad projektem można podzielić na:

1. Wstępne planowanie i analiza:

- określenie wymaganych funkcjonalności,
- analiza dostępnych narzędzi i bibliotek do ich realizacji.

2. Implementacja wersji bazowej:

- stworzenie działającej aplikacji z każdą funkcjonalnością w podstawowej formie,

¹<https://github.com/et1141/Piano-Learning-Tool>

- celowe pomijanie szczegółów i niedoskonałości.

3. Iteracyjne ulepszanie i dokumentowanie:

- odświeżenie kodu źródłowego danej funkcjonalności:
 - poprawa błędów, jeśli były zauważone,
 - implementacja usprawnień,
 - poprawa jakości i bezpieczeństwa kodu.
- przygotowanie opisu danego modułu.

Czynności z kroku 3 często, a nawet zazwyczaj, wykonywane były równolegle.

2.2. Przegląd kluczowych technologii i narzędzi

Wszystkie technologie, biblioteki oraz narzędzia wykorzystane w projekcie zostały wyszczególnione w opisach odpowiednich modułów systemu w sekcji 2.5.. Poniżej przedstawiono najważniejsze z nich, wraz z krótkim opisem oraz uzasadnieniem wyboru. Przybliżone zostało również działanie komponentów odpowiedzialnych za generowanie zapisu nutowego oraz animacji treningowej. Fragmenty tej sekcji mogą częściowo powielać treści z innych rozdziałów. Jej celem jest zebranie opisu i uzasadniania wyboru dla wszystkich najważniejszych technologii w jednym miejscu.

Frontend: HTML, CSS i JavaScript

Warstwa frontendowa została zbudowana jako aplikacja typu `single-page`. Podział na wiele podstron nie byłby uzasadniony, ponieważ interfejs składa się jedynie z kilku sekcji o ograniczonej liczbie elementów. Do stylowania wykorzystano framework `W3.CSS` [10], a logika interfejsu została zaimplementowana w języku `JavaScript`. Strona jest dość prosta, brak w niej formularzy, zarządzania skomplikowanym stanem czy rozbudowanych interakcji — użycie cięższych frameworków takich jak `React` czy `Vue` byłoby nieuzasadnione.

Backend: Python i Flask

Ze względu na bogactwo bibliotek oraz narzędzi do pracy z muzyką, warstwa serwerowa serwera została zbudowana w języku `Python`. Do budowy API użyto frameworka `Flask`, wybrany ze względu na prostotę i elastyczność w tworzeniu endpointów typu REST, co pozwoliło na szybkie prototypowanie i rozwój aplikacji. Framework `Django` nie został zastosowany, oferuje wiele rozbudowanych mechanizmów, które w kontekście tego projektu byłyby zbędne i narzucałyby dodatkową, niepotrzebną strukturę.

Baza danych: SQLite[5]

Jako system zarządzania bazą danych wykorzystano SQLite3. Opis wraz z uzasadnieniem wyboru dostępne są w sekcji 2.3.

Modele transkrypcji: Transkun i Basic Pitch

Kluczowym komponentem systemu są modele sztucznej inteligencji dokonujące konwersji nagrań audio do formatu MIDI.

- **Transkun** — model oparty na architekturze transformera, wyspecjalizowany w transkrypcji nagrań fortepianowych. Osiąga wyniki *state-of-the-art* na zbiorze danych MAESTRO, przez co wybrany jest jako domyślny model do transkrypcji.
- **Basic Pitch** — uniwersalny model opracowany przez *Spotify*, dobrze sprawdzający się dla większości instrumentów, oraz utworów z wieloma instrumentami jednocześnie, czy nawet wokalem. Stosowany jest jako alternatywa, gdy w nagraniu występują inne instrumenty niż pianino.

Generowanie animacji treningowej: synthviz

Do wizualizacji gry na pianinie wykorzystano bibliotekę **synthviz**. Jest to proste rozwiązanie typu *open-source*, łatwe w integracji i konfiguracji, co przesądziło o jego wyborze do realizacji tego zadania. Jak podaje autor, jej implementacja oparta została na wpisie blogowym Davida Barry’ego „*Making Synthesia-style videos in Ubuntu*”², w którym przedstawiono koncepcję tworzenia prostych animacji w stylu programu *Synthesia*.

Działanie funkcji sprowadza się do generowania kolejnych klatek przedstawiających klawiaturę pianina oraz spadające prostokąty symbolizujące nuty z pliku MIDI. Proces przebiega w następujących krokach:

1. **Analiza pliku MIDI** – na podstawie zdarzeń `note_on` oraz `note_off` określany jest czas rozpoczęcia i zakończenia każdej nuty.
2. **Mapowanie na klawiaturę** – każda nuta odwzorowywana jest na odpowiedni klawisz, a jej czas trwania odpowiada długości prostokąta.
3. **Generowanie klatek** – dla każdej ramki rysowana jest klawiatura oraz aktualne nuty w ruchu. Klawisze aktualnie grane są dodatkowo podświetlane.
4. **Składanie animacji** – wygenerowane obrazy łączone są w sekwencję wideo z wykorzystaniem narzędzia `ffmpeg` [18]. Równolegle dołączany jest dźwięk uzyskany z pliku MIDI przy pomocy syntezy `timidity` [19]).

²https://pappubahry.com/misc/piano_diaries/synthesia/

Pobieranie nagrań: yt-dlp

Narzędzie `yt-dlp` pozwala na pobieranie nagrań dostępnych online. Jest to projekt *open-source*, wykorzystywany w tysiącach stron internetowych ³. Narzędzie wydawało się być najlepszym spośród dostępnych rozwiązań tego typu.

Generowanie zapisów nutowych: music21 + LilyPond

Do obsługi konwersji plików MIDI na zapis nutowy wykorzystano moduł `converter` biblioteki `music21` [6] Funkcja `converter.parse()`⁴ ładuje plik MIDI i zwraca strukturę `Stream`, którą następnie można zapisać w różnych formatach notacji muzycznej.

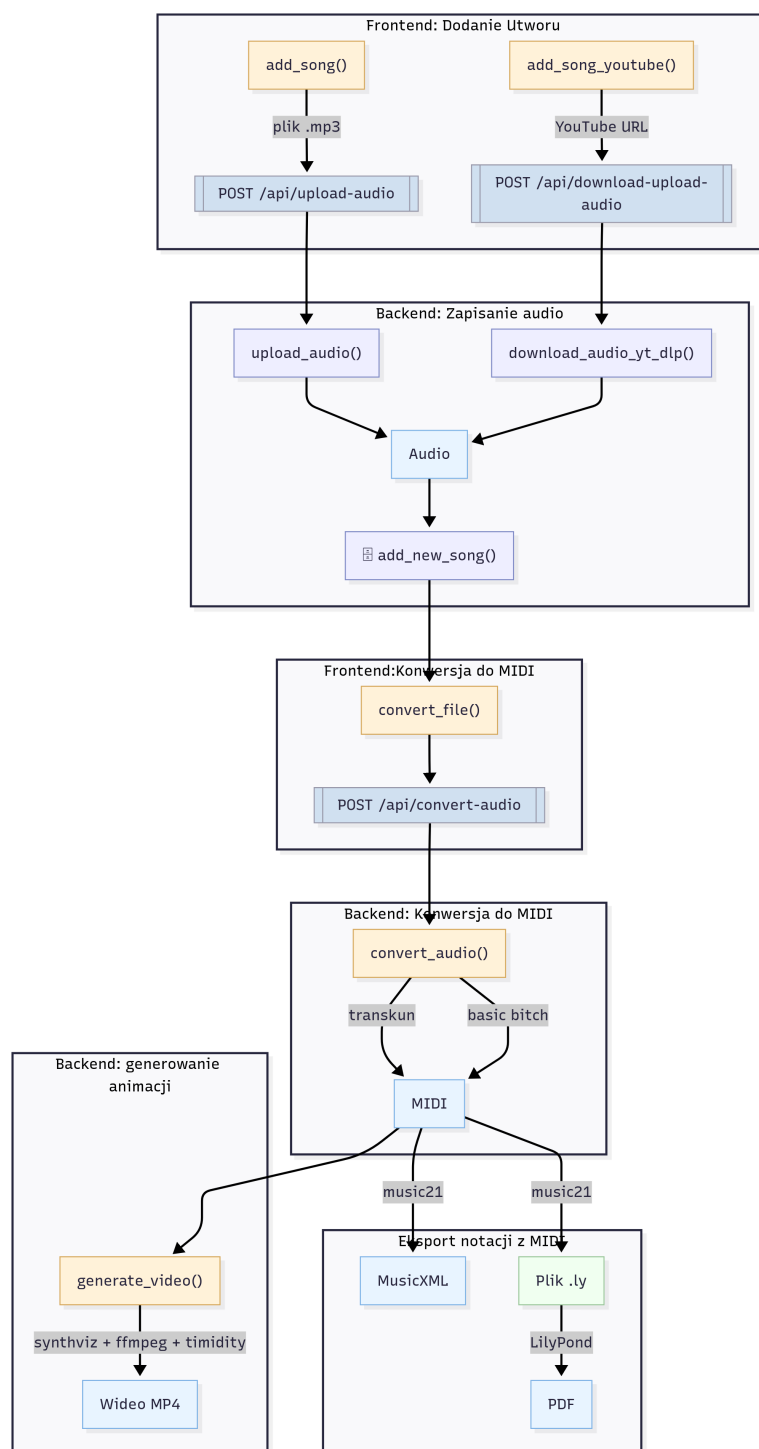
1. MIDI → PDF: Do tworzenia zapisu nutowego w formacie PDF biblioteka `music21` dodatkowo korzysta z oprogramowania `LilyPond`, będącym narzędziem open source do profesjonalnego składania nut. W kroku pośrednim tworzony jest plik `.ly`, który następnie kompilowany ⁵ jest przez `LilyPond`.
2. MIDI → MusicXML Eksport do formatu `MusicXML` realizowany jest w pełni przez bibliotekę `music21`, bez konieczności użycia dodatkowych narzędzi.

Przepływ sterowania podczas powstawania poszczególnych formatów utworu zilustrowany został na diagramie 2.1.

³<https://github.com/yt-dlp/yt-dlp/blob/master/supportedsites.md>

⁴<https://www.music21.org/music21docs/moduleReference/moduleConverter.html>

⁵<https://lilypond.org/doc/v2.24/Documentation/learning/compiling-a-file>



Rysunek 2.1: Diagram prezentujący przepływ sterowania podczas powstawania różnych formatów utworu wygenerowany przy użyciu narzędzia Mermaid Chart [11]

2.3. Dane

Projekt wykorzystuje bazę danych **SQLite3**, wybraną ze względu na prostotę, popularność i brak konieczności uruchamiania dodatkowego serwera. **SQLite3** jest najczęściej używanym silnikiem bazy danych na świecie i bez problemu radzi sobie w obsłudze stron internetowych z małym i średnim obciążeniem. Należy jednak pamiętać, że nie zaleca się jej stosowania w projektach wymagających dużej liczby równoczesnych zapisów. Gdyby aplikacja została upubliczniona i liczba użytkowników gwałtownie wzrosła, należałoby by rozważyć przejście na inny system - na przykład **PostgreSQL**.

Struktura tabel

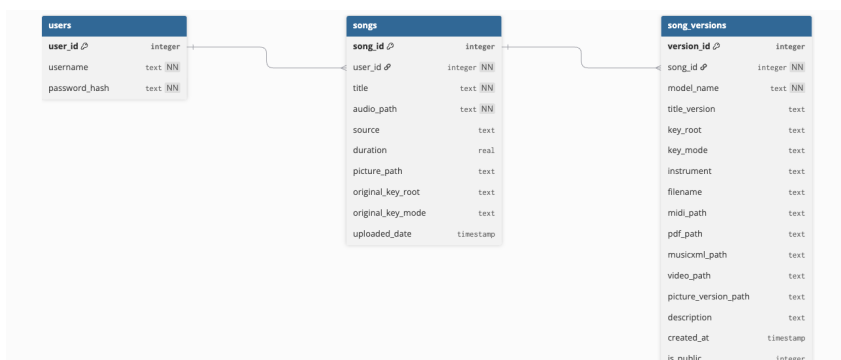
Diagram bazy danych przedstawiony jest na rysunku 2.2.

- **users** — przechowuje dane o użytkownikach: nazwę użytkownika oraz zaszyfrowane hasło. Kluczem głównym jest `user_id`.
- **songs** — zawiera podstawowe metadane utworów takie jak tytuł (`title`), długość nagrania (`duration`) oraz oryginalną tonację (`original_key_root`, `original_key_mode`). A oprócz tego ścieżkę do pliku audio (`audio_path`), źródło (`source` - link *YouTube*) oraz ścieżkę do miniatury (`picture_path`). Kluczem głównym jest `song_id`.
- **song_versions** — przechowuje dodatkowe informacje o utworach i pozwala zapisywać różne wersje tego samego utworu. Zawiera nazwę użytego modelu (`model_name`) przy transkrypcji, dane o tonacji (`key_root`, `key_mode`) (w przypadku gdy została ona zmieniona), instrument, ścieżki do wygenerowanych plików (`midi_path`, `pdf_path`, `musicxml_path`, `video_path`) oraz pola `title_version` i `picture_version_path`, które nadpisują wartości z tabeli **songs**, jeśli użytkownik chce zmienić tytuł lub miniaturę tylko dla jednej wersji. Każda wersja ma też znacznik `is_public`, który określa, czy dany utwór może być widoczny przez pozostałych użytkowników.

W projekcie pominięto implementację dodawania nowych użytkowników oraz procesu logowania. Są to funkcjonalności bardzo powtarzalne i szeroko omówione. Gdyby miała powstać wersja produkcyjna aplikacji to, jak wspomniano wcześniej, należałoby rozważyć zmianę bazy danych **SQLite**. Funkcjonalność logowania zostałaby wtedy dopisana, a sama tabela rozszerzona o dodatkowe pola, takie jak adres e-mail czy informacja o statusie aktywacji konta.

Gdyby nie to, że użytkownik może modyfikować tonację utworu, tabele **songs** oraz **song_versions** mogłyby zostać połączone. Jednak, w przypadku gdy użytkownik chciałby zapisać zmodyfikowany utwór jako nowy, tak, by zachować w galerii również

utwór oryginalny, to tworząc nowy rekord należałoby skopiować również dane niezależne od zmiany tonacji - takie jak `audio_path` czy `source`. Z tego właśnie powodu, by uniknąć replikowania, informacje o utworach zostały podzielone na dwie tabele.



Rysunek 2.2: Schemat bazy danych, wygenerowany przy użyciu narzędzia `dbdiagram.io` [12].

2.4. Frontend

Strona zbudowana została jako single-page application. Całość ładuje się raz, a przełączanie między poszczególnymi widokami nie wymaga ładowania - odpowiednie sekcje są pokazywane lub ukrywane.

Interfejs strony internetowej bazuje na szablonie przygotowanym przez serwis `W3Schools`⁶. Strona jest minimalistyczna i czytelna. Wszystkie elementy (poza oknem dialogowym) stylowane są w odcieniach czerni, bieli oraz szarości. Wykorzystywany jest framework `W3.CSS` [10], dzięki któremu większość stylowania ogranicza się do przypisywania gotowych klas, takich jak `w3-container`, `w3-button`, `w3-black`, `w3-input` i inne.

Strona wyświetla się poprawnie na urządzeniach mobilnych. Przy małej szerokości ekranu pasek nawigacyjny dzieli się na dwa rzędy. Dodatkowo, na telefonach ukrywany jest napis `Piano Learning Tool`, żeby zaoszczędzić miejsce. Efekt ten został osiągnięty poprzez wykorzystanie `media query`.

Interakcja z użytkownikiem

- **Alerty:** Gdy użytkownik próbuje wykonać coś, co przy obecnym stanie nie jest możliwe (np. kliknięcie przycisku `Upload` bez wybrania pliku), pojawia się klasyczny przeglądarkowy `alert()`.
- **Okno dialogowe:** Niektóre z dostarczonych funkcjonalności wymagają czasu dłuższego niż kilka sekund, czasami nawet ponad minutę. Dlatego w aplikacji

⁶https://www.w3schools.com/w3css/tryit.asp?filename=tryw3css_templates_band&stacked=h

zaimplementowano okno dialogowe do przekazywania informacji o stanie realizowanych zadań. Okno jest również wykorzystywane w przypadku błędów. Funkcja odpowiedzialna za wyświetlanie komunikatu to `show_user_message`

- **Wskaźniki ładowania:** Operacje, które trwają dłużej (jak transkrypcja audio czy tworzenie wideo), pokazują animację spinnera z zestawu **font-awesome**.
- **Okno modalne:** Edycja metadanych utworu (np. tytuł, tonacja, opis) odbywa się w wyskakującym oknie modalnym.
- **Podświetlanie elementów:** Wybrane przyciski i utwory wyróżniają się inną barwą, która zmienia się również przy najechaniu kursorem.

2.5. Funkcjonalności

2.5.1. Dodawanie nowego utworu i jego transkrypcja

Wykorzystane komponenty:

- **Logika w przeglądarce:**
 - `add_song` — obsługuje proces dodawania nowego utworu, gdy jego źródłem jest plik mp3.
 - `add_song_youtube` — obsługuje proces dodawania nowego utworu, gdy jego źródłem jest link.
 - `convert_file` — odpowiada za transkrypcję utworu. Jest to funkcja pomocnicza wykorzystywana przez `add_song` oraz `add_song_youtube`.
 - `unlock_new_song_lock` — funkcja wykorzystywana w razie błędu do odblokowania blokady po stronie serwera.
- **Logika serwera:**
 - `/api/upload-audio` — dodaje nowy utwór do bazy danych, gdy jego źródłem jest plik mp3.
 - `/api/download-upload-audio` — pobiera nagranie z dostarczonego linku, a następnie dodaje je do bazy danych.
 - `/api/convert-audio` — konwertuje nagranie audio do formatu MIDI, dokonuje jego analizy i aktualizuje utwór w bazie danych o uzyskane informacje.
 - `/api/unlock-upload` — odblokowuje blokadę lock po stronie serwera (na żądanie klienta).
- **Wykorzystane biblioteki i narzędzia:**

- `yt-dlp` — pobieranie nagrań audio.
- `threading.Lock` — blokowanie przesyłu wielu utworów jednocześnie.
- `transkun`, `basic pitch` — modele wykorzystywane przy transkrypcji.
- `music21` - analiza utworu, rozpoznanie tonacji.

Opis działania:

Proces dodawania nowego utworu podzielić można na dwa główne etapy: przesłanie lub pobranie nagrania oraz jego transkrypcja. Po stronie *frontend* obsługiwany jest (w zależności od źródła utworu) przez funkcję `add_song()` lub `add_song_youtube()`. Obie są bardzo podobne i różnią się jedynie tym, że w pierwszym przypadku użytkownik przesyła plik `mp3`, natomiast w drugim podaje link do nagrania.

W pierwszym etapie, w zależności od źródła utworu, wywoływane jest żądanie do odpowiedniego punktu końcowego (endpointu) `/api/upload-audio` lub `/api/download-upload-audio`. Efekt działania obu z nich jest taki sam: nagranie utworu zostaje zapisane po stronie serwera i trafia do bazy danych. Jeśli etap ten zakończy się sukcesem, wykonywana jest transkrypcja. Niezależnie od źródła nagrania, uruchamiana jest funkcja `convert_file()`, która wysyła żądanie do endpointu `/api/convert-audio`. Ścieżka do nagrania audio utworu zostaje pobrana z bazy danych, a następnie dokonywana jest transkrypcja przy pomocy wybranego wcześniej przez użytkownika modelu. W jej wyniku powstaje plik MIDI, który zostaje zapisany i wykorzystany do analizy utworu przy pomocy biblioteki `music21`. Na koniec utwór zostaje uaktualniony w bazie danych o uzyskane metadane oraz ścieżkę do pliku MIDI. Cały proces został zilustrowany na rysunku 2.3.

Mechanizm blokowania wielu przesyłów jednocześnie

W pierwotnej wersji system pozwalał na rozpoczęcie wielu żądań dodania nowego utworu jednocześnie. Zostało to zmienione i obecnie możliwym jest rozpoczęcie tylko jednego żądania tego typu. Wynika to z tego, że transkrypcja oraz pobieranie są zasobożerne. W przypadku dłuższych nagrań mogą trwać ponad minutę. Obsługa wielu jednoczesnych operacji tego typu mogłyby łatwo stać się wąskim gardłem całej aplikacji, a także prostym sposobem na przeprowadzenie ataku.

Z tego powodu został wprowadzony mechanizm blokady oparty na obiekcie `threading.Lock`, który kontroluje dostęp do endpointów odpowiedzialnych za dodanie nowego utworu. Lock aktywowany jest na samym początku działania endpointów `/api/upload-audio` lub `/api/download-upload-audio`. Co ważne, nie jest on zwalniany po pierwszym etapie — pozostaje aktywny aż do zakończenia transkrypcji, czyli do końca działania funkcji `/api/convert-audio`. Dzięki temu aplikacja nie pozwala na rozpoczęcie nowego procesu dodawania utworu (niezależnie od źródła) dopóki poprzedni nie zakończy się całkowicie. Dokładniejszy opis mechanizmu blokowania:

1. Na początku każdego przesyłu podejmowana jest próba zablokowania obiektu `threading.Lock`. Jeśli operacja się nie powiedzie (inny proces już trwa), zwracany jest kod błędu HTTP 429 (`Too many requests`).
2. Po zablokowaniu rejestrowany jest czas rozpoczęcia przesyłania. Na tej podstawie działa dodatkowy mechanizm bezpieczeństwa, który wymusza odblokowanie po upływie określonego limitu czasu (obecnie limit wynosi 5 minut).
3. Zwalnianie blokady:
 - Docelowym miejscem zwalniania blokady jest zakończenie działania endpointu `/api/convert-audio`. Zwolnienie następuje w bloku `finally`, co gwarantuje poprawne odblokowanie niezależnie od wyniku operacji.
 - Jednocześnie, jeśli na którymkolwiek etapie pojawi się błąd to blokada zostaje zwolniona przy pomocy funkcji `unlock_new_song_lock()`. W przypadku wystąpienia błędów po stronie klienta do odblokowania służy endpoint `/api/unlock-upload`.

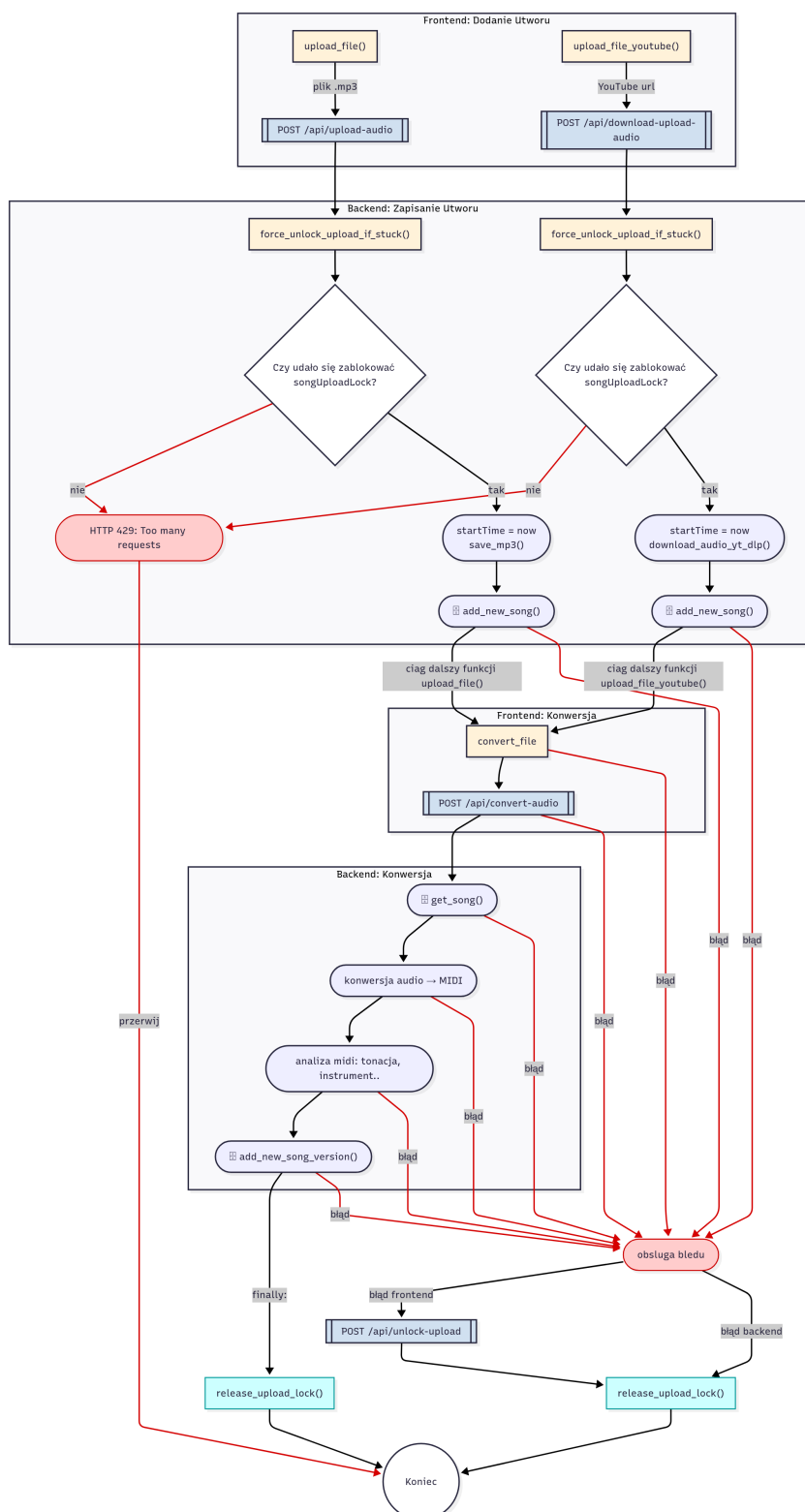
2.5.2. Generowanie wideo treningowego

Wykorzystane komponenty:

- **Logika w przeglądarce:**
 - `generate_training_video` — odpowiada za cały proces od strony użytkownika: pobranie tytułu utworu, wysłanie żądania wygenerowania wideo, aktualizację interfejsu oraz załadowanie pliku, gdy będzie gotowy.
 - `wait_for_video` — funkcja pomocnicza która cyklicznie wysyła zapytania HEAD do serwera, sprawdzając, czy wideo jest już gotowe.
- **Logika serwera:**
 - `/api/get-video` — główny endpoint. Przy żądaniu HEAD informuje, czy wideo jest już dostępne. Przy zapytaniu GET zwraca gotowy plik, a jeśli pliku nie ma, inicjuje jego wygenerowanie.
 - `generate_video` — uruchamia proces tworzenia wideo, zapisuje je i aktualizuje bazę danych.

Proces działa następująco:

1. Główny endpoint `/api/get-video` sprawdza, czy istnieje plik wideo:
 - Jeśli tak — od razu zwraca kod 200 (OK) (dla żądania HEAD) lub gotowy plik (dla GET).



Rysunek 2.3: Przepływ sterowania dla sekcji „Upload”, wygenerowany przy użyciu narzędzia Mermaid Chart

- Jeśli nie — uruchamiana jest funkcja `generate_video`. Po zakończeniu generowania zwracany jest kod 201 (`Created`) dla żądań `HEAD`, lub nowo wygenerowany plik dla żądań `GET`.
 - Jeśli wideo dla tego utworu jest już w trakcie generowania i nadejdzie kolejne żądanie, zwracany jest kod 429.
2. Funkcja `generate_training_video()` wywoływana jest automatycznie po wejściu do omawianej sekcji. Wyświetla w interfejsie odpowiednie komunikaty, a następnie wywołuje funkcję pomocniczą `wait_for_video()`, która co 4 sekundy wysyła żądanie `HEAD` do `/api/get-video`. Poniżej opisane są trzy możliwe scenariusze zakończenia oczekiwania na wideo w zależności od kodu zwróconego przez serwer, oraz opis co po nich następuje po stronie klienta.
- 200: wideo było już wcześniej wygenerowane. Zostaje ono natychmiast załadowane i wyświetlone użytkownikowi.
 - 201: wideo zostało właśnie wygenerowane (proces mógł trwać kilkadziesiąt sekund). Sprawdzane jest, czy użytkownik nadal znajduje się w sekcji generowania wideo i ma wybrany ten sam utwór. Jeśli tak, to świeżo wygenerowana animacja zostaje pokazana. W przeciwnym wypadku (gdy użytkownik zmienił sekcję lub odtwarza wideo dla innego utworu), wyświetlany jest jedynie komunikat informujący że wideo zostało wygenerowane.
 - 429: proces po stronie klienta zostaje zakończony, ponieważ kod ten oznacza, że wideo dla danego utworu jest już w trakcie generowania. Innymi słowy, po stronie klienta, istnieje wcześniejsze wywołanie funkcji `wait_for_video()`, które nadal oczekuje na odpowiednią odpowiedź od serwera i po jej otrzymaniu odpowiednio zareaguje (ładując plik lub wyświetlając odpowiedni komunikat).

Proces generowania animacji treningowej jest złożony obliczeniowo. Zajmuje nawet połowę czasu potrzebnego do dokonania transkrypcji. Aby zabezpieczyć serwer przed wykonywaniem niepotrzebnej pracy oraz zminimalizować możliwość ataków zastosowano:

- **podejście leniwe:** wideo nie jest generowane automatycznie przy dodawaniu utworu, lecz dopiero przy pierwszym żądaniu klienta,
- **blokadę dla pojedynczego utworu:** po rozpoczęciu generowania animacji dla danego utworu jego id zostaje zapisane w zbiorze `videoGenerationJobs`, chronionym przez obiekt `threading.Lock()`. W razie pojawienia się kolejnego żądania o wygenerowanie wideo tego samego utworu serwer nie uruchamia drugiego procesu, lecz zwraca status 429.

Możliwe jest równoczesne generowanie animacji dla wielu różnych utworów. Potencjalnie mogłoby to również stanowić punkt ataku mającego na celu przeciążenie

serwera. Nie powinno się to jednak zdarzyć w typowym scenariuszu, ponieważ zablokowane jest wysyłanie wielu żądań dodawania nowych utworów jednocześnie (a muszą one zostać dodane, zanim będzie można wygenerować dla nich wideo). Oczywiście użytkownik mógłby nadal dodać kilka utworów pojedynczo, a następnie wysłać równoległe żądania o generowanie animacji. W wersji produkcyjnej należałoby zatem dodatkowo to zabezpieczyć, ograniczając liczbę jednoczesnych żądań przypadających na jednego użytkownika. Można to osiągnąć w bardzo prosty sposób, sprawdzając rozmiar zbioru `videoGenerationJobs`.

Aby wideo odpowiadało aktualnej wersji utworu, każdorazowa zmiana tonacji (np. modyfikacja pola `key_root`) skutkuje przetworzeniem pliku MIDI oraz usunięciem dotychczasowego pliku wideo.

2.5.3. Galeria utworów, zmiana tonacji oraz generowanie partytury

Wykorzystane komponenty:

- **Logika w przeglądarce:**

- `load_songs_gallery` — odpowiada za wygenerowanie galerii utworów.
- `manage_song` — wyświetla okno modalne do edycji parametrów utworu.
- `save_song` — pobiera dane z okna modalnego i zapisuje je w bazie danych. W zależności od wybranej opcji, aktualizuje istniejący utwór lub zapisuje go jako nową wersję.
- `delete_song` — usuwa wskazany utwór z bazy.
- `download_song_file`, — umożliwia pobranie utworu w formatach mp3, MIDI, MusicXML lub PDF.

- **Logika serwera:**

- `/api/get_audio`, `/api/get_midi`, `/api/get_musicxml`, `/api/get_pdf` — zwracają odpowiedni plik. W przypadku MusicXML lub PDF, jeżeli plik nie istnieje, endpoint odpowiada również za jego wygenerowanie.
- `/api/update-song` — wykorzystywany przez `save_song`; aktualizuje dane utworu.

- **Biblioteki i narzędzia:**

- `music21` — do generowania partytur i konwersji z MIDI do MusicXML oraz PDF.

Generowanie zapisu nutowego: Utwór można wyeksportować jako klasyczny zapis nutowy w formacie PDF, jako plik MusicXML lub MIDI. Dla dwóch pierwszych formatów, zastosowano leniwe podejście - pliki nie są tworzone od razu, podczas

dodawania utworu, a dopiero przy pierwszym żądaniu użytkownika. Pozwala to oszczędzić miejsce i czas pracy serwera. Raz wygenerowany plik zostaje zapisany i jest dostępny przy kolejnych żądaniach natychmiastowo.

Generowanie zapisu nutowego w formatach PDF oraz MusicXML bazuje na wygenerowanym już zapisie MIDI, dzięki czemu nie jest bardzo zasobożerne. Mimo tego, podobnie jak w przypadku generowania wideo, warto zabezpieczyć się przed sytuacją, w której użytkownik wysyła wielokrotnie żądanie wygenerowania danego pliku. Każde uruchamiałoby osobny proces serwera przez co wykonywane byłby wielokrotnie dokładnie to samo zadanie.

Czas generowania zapisu nutowego dla znacznej większości utworów można oszacować z góry przez 7 sekund. Zastosowano proste zabezpieczenie, opierające się tym, na blokadzie na siedmiosekundowej blokadzie przycisku odpowiedzialnego za wysyłanie żądania pobrania pliku, po każdym kliknięciu. Próba wysłania żądania w krótszym czasie, spowoduje jedynie wyświetlenie odpowiedniego komunikatu:

```
function download_song_xml(versionId) {
  if (wasMusicXmlClicked){
    show_user_message("File already downloading", true);
    return;
  }
  wasMusicXmlClicked = true;
  const xmlUrl = `${BACKEND_URL}/api/get-musicxml?${versionId}`;
  const link = document.createElement('a');
  link.href = xmlUrl;
  link.click();

  setTimeout(() => {
    wasMusicXmlClicked = false;
  }, 7000);
}
```

Powyższe rozwiązanie działa, ale ma wadę — kliknięcie przycisku pobierania dla jednego utworu blokuje możliwość pobierania pliku dla wszystkich innych. Każdy utwór powinien mieć zatem własną blokadę. Możliwe byłoby osiągnięcie tego poprzez użycie mapy zamiast pojedynczej zmiennej globalnej. Można to jednak zrobić prościej i bardziej elegancko poprzez przypisanie stanu bezpośrednio do przycisku:

```
function download_song_xml(versionId, event) {
  const button = event.target;

  if (button.dataset.cooldown === 'true') {
```



```
        show_user_message("File already downloading", true);
        return;
    }

    button.dataset.cooldown = 'true';
    const xmlUrl = `${BACKEND_URL}/api/get-musicxml?${versionId}`;
    const link = document.createElement('a');
    link.href = xmlUrl;
    link.click();

    setTimeout(() => {
        button.dataset.cooldown = 'false';
    }, 7000);
}
```

Efekt jest taki, że każdemu przyciskowi przypisany jest osobny stan blokady i kliknięcia nie wpływają na siebie nawzajem.

Dla bardzo długich utworów lub gdy serwer jest obciążony, generowanie pliku mogłoby trwać dłużej niż wspomniane 7 sekund. Użytkownik może wtedy kliknąć ponownie i wysłać to samo żądanie, które zostanie zrealizowane. W praktyce jednak zdarzyć się to może bardzo rzadko, a ewentualne podwójne wywołanie generowania nie stanowi dużego problemu. Powtórzenie takiej sytuacji ponownie ma pomijalne prawdopodobieństwo zajścia. Uzyskana została zatem prostota implementacji, kosztem minimalnego ryzyka ponownego wykonania niektórych zapytań.

2.5.4. Rozpoznawanie nut z mikrofonu w czasie rzeczywistym

Wykorzystane komponenty:

- **Logika w przeglądarce:**

- `toggle_listening` — przełącza tryb nasłuchiwanie mikrofonu.
- `start_listening` — uruchamia nasłuchiwanie z mikrofonu, inicjalizuje obiekt `AudioContext` oraz `AnalyserNode`, a następnie rozpoczyna pętlę nasłuchiwanie.
- `stop_listening` — zatrzymuje pętlę nasłuchiwanie.
- `listen_loop` — pobiera próbki dźwięku z obiektu `AnalyserNode`, oblicza jego wysokość przy użyciu funkcji `findPitch` z biblioteki `pitchy`, a następnie przelicza wynik na numer MIDI ⁷ i nazwę nuty, po czym wyświetla wynik w interfejsie.

- **Biblioteki i narzędzia:**

⁷https://inspiredacoustics.com/en/MIDI_note_numbers_and_center_frequencies

- `pitchy` — prosta biblioteka do wykrywania wysokości dźwięku, napisana w całości w języku JavaScript. Jest przystosowana do pracy w czasie rzeczywistym [16].

Proces rozpoznawania dźwięków podzielić można na etapy:

1. Pobranie próbki dźwięku
2. Obliczenie częstotliwości oraz czystości dźwięku przy pomocy funkcji `findPitch` z biblioteki `pitchy`.
3. Jeśli czystość dźwięku jest wyższa niż 80%, to zgodnie z poniższym wzorem przeliczamy jego częstotliwość na numer MIDI

$$n = 69 + 12 \cdot \log_2 \left(\frac{f}{440 \text{ Hz}} \right)$$

Dodatkowo zastosowano prostą histerezę: nowa nuta pojawia się w interfejsie dopiero wtedy, gdy utrzyma się przez kilka kolejnych próbek, co poprawia precyzję i ogranicza migotanie wskazań w interfejsie.

4. Na podstawie numeru MIDI można łatwo odczytać nazwę nuty — korzystając z tabeli mapującej numery MIDI na nazwy nut i oktawy ⁷.

2.5.5. Prosta gra 3D do nauki nut

Kod źródłowy gry znajduje się w osobnym repozytorium GitHub ⁸. Gra do projektu głównego została zaimportowana statycznie. Do budowy sceny wykorzystano framework `A-Frame` [13] oraz bibliotekę `Spatial Design System` [14], która udostępnia gotowe komponenty interfejsu. Modele wykorzystane do budowy sceny pochodzą ze strony `poly.pizza` [15]. Model pianina został przygotowany przez `polygonrunway` i uzupełniony o animacje naciskania klawiszy w programie `Blender`. Sama implementacja gry jest bardzo prosta - składa się z kilku komponentów, które odpowiadają za logikę rozgrywki, obsługę mikrofonu oraz interfejsu. Wyszczególnienie każdej funkcji zostaje zatem pominięte. Warto jednak wspomnieć o napotkanym w czasie implementacji problemie dotyczącym ukrywania przycisków z biblioteki `Spatial Design System`. W niektórych momentach poszczególne przyciski powinny być ukryte — przykładowo, po otwarciu panelu ustawień, przycisk odpowiedzialny to powinien zostać ukryty. Jeden z autorów biblioteki `Spatial Design System` zaleca by komponenty ukrywać poprzez ustawienie parametru `scale` na zero:

```
this.btnOpen.setAttribute("scale", "0 0 0")
```

⁸<https://github.com/et1141/simple-3d-music-notes-learning-game>

Okazało się jednak, że takie podejście nie działa. Powodem jest animacja kliknięcia wbudowana w komponent `ar-button`. Symuluje ona ruch przycisku, wykorzystując do tego właśnie manipulację parametrem `scale`. Animacja ta trwa ułamek sekundy. Zmiana skali na zero wykonywała się jednak od razu po kliknięciu, a w efekcie była natychmiast nadpisywana. Dodanie opóźnienia rozwiązało problem.

```
setTimeout(() => this.btnOpen.setAttribute("scale", "0 0 0"), 200);
```


Rozdział 3.

Podręcznik użytkownika

3.1. Funkcjonalności

Sekcja ta opisuje moduły dostarczane przez aplikację i jednocześnie może być traktowana jako przewodnik po jej obsłudze.

3.1.1. Dodawanie nowego utworu i jego transkrypcja

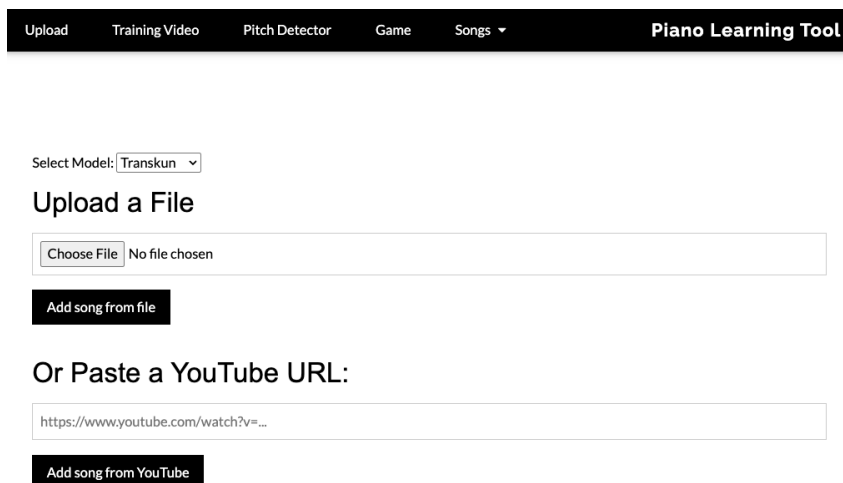
Użytkownik ma możliwość dodania nowego utworu na dwa sposoby: poprzez wgranie pliku audio w formacie **mp3** lub poprzez podanie linku do nagrania w serwisie *YouTube*. W praktyce działają również linki z innych serwisów, jednak pełne wsparcie dla dowolnego źródła nie jest gwarantowane.

Wraz z przesłaniem nagrania, dokonywana jest jego transkrypcja — generowany jest odpowiadający mu plik **MIDI**. Konwersja możliwa jest przy pomocy jednego spośród dostępnych modeli do transkrypcji:

- **Transkun:** Jest to model wybierany domyślnie. Zalecany w przypadku nagrań fortepianowych, dla których osiąga najlepsze spośród znanych wyniki.
- **Basic Pitch:** Model od *Spotify*, dobrze radzi sobie także z nagraniami zawierającymi wiele instrumentów lub wokół.

Po zakończeniu transkrypcji, utwór pojawia się w galerii (sekcja *Songs*). Wygenerowany plik **MIDI** jest gotowy do pobrania i wykorzystania w pozostałych modułach systemu.

Uwaga: w przypadku konwersji wielu utworów w krótkim czasie, gdzie źródłem jest link *YouTube*, funkcjonalność ta może zostać tymczasowo zablokowana. Nie wynika to z problemu po stronie aplikacji, lecz z protokołu zabezpieczającego **SABR** (**S**ignature and **B**ot **R**isk). W takiej sytuacji adres IP użytkownika może zostać



The screenshot shows the top navigation bar of the 'Piano Learning Tool' with links: Upload, Training Video, Pitch Detector, Game, Songs, and Piano Learning Tool. Below the navigation bar, there is a 'Select Model' dropdown menu set to 'Transkun'. The main section is titled 'Upload a File' and contains a file upload area with a 'Choose File' button and the text 'No file chosen'. Below this is a black button labeled 'Add song from file'. The next section is titled 'Or Paste a YouTube URL:' and contains a text input field with the placeholder 'https://www.youtube.com/watch?v=...'. Below the input field is a black button labeled 'Add song from YouTube'.

Rysunek 3.1: Interfejs aplikacji wraz z otwartą sekcją *Upload*, umożliwiającą dodanie nowego utworu

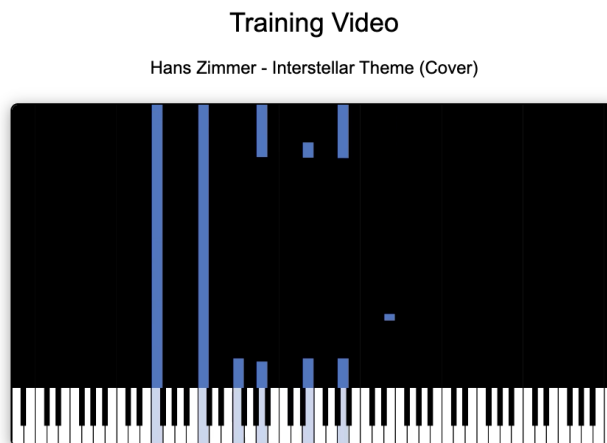
zablokowany przez serwis *YouTube*. Problem ten rozwiązać można korzystając z VPN lub zmieniając adres IP w inny sposób.

3.1.2. Generowanie wideo treningowego

Sekcja *Training Video* pozwala na wyświetlenie animacji z klawiaturą pianina podczas gry danego utworu. Nagrania tego typu są popularne wśród początkujących i znacznie ułatwiają naukę gry danego utworu. Na wstępie warto jednak zaznaczyć, że pozostanie jedynie przy tej metodzie nauki w dłuższej perspektywie może prowadzić do ograniczeń. Uczenie się gry na pianinie tylko przez odtwarzanie wizualizacji, nie rozwija umiejętności czytania nut i spowalnia postęp w improwizacji, dostrzeganiu analogii, czy rozumieniu harmonii. Dlatego nauka w ten sposób powinna być traktowana jako sposób na ułatwienie pierwszego kontaktu z instrumentem i zdobycie dodatkowej motywacji do dalszej nauki. Nie powinna jednak być docelową i jedyną metodą nauki.

Nagłówek *Songs* wyświetla menu rozwijane zawierające dostępne utwory. Kliknięcie na wybrany utwór powoduje wyświetlenie jego animacji treningowej. Nagłówek ten jest także elementem paska nawigacyjnego i otwiera galerię utworów, w której również można zmienić utwór, dla którego wyświetlana obecnie jest animacja.

Animacje nie są generowane automatycznie podczas dodawania utworu do systemu. Z tego powodu przy pierwszej próbie ich wyświetlenia należy liczyć się z kilkudziesięciosekundowym czasem oczekiwania. Kolejne odtworzenia animacji odbywają się już natychmiastowo. Generowanie odbywa się w tle - w jego trakcie możliwe jest korzystanie z aplikacji, w tym wyświetlanie animacji treningowych innych utworów



Rysunek 3.2: Interfejs sekcji **Training Video** z animacją treningową utworu

(dla których była ona wygenerowana wcześniej). Gdy wideo treningowe jest gotowe, wyświetli się odpowiedni komunikat.

3.1.3. Galeria utworów, zmiana tonacji oraz generowanie partytury

Sekcja **Songs** odpowiada za wyświetlenie dostępnych utworów oraz zarządzanie nimi. Jednocześnie pozwala pobrać odpowiadające im pliki **mp3**, **MIDI**, **PDF** (zapis nutowy) lub **MusicXML**.

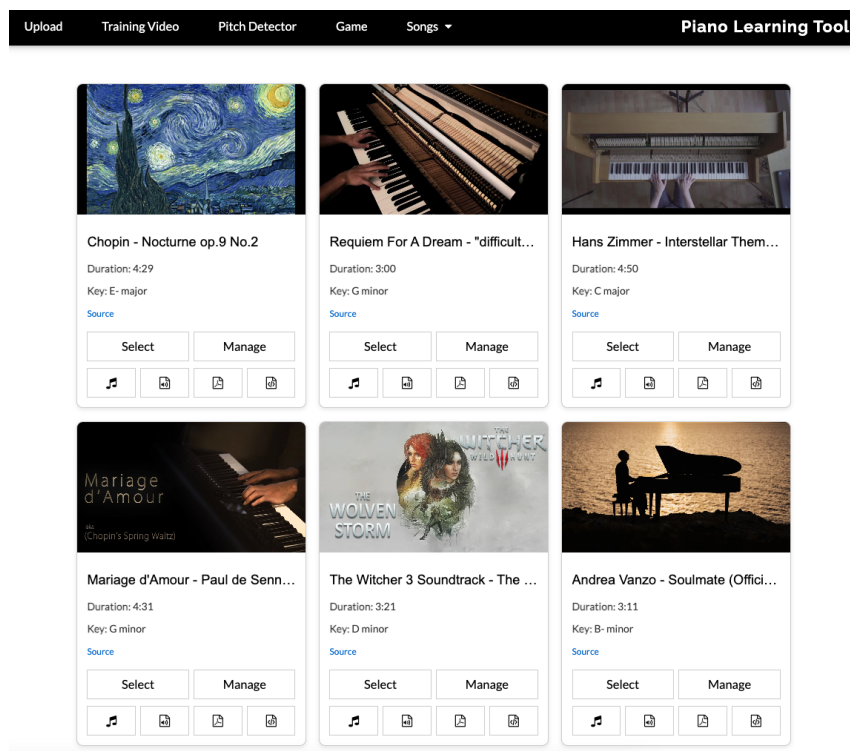
Zarządzanie utworami Po kliknięciu przycisku **Manage** wyświetlone zostaje okno modalne pozwalające na modyfikacje niektórych informacji o utworze. Zmodyfikowanie *Key root* spowoduje zmianę tonacji utworu.

Po wprowadzeniu zmian, utwór zapisać można również jako nową wersję - przydatne jest to właśnie wtedy, gdy użytkownik zmienia tonację utworu, a jednocześnie chce zachować wersję oryginalną. Okno modalne pozwala również na usunięcie utworu.

3.1.4. Rozpoznawanie nut z mikrofonu w czasie rzeczywistym

Moduł *Pitch Detector* pozwala na rozpoznawanie pojedynczych nut z mikrofonu. Po rozpoczęciu nasłuchiwanie, detektor w czasie rzeczywistym klasyfikuje zagrane lub zaśpiewane dźwięki.

Aby rozpocząć detekcję, należy kliknąć przycisk *Start Listening*, a następnie udzielić zgody na korzystanie z mikrofonu. Od tego momentu, dopóki nie przerwiemy nasłuchiwanie, dźwięki będą automatycznie wykrywane i wyświetlane w interfejsie. W przypadku dźwięków bardzo niskich (**A0-C2**), dla osiągnięcia poprawnej detekcji,



Rysunek 3.3: Interfejs sekcji Songs wyświetlający galerię utworów

Edit Song Version

Title

Hans Zimmer - Interstellar Theme (Cover)

Key Root

C

Picture URL

thumbnails/Hans_Zimmer_-_Interstellar_Theme_Cover.jpg

Description

Is Public

No

Save Changes

Save as New Version

Delete

Rysunek 3.4: Interfejs okna modalnego do zarządzania utworem

Single Note Detector

Use your microphone to detect notes in real time.

Stop Listening

Last Played Note: D6

Rysunek 3.5: Interfejs sekcji *Pitch Detector* pozwalającej na klasyfikację dźwięków z mikrofonu w czasie rzeczywistym

wymagany jest mikrofon dobrej jakości oraz brak zakłóceń w otoczeniu. Pozostałe dźwięki są rozpoznawane bezproblemowo.

3.1.5. Prosta gra 3D do nauki nut

Aplikacja zawiera również prostą grę edukacyjną, której celem jest ćwiczenie szybkiego odczytywania nut z pięciolinii oraz zagrywania ich.

Zasady gry

1. Przed rozpoczęciem rozgrywki użytkownik ustala parametry gry: początkowy czas (np. 20s) oraz bonus czasowy za zagranie poprawnej nuty.
2. Po rozpoczęciu gry licznik czasu zaczyna odliczanie w dół. Gra trwa dopóki pozostały czas > 0 .
3. Na pięciolinii wyświetlana jest losowo wybrana nuta do zagrania.
4. Jeśli zagrana zostanie wyświetlona nuta, to pozostały czas zwiększa się o ustaloną wcześniej liczbę sekund, a następnie wyświetlana zostaje kolejna nuta.
5. Gra kończy się, gdy licznik spadnie do zera.

Nuty mogą zostać zagrane przy pomocy instrumentu lub zaśpiewane. Wymagane jest wcześniejsze udzielenie w przeglądarce pozwolenia na korzystanie z mikrofonu. Alternatywnie gracz ma możliwość wprowadzania nut za pomocą klawiatury komputera. Nuty krzyżkowe wprowadza się wtedy poprzez użycie kombinacji klawisza **Shift** oraz odpowiedniej litery (np. **Shift** + **F** = **F#**).

Interfejs gry

Interfejs składa się z panelu ustawień, w którym można skonfigurować parametry gry oraz panelu rozgrywki wyświetlającego aktualną nutę, wynik i pozostały czas. Elementy interfejsu wyświetlane są w przestrzeni trójwymiarowej (scenie 3D).



Rysunek 3.6: Interfejs gry do nauki nut podczas rozgrywki



Rysunek 3.7: Interfejs gry do nauki nut: panel ustawień

3.2. Sposób uruchomienia

Cały proces należy rozpocząć od sklonowania repozytorium projektu:

```
git clone https://github.com/et1141/Piano-Learning-Tool.git
cd Piano-Learning-Tool
```

Dalsza część budowy i uruchomienia aplikacji możliwa jest na dwa sposoby opisane w poniższych sekcjach.

3.2.1. Sposób 1: ręczna instalacja i konfiguracja środowiska

Instalacja wymaganych narzędzi

Do poprawnego działania aplikacji potrzebne są dodatkowe narzędzia systemowe, które należy zainstalować:

- **Python 3.10.13** — wymagany do uruchomienia backendu aplikacji; wersja ta jest konieczna do poprawnego działania biblioteki **Basic Pitch**,
- **FFmpeg** — używane przez **Synthviz** do generowania wideo,
- **Timidity** — służy do syntezy audio z plików MIDI,
- **LilyPond** — używane przez bibliotekę **music21** do generowania zapisu nutowego w PDF.

Przykład instalacji:

- macOS: `brew install ffmpeg timidity lilypond`
- Ubuntu: `sudo apt install ffmpeg timidity lilypond`

Tworzenie środowiska i instalacja paczek

- Utworzenie i aktywacja środowiska

```
python -m venv piano-env
source piano-env/bin/activate
```
- Instalacja zależności

```
pip install --upgrade pip
pip install -r requirements.txt
```

Część paczek w pliku `requirements.txt` ma przypisaną wersję, aby zapewnić powtarzalność środowiska i uniknąć sytuacji, w której aktualizacja którejś z bibliotek spowodowałaby niekompatybilność lub błędy w działaniu aplikacji.

Uruchomienie

Po wykonaniu powyższych kroków można uruchomić serwer backend poleceniem:

```
python backend_server.py
```

Serwer dostępny będzie pod adresem `http://localhost:8000`. Aby uruchomić stronę, wystarczy otworzyć w przeglądarce plik `index.html` lub po prostu przejść pod adres `http://localhost:8000`.

3.2.2. Sposób 2: Docker

Drugim sposobem uruchomienia aplikacji jest wykorzystanie narzędzia **Docker**. Instalacji wymaganych bibliotek systemowych i środowiska uruchomieniowego odbywa się wtedy automatycznie w kontenerze.

Budowa obrazu

Aby zbudować obraz **Docker** należy w katalogu głównym projektu uruchomić polecenie:

```
docker build -t piano-learning-app .
```

Podczas budowania obrazu zostaną:

- zainstalowane wszystkie wymagane narzędzia systemowe (m.in. `ffmpeg`, `timidity`, `lilypond`),
- zainstalowane pakiety **Python**, wyszczególnione w pliku `requirements.txt`,
- skopiowane pliki projektu do katalogu roboczego kontenera.

Uruchomienie kontenera

Aby uruchomić aplikację w kontenerze należy wykonać:

```
docker run -d -p 8000:8000 piano-learning-app
```

Po uruchomieniu kontenera serwer dostępny będzie pod adresem `http://localhost:8000`. Aby uruchomić stronę, wystarczy otworzyć w przeglądarce plik `index.html` lub po prostu przejść pod adres `http://localhost:8000`.

Udostępnienie aplikacji zdalnie

W celu udostępnienia aplikacji przez Internet można skorzystać z narzędzia **ngrok**, które tworzy tunel HTTPS do lokalnego serwera. W tym celu, w osobnym terminalu należy uruchomić:

```
ngrok http 8000
```

Polecenie to wygeneruje publiczny adres URL w domenie `ngrok-free.app`, który będzie przekierowywał ruch do backendu działającego w kontenerze **Docker**.

Rozdział 4.

Podsumowanie

4.1. Konkluzja

Głównym celem projektu było stworzenie narzędzia wspierającego naukę gry na pianinie poprzez umożliwienie automatycznej transkrypcji dowolnego utworu na podstawie jego nagrania audio. Został on w pełni osiągnięty. Aplikacja umożliwia użytkownikowi przesyłanie nagrań (również z serwisu *You Tube*), a następnie eksport zapisu nutowego w kilku formatach. Dodatkowo możliwe jest generowanie animacji wspierającej naukę gry, czy dokonywanie transpozycji utworu. System został dodatkowo rozbudowany o możliwość detekcji nut z mikrofonu w czasie rzeczywistym oraz prostą grę edukacyjną do nauki szybkiego rozpoznawania i grania nut.

4.2. Dalszy kierunek rozwoju projektu

- Trening własnych modeli opartych na architekturze transformer (jak Transkun) dla innych instrumentów lub dla nagrań z wieloma instrumentami.
- Implementacja systemu ewaluacji odegranego przez użytkownika utworu: Nuty mogłyby być odczytywane bezpośrednio z keyboardu po podłączeniu go do komputera lub rozpoznawane z mikrofonu. Dodatkowo, w celu treningu tempa gry, wyświetlana byłaby pięciolinia z nutami do zagrania w danym momencie. Po skończeniu utworu generowane byłoby podsumowanie dotyczące poprawności momentu zagrania nut oraz długości ich trwania - wyszczególnione byłyby nuty zagrane za późno, grane za krótko lub długo oraz pomyłki.
- Doprowadzenie projektu do wersji produkcyjnej: dodanie logowania i potwierdzania użytkowników.
- Rozbudowanie systemu zarządzania utworami przez użytkownika: utwory ulubione, ranking, podział na kategorie.

- Rozbudowanie gry edukacyjnej do nauki nut.
- Analiza i edycja dodatkowych metadanych utworu, takich jak tempo, gatunek muzyczny i inne.

Bibliografia

- [1] Rachel M. Bittner, Juan José Bosch, David Rubinstein, Gabriel Meseguer-Brocal, and Sebastian Ewert. *A Lightweight Instrument-Agnostic Model for Polyphonic Note Transcription and Multipitch Estimation*, 2022. <https://arxiv.org/abs/2203.09893>.
- [2] Yujia Yan and Zhiyao Duan. *Scoring Time Intervals using Non-Hierarchical Transformer For Automatic Piano Transcription*, 2024. <https://arxiv.org/abs/2404.09466>.
- [3] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. *Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset*, 2019. <https://openreview.net/forum?id=r1lYRjC9F7>.
- [4] Armin Ronacher et al, *Flask* — lightweight WSGI web application framework. <https://flask.palletsprojects.com/>.
- [5] SQLite Consortium. *SQLite* — SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. <https://www.sqlite.org/>.
- [6] Michael Cuthbert et al. *music21* — Python-based toolkit for computer-aided musicology. <https://www.music21.org/music21docs/>.
- [7] Carlos Cancino-Chacón, Silvan David Peter, Emmanouil Karystinaios, Francesco Foscari, Maarten Grachten, and Gerhard Widmer. *Partitura - A Python Package for Symbolic Music Processing*, 2022. Available at: <https://arxiv.org/abs/2206.01071>.
- [8] *yt-dlp* — A feature-rich command-line audio/video downloader, fork of *youtube-dl* with additional features and fixes. <https://github.com/yt-dlp/yt-dlp>.
- [9] *Synthviz* — Python library for creating visualizations of piano-playing from MIDI files. <https://github.com/tstone/synthviz>.
- [10] W3Schools. *W3.CSS* — Modern CSS framework developed by the W3Schools team. <https://www.w3schools.com/w3css/>.

- [11] Knut Sveidqvist et al. *Mermaid* — JavaScript based diagramming and charting tool that renders Markdown-inspired text definitions to create and modify diagrams dynamically <https://mermaid.js.org/>.
- [12] Holistics Software. *dbdiagram.io* — A free, simple tool to draw ER diagrams by just writing code. <https://dbdiagram.io/home>.
- [13] Diego Marcos, Mozilla VR team *A-Frame* A web framework for building 3D/AR/VR experiences. <https://aframe.io/>.
- [14] *Spatial Design System* — A set of AR/VR primitives that simplify application development by providing detailed guidelines and ready-to-use components. <https://sds.spatialhub.cz/>.
- [15] *Polly Pizza*: A library of thousands of free low-poly 3D models. <https://poly.pizza/>.
- [16] *pitchy* — Simple pitch-detection library written entirely in JavaScript that aims to be fast and accurate enough to be used in real-time applications such as tuners. <https://www.npmjs.com/package/pitchy>.
- [17] *Piano Marvel* — Interactive piano learning software with real-time assessment of played notes. <https://pianomarvel.com>.
- [18] *ffmpeg* — Complete, cross-platform solution to record, convert and stream audio and video. <https://ffmpeg.org>.
- [19] *timidity* — Software synthesizer that can play MIDI files without a hardware synthesizer.
<https://timidity.sourceforge.net/#info>.