WIMU 2023Z

Dokumentacja końcowa

Łukasz Pokorzyński (300251) Olga Sapiechowska (302687) Michał Wiszenko (300285)

17 stycznia 2024

Temat projektu

Narzędzie pozwalające na wizualizację tokenizacji plików MIDI przez bibliotekę MidiTok

Opis funkcjonalności programu

Stworzone narzędzie będzie aplikacją internetową, która pozwoli na przegląd tokenów wygenerowanych poprzez bibliotekę MidiTok[1] na podstawie pliku formatu MIDI oraz wizalizację metryk, które mogą być wyczytane z tego formatu w celu dokładnej analizy pliku i danych się w nim znajdujących. Wśród tych metryk możemy znaleźć bardzo podstawowe takie jak klucz, metrum, tempo[2], a także bardziej zaawansowane związane z tonami oraz rytmiką utworu[3]. Decyzja, jakie dokładnie metryki zostaną zaimplementowane, zostanie podjęta na dalszym etapie rozwoju projektu.

Planowane funkcjonalności:

- wgranie pliku MIDI z urządzenia;
- wizualizacja tokenizacji pliku MIDI poprzez bibliotekę MidiTok;
- możliwość zmiany parametrów oraz sposobu tokenizacji poprzez bibliotekę MidiTok;
- wizualizacja metryk symbolicznych na podstawie pliku MIDI.

W razie wystarczających zasobów czasowych, chcielibyśmy zrealizować również następujące funkcjonalności:

- stworzenie pliku MIDI "na żywo"[4].

Nie gwarantujemy jednak, że zostaną one zaimplementowane i traktujemy je jako funkcjonalności w pełni opcjonalne.

Przewidywany stack technologiczny

- **Frontend**: TypeScript, React
- **Backend**: Python, FastAPI/Flask
- **Tokenizacja plików MIDI**: MidiTok
- **Hosting aplikacji**: Heroku
- **System kontroli wersji**: Git
- **Hosting repozytorium**: GitLab
- **Testy**: pytest, jest

Harmonogram prac

- 30.10 - 5.11: Przygotowanie repozytorium, zapoznanie się z formatem MIDI, dokumentacją narzędzi (np. MidiTok); Stworzenie podstawowego programu w Pythonie z MidiTok
- 6.11 - 12.11: Rozwój programu: wgrywanie pliku MIDI, przetwarzanie pliku, możliwość doboru typu tokenizacji
- 13.11 - 19.11: Poprawki do programu; Stworzenie minimalnego frontendu, wyświetlanie rezultatów w prostym formacie; podstawowa komunikacja z backendem
- 20.11 - 26.11: Poprawki frontendu; Przekazanie funkcjonalnego prototypu; Rozpoczęcie dalszego rozwoju
- 27.11 - 10.12: Implementacja wybranych metryk symbolicznych, finalizacja komunikacji między frontendem i backendem
- 11.12 - 24.12: Praca nad interfejsem graficznym aplikacji, wizualizacja zaimplementowanych metryk symbolicznych
- 25.12 - 31.12: Przerwa świąteczna - brak przewidzianych prac
- 1.01 - 7.01: Ostateczny rozwój interfejsu graficznego; Wprowadzanie poprawek
- 8.01 - 14.01: Oddanie projektu, możliwe poprawki w razie uwag
- 15.01 - 28.01: Bufor czasowy; (Ewentualne) Prace nad artykułem

Kolejne kroki są zaplanowane sztywno według tygodni, ale dopuszczamy możliwość szybszego rozpoczynania kolejnych etapów, jak i możliwość przedłużania zadań na kolejny tydzień, czego postaramy się unikać.

Zmiany względem dokumentacji wstępnej projektu

Niezmieniony stack technologiczny. Doprecyzowanie:

- backend z FastAPI,
- metryki z MusPy.

[1] https://miditok.readthedocs.io/en/v2.1.7/ [Dostęp zdalny 26.10.2023]
[2] https://craffel.github.io/pretty-midi/ [Dostęp zdalny 26.10.2023]
[3] https://salu133445.github.io/muspy/metrics.html [Dostęp zdalny 26.10.2023]
[4] https://developer.mozilla.org/en-US/docs/Web/API/Web_MIDI_API [Dostęp zdalny 26.10.2023]

Napotkane problemy

Tokenizery z MIDItok dają wyjście w czterech różnych formatach, przez co ich wyświetlanie różni się od podstawowego, zamierzonego efektu. Wizualizacja każdego typu wyjścia musiała zostać odpowiednio przemyślana.

Niezrealizowane możliwości rozbudowy

- stworzenie pliku MIDI "na żywo" (jedyną dostępną w tej chwili opcją jest dodanie pliku z urządzenia);
- rozszerzenie funkcjonalności aplikacji na wszystkie tokenizery z MIDItok;
- stworzenie kontekstu użytkownika.

Stopień wywiązania się z harmonogramu

Do grudnia prace postępowały zgodnie z harmonogramem. Ostatecznie implementacja metryk przesunęła się na styczeń, a finalizacja frontendu została przesunięta o tydzień. Oddanie również nastąpiło później niż planowano, lecz z niewielkim opóźnieniem.

Specyfikacja techniczna

Struktura aplikacji

Aplikacja została zaprojektowana zgodnie z wzorcem architektonicznym architektury wielowarstwowej. Występuje twardy rozdział na część frontendową oraz backendową.

```
|-- docker-compose.yaml
|-- README.md
|-- backend
    |-- poetry.lock
    |-- pyproject.toml
    |-- Dockerfile
    |-- core
        |-- main.py
        |-- constants.py
        |-- api
            |-- ...
        |-- data
            |-- ...
        |-- service
            |-- ...
    |-- tests
        |-- ...
|-- frontend
    |-- package.json
    |-- package-lock.json
    |-- tsconfig.json
    |-- Dockerfile
    |-- src
        |-- index.tsx
        |-- App.tsx
        |-- components
            |-- ...
        |-- interfaces
            |-- ...
    |-- public
        |-- ...
|-- node_modules
    |-- ...
```

Proces developerski

W celu zachowania zasad clean code, przed wrzuceniem commita na brancha, zaleca się wykonanie pre-commita. Aby uruchomić pre-commit, należy użyć komendy:

```
cd backend
pre-commit run --all-files
```

W skład skryptu pre-commit wchodzą:

- black (formatowanie)
- ruff (linting)
- isort (sortowanie importów)
- mypy (weryfikacja typowania)

Budowanie i uruchamianie aplikacji

Aplikacja frontendowa

Podstawowe uruchamianie aplikacji:

```
cd frontend
npm install
npm run dev
```

Uruchamianie aplikacji przy pomocy Dockera:

```
cd frontend
docker build . -t frontend
docker run frontend -p 3000:3000
```

Aplikacja backendowa

Podstawowe uruchamianie aplikacji:

```
cd backend
poetry shell
poetry install
python -m core.main
```

lub

```
poetry run python -m core.main
```

Uruchamianie aplikacji przy pomocy Dockera:

```
cd backend
DOCKER_BUILDKIT=1 docker build --target=runtime . -t backend
docker run backend -p 8000:8000
```

Docker Compose

Możliwe jest również uruchomienie całego projektu przy użyciu Docker Compose:

```
docker-compose up --build
```

Testowanie aplikacji

Aplikacja frontendowa

Testy jednostkowe uruchamiane są przy użyciu *jest*:

```
cd frontend
npm install
npm run test
```

Aplikacja backendowa

Testy jednostkowe uruchamiane są przy użyciu *pytest*:

```
poetry shell
poetry install
pytest
```

lub:

```
poetry run pytest
```

Logi

Zaimplementowano *middleware* na bazie *starlette*, który przy użyciu modułu *logging* tworzy logi dla każdego zapytania do serwera. Pojedynczy wpis w logach zawiera podstawowe dane dla pojedynczego zapytania oraz odpowiedzi serwera, jak również czas przetwarzania zapytania. Domyślnie logi zapisywane są do pliku *logfile.log*.

Deployment

Obie aplikacje są hostowane na Heroku. Aplikacja frontendowa jest dostępna pod adresem: https://wimu-frontend-ccb0bbc023d3.herokuapp.com

Repozytorium

- System kontroli wersji: Git
- Hosting repozytorium: GitLab
- Link do repozytorium: https://gitlab-stud.elka.pw.edu.pl/lpokorzy/wimu-miditokvisualizer