Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Kierunek: **INF**
Specjalność: **INS**

PRACA DYPLOMOWA INŻYNIERSKA

Projekt aplikacji webowej służącej do muzycznego samokształcenia

Jacek Kozicki

Opiekun pracy
Dr inż. Tomasz Kubik

Słowa kluczowe: REST API, teoria muzyki, Flask, Angular, Python, TypeScript

WROCŁAW 2022

Streszczenie

Ninejsza praca inżynierska to dokumnetcja projektowa aplikacji webowej opartej na REST API służącej do wspierania muzycznego samokształcenia. Aplikacja ma pomagać muzykom w opanowaniu teorii muzyki przy pomocy wirtualnych instrumentów oraz interaktywnych ćwiczeń. Praca skupia się przede wszystkim na stronie technicznej projektu. Pierwsze rozdziały służą omówieniu zastosowanych technologii, architektury i podjętych decyzji projektowych. Następnie omówiona jest implementacja systemu. Na końcu przedstawione są zrealizowane cele oraz perspektywy na rozwój aplikacji.

Słowa kluczowe: aplikacja internetowa, Angular, TypeScript, Python, Flask, REST API, teoria muzyki

Abstract

This bachelor thesis is a design document of a web application based on REST API. Application is designed to help musicians in developing their knowledge of music theory by using virtual instruments and interactive excercises. The document focuses on technical aspect of application rather than music theory. First chapters are focused on describing decisionmaking process used while making decisions about system architecture, choosing technologies and project design. Next chapters are used to present implementation details and final results. The document ends by describing future prospects for the app.

Keywords: web application, Angular, TypeScript, Flask, Python, REST API, music theory

Spis treści

Spis rysunków

Spis tabel

Spis listingów

Skróty

API (ang. *Application Programming Interface*)
REST (ang. *Representational State Transfer*)
HTML (ang. *HyperText Markup Language*)
URI (ang. *Uniform Resource Identifier*)
HTTP (ang. *Hypertext Transfer Protocol*)
JSON (ang. *JavaScript Object Notation*)

Rozdział 1

Wstęp

1.1. Wprowadzenie

Odkąd internet zaczął stawać się dobrem powszechnym, pojawia się w nim wiele materiałow edukacyjnch. Dostęp do darmowych lekcji i podobnych zasobów pozwala na kształcenie się w niemal dowolnym kierunku bez potrzeby wychodzenia z domu. Aktualne możliwości samorozwoju są więc bardzo duże, chociaż nie wszystkie materiały dostępne za darmo są wysokiej jakości, często nawet trafiają się w nich błędy. Niektóre z nich są nieprzyjazne dla użytkownika lub niekompletne.

Aplikacja stworzona w ramach tej pracy docelowo ma posłużyć jako darmowy zasób edukacyjny. Projekt skupiał się będzie w szczególności na gitarze basowej, ale będzie łatwy do rozbudowania o inny rodzaj gitary oraz kolejne instrumenty, takie jak np. pianino. Niniejsza praca porusza więc dwa istotne zagadnienia: teorię muzyki oraz projektowanie aplikacji webowych.

Teoria muzyki jest często pomijana przez muzyków amatorów, którzy edukują sie sami - oprócz tego, że same materiały mogą być słabej jakości, trudno jest zacząć przekładać teorię na praktykę. Na przeciw tym temu problemowi starają sie wyjść aplikacje takie jak Rocksmitch czy musicca, które są inspiracją dla niniejszej pracy. Rocksmith to gra pozwalająca na granie piosenek, do których tabulatura jest wyświetlana na ekranie w czasie rzeczywistym. Jako kontreler wykorzystuje się gitarę elektryczną lub basową ze specjalnym złączem. Oprócz tego, Rocksmith posiada również wiele przydatnych narzędzi, m.in. stroik czy efekty modyfikujące brzmienie gitary. Istotnymi wadami są natomiast: cena, wymaganie specjalnego złącza oraz duże obciążenie komputera. Aplikacja musicca jest duża bliższą inspiracją - posiada interaktywny gryf i możliwość wyszukiwania akordów czy interwałów. W tej aplikacji za minus można uznać interfejs użytkownika - przejście z wyszukiwania interwałów do akordów wymaga zmiany widoku, a w kolejnym widoku trzeba ponownie wybrać instrument.

Aplikacja stworzona w ramach tej pracy to próba stworzenia przyjaznego użytkownikowi środowiska do nauki teorii muzyki, która wolna będzie od wyżej wymienionych problemów.

1.2. Cel i zakres pracy

1.2.1. Cel pracy

Celem pracy jest stworzenie aplikacji, która wspomaga muzyków w nauce teorii muzyki poprzez wizualizację oraz interaktywne ćwiczenia. Aplikacja powinna być łatwo rozszerzalna o kolejne aspekty teoretyczne oraz dodatkowe rodzaje ćwiczeń.

1.2.2. Zakres pracy

Zakres pracy obejmuje:

- wybór technologii,
- zaprojektowanie aplikacji webowej,
- implementację aplikacji,
- projekt i implementację testów aplikacji,
- udokumentowanie procesu w pracy dyplomowej.

1.3. Układ pracy

Praca została podzielona na rozdziały opisujące kolejne etapy tworzenia aplikacji:

- wybór technologii - analiza dostępnych rozwiązań i uzasadnienie podjętego wyboru,
- analiza wymagań - przegląd wymagań funkcjonalnych i niefunkcjonalnych stawianych wobec projektowanej aplikacji oraz rozważanie przypadków użycia, które powinna ona spełniać,
- projekt aplikacji - opis architektury systemu,
- implementacja aplikacji - omówienie implementacji poszczególnych fragmentów systemu wraz z fragmentami kodu, prezentacja finalnych efektów,
- podsumowanie - omówienie spełnionych celów pracy oraz perspektyw na dalszy rozwój aplikacji.

Rozdział 2

Użyte technologie

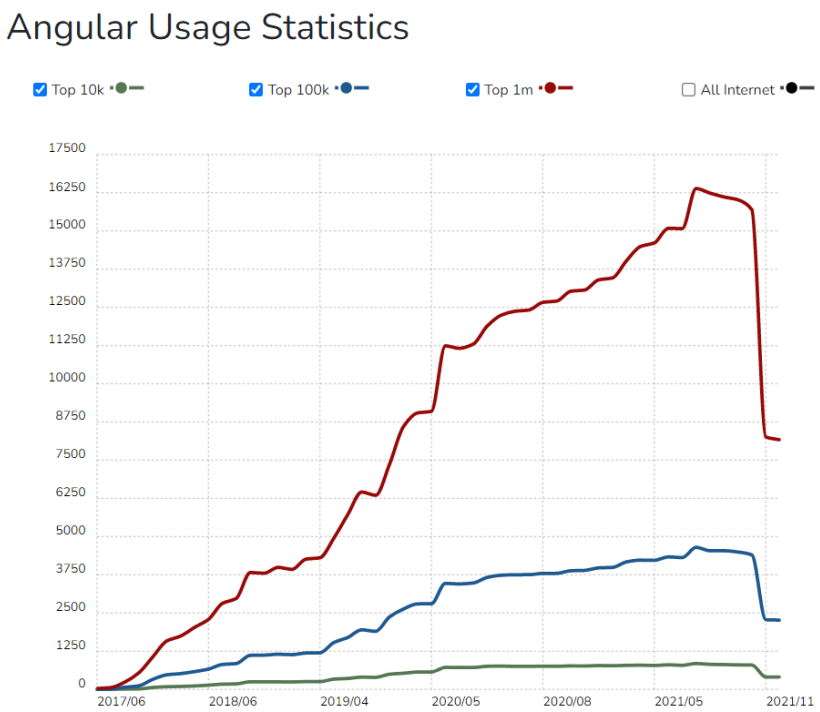
Na wybór technologii wpływ miały następujące czynniki:

- możliwość implementacji założonych funkcjonalności - technologia musi być dobrana odpowiednio do zadania,
- popularność technologii - wpływa na dostępność materiałów oraz pokazuje czy dana znajomość danej technologii ma szanse przydać się na rynku pracy,
- preferencja personalna - ostateczna decyzja zależała również od tego, na ile dobrze autor znał proponowane rozwiązania.

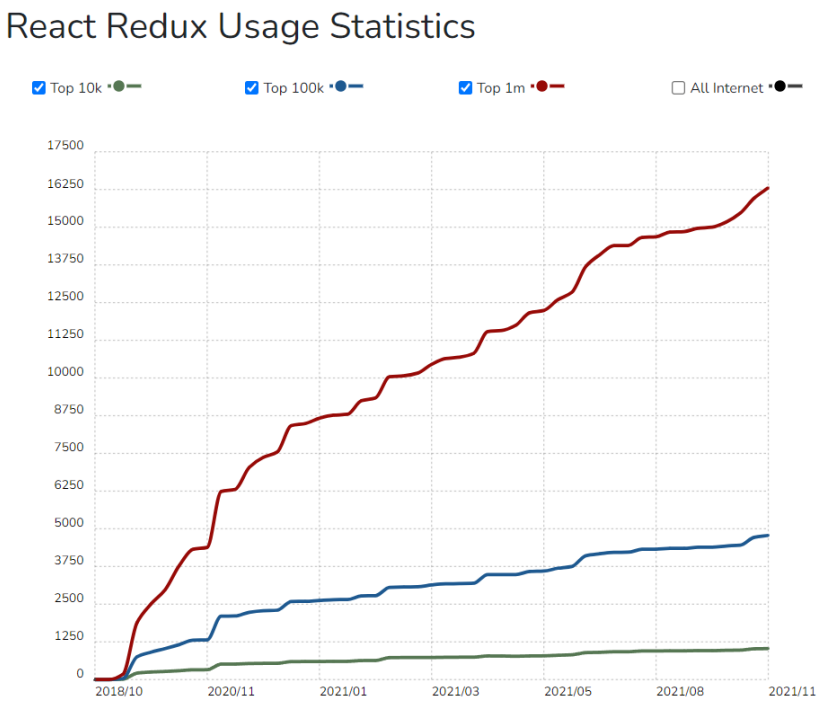
2.1. Technologie frontendowe

2.1.1. Wybór technologii

Podczas wyboru technologi frontendowej rozważane były przede wszystkim frameworki React oraz Angular.
Wg trends.builtwith - strony analizującej trendy technologiczne w branży popularność obu frameworków jest zbliżona - na dzień 26.01.2022 295,780 stron internetowych zbudowanych jest z wykorzystaniem Angulara[2], a z wykorzystaniem Reacta 309,581[6]. Wykresy 2.1 oraz 2.2 przedstawiają popularność wspomnianych bibliotek w okresach 06.2017-10.2021 oraz 10.2018-10.2021 na podstawie liczby stron korzystających z danego rozwiązania spośród miliona, 100 tysięcy oraz 10 tysięcy stron generujących największy ruch.



Rys. 2.1: Statystyki wykorzystania frameworka Angular na stronach generujących największy ruch[2]



Rys. 2.2: Statystyki wykorzystania frameworka React na stronach generujących największy ruch[6]

	Angular	React
Liczba działających stron	139,795	265,454
Procent spośród miliona stron generujących największy ruch	1.94%	2.06%
Procent spośród 100tys. stron generujących największy ruch	5.39%	5.98%
Procent spośród 10tys. stron generujących największy ruch	9.67%	12.35%

Tab. 2.1: Porównanie statystyk użycia Angulara i Reacta

Na podstawie wykresów można łatwo zauważyć spadającą popularność angulara, należy jednak pamiętać, że decyzja odnośnie wyboru technologii, podejmowana była na początkowym etapie pracy, a więc tendencja spadkowa nie była wtedy jeszcze tak widoczna. Przed tak znaczącym spadkiem popularności Angulara obie technologie były wybierane z podobną częstotliwością, więc ostateczna decyzja sprowadziła się do preferencji autora - w ramach zajęć na uczelni oraz prywantych projektów miał on styczność z obydwoma bibliotekami, ale to z Angularem miał większe doświadczenie, w związku z czym wybrany został właśnie ten framework.

2.1.2. Opis wybranej technologii

Angular to kompleksowy framework do projektowania oraz tworzenia wydajnych aplikacji typu SPA (Single Page Application)[1]. Podstawowymi elementami składowymi Frameworka Angular są komponenty Angular, które są zorganizowane w NgModules. NgModules zbierają powiązany kod w zestawy funkcjonalne; aplikacja Angular jest definiowana przez zestaw NgModules. Aplikacja zawsze ma co najmniej moduł główny, który umożliwia ładowanie początkowe, i zazwyczaj ma o wiele więcej modułów funkcji.
Podczas implementacji korzystano głównie z komponentów i serwisów. Komponenty definiują widoki, które są zestawami elementów ekranu, które Angular może wybierać i modyfikować zgodnie z logiką programu i danymi. Serwis w Angularze jest zwykłą klasą TypeScript. Opatrzony jest dodatkowym dekoratorem, który może mieć różne właściwości konfiguracyjne. Serwis może działać jako singleton dla całej aplikacji lub może zostać powołany dla konkretnego komponentu.

2.2. technologie backendowe

2.2.1. Wybór technologii

Podczas wyboru technologii backendowej głównymi kandydatami były Python[5] z frameworkiem Flask[3] oraz Java[4] z frameworkiem Spring[7].

Tutaj głównym kryterium okazal sie rozmiar aplikacji - w planowanej implementacji backend jest stosunkowo niewielki, wiekszosc implementacji lezy po stronie frontendu, natomiast Flask jest lepiej przystosowany do tworzenia niewielkich aplikacji niz Spring.

2.2.2. Opis wybranej technologii

Flask określany jest mianem micro framework, co oznacza, że nie wymaga żadnych dodatkowych narzędzi ani bibliotek. Służy on do budowania aplikacji sieciowych i charakteryzuje się brakiem wbudowanych komponentów do obsługi bazy danych czy walidacji formularzy, ale zamiast tego jest on skonstruowany w taki sposób, aby był łatwo rozszerzalny o wszystkie potrzebne funkcjonalności.
Wśród zalet flaska wymieniane są przede wszystkim:

- pytoniczność kodu - kod pisany w tym frameworku pozwala na stosowanie dobrych praktyk programowania charakterystynczych dla języka Python,
- łatwość nauki - z uwagi na to, że kod bardzo przypomina ten pisany w czystym Pythonie a sam framework posiada niewiele komponentów, jest on uznawany za łatwy do przyswojenia.

2.3. Wersjonowanie oraz pozostałe narzędzia

2.3.1. Wersjonowanie

Do wersjonowania aplikacji wykorzystano system kontroli wersji GIT. Zdalne repozytorium zostało założone na platformie github. Podczas pracy z GITem korzystano przede wszystkim z konsoli git bash oraz interfejsów graficznych udostępnianych przez używane środowiska programistyczne.

2.3.2. Środowiska programistyczne

Do pracy z kodem pisanym w języku Python wykorzystano środowisko PyCharm - jest ono bardzo intuicyjne, wspiera formatowanie kodu według standardu PEP8, a oprócz tego posiada ono wbudowane wsparcie dla projektów opartych o framework Flask.
Do pracy z frameworkiem Angular wykorzystano edytor kodu Visual Studio Code. Edytor nie posiada wbudowanych narzędzi do pracy w tej technologii, ale jest otwarty na rozszerzenia, a z uwagi na swoją popularność, jest ich dużo i są dobrej jakości. W ramach projektu przydały się przede wszystkim rozszerzenia: SonarLint i Auto Rename Tag.

Rozdział 3

Analiza wymagań

3.1. Wymagania funkcjonalne

Dla implementowanej aplikacji zdefiniowano następujące wymagania funkcjonalne:

- Użytkownik może sprawdzić jakie dźwięki należa do wybranych skal,
- Użytkownik może sprawdzić jakie dźwięki tworzą dany interwał,
- Użytkownik może sprawdzić jakie dźwięki tworzą dany akord,
- Użytkownik może wchodzić w interakcję z wirtualnym gryfem gitary, aby odtworzyć dźwięki,
- Użytkownik może sprawdzić swoją wiedzę teoretyczną w interaktywnych ćwiczeniach,
- Użytkownik może trenować ucho muzyczne za pomocą specjalnych ćwiczeń.

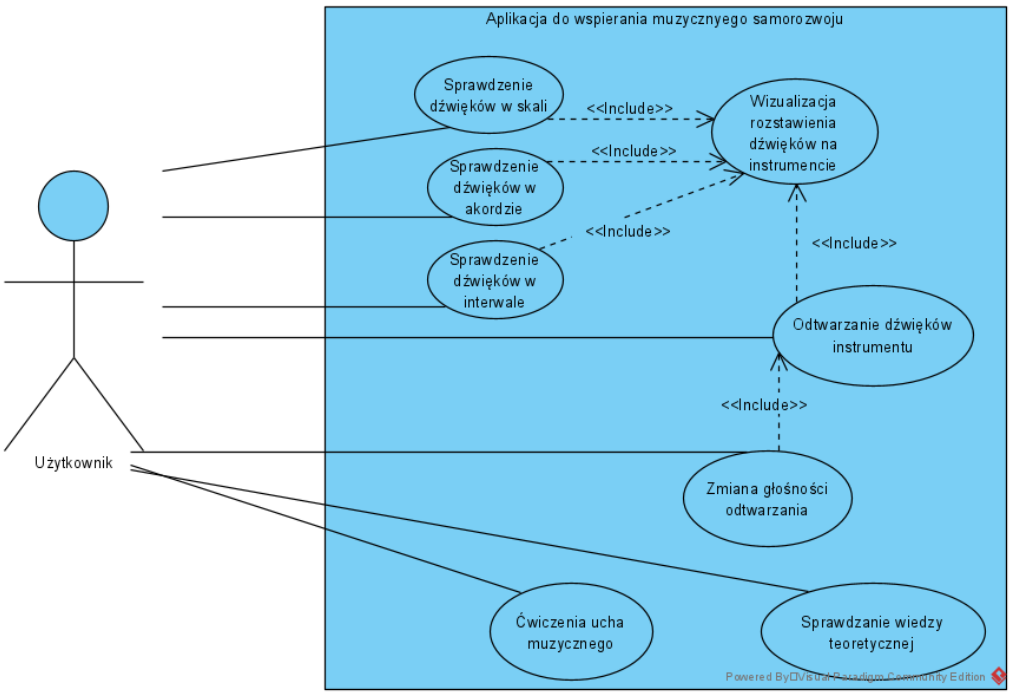
3.2. Wymagania niefunkcjonalne

Dla implementowanej aplikacji zdefiniowano następujące wymagania niefunkcjonalne:

- Aplikacja powinna działać na najpopularniejszych przeglądarkach internetowych,
- Aplikacja powinna być responsywna,
- Aplikacja nie wymaga od użytkownika zakładania konta,
- Pierwsze załadowanie aplikacji nie powinno trwać dłużej 2 sekundy,
- Aplikacja powinna być wykonana przy użyciu bibliotek Angular oraz Flask,
- Aplikacja powinna być zbudowana w taki sposób, aby jej dalszy rozwój był jak najprostszy,
- Aplikacja powinna posiadać intuicyjny interfejs użytkownika

3.3. Omówienie przypadków użycia

3.3.1. Diagram przypadków użycia



Rys. 3.1: Diagram przypadków użycia aplikacji

3.3.2. Scenariusze wybranych przypadków użycia

Poniżej opisano Scenariusze dla następujących przypadków użycia: ćwiczenie ucha muzycznego, sprawdzanie dźwięków w skali, odtwarzanie dźwięków instrumentu, zmiana głośności odtwarzania.

Ćwiczenie ucha muzycznego

1. Za pomocą paska nawigacji użytkownik wchodzi na podstronę z ćwiczeniem ucha muzycznego.
2. Użytkownik widzi swój aktualny wynik oraz guziki służące do obsługi ćwiczenia.
3. Po wciśnięciu przycisku, użytkownik słyszy wylosowany dźwięk. Guzik może być wciśnięty wielokrotnie w celu ponownego odsłuchania dźwięku.
4. Użytkownik wybiera odpowiedź z listy lub klika guzik reset.
   4.1. Po wybraniu poprawnej odpowiedzi losowany jest nowy dźwięk, a wynik zwiększa się o 1pkt.
   4.2. Po wybraniu zlej odpowiedzi lub wciśnięciu guziku reset, losowany jest nowy dźwięk, a wynik ustawiany jest na 0pkt.

Sprawdzanie dźwięków w skali

1. Za pomocą paska nawigacji użytkownik wchodzi na podstronę z wybranym instrumentem.
2. Użytkownik widzi wirtualny instrument oraz selektor zapytań.
3. Użytkownik wybiera interesujące go zapytanie i zatwierdza to guzikiem.
4. Na wirtualnym instrumencie zaznaczają się odpowiednie dźwięki.

Odtwarzanie dźwięków instrumentu

1. Za pomocą paska nawigacji użytkownik wchodzi na podstronę z wybranym instrumentem.
2. Użytkownik widzi wirtualny instrument oraz selektor zapytań.
3. Użytkownik naciska na guzik odpowiadający wybranemu dźwiękowi na instrumencie.
4. Wybrany przez użytkownika dźwięk jest odtwarzany.

Zmiana głośności odtwarzania

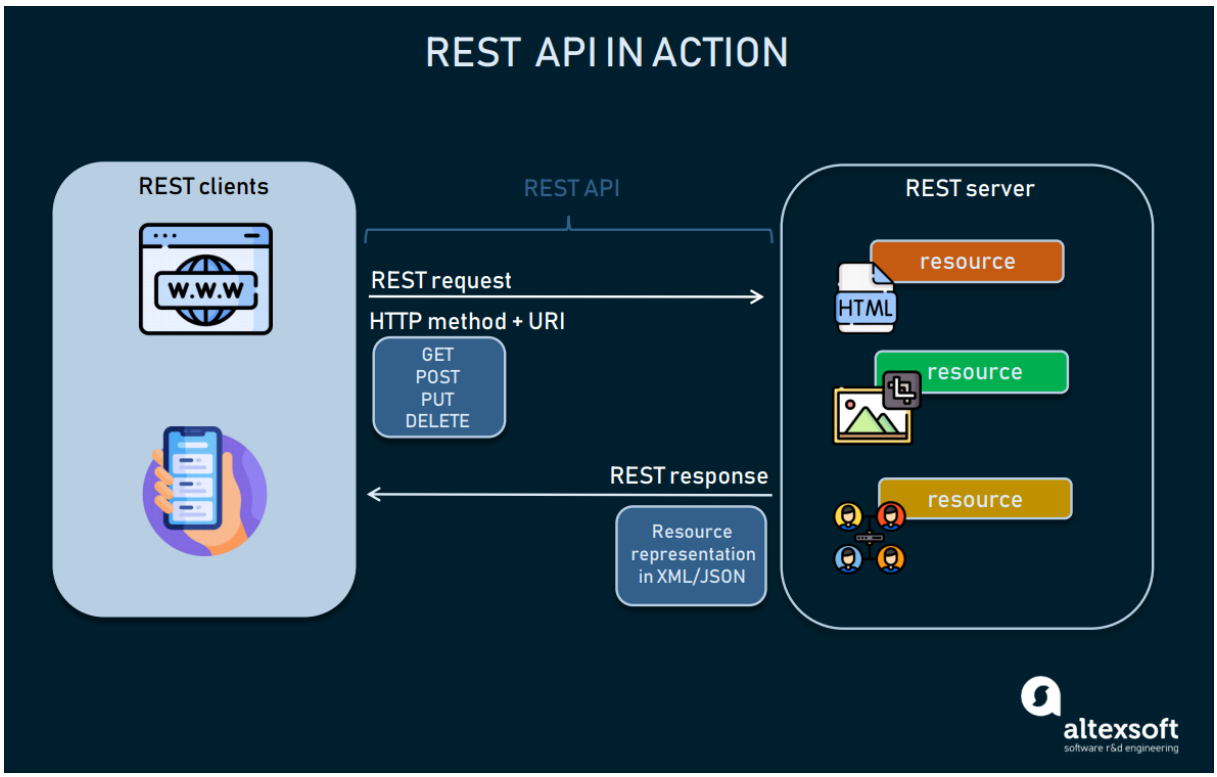
1. Za pomocą paska nawigacji użytkownik wchodzi na podstronę z wybranym instrumentem.
2. Użytkownik widzi wirtualny instrument oraz selektor zapytań.
3. Użytkownik odtwarza dowolny dźwięk poprzez kliknięcie na wirtualny instrument.
4. Użytkownik zmienia głośność instrumentu korzystając z suwaka regulacji głośności.
5. Użytkownik ponownie odtwarza dowolny dźwięk poprzez kliknięcie na wirtualny instrument - głośność jest dostosowana do jego oczekiwań.

Rozdział 4

Projekt aplikacji

4.1. Architektura systemu

Aplikacja bazuje na REST API - komunikacja pomiędzy aplikacją frontendową i backendową odbywa się w sposób przedstawiony na rysunku 4.1. Aplikacja kliencka wysyła zapytania HTTP na odpowiednie URI, a w odpowiedzi otrzymuje dane w formacie JSON. Typowy sposób tworze-



Rys. 4.1: Architektura komunikacji w systemie opartym na REST API[8]

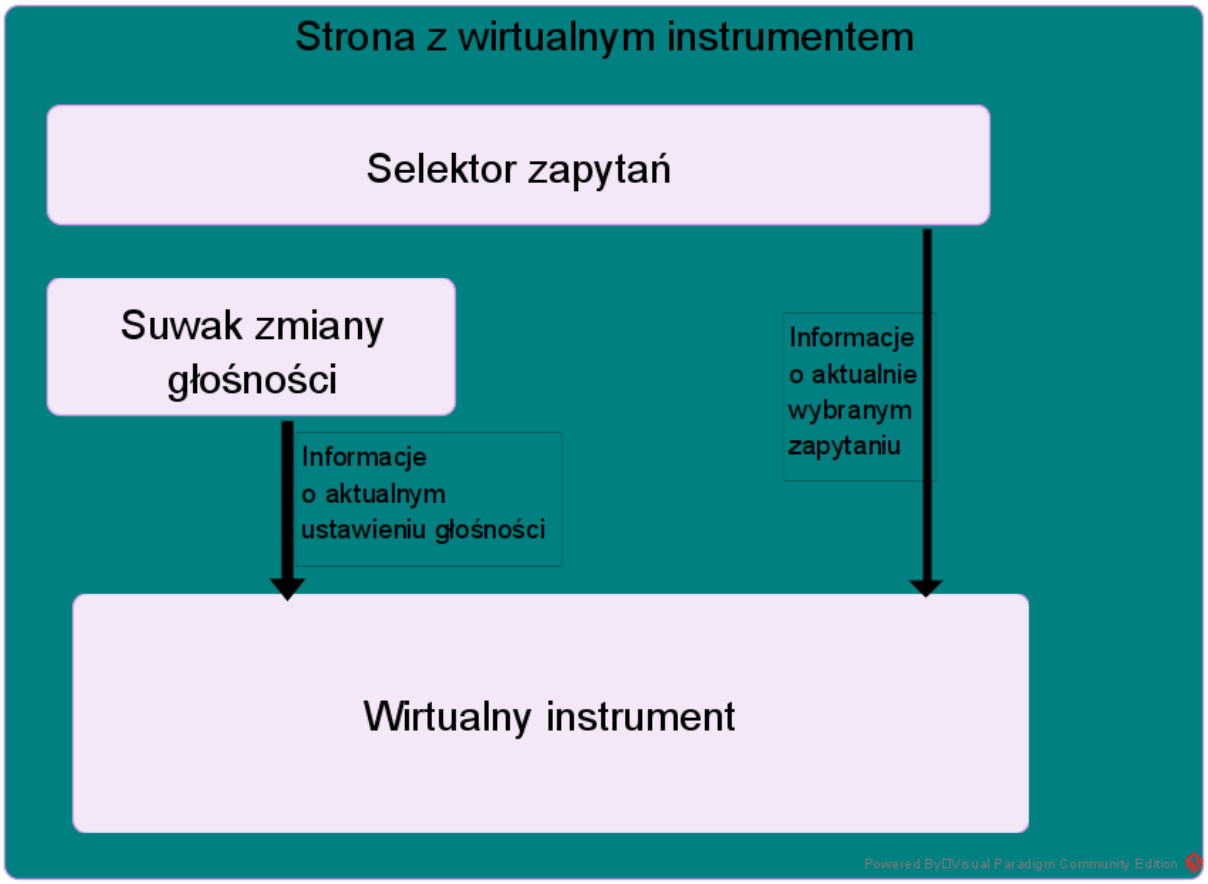
nia endpointów w REST API to nazywanie endpointów nazwami kolekcji, a następnie uszczegóławianie zapytania w kolejnych częściach adresu[9]. Przykładowe endpointy w REST API powinny wyglądać następująco:

API/people - endpoint zwracający zawartość kolekcji “people”
API/people/:id - endpoint zwracający element kolekcji “people” o zadanym id
API/pepole/:id/name - endpoint zwracający wartość pola “name” elemntu kolekcji “people” o zadanym id

Rozwiązanie REST API pozwala na łatwe przesyłanie informacji o potrzebnych obiektach, takich jak skale czy akordy - JSON nie ogranicza wielkości wiadomości czy liczby elementów, więc takie obiekty świetnie nadają się do przekazywania tablic o nieznanych z góry rozmiarach. REST API pozwala również w prosty sposób zaimplementować mechanizmy obsługi błędów poprzez dołączanie do przekazywanego obiektu statusów HTTP, które opisują efekt zapytania czy nawet dodatkowej wiadomości opisującej błąd w dokładniejszy sposób.
Kolejną zaletą stosowania tego podejścia jest jego skalowalność. Każda funkcjonalność jest przypisana pod osobny adres, więc łatwo dodawać nowe poprzez tworzenie kolejnych endpointów.

4.2. Architektura aplikacji klienta

Korzystając z możliwości oferowanych przez framework Angular zdecydowano się rozbić aplikację na mniejsze elementy - charakterystyczne dla tej biblioteki komponenty oraz serwisy. Każda podstrona składa się z odpowiednich, połączonych ze sobą komponentów. Takie podejście pozwala na wielokrotne używanie tego samego kodu - ten sam komponent można wykorzystać w kilku miejsach na stronie lub na wielu różnych podstronach. Pozwala ono również na generowanie wielu podobnych podstron poprzez zamianę pojedynczych komponentów.
Diagram 4.2 przedstawia schemat jednej z podstron w aplikacji - podstrony zawierającej wirtu-



Rys. 4.2: Diagram podstrony z wirtualnym instrumentem z wyróżnieniem komponentów i zależności między nimi

alny instrument. Jak widać sam wirtualny instrument jest osobnym komponentem, co pozwala

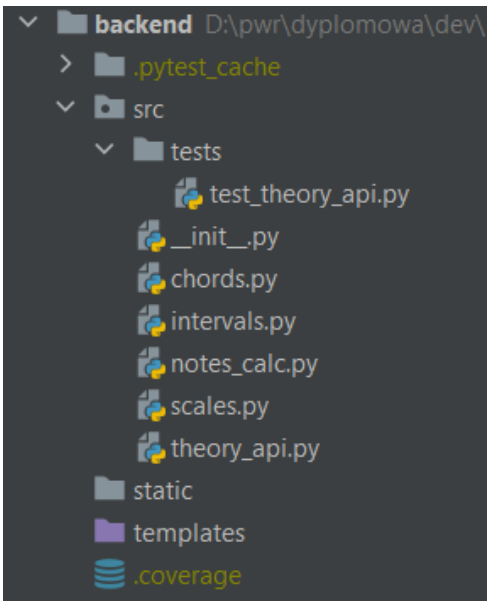
na stworzenie części aplikacji dedykowanej dla gitary akustycznej oraz części aplikacji dedykowanej dla gitary basowej, a potencjalnie również dla dowolnych innych instrumentów. W tym celu należy jedynie zmienić komponent implementujący instrument.
Warto również zwrócić uwagę, na to, że oprócz widocznych na diagramie mniejszych komponentów, pojedynczy komponent może zawierać w sobie serwisy. Pozwalają one m.in. na pobieranie danych z API czy dostęp do zasobów lokalnych serwera.

Rozdział 5

Implementacja aplikacji

5.1. Implementacja Backendu

Schemat projektu aplikacji backendowej przedstawiono na rysunku 5.1. Najważniejszymi modułami są pliki notes_calc.py i theory_api.py - odpowiadają one kolejno za obliczanie dźwięków w danych akordach, skalach czy interwałach oraz za tworzenie endpointów aplikacji.
Pliki chords.py, scales.py, intervals.py to pliki pomocnicze zawierające definicje wzorców wykorzystowanych przez notes_calc oraz listy wszystkich dostępnych obiektów.
Pliki testów jednostkowych znajdują się w katalogu tests.



Rys. 5.1: Schemat projektu aplikacji Frontendowej

5.1.1. Endpointy

Endpointy deklarowane są w sposób typowy dla frameworka Flask, czyli poprzez dekoratory funkcji opisujące ścieżkę oraz dostępne metody. Wszystkie endpointy zawarte w API zwracają obiekty JSON. Elementy ścieżki zapisane jako “<name>” to parametry ścieżki - zmienne, które są poźniej wykorzystywane w celu uszczegóławiania zapytania. W głównym endpoincie API, przedstawionym na listnigu 5.1, ścieżka opisana jest trzema parametrami, z których każdy coraz bardziej zawęża zapyanie. Początkowo obszar poszukiwania odpowiedzi jest określany przez

obj_type - wybierana jest jedna 3 kategorii: skala, interwał lub akord. Następnie podawana jest nuta główna, a na końcu nazwa obiektu.

Listing 5.1: Główny endpoint stworzonego API

```
@theory_api.route('/<obj_type>/<root>/<name>', methods=['GET'])
def get_obj(root: str, name: str, obj_type: str) -> flask.Response:
    # get root from link, check if it's a legal note and cast to
      ↪ proper format
    print(f"root:{root}; name:{name}; obj_type:{obj_type}")
    root = cast_root(root)
    if root is None:
        return wrong_request("Incorrect root note")

    # try to find proper object, if there's no matching obj type -
      ↪ wrong request
    obj = None
    if obj_type == 'chord':
        obj = find_chord(root, name)
    elif obj_type == 'scale':
        obj = find_scale(root, name)
    elif obj_type == 'interval':
        obj = find_interval(root, name)
    else:
        return wrong_request("Incorrect obj_type in request")

    # obj type was correct, but interval wasn't found - wrong
      ↪ interval in request
    if obj is None:
        return wrong_request(f"Incorrect {obj_type} name")

    # all went smooth - return requested sound
    return jsonify(obj)
```

5.1.2. Obliczanie dźwięków

W celu obliczania dźwięków wykorzystano następujące założenia:

- każdy dźwięk ma wartość liczbową od 0 do 11, zaczynając od dźwięku 'E',
- przez dźwięk rozumie się również półtony,
- oktawy to wielkości okresowe (po ostatnim dźwięku w oktawie następuje powrót do pierwszego),
- odległość między półtonami wynosi 1 jednostkę.

Korzystając z takich założeń wszystkie akordy, skale i interwały opisano za pomocą odległości od poprzeniego dźwięku. Pierwszy dźwięk to zawsze nuta główna, drugi dźwięk opisany jest odległością od nuty głównej, trzeci dźwięk odległością od drugiego dźwięku itd.

Listing 5.2: Wzorce opisujące przykładowe akordy

```
seventh_pattern = (4, 3, 3)
minor_seventh_pattern = (3, 4, 3)
major_seventh_pattern = (4, 3, 4)
diminished_seventh_pattern = (3, 3, 3)
```

Na podstawie opisanych założeń oraz wzorców stworzono funkcje obliczające dźwięki zawarte w skalach, akordach i interwałach na podstawie ich nazwy oraz nuty głównej. Na listingach 5.3, 5.4 przedstawiono funkcje dla akordów oraz skal.

Listing 5.3: Funkcja obliczająca dźwięki w skali

```
def find_scale(root: str, scale_name: str) -> list:
    curr_sound = notes_mapping.get(root)
    pattern = scales_patterns.get(scale_name)
    if pattern is None:
        return None
    scale = [root]
    for step in pattern:
        curr_sound += step
        sound = sound_from_val(curr_sound % 12)
        scale.append(sound)
    return scale
```

Listing 5.4: Funkcja obliczająca dźwięki w akordzie

```
def find_chord(root: str, chord_name: str) -> list:
    curr_sound = notes_mapping.get(root)
    pattern = chords_patterns.get(chord_name)
    if pattern is None:
        return None
    chord = [root]
    for step in pattern:
        curr_sound += step
        sound = sound_from_val(curr_sound % 12)
        chord.append(sound)
    return chord
```

5.2. Obserwacje i uwagi

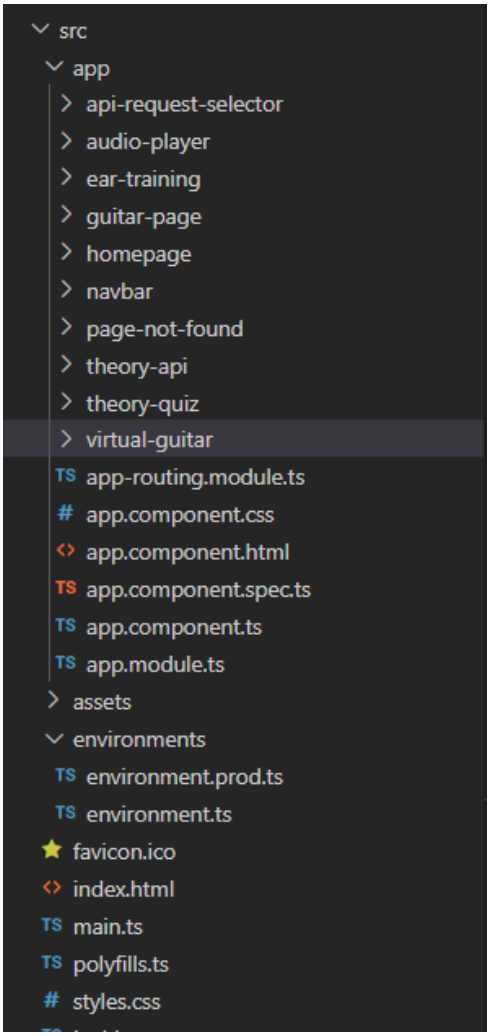
Po zaimplementowaniu całości aplikacji, zauważono, że obliczenia są na tyle niewymagające, że mogłyby być wykonywane po stronie frontendu zamiast dodatkowego API. Warto jednak zauważyć, że API można łatwo rozwinąć o dodatkowe funkcjonalności, takie jak np. połączenie z bazą danych, co mogłoby okazać się przydatne w wypadku rozwoju aplikacji o konta użytkowników.

5.3. Implementacja Frontendu

Schemat projektu aplikacji frontendowej jest typowy dla aplikacji implementowanych w Angularze. Program podzielony jest na fragmenty zwane komponentami oraz serwisami. Każdy komponent oraz każdy serwis jest przechowywany w osobnym folderze. Katalog assests zawiera zasoby wykorzystywane w aplikacji, takie jak dźwięki czy obrazki. Oprócz komponentów stworzonych przez użytkownika wykorzystywane są również pliki wygenerowane za pomocą Angular CLI znajdujące się w katalogu głównym:

app-routing.module.ts - moduł pozwalający na zarządzanie routingiem wewnątrz projektu
app.module.ts - główny moduł aplikacji opisujący elementy importowane przez aplikację oraz deklarowane w niej komponenty i serwisy
app.* - pozostałe pliki kompenentu głównego
environment.prod.ts - plik zawierające zmienne środowiskowe dla środowiska produkcyjnego

environment.ts - zmienne środowiskowe do wykorzystania podczas tworzenia i testowania aplikacji

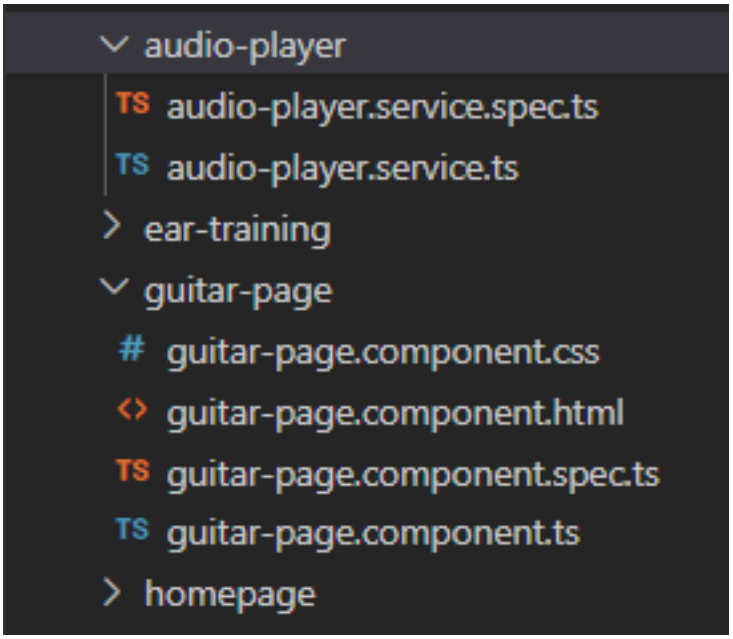


Rys. 5.2: Schemat projektu aplikacji Frontendowej

Każdy komponent składa się z 4 plików:

component-name.css - plik opisujący style wykorzystane w obrębie tego komponentu,
component-name.html - plik opisujący rozkład elementów w danym komponencie. Angular pozwala na wykorzystywanie tam selektorów innych komponentów oraz dodatkowych dyrektyw, takich jak np. ngIf czy ngFor,
component-name.ts - plik zawierający konstruktor komponentu oraz skrypty napisane przez użytkownika dla danego komponentu,
component-name.spec.ts - plik zawierający testy jednostkowe komponentu.

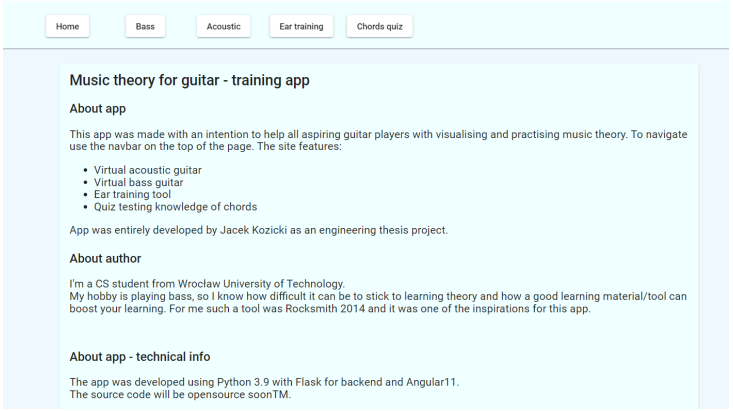
Serwisy natomiast składają się domyślnie z 2 plików: service-name.ts oraz service-name.spec.ts. Pełnią one takie same role jak w wypadku komponentów - pierwszy z nich zawiera klasę serwisu, natomiast drugi jej testy jednostkowe.



Rys. 5.3: Katalogi audio-player oraz guitar-page pokazują schemat plików odpowiednio serwisu i komponentu

5.3.1. Strona główna

Na stronie głownej wyświetlony jest panel z krótkim tekstem o aplikacji oraz jej autorze.
Na górze strony widoczny jest pasek nawigacji, który pozwala na przejście do wszystkich funkcjonalności aplikacji.
To na tę stronę przekierowywany jest użytkownik jeśli próbuje dostać sie na nieistniejącą podstronę.



Rys. 5.4: Katalogi audio-player oraz guitar-page pokazują schemat plików odpowiednio serwisu i komponentu

5.3.2. Nawigacja

Do nawigacji podczas korzystania z aplikacji służy pasek nawigacji umieszczony na górze strony. Każdy z guzików jest odnośnikiem do odpowiednich podstron. Próba przejścia na podstrony, które nie są wymienione na pasku nawigacji skutkuje przekierowaniem stronę główną.

Implementacja nawigacji opiera się na wykorzystaniu modułu routing zawaratego w Angularze. W pliku app-routing.module.ts zdefiniowano wszystkie ścieżki obsługiwane przez aplikację. Można je zobaczyć na listingu 5.5. szczególnie warto zwrócić uwagę na ostatni wpis w tablicy. Wpis path: '**' to tzw. dzika karta (ang. wildcard route) - ścieżka, która jest wykorzystywana, kiedy użytkownik próbuje dostać się na dowolną podstronę, która nie została opisana w tablicy przed wystąpieniem tego wpisu. Ze względu na sposób działania routingu, ważne, aby dzika karta była na końcu tablicy: podczas szukania ścieżki porównywana jest podana ścieżka i kolejne wartości pola "path" w tablicy Routes. Porównanie dowolnej ścieżki z symbolem '**' zawsze zwraca wartość prawda, więc umieszczenie tego symbolu jako wcześniejszego wpisu, zablokowałoby dostęp do ścieżek zadeklarowanych później.

Listing 5.5: Ścieżki zdefiniowane w aplikacji

```
const routes: Routes = [
 {path: 'test', component: VirtualGuitarComponent, pathMatch: 'full'
    ↪ },
 {path: 'api', component: ApiRequestSelectorComponent, pathMatch: '
    ↪ full'},
 {path: 'bass', component: GuitarPageComponent, pathMatch: 'full'},
 {path: 'acoustic', component: GuitarPageComponent, pathMatch: 'full
    ↪ '},
 {path: 'earTraining', component: EarTrainingComponent, pathMatch: '
    ↪ full'},
 {path: 'practice', component: TheoryQuizComponent, pathMatch: 'full
    ↪ '},
 {path: '**', component: HomepageComponent}
];
```

5.3.3. Komunikacja z API

W celu komunikacji z API zaimplementowano serwis TheoryApiService oraz modele danych odpowiadające tym z API, czyli klasy: Chord, Interval oraz Scale. Klasę Scale przedstawiono na listingu 5.6. Serwis służy do wysyłania zapytań HTTP na zadany adres i rzutuje otrzymane dane na rządany typ.

Listing 5.6: Klasa Scale wykorzystywana do zapytań i przechowywania danych o skalach

```
export class Scale {
    constructor(
        public name: string,
        public notes: string[],
    ) { }
}
```

TheoryApiService wykorzystuje zmienną środowiskową API_URL pod którą przechowywany jest adres API, dzięki czemu można modyfikować adres API bez ingerencji w kod. Zastosowanie zmiennych środowiskowych jest szczególnie istotne kiedy dochodzi do dystrybucji aplikacji - adresy API mogą się zmieniać i nie powinno to powodować potrzeby aktualizacji aplikacji. Przykładowe zapytanie do API z wykorzystaniem zmiennej środowiskowej można zobaczyć na listingu 5.7.

Listing 5.7: Metoda klasy TheoryApiService wysyłająca zapytanie o dźwięki akordu do API

```
getChord(rootNote: string, name: string): Observable<Chord[]> {
```

```
        return this.http
        .get<Chord[]>(`${API_URL}/chord/${rootNote}/${name}`).pipe(
            tap(_ => console.log('fetched')),
            catchError(TheoryApiService._handleError)
        );
    }
```

Takie rozwiązanie pozwala na wykorzystywanie tego samego serwisu w wielu komponentach poprzez wstrzykiwanie zależności. W Angularze proces ten odbywa się poprzez oznaczenie klasy serwisu dekoratorem Injectable oraz zadeklarowanie odpowiedniej zależności w konstruktorze komponentu (serwis przekazuje się jako argument konstruktora i przypisuje do odpowiedniego pola w klasie kompenentu). Przykład wstrzykiwania zależności można zobaczyć na listingu 5.8, do kompenentu są tam wstrzykiwane dwie zależności: TheoryApiService oraz ActivatedRoute.

Listing 5.8: Wstrzykiwanie zależności do komponentu GuitarPageComponent

```
constructor(api: TheoryApiService, route: ActivatedRoute) {
    this.theoryApi = api;
    this.route = route;
}
```

5.3.4. Podstrony z wirtualnymi instrumentami

Podstrony z wirtualnymi instrumentami zostały stworzone poprzez połączenie ze sobą mniejszych komponentów: selektora zapytań, suwaka regulującego głośność oraz wirtualnego instrumentu.

Rys. 5.5: Widok strony z gitarą basową przed wykonaniem zapytania

W aplikacji zostały zaimplementowane dwa instrumenty: gitarę akustyczną oraz gitarę basową. Bazują one na tym samym komponencie, ponieważ na poziomie abstrakcji, na którym operowano podczas implementacji róznica między nimi to tylko liczba strun.

Na każdym z instrumentów po wybraniu zapytania podświetlają się odpowiednie dźwięki. Wynik przykładowego zapytania można zobaczyć na rysunkach 5.6, 5.7 i 5.8. Wybrano zapytanie o skalę F Natural Major (F-dur). W skład tej skali wchodzą dźwięki: F,G,A,A#,C,D,E i te dźwięki są zaznaczone kolorem czerwonym.
Jeśli użytkownika interesują tylko dźwięki należące do wybranego zapytania, może on wyłączyć wyświetlanie pozostałych dźwięków przyciskając guzik “show notes on hover” - powoduje on ukrycie wszystkich dźwięków, które nie należą do aktualnego zapytania i wyświetlanie ich dopiero po najechaniu kursorem.

Kliknięcie na dowolny dźwięk powoduje odtworzenie jego próbki dźwiękowej. Za odtwarzanie dźwięków odpowiada serwis AudioPlayerService. Klasa ta definiuje ścieżki do próbek dźwiękowych danego instrumentu oraz posiada metodę pozwalającą na odtworzenie dźwięku po stronie klienta.

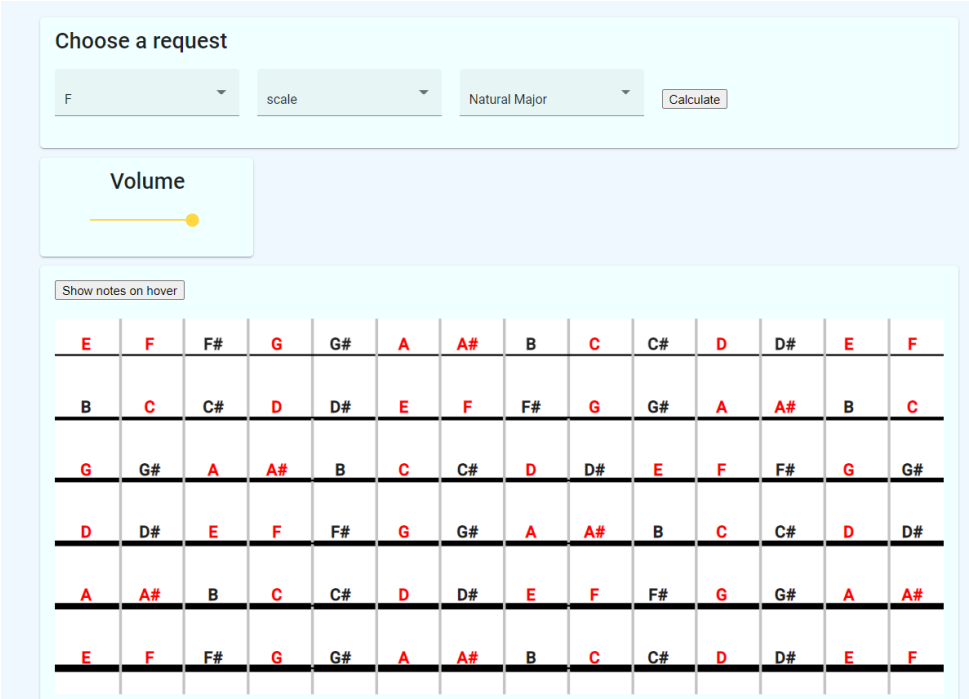
Listing 5.9: Metoda odtwrzająca zadany dźwięk

```
playNote(guitar_string:string, note:string, volume:number)
 {
   let audio = new Audio();
   //change format, file path can't contain '#' symbol
   if(note.length > 1 && note[1] == '#')
     note = note[0] + 's' + note.substring(2);
   audio.src = this.samplesDirectory + guitar_string + "/" + note +
      ↪ ".wav";
   audio.volume = volume
   audio.load();
   audio.play();
 }
```

Nagrania próbowano pozyskać z serwisów oferujących darmowe próbki oraz gotowych paczek, jednak nie udało się znaleźć pełnego, darmowego zestawu, który można byłoby wykorzystać w ramach aplikacji, więc zdecydowano się na własnoręczne nagranie gitar w domowych warunkach.

Rys. 5.6: Widok strony z gitarą basową po wykonanu zapytania

Rys. 5.7: Widok strony z gitarą basową po wykonaniu zapytania i przełączeniu opcji hover

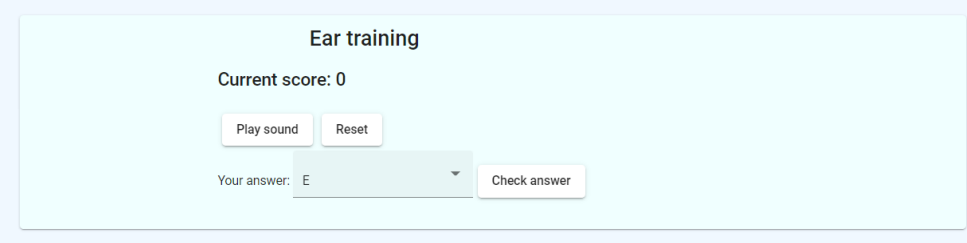


Rys. 5.8: Widok strony z gitarą akustyczną po wykonaniu zapytania

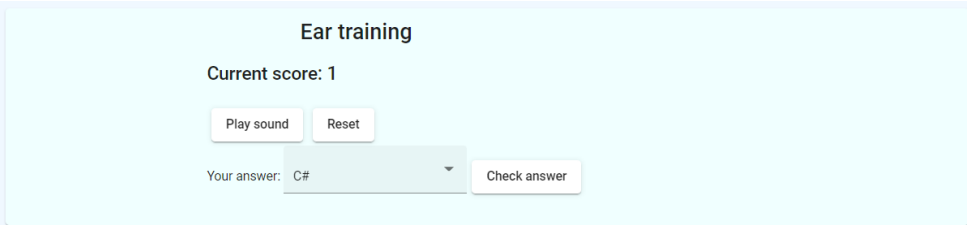
5.3.5. Trening słuchu

Podstrona służąca do treningu ucha muzycznego to kolejny komponent wykorzystujący AudioService poprzez wstrzykiwanie zależności.
Ćwiczenie polega na rozpoznawaniu odtwarzanego dźwięku ze słuchu. Użytkownik może użyć przycisku “Play sound” aby odtworzyć próbkę. Guzik jest wieloktrotengo użytku. Po odsłuchaniu próbki, użytkownik wybiera jeden z dźwięków z listy rozwijanej i zatwierdza wybór guzikiem “Check answer”. W wypadku udzielenia poprawnej odpowiedzi zwiększa się liczba punktów, a w wypadku złej, liczba punktów jest ustawiana na 0. W obu wypadkach losowany jest nowy dźwięk, aby użytkownik mógł kontynuować ćwiczenie.

Na stronie jest również guzik reset, który zeruje wynik użytkownika oraz losuje nową próbkę dźwiękową.



Rys. 5.9: Początkowy stan komponentu treningu słuchu



Rys. 5.10: Stan komponentu treningu słuchu po udzieleniu poprawnej odpowiedzi

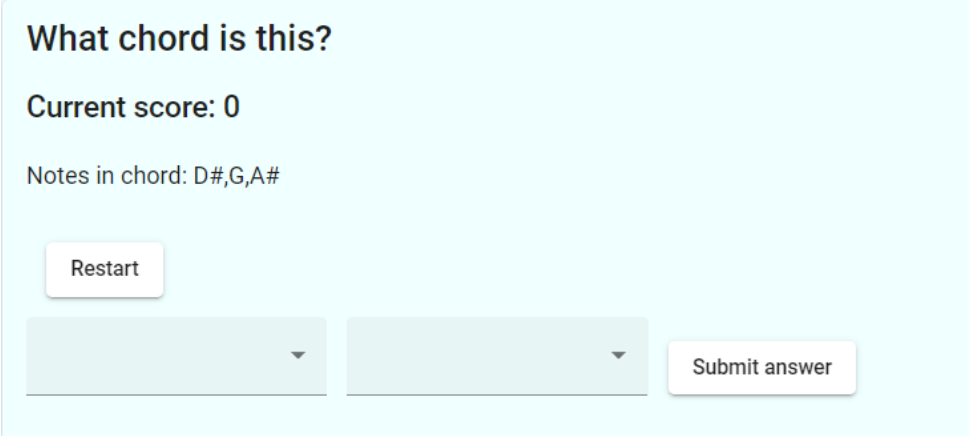
5.3.6. Quiz

Ten komponent jest zależny od serwisu TheoryApiService. Podstrona z quizem z wiedzy dotyczącej akordów działa podobnie do podstrony z treningiem słuchu - najpierw użytkownikowi zadawane jest pytanie, później wybiera on odpowiedź za pomocą list rozwijanych i zatwierdza wybór guzikiem. Po udzieleniu dobrej odpowiedzi jego wynik jest inkrementowany, a po niepoprawnej wynik jest zerowany. W obu wypadkach po udzieleniu odpowiedzi losowane jest nowe pytanie. Na stronie jest również guzik reset losujący nowe pytanie i zerujący wynik.
Zaimplementowane pytania polegają na rozpoznawaniu losowo wybieranych akordów po budujących je dźwiękach, łatwo jednak rozwinąć go dodatkowe pytania dodając do list np. interwały.

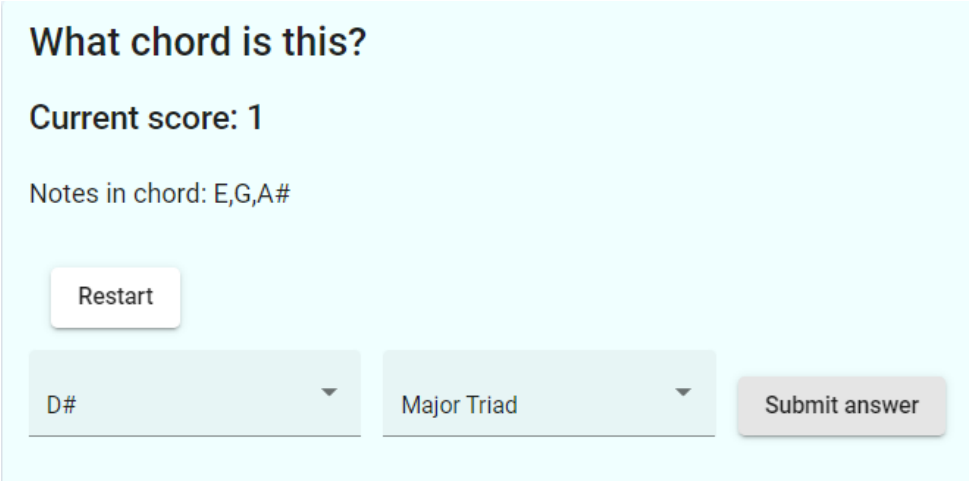
Listing 5.10: Metoda generująca losowy akord

```
getChord()
 {

   let chordName = this.chordsNames[Math.floor(Math.random() * this.
      ↪ chordsNames.length)];
   let noteName  = this.notes[Math.floor(Math.random() * this.notes.
      ↪ length)];
   this.currentChord = chordName;
   this.currentNote = noteName;
   this.notesInChord = null;

   if (noteName.includes('#') ){
     noteName = noteName[0] + 'S';
   }
   let split: number = chordName.indexOf('␣');
   chordName = chordName.toLowerCase();
```

```
    chordName = chordName.substr(0, split) + '_' + chordName.substr(
      ↪ split+1);
    console.log(noteName, chordName)
    this.theoryApi.getChord(noteName, chordName).subscribe(data => {
      this.notesInChord = data;
    });

  }
```



Rys. 5.11: Początkowy stan komponentu treningu słuchu

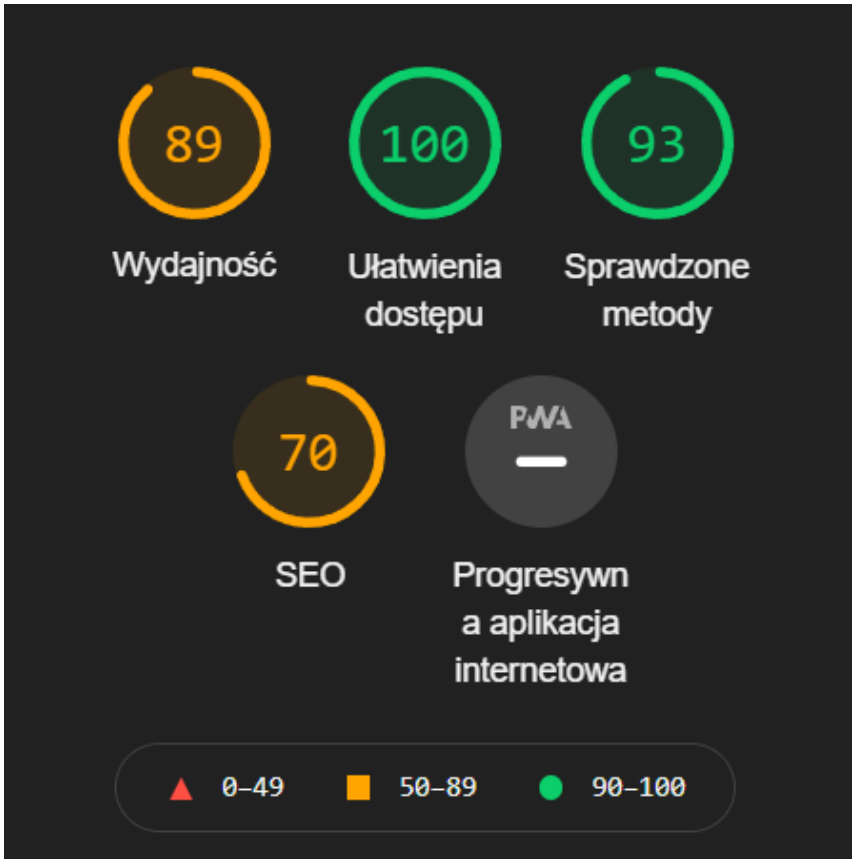


Rys. 5.12: Stan komponentu treningu słuchu po udzieleniu poprawnej odpowiedzi

5.4. Testowanie aplikacji

5.4.1. Lighthouse

Wtyczka lighthouse pozwala na przetestowanie aplikacji webowych biorąc pod uwagę charakterystyczne dla nich cechy, takie jak czas do załadowania czy optymalizacja pod kątem wyszukiwania strony przez silniki wyszukiwania.



Rys. 5.13: Wyniki testu Lighthouse

Aplikacja przeszła testy z dobrymi wynikami. Jednym ze znalezionych przez wtyczkę problemów okazała się optymalizacja pod kątem silników wyszukiwania - w celu poprawy należałoby przede wszystkim zwiększyć ilość metadanych opisujących stronę.
Kolejnym problemem jest wydajność aplikacji - tutaj informacja o wyniku <90 jest dużo bardziej niepokojąca, ponieważ wydajność bezpośrednio wpływa na odczucia użytkownika. W celu poprawienia wydajności należałoby przede wszystkim przeanalizować kod pod kątem funkcji, które mogą opóźniać ładowanie elementów strony. Należy jednak pamiętać, że podczas testów zarówno frontend, backend jak i aplikacja testowa były uruchomione na tej samej maszynie, co może znacząco wpływać na wyniki testów wydajnościowych.

Rozdział 6

Podsumowanie

6.1. Zrealizowane cele i założenia

W ramach pracy postawiono następujące cele:

- wybór technologii,
- zaprojektowanie aplikacji webowej,
- implementację aplikacji,
- projekt i implementację testów aplikacji,
- udokumentowanie procesu w pracy dyplomowej.

Wszystkie powyżej wymienione cele udało się spełnić.

W celu wyboru technologii zastosowano kryteria takie jak popularność danej technologii oraz jej dostosowanie do zadania. Przed zaprojektowaniem aplikacji dokonano analizy wymagań funckjonalnych, niefunkcjonalnych oraz przypadków użycia.

implementacja aplikacji skupiała się przede wszystkim na spełnieniu wyznaczonych przypadków użycia przedstawionych na rysunku 3.1. Udało się zaimplementować wszystkie zaprezentowane tam przypadki użycia, jednak ćwiczenia interaktywne stoją na stosunkowo niskim poziomie - zaimplementowano tylko dwa rodzaje ćwiczeń. Żadne z ćwiczeń nie korzysta z mikrofonu, co prawda nie było to wymogiem, ale na pewno byłoby to dużo bardziej angażujące dla użytkownika niż wybór odpowiedzi z listy rozsuwanej.

6.2. Dalszy rozwój aplikacji

Aplikacja ma duże możliwości na dalszy rozwój. Szczególnie warte uwagi pomysły to: konta użytkowników, dodatkowe instrumenty, dodatkowe ćwiczenia, używanie mikrofonu, lekcje.

6.2.1. Konta użytkowników

Do aplikacji można dodać konta użytkowników, które pozwalałaby na zachowywanie wyników z ćwiczeń pomiędzy sesjami. Mając zapisane wyniki z ćwiczeń możnaby nawet stworzyć tablice wyników. Wraz z dodawaniem kolejnych ćwiczeń konta byłyby coraz bardziej użyteczne. Kolejne funkcjonalności również mogłyby korzystać z tej funkcjonalności, np. po dodaniu lekcji, konta pozwalałyby zapamiętać lekcje, które użytkownik już przejrzał.

6.2.2. Dodatkowe instrumenty

Ze względu na to jak stworzono podstronę z wirtualnym instrumentem i serwis do odtwarzania dźwięków stronę łatwo wzbogacić o dodatkowe instrumenty, takie jak np. pianino. Oczywiście nie na każdym instrumencie łatwo zaprezentować graficznie dźwięki, więc trzebaby rozważyć odpowiedni dobór instrumentów.

6.2.3. Dodatkowe ćwiczenia

Dodatkowe ćwiczenia można implementować jako kolejne podstrony niezależnie od reszty treści strony, więc aplikacja jest pod tym względem aplikacja jest rozszerzalna. Możnaby stworzyć podobny quiz do tego z akordami, ale zawierający tylko skale lub zupełnie inne rodzaje ćwiczeń jak np. wybranie kolejnych dźwięków w zadanej skali.

6.2.4. Wykorzystanie mikrofonu

Wykorzystanie mikrofonu otworzyłoby wiele możliwości, np. ćwiczenia w których odpowiedzią jest zagranie odpowiedniego dźwięku, interaktywny stroik pokazujący czy dźwięk jest za niski czy za wysoki.

6.2.5. Lekcje

Warto rozważyć podstronę która zawierałaby krótkie lekcje teorii muzyki, które następnie możnaby połączyć z interaktymi ćwiczeniami. Tutaj można byłoby również wykorzystać funkcjonalność kont po ich implementacji do zapamiętywania które lekcje użytkownik już przejrzał, tworzenia notatek czy dyskusji o materiale zawartym w lekcji.

Literatura

[1] Angular documentation. https://angular.io/docs. Dostęp: 25.01.2022.

[2] Angular usage statistics. https://trends.builtwith.com/framework/Angular. Dostęp: 25.01.2022.

[3] Flask documentation. https://flask.palletsprojects.com/en/2.0.x/. Dostęp 26.01.2022.

[4] Java documentation. https://docs.oracle.com/en/java/. Dostęp 26.01.2022.

[5] Python 3.9 documentation. https://docs.python.org/3.9/. Dostęp 26.01.2022.

[6] React redux usage statistics. https://trends.builtwith.com/javascript/React-Redux. Dostęp: 25.01.2022.

[7] Spring documentation. https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html. Dostęp 25.01.2022.

[8] Rest api design. *alexsoft.com*, Mar. 2021. https://www.altexsoft.com/blog/rest-api-design/.

[9] R. D. John Au-Yeung. Rest api design. *stackoverflow.blog*, Mar. 2020. https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design/.

Dodatek A

Instrukcja uruchomienia

Uruchomienie systemu opiera się na uruchomieniu dwóch mniejszych aplikacji: aplikacji backendowej napisanej w języku Python oraz Frontendowej napisanej w języku TypeScript.

A.1. Uruchamianie backendu

1. W katalogu backend uruchomić konsolę Windows.
2. Zainstalować potrzebne zależności komendą: pip install -r requirements.txt.
3. Wpisać komendę: SET FLASK_APP=src.
4. Wpisać komendę: python -m flask run.
5. Aplikacja jest uruchomiona aż do wyłączenia konsoli lub przerwania działania (np. za pomocą skrótu CTRL+C).

A.2. Uruchamianie frontendu

1. Zainstalować NPM.
2. Za pomocą NPM Zainstalować Angular CLI - w konsoli wpisać komendę: npm install -g @angular/cli.
3. Jeśli backend nie jest uruchomiony na tej samej maszynie, należy ustawić odpowiedni adres w pliku environment.prod.ts lub environment.ts (dla uruchomienia w środowisku debugowym).
4. Uruchomić program komendą ng serve.

Dodatek B

Opis załączonej płyty CD/DVD

Na płycie znajduje się katalog zawierający gotowy do wybudowania projekt z podziałem na podkatalogi: backend oraz frontend. Folder backend zawiera aplikację backendową napisaną w języku Python, natomiast w drugim katalogu znajduje się aplikacja frontendowa napisana w języku TypeScript oraz pliki potrzebne do jej poprawnego funkcjonowania, takie jak obrazy czy próbki dźwięku.