Politechnika Śląska

PROJEKT INŻYNIERSKI

„Wspomaganie tworzenia tabulatur gitarowych z wykorzystaniem analizy dźwięku instrumentu”

Paweł DRZAZGA
Nr albumu 301728

Kierunek: Informatyka
Specjalność: Grafika Komputerowa i Oprogramowanie

PROWADZĄCY PRACĘ
dr inż. Krzysztof Hanzel
KATEDRA Telekomunikacji i Teleinformatyki
Wydział Automatyki, Elektroniki i Informatyki

KATOWICE Rok 2025

Tytuł pracy:

Wspomaganie układania tabulatur gitarowych z wykorzystaniem analizy dźwięku instrumentu

Streszczenie:

Projekt inżynierski dotyczy zaimplementowania edytora notacji muzycznej dla gitarzystów w postaci jednostronicowej aplikacji webowej (SPA). Wyróżniającą się funkcjonalnością tej aplikacji jest automatyczne układanie tabulatury na podstawie analizy dźwięku instrumentu, wykorzystującej algorytmy przetwarzania sygnału, takie jak FFT oraz AMDF. Dodatkowo aplikacja posiada funkcje: symulacja odtwarzania melodii z tabulatury, system kont użytkowników, zarządzanie projektami użytkownika. W pracy omówiono zasady dotyczące zapisu tabulaturowego i specyfikę instrumentów gitarowych, istotne dla kontekstu problemu. Szczegółowo przedstawiono zagadnienia związane z analizą sygnałów dźwiękowych, w tym metod uzyskiwania częstotliwości podstawowej. Zdefiniowano wymagania dla systemu i opisano jego specyfikację zewnętrzną oraz wewnętrzną. Porównano finalną implementację z początkowym założeniami i przedstawiono możliwe kierunki rozwoju aplikacji.

Słowa kluczowe:

Analiza sygnału dźwiękowego, częstotliwość podstawowa, tabulatura gitarowa, aplikacja webowa

Thesis title:

Aiding guitar tablature composition through instrument sound analysis

Abstract:

The engineering project focuses on the implementation of a music notation editor for guitarists in the form of a single-page web application (SPA). The key feature of this application is the automatic arrangement of tablature based on instrument sound analysis, using signal processing algorithms such as FFT and AMDF. Additionally, the application includes functions such as melody playback simulation from tablature, a user account system, and project management.

The paper discusses the principles of tablature notation and the specific characteristics of guitars, which are important in the context of the problem. Issues related to sound signal analysis, including methods for determining the fundamental frequency, are presented in detail. The system requirements are defined, and both external and internal specifications are described. The final implementation is compared with the initial assumptions, and potential directions for further application development are presented.

Keywords:

Sound signal analysis, fundamental frequency, guitar tablature, web application

Spis treści

Rozdział 1

Wstęp

Tworzenie kompozycji muzycznych jest złożonym i często długotrwałym procesem, składającym się z wielu przeplatających się przez siebie etapów. Twórca, po zaczerpnięciu inspiracji z wybranego źródła oraz utworzeniu koncepcji swojego dzieła, wkracza w stadium, którego nieodzowną częścią staje się zapis postępów jego pracy w odpowiedniej notacji muzycznej. Już na samym początku, w fazie planowania struktury, podczas układania pierwszych motywów i melodii, zachodzi konieczność ich odpowiedniego zanotowania. Standardowa notacja muzyczna, inaczej zapis nutowy, pozwala na precyzyjne odzwierciedlenie zamysłu kompozytora w czytelnej dla wykonawców formie. Stanowi symboliczny język opisujący między innymi cechy dźwięków muzycznych, rytmiki, dynamiki oraz artykulacji. Podstawą zapisu jest pięciolinia, na której umieszczane są jednostki rytmiczne w postaci nut i pauz. Wysokość nut, wraz z znakami chromatycznymi, stanowi o wybranym dźwięku jaki reprezentują w obrębie gamy muzycznej.

Mimo swojej uniwersalności i precyzji, zapis nutowy nie zawsze jest dogodnym rozwiązaniem dla każdego instrumentalisty. Wymaga od grającego dobrej znajomości teorii muzyki oraz zrozumienia specyfiki instrumentu. Notacja ta często nie jest wprost powiązana z budową danego narzędzia muzycznego, co zmusza muzyka do operowania na pewnym poziomie abstrakcji.

Kontrast w dziedzinie czytania nut dla instrumentów klawiszowych i strunowych (szyjkowych) dobrze obrazuje przywołany problem. Klawisze pianina odpowiadają kolejnym oktawom w sekwencji dźwięków gamy muzycznej, analogicznie do sposobu ich przedstawienia na pięciolinii. W przypadku gitary dźwięk wydobywa się poprzez szarpnięcie struny, a jego wysokość zależy od siły napięcia oraz długości, czyli od miejsca, w którym struna jest dociśnięta do progu. Granie z nut na gitarze wymaga od instrumentalisty bardzo dobrej znajomości rozłożenia dźwięków na gryfie, ponieważ zapis nutowy nie wskazuje jednoznacznie konkretnych miejsc ich występowania, w przeciwieństwie do pianina – każdemu klawiszowi odpowiada określony dźwięk. Nawet

nieznaczna zmiana stroju gitary, czyli bazowych dźwięków każdej struny, wpłynie na przesunięcie pozostałych dźwięków na gryfie.

Gitarzyści często sięgają po uproszczoną formę zapisu notacji muzycznej, czyli tabulaturę. Główną cechą odróżniającą tabulaturę od standardowej notacji muzycznej jest wskazanie konkretnych miejsc występowania danych dźwięków na instrumencie oraz sposób ich wydobycia. Podstawą zapisu tabulatury jest układ linii reprezentujących struny instrumentu, na którym umieszczane są numery progów wraz z oznaczeniami technik artykulacyjnych. Forma zapisu notacji na tabulaturze jest ściśle dostosowana do budowy danego instrumentu, dzięki temu proces tworzenia muzyki dla tych instrumentów strunowych przebiega sprawniej.

W Internecie dostępnych jest wiele edytorów, które umożliwiają efektywne tworzenie tabulatur i zapisu nutowego. Jednym z najpopularniejszych narzędzi tego typu jest „Guitar Pro". Jest to aplikacja na komputery klasy pc, pozwalająca nie tylko na tworzenie i edytowanie tabulatur, ale również na odtwarzanie utworów i współpracę z plikami muzycznymi takimi jak MIDI. Musical Instrument Digital Interface jest standardem komunikacji cyfrowej, pozwalającym na przesyłanie informacji muzycznych takich jak opis dźwięku, jego wysokość, głośność czy czas trwania. Jednak dane te nie zawierają rzeczywistego sygnału audio. W związku z tym „Guitar Pro" podobnie jak inne aplikacje, mimo możliwości przetwarzania danych muzycznych nie umożliwiają bezpośredniej analizy audio, co nie pozwala na automatyczne tworzenie tabulatury na podstawie sygnału wejściowego.

Celem niniejszej pracy jest zaprojektowanie i zaimplementowanie kreatora tabulatur gitarowych w postaci jednostronicowej aplikacji webowej. Aplikacja pozwoli na sprawne tworzenie oraz edytowanie notacji muzycznej dla gitarzystów. Wyróżniającą się funkcjonalnością aplikacji, na tle innych popularnych serwisów, jest automatyczne zapisywanie zagranych dźwięków z instrumentu na wirtualnej tabulaturze. Pozwoli to na odwzorowywanie utworów bez konieczności ręcznego wprowadzania nut. Dodatkowo aplikacja zostanie wyposażona w manualny edytor oraz odtwarzacz tabulatury z regulacją tempa, sprzyjający nauce zapisanych utworów. Dzięki systemowi kont, użytkownik będzie mógł w każdej chwili wrócić do komponowania utworu, bez obawy o utratę swojej pracy.

Niniejsza praca została podzielona na siedem rozdziałów omawiających najważniejsze aspekty projektu.

Rozdział 1. Wstęp – wprowadzenie w zagadnienie projektu oraz określenie celu i zakresu pracy.

Rozdział 2. Analiza tematu – omówienie zagadnień związanych z przetwarzaniem sygnału audio oraz dekodowaniem dźwięków na zapis w notacji tabulatorowej.

Rozdział 3. Wymagania i narzędzia – opis wymagań funkcjonalnych i niefunkcjonalnych dla aplikacji, diagram przypadków użycia oraz omówienie wykorzystanych technologii.

Rozdział 4. Specyfikacja zewnętrzna – prezentacja interfejsu użytkownika wraz z przykładami działania.

Rozdział 5. Specyfikacja wewnętrzna – przedstawienie idei implementacyjnej oraz architektury systemu w postaci opisów i diagramów UML.

Rozdział 6. Weryfikacja i walidacja – sposób testowania aplikacji, przeprowadzone testy oraz wykryte i usunięte błędy.

Rozdział 7. Podsumowanie i wnioski – zestawienie osiągniętego rezultatu wraz z postawionymi celami oraz wymaganiami projektu, a także rozwój projektu w przyszłości.

Wkład autora w realizację pracy obejmuje zaprojektowanie oraz zaimplementowanie kreatora tabulatury gitarowej w formie aplikacji webowej z systemem kont użytkowników oraz modułem audio, służącym do analizy dźwięku instrumentu.

Rozdział 2

Analiza tematu

2.1. Specyfika instrumentów strunowych

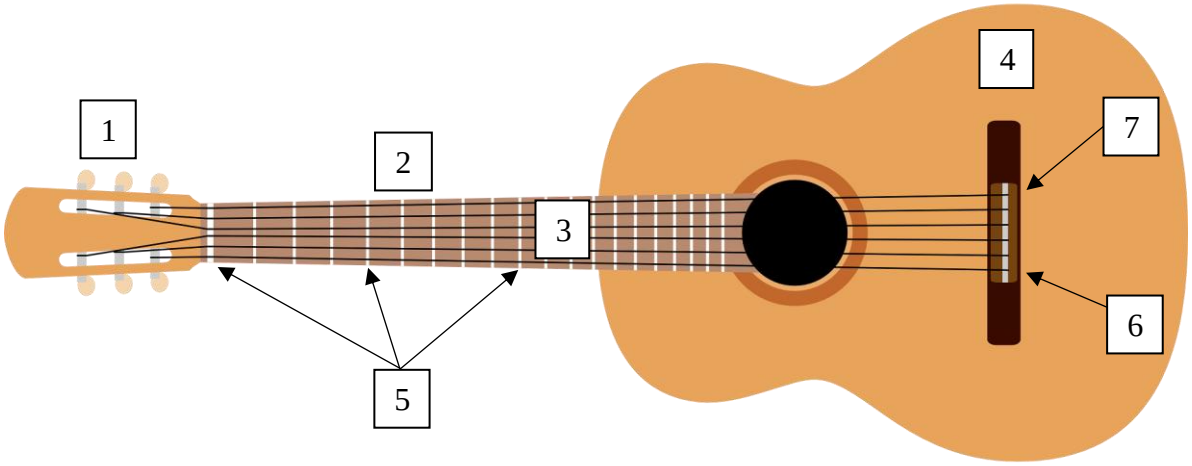
Źródłem dźwięku instrumentów strunowych, inaczej chordofonów, są drgające struny. Drgania te wywoływane są poprzez szarpanie, uderzanie lub pocieranie strun, a za ich wzmocnienie odpowiadają różne elementy konstrukcyjne instrumentu, np. pudło rezonansowe gitary akustycznej. W wyniku tych drgań powstają fale akustyczne o określonej częstotliwości, które przez ludzkie ucho odbierane są jako dźwięk.

Chordofony są szeroką kategorią instrumentów muzycznych. Zgodnie z podziałem organologicznym Hornbostela-Sachsa, do instrumentów strunowych (tabela 2.1), ze względu na sposób wzbudzania drgań, należą instrumenty uderzane, smyczkowe i szarpane, w tym szarpane szyjkowe (ze względu na konstrukcję) [1].

Tabela 2.1. Wybrane przykłady poszczególnych instrumentów strunowych

Chordofony szarpane	Gitara, mandolina, lutnia, harfa, santur
Chordofony szarpane - szyjkowe	Gitara, mandolina, lutnia
Chordofony uderzanie	Fortepian, pianino
Chordofony smyczkowe	Skrzypce, wiolonczela, kontrabas

Z uwagi na specyfikę niniejszej pracy, skupiono się na podgrupie instrumentów strunowych szarpanych-szyjkowych, należących do chordofonów złożonych, których nośnik strun i rezonator są na stałe scalone. Szyjka, potocznie nazywana też gryfem, łączy korpus z główką i stanowi przedłużenie strun poza korpus instrumentu. Pod strunami znajduje się podstrunnica, będąca polem, na którym instrumentalista za pomocą docisku palców może swobodnie skracać długość drgającej struny, czyli zmieniać wysokość dźwięku. W zależności od rodzaju i wariantu instrumentu, na podstrunnicy znajdują się progi - wypukłe elementy ułatwiające wydobycie określonych dźwięków muzycznych. Na rysunku 2.1 przedstawiono i opisano gitarę klasyczną.



Rysunek 2.1. Gitara klasyczna. 1 – główka, 2 – szyjka, 3 – podstrunnica, 4 – korpus (rezonator), 5 – progi (kolejno: 0 - siodełko, 4, 9), 6 – struna szósta, 7 – struna pierwsza

W standardowej gitarze klasycznej, a także jej odmianach – gitarach elektrycznych i akustycznych, zamocowanych jest sześć strun o różnych średnicach, wykonanych z nylonu bądź stali z powłoką w przypadku pozostałych. Zazwyczaj pierwsza struna, czyli znajdująca się na dole, gdy instrument trzymany jest w pozycji grającej, jest najcieńsza (0.008 – 0.012 cala w gitarze elektrycznej), a każda kolejna jest grubsza o kilka tysięcznych cala. Wybór szerokości oraz materiału strun zależy od typu i konstrukcji instrumentu oraz od jego stroju.

Strojem otwartym określa się zestawienie dźwięków, jakie wydają struny na zerowym progu (tabela 2.2). Nazwa stroju otwartego pochodzi od struny o najniższym dźwięku oraz ustalonych interwałów (odległości tonowych) pomiędzy kolejnymi strunami (tabela 2.2) [2].

Tabela 2.2. Strój otwarty E-Standard [2]

Numer struny	**Dźwięk muzyczny**	**Częstotliwość (Hz)**
6	E2	82.4
5	A2	110
4	D3	146.8
3	G3	196
2	B3	246.9
1	E4	329.6

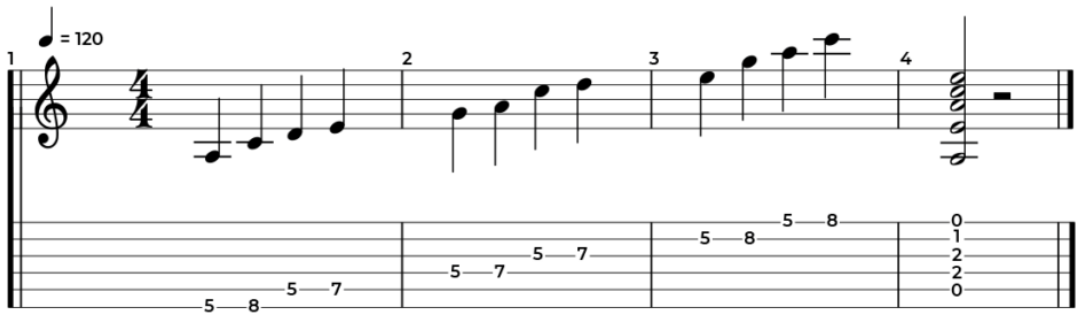
Tabela 2.3. Interwały w strojach standardowych

Struna	**Przejście**	**Interwał**
6. → 5.	+5 półtonów	Kwarta czysta
5. → 4.	+5 półtonów	Kwarta czysta
4. → 3.	+5 półtonów	Kwarta czysta
3. → 2.	+4 półtony	Tercja wielka
2. → 1.	+5 półtonów	Kwarta czysta

2.2. Definicja tabulatury

Tabulatura (TAB) dla instrumentów strunowych w swojej tradycyjnej postaci nie jest tak precyzyjnym i uniwersalnym narzędziem jak notacja muzyczna, co zauważył Denyer w swojej publikacji [2]. Ograniczenia, takie jak brak informacji o długości trwania nut i rytmie, sprawiają, że zapis w tym systemie nie jest wystarczający do dokładnego odwzorowania utworu i stanowi jedynie formę skrótu myślowego. Mimo to, niski próg wejścia w kontekście wiedzy muzycznej jest przyczyną popularności tabulatury u początkujących muzyków.

System TAB opiera się na siatce złożonej z poziomych i równoległych linii, których ilość jest równa liczbie strun danego instrumentu – w przeciwieństwie do standardowej notacji muzycznej, gdzie używana jest pięciolinia, niezależnie od jej przeznaczenia (rysunek 2.2). Pierwsza, górna linia odpowiada pierwszej strunie instrumentu natomiast ostatnia, n-ta linia – ostatniej n-tej strunie. Tabulaturę czyta się wierszami, od lewej do prawej strony. Liczby umieszczone na liniach odpowiadają numerom progów na podstrunnicy instrumentu.



Rysunek 2.2 Porównanie zapisu standardowej notacji muzycznej i tabulatury

Często występującym elementem jest oznaczenie tempa oraz metrum utworu. Tempo określa, ile jednostek rytmicznych (najczęściej ćwierćnut) mieści się w jednej minucie. Metrum określa regularne grupowanie nut w jednostki zwane taktami, dzielącymi

tabulaturę na fragmenty za pomocą pionowych linii. Górna liczba (numerator) określa liczbę uderzeń w takcie, natomiast dolna liczba (denominator) wskazuje jaka wartość nuty (tabela 2.4) odpowiada jednemu uderzeniu. Jak przedstawiono na rysunku 2.2, w jednej minucie utworu mieści się sto dwadzieścia ćwierćnut (tempo), a w jednym takcie znajdują się cztery uderzenia, z których każde ma wartość ćwierćnuty (metrum).

Tabela 2.4. Tabela wartości rytmicznych nut

Nazwa jednostki	**Symbol**	**Czas trwania w takcie**
Pełna nuta (1)	𝅝	Cały takt
Półnuta (2)	𝅗𝅥	½ Taktu
Ćwierćnuta (4)	♩	¼ Taktu
Ósemka (8)	♪	⅛ Taktu
Szesnastka (16)	♬	1⁄16 Taktu

W zapisie tabulatorowym istnieją również oznaczenia różnych technik artykulacyjnych (tabela 2.5), czyli specjalnych metod wykonania danego dźwięku lub ich zestawu. Umożliwia to instrumentaliście precyzyjne odwzorowanie zamierzonego efektu dźwiękowego. Techniki wykonywane na pojedynczej strunie, dzielą się na trzy główne grupy: techniki młoteczkowe (ang. *hammering-techniques*), slajdy (ang. *slides*) oraz podciągnięcia (ang. *bending*) [2].

Techniki młoteczkowe polegają na płynnym przejściu pomiędzy dźwiękami, bez konieczności dodatkowego wzbudzania struny ręką szarpiącą (ang. *picking-hand*). *Hammer-on* to technika wydobycia wyższego dźwięku poprzez energiczne uderzenie palcem ręki dociskającej w strunę na wyższym progu. Natomiast *pull-off* polega na pociągnięciu struny podczas przejścia na niższy próg, tak aby wybrzmiał dźwięk. Połączenie tych obu technik nazywane jest jako *legato* i pozwala na osiągnięcie płynności i ciągłości dźwięku. Innym wariantem technik młoteczkowych jest *tapping*, zaawansowana technika, w której używa się ręki grającej do wykonywania hammer-on'ów i pull-off'ów.

Slajdy mają na celu uzyskanie podobnego efektu płynnego przejścia jak techniki młoteczkowe, jednak różnią się sposobem wykonania. W tej technice używany jest tylko jeden palec, dociskając strunę jest przeciągany z jednego progu na drugi.

Podciągnięcia służą do uzyskania wyższego dźwięku, poprzez zwiększenie napięcia struny. Wykonywane są poprzez palec dociskający, który odchyla strunę w górę lub w dół, zwiększając jej długość. Najczęściej stosowanymi wariantami podciągnięć są podciągnięcia o cały ton, pół tonu oraz podciągnięcie z powrotem do pierwotnej pozycji.

Tabela 2.5. Tabela oznaczeń technik artykulacyjnych

Technika artykulacyjna	**Oznaczenie na tabulaturze**
Hammer-on	Litera „h”, np. „7 h 9”
Pull-off	Litera „p”, np. „9 p 7”
Tapping	Litera „T” pod wierszem siatki TAB.
Legato	Łuk łączący liczby z góry (h) lub z dołu (p)
Slajd	Ukośna linia, np. „7 / 9” lub „9 \ 7”
Podciągnięcie o cały ton	Strzałka wygięta w górę z etykietą „full”
Podciągnięcie o pół tonu	Strzałka wygięta w górę z eytkietą ½
Podciągnięcie z powrotem	Dodatkowa strzałka wygięta w dół

W kilku ostatnich dekadach notacja tabulatury została znacznie rozwinięta. Dodano do niej wiele nowych oznaczeń, które umożliwiają dokładniejsze odwzorowanie skomplikowanych technik gry oraz elementy sprawiające, że zapis ten stał się bardziej uniwersalny. Za przykład rozwoju tabulatury może posłużyć portal Songsterr. Przedstawia współczesną, interaktywną formę tabulatury, uwzględniającą nowe oznaczenia, takie jak wartości rytmiczne nut, alternatywne zakończenia powtórzonych sekcji, szczególne sposoby zagrania dźwięku czy akcenty [3].

W niniejszej pracy skoncentrowano się na programowym zaprojektowaniu i zaimplementowaniu tabulatury gitarowej w swojej tradycyjnej postaci. Dodatkowo zadbano o pełną precyzję w zapisie rytmicznym, przy uwzględnieniu wartości rytmicznych nut oraz o odpowiednie zaplanowanie taktu. Zgodność z teorią muzyki zapewnia, że nuty są rozmieszczone zgodnie z taktem oraz metrum. Połączenie zapisu tabulatorowego z niektórymi elementami standardowej notacji muzycznej umożliwi interaktywne odtwarzanie tabulatury oraz prostą edycję z wykorzystaniem narzędzia analizującego dźwięk prawdziwego instrumentu.

2.3. Analiza sygnału dźwiękowego

W kontekście analizy dźwięku, którego źródłem jest instrument w formie gitary lub jej wariantów, kluczowym zagadnieniem jest rozpoznawanie poszczególnych częstotliwości sygnału a następnie poprawne odnalezienie dźwięku fundamentalnego. W tym podrozdziale zdefiniowane oraz omówione zostaną ważne pojęcia i metody analizy sygnału audio.

2.3.1. Sygnał dźwiękowy

Sygnał, w szerokim ujęciu, to model pewnej określonej mierzalnej wartości, która zmienia się w czasie. Wartość ta, generowana jest przez zjawiska fizyczne bądź systemy a przedstawiać ją można za pomocą matematycznej funkcji w dziedzinie czasu. W książce

„*Signals and Systems*” Oppenheim i Willsky wyjaśniają, że sygnał jest funkcją zmiennej niezależnej, która reprezentuje pewne zmieniające się wielkości np. napięcie, ciśnienie, temperaturę, czy inne fizyczne zjawiska [4].

Dźwięk jest zjawiskiem akustycznym, polegającym na rozprzestrzenianiu się fal akustycznych, czyli na cyklicznych zmianach ciśnienia w ośrodku sprężystym takim jak powietrze. Okresowe zmiany w poziomie ciśnienia są wynikiem drgań cząsteczek w medium, które przenoszą energię od źródła do odbiorcy. W kontekście sygnałów, dźwięk można traktować jako sygnał mechaniczny. Sygnał ten reprezentuje zmienną wielkość fizyczną, czyli zmiany ciśnienia akustycznego w czasie. W matematycznym ujęciu, jedną z postaci sygnału akustycznego jest funkcja czasu $p(t)$, gdzie p reprezentuje ciśnienie akustyczne w pewnym punkcie przestrzeni w dziedzinie czasu t.

2.3.2. Charakterystyka fal dźwiękowych

Fale dźwiękowe są typem fal podłużnych i mogą być opisywane za pomocą parametrów i reprezentacji charakterystycznych dla sygnałów, takich jak częstotliwość, amplituda, okres oraz widmo.

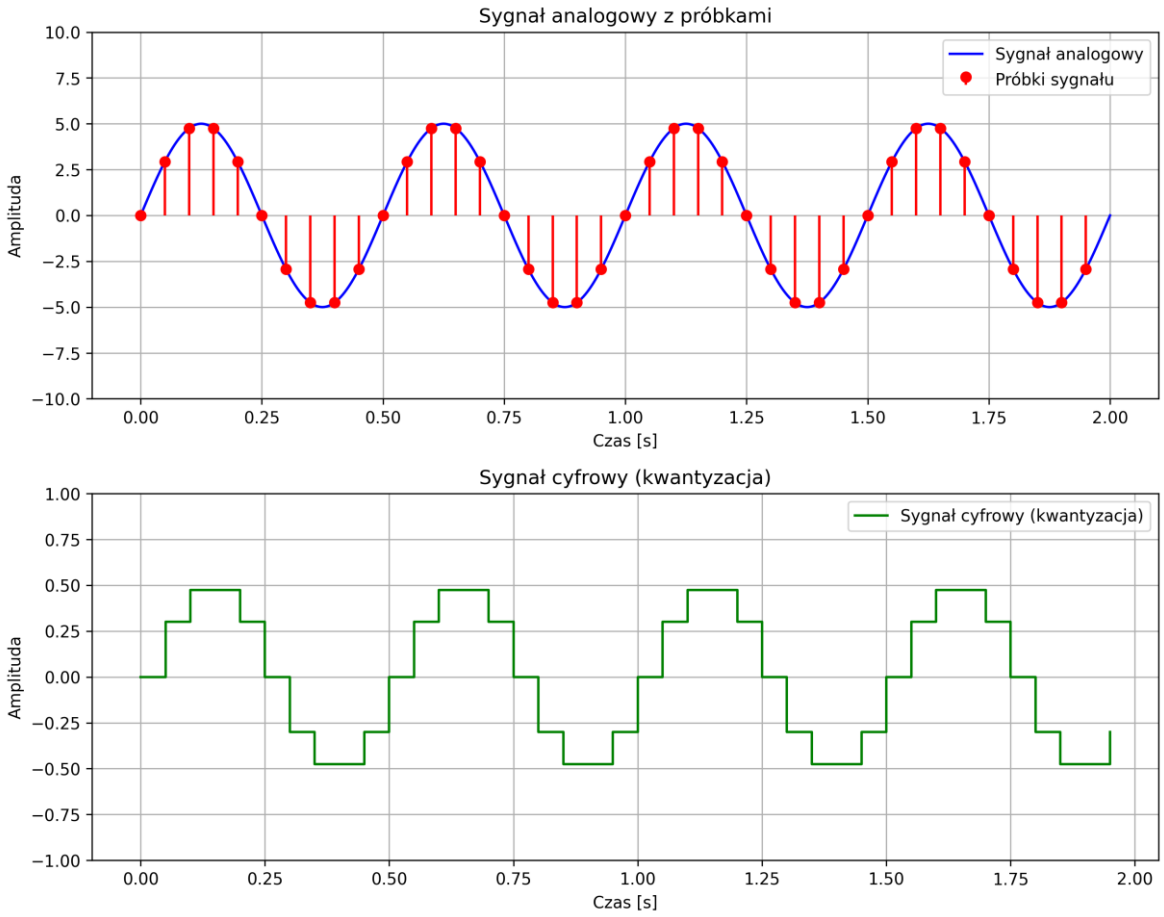
- Częstotliwość (f) opisuje liczbę cykli zmian ciśnienia na sekundę i jest wyrażana w hercach (Hz). Jej wartość dla sygnału dźwiękowego określa wysokość tonu – niski ton ma niższą częstotliwość, wysoki ton – wyższą częstotliwość (przykład w tabelach 2.2 i 2.3).
- Amplituda to maksymalna różnica pomiędzy minimalną lub maksymalną wartością zmiany ciśnienia od poziomu równowagi. Im większa jest amplituda tym sygnał dźwiękowy jest odbierany jako głośniejszy.
- Okres (T) jest czasem trwania jednego cyklu fali wyrażanym w sekundach (s) i zarazem odwrotnością częstotliwości ($T = 1/f$).
- Widmo sygnału dźwiękowego przedstawia składowe częstotliwościowe fali dźwiękowej. Reprezentacja dźwięku w dziedzinie częstotliwości jest kluczowa podczas analizy harmonicznych.

Oprócz swojej podstawowej częstotliwości, fale dźwiękowe składają się także z częstotliwości harmonicznych, które są całkowitymi wielokrotnościami częstotliwości podstawowej (fundamentalnej). Harmoniczne mają wpływ na barwę dźwięku, co czyni je istotnymi elementami w analizie widmowej.

2.3.3. Sygnał cyfrowy a sygnał analogowy

Sygnał dźwiękowy jako funkcja ciśnienia akustycznego jest funkcją ciągłą. Oznacza to, że wartości zmieniają się w sposób płynny, w każdym punkcie dziedziny czasu. Teoretycznie taki sygnał, czyli sygnał analogowy, może przyjmować nieskończoną liczbę wartości z zakresu amplitudy lub częstotliwości, lecz w praktyce zakresy te są ograniczone przez fizyczne właściwości urządzeń rejestrujących bądź odtwarzających dźwięk. Aby skutecznie przechować lub przetworzyć informację niesioną przez taki sygnał, w systemach cyfrowych, należy podjąć próbę jego reprezentacji w sposób dyskretny [5].

Dyskretyzacja sygnału analogowego, czyli przejście na sygnał cyfrowy, polega na przekształceniu sygnału w postać zawierającą jedynie wartości z skończonego przedziału w danym punkcie czasowym. Proces ten w głównej mierze obejmuje etap próbkowania oraz kwantyzacji. Próbkowanie polega na pomiarze wartości sygnału w regularnych odstępach czasowych, zwanych okresem próbkowania, a wynikiem jest dyskretny zbiór próbek. Im krótszy jest ten okres, tym dokładniej odwzorowany jest oryginalny sygnał [5]. Następnie każdej próbce przypisywany jest pewien poziom amplitudy z skończonego zbioru możliwych wartości – proces kwantyzacji (rysunek 2.3).



Rysunek 2.3. Porównanie sygnału analogowego oraz sygnału cyfrowego

Na rysunku 2.3. przedstawiono sygnał analogowy jako sinusoidalną funkcję ciągłą z zakresem wartości amplitudy $[-10,10]$ oraz sygnał cyfrowy w postaci funkcji schodkowej z znormalizowanym zakresem amplitudy po kwantyzacji w przedziale $[-1,1]$. Naniesiono wizualizację próbkowania, której częstotliwość (ang. *sample rate*) wynosi $f_s = 20\,Hz$. W tym przykładzie w ciągu sekundy wartość sygnału pobierana jest dwudziestokrotnie, co każdorazowo następuje po upłynięciu okresu $T = 0.05\,s$. Uzyskany w ten sposób sygnał cyfrowy jest podstawą do jego dalszej analizy pod kątem częstotliwości.

2.3.4. Analiza sygnału

Sygnał dźwiękowy, w formie sygnału cyfrowego, stanowi dane, które można poddać różnym metodom analizy w celu wyciągnięcia informacji, takich jak dominująca częstotliwość, czy składowe harmoniczne. Istnieją dwie główne dziedziny analizy sygnału dźwiękowego, które znajdują swoje zastosowanie, w zależności od charakterystyki analizowanego sygnału.

- Analiza w dziedzinie czasu jest niczym innym jak badaniem zmian sygnału w funkcji czasu. To najprostsza forma analizy sygnału, polega na analizie amplitud w każdej jednostce czasowej. W ten sposób można wykrywać zdarzenia dźwiękowe i czas ich trwania, wykrywać szumy i zakłócenia oraz rozpoznawać dźwięki.
- Analiza w dziedzinie częstotliwości polega na badaniu składowych częstotliwościowych sygnału dźwiękowego. Pozwala na wyciągnięcie informacji jakie częstotliwości są obecne w sygnale oraz jaka jest ich intensywność. To szczególnie istotne podejście podczas rozpoznawania tonalności dźwięków czy detekcji akordów muzycznych.

2.4. Metody analizy w dziedzinie czasu

Techniki analizy sygnału pozostające w reprezentacji czasowej, w kontekście detekcji fundamentalnej częstotliwości, opierają się na bezpośrednim badaniu zmian amplitudy w kolejnych próbkach. Wykorzystują właściwości sygnału, takie jak liczba przecięć z osią czasu, różnice między kolejnymi próbkami, czy okresowość. Najprostsze metody polegają na operacjach zliczania, natomiast inne, zaawansowane algorytmy porównują fragmenty sygnału w celu znalezienia najbardziej prawdopodobnego okresu drgań. Podejścia te charakteryzują się stosunkowo niską złożonością obliczeniową, co sprawia, że są użyteczne w zastosowaniach wymagających szybkiej analizy.

2.4.1. Zero Crossing Rate

W ogólnym znaczeniu Zero-Corssing jest argumentem, dla którego znak funkcji matematycznej ulega zmianie na przeciwny – jest to miejsce zerowe o nieparzystej krotności. W kontekście analizy sygnału w dziedzinie czasu, Zero Corssing Rate (ZCR) jest prostą metodą detekcji takich punktów w przebiegu badanego sygnału, stosowaną do estymacji podstawowej częstotliwości dźwięku [6].

Głównym założeniem metody ZCR jest obserwacja, że liczba oscylacji przebiegu sygnału jest proporcjonalna do jego częstotliwości w danym odcinku czasowym. Przyjmując, że każde przejście przez zero odpowiada połowie pełnego cyklu fali, to liczbę pełnych cykli C w sygnale można zdefiniować za pomocą wzoru (1).

$$C = \frac{1}{2}\sum_{j}^{N-1} 1\left(x_j \cdot x_{j-1} < 0\right) \tag{1}$$

Gdzie N – liczba próbek, x_j – wartość próbki sygnału w chwili j, $1(\cdot)$ – funkcja indykatorowa, zwracająca jeden, jeżeli wyrażenie w nawiasie jest prawdzie lub zero w przeciwnym przypadku. Czas w sekundach t, w którym badany jest sygnał można określić jako liczbę próbek podzieloną przez częstotliwość próbkowania f_s (2).

$$t = \frac{N}{f_s} \tag{2}$$

Wynikową częstotliwość podstawową f_0 metody ZCR dla badanego sygnału, próbkowanego z częstotliwością f_s wyznacza się wzorem (3).

$$f_0 = \frac{C}{t} = \frac{C \cdot f_s}{N} \tag{3}$$

ZCR w swojej prostocie jest szybkim i wydajnym algorytmem, dobrze działającym w przypadku sygnałów o sinusoidalnym charakterze, takich jak czyste tony. Jednakże dla dźwięków muzycznych, które zawierają liczne harmoniczne, metoda ta może generować błędne estymacje częstotliwości podstawowej i wymaga odpowiedniego wygładzenia sygnału, na przykład poprzez filtrację dolnoprzepustową [7].

2.4.2. YIN

Metoda YIN autorstwa Alain de Cheveigne'a oraz Hideki Kawahara jest jedną z najskuteczniejszych technik estymacji fundamentalnej częstotliwości w dziedzinie czasu. Opiera się na klasycznej metodzie autokorelacji i wprowadza szereg ulepszeń w celu zredukowania błędów związanych z wykrywaniem częstotliwości, szczególnie

w sygnałach muzycznych i mowie. Minimalizuje trudności analizy związane z subharmonicznymi oraz aperiodycznością sygnału.

Algorytm YIN estymuje okres sygnału, za pomocą funkcji różnicowej $d_t(\tau)$, która mierzy różnicę między oryginalnym sygnałem a jego przesuniętą w czasie wersją dla opóźnienia τ (4).

$$d_t(\tau) = \sum_{j=1}^{W} \left(x_j - x_{j+\tau}\right)^2 \tag{4}$$

Gdzie W – rozmiar okna analizy. Funkcja $d_t(\tau)$ minimalizuje błędy związane z zmianami amplitudy i dynamicznie dostosowuje się do charakterystyki analizowanego sygnału. W celu zapewnienia jednolitej analizy dla różnych wartości τ algorytm dokonuje normalizacji funkcji różnicowej (5).

$$d'_t(\tau) = \begin{cases} 1 & dla\ \tau = 0 \\ d_t(\tau) \Big/ \left[\frac{1}{\tau}\sum_{j=1}^{\tau} d_t(j)\right] & dla\ \tau \neq 0 \end{cases} \tag{5}$$

Unika w ten sposób błędnych estymacji wynikających z dominacji harmonicznych w sygnale. Następnie algorytm wprowadza stały próg wartości funkcji różnicowej (zazwyczaj 0.1) aby wyeliminować fałszywe minima i wybrać najmniejszą wartość τ spełniającą warunek (6).

$$d'_t(\tau) < 0.1 \tag{6}$$

W pobliżu wyznaczonego minimum funkcji różnicowej τ_{min}, YIN stosuje interpolację paraboliczną (7).

$$\tau_{interp} = \tau_{min} + \frac{d'_t(\tau_{min} - 1) - d'_t(\tau_{min} + 1)}{2\left(2d'_t(\tau_{min}) - d'_t(\tau_{min} - 1) - d'_t(\tau_{min} + 1)\right)} \tag{7}$$

Uzyskane wartości częstotliwości f_0 są dokładniejsze, nawet jeżeli rzeczywisty okres nie jest dokładnie wielokrotnością częstotliwości próbkowania. Ostatecznie wybierana jest najbardziej stabilna wartość częstotliwości podstawowej (8), spośród kolejnych ramek. Pozwala to na uniknięcie gwałtownych zmian estymowanej częstotliwości [9].

$$f_0 = \frac{f_s}{\tau_{interp}} \tag{8}$$

2.4.3. Average Magnitude Differential Function

Metoda Average Magnitude Difference Function (AMDF), podobnie jak ZCR oraz YIN, jest techniką estymacji częstotliwości podstawowej. Jako alternatywa do metod autokorelacyjnych, eliminuje konieczność wykonywania operacji mnożenia, co sprawia, że jest efektywniejsza obliczeniowo. Podobnie jak YIN, polega na porównywaniu oryginalnego sygnału z jego przesuniętą w czasie wersją z różnicą.

W przeciwieństwie do klasycznej funkcji autokorelacji, AMDF nie sumuje iloczynów wartości sygnału, a oblicza różnice amplitud między przesuniętym sygnałem a oryginałem za pomocą funkcji różnicowej wyrażonej wzorem (9).

$$d(\tau) = \frac{1}{L}\sum_{j=1}^{L}\left|x_j - x_{j+\tau}\right| \tag{9}$$

Gdzie L – długość analizowanego okna. Funkcja różnicowa osiąga minima dla wartości opóźnienia τ_{min}, które odpowiadają okresowi próbkowania sygnału T_s. Pozwala to, na estymację jego częstotliwości podstawowej (10) [10, 11].

$$f_0 = \frac{f_s}{\tau_{min}} = \frac{1}{\tau_{min}\, T_s} \tag{10}$$

2.5. Metody analizy w domenie częstotliwości

Przekształcenie sygnału czasowego w jego reprezentację widmową umożliwia identyfikację składowych harmonicznych i dominujących częstotliwości. Techniki analizujące sygnał w dziedzinie częstotliwości wykorzystują matematyczne operacje dekompozycji sygnału. Pozwala to na precyzyjne określenie struktury częstotliwościowej sygnału. Niektóre metody bazują na transformacjach, rozkładając sygnał na sinusoidalne komponenty, inne natomiast skupiają się na analizie widma sygnału i jego harmonicznych. Oferują dużą dokładność kosztem większej złożoności obliczeniowej.

2.5.1. FFT

Szybka transformata Fouriera (ang. *Fast Fourier Transform*) to algorytm pozwalający na obliczenie dyskretnej transformaty Fouriera (DFT) w efektywny sposób, redukując liczbę operacji z $O(N^2)$ do $O(N \log N)$. Została opracowana przez Jamesa W. Cooleya i Johna W, Tukeya w 1965 roku i do tej pory jest podstawą do analizy różnych sygnałów w dziedzinie częstotliwości [12].

DFT jest matematycznym narzędziem umożliwiającym przekształcenie sygnału z dziedziny czasu do domeny częstotliwości. Analizuje sygnał jako sumę sinusoidalnych składowych, pozwalając na określenie jakie częstotliwości są obecne w sygnale wraz z ich amplitudami i fazami. Dla sygnału dyskretnego w dziedzinie czasu x_n o długości N próbek, współczynniki Fouriera w dziedzinie częstotliwości uzyskuje się stosując wzór (11).

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}kn}, \quad k = 0,1, \dots, N-1 \tag{11}$$

Gdzie X_k – wartość w dziedzinie częstotliwości odpowiadająca indeksowi k, $e^{-j\frac{2\pi}{N}kn}$ – współczynniki reprezentujące bazowe funkcje sinusoidalne. Metoda ta jest nieefektywna dla dużych zbiorów danych, ze względu na wysoką złożoność obliczeniową. Dla każdego punktu widma X_k, których jest N, DFT sumuje N składników, co przekłada się na złożoność kwadratową.

FFT to zoptymalizowana wersja algorytmu transformaty dyskretnej. Działa w oparciu o paradygmat „dziel i zwyciężaj”, czyli rekurencyjnie dzieli problem na dwa mniejsze pod-problemy. Próbki są dzielone na dwie grupy o indeksach parzystych oraz o indeksach nieparzystych (12).

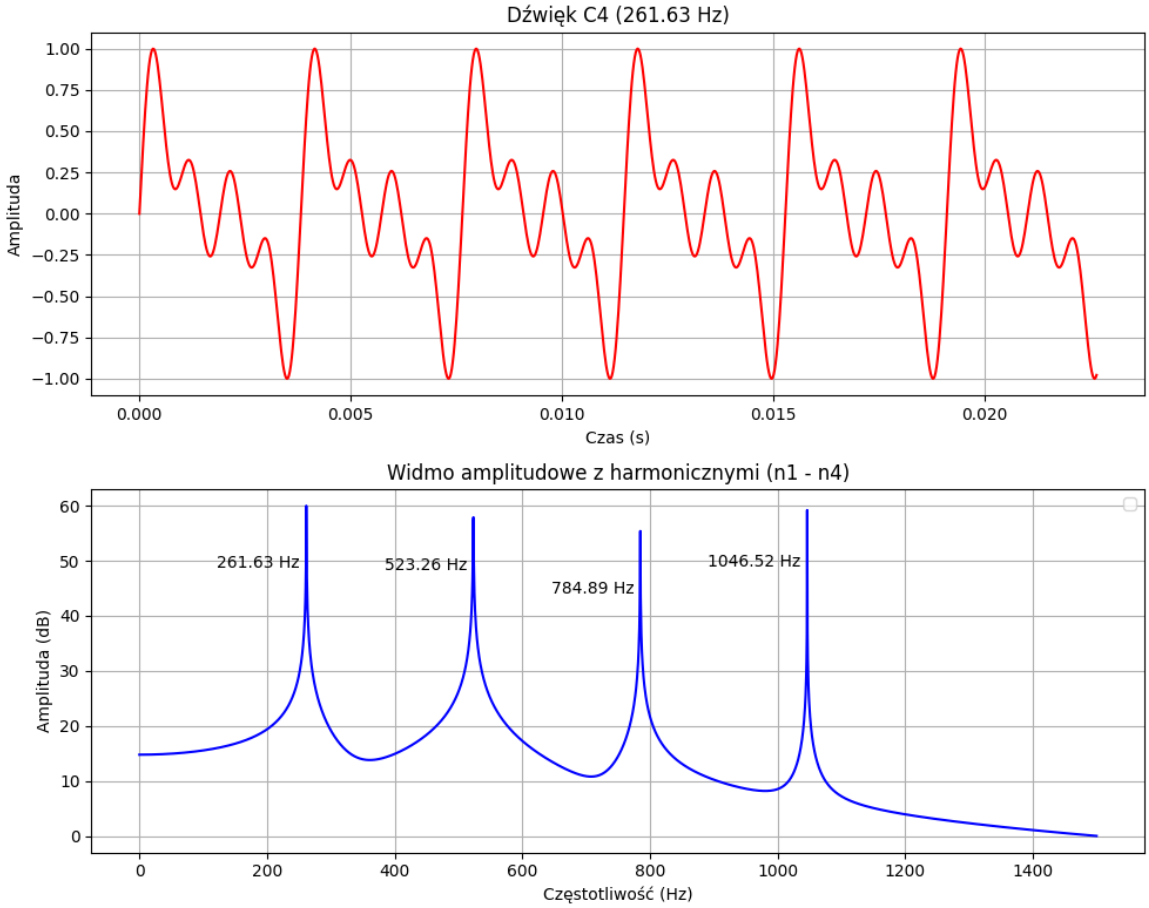
$$X_k = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} e^{-j\frac{2\pi}{N}2nk} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} e^{-j\frac{2\pi}{N}(2n+1)k} = E_k + e^{-j\frac{2\pi}{N}k} O_k, \quad k = 0, \dots \frac{N}{2} - 1 \tag{12}$$

Gdzie E_k – próbka o indeksie parzystym (ang. *even*), O_k – próbka o indeksie nieparzystym (ang. *odd*). Następnie, za pomocą operacji motylkowej (ang. *butterfly operation*) algorytm składa wyniki. Znając wartości E_k oraz O_k oblicza dwie składowe X o indeksach k oraz $k + \frac{N}{2}$ jak przedstawiono we wzorze (13).

$$\begin{aligned} X_k &= E_k + e^{-j\frac{2\pi}{N}k} O_k \\ X_{k+\frac{N}{2}} &= E_k - e^{-j\frac{2\pi}{N}k} O_k \end{aligned} \tag{13}$$

Jeżeli liczba próbek N jest potęgą dwójki, a N jest dzielone przez 2 aż do osiągnięcia $N = 1$, to liczba poziomów podziału rekurencyjnego wyniesie $\log_2 N$. Każdy poziom wykonuje N operacji motylkowych sumowania i mnożenia czynników, zatem łączna liczba operacji dla FFT wynosi $N \log_2 N$, co czyni ją znacznie szybszą, od metody DFT

o złożoności $O(N^2)$ [13]. Wynikiem zastosowania FFT na sygnale w dziedzinie czasu, jest zestawienie N współczynników X_k. Uzyskane składowe odpowiadają poszczególnym częstotliwościom w spektrum sygnału. W zależności od specyfiki sygnału uzyskuje się widma amplitudowe (rysunek 2.4) oraz fazowe o różnej symetryczności.



Rysunek 2.4. Porównanie sygnału w dziedzinach czasu i częstotliwości

Dla sygnałów zespolonych np. prądu przemiennego, o długości N próbek, próbkowanych z częstotliwością f_s, częstotliwość każdej k-tej składowej jest określana za pomocą wzoru (14).

$$f_k = \frac{k \cdot f_s}{N}, \quad k = 0,1 \dots N-1 \tag{14}$$

Widmo takich sygnałów jest niesymetryczne – dla dodatnich i ujemnych częstotliwości współczynniki amplitudowe oraz fazowe mogą się różnić. Aby uzyskać widmo amplitudowe, czyli wartości amplitud dla poszczególnych częstotliwości, należy obliczyć moduł ich składowych (15).

$$|X_k| = \sqrt{Re(X_k)^2 + Im(X_k)^2} \tag{15}$$

Gdzie Re – część rzeczywista, Im – część urojona współczynnika. Indeks k_{max} (16) najwyższej wartości uzyskanej amplitudy stanowi podstawę do wyznaczenia częstotliwości dominującej w sygnale (17).

$$k_{max} = \arg\max_k |X_k| \tag{16}$$

$$f_0 = \frac{k_{max} \cdot f_s}{N} \tag{17}$$

W przypadku sygnałów rzeczywistych, takich jak omawiany sygnał dźwiękowy, widmo uzyskane za pomocą FFT jest symetryczne i nie zawiera ujemnych częstotliwości. Z zależności (18) wynika potrzeba analizy częstotliwości jedynie z zakresu od 0 do $f_s/2$, czyli tak zwanej częstotliwości Nyquista.

$$X_k = X_{\frac{N}{2}+k}, \;\; k = 0, \ldots \frac{N}{2} - 1 \tag{18}$$

Częstotliwość każdej k-tej składowej sygnału rzeczywistego wyraża się za pomocą wzoru (19).

$$f_k = \frac{k \cdot f_s}{2N}, \;\; k = 0{,}1 \ldots \frac{N}{2} - 1 \tag{19}$$

Natomiast częstotliwości fundamentalnej odpowiada składowa k_{max} o najwyższej amplitudzie i można ją obliczyć korzystając ze wzoru (20).

$$f_0 = \frac{k_{max} \cdot f_s}{2N} \tag{20}$$

Algorytm FFT dobrze działa na dużych zbiorach danych oraz w czasie rzeczywistym ze względu na swoją niską złożoność obliczeniową. Sprawdza się jako narzędzie do analizy dźwięku audio, w szczególności pod względem analizy częstotliwości składowych sygnału, np. identyfikacji akordów. Jednakże analiza niskich częstotliwości, pod kątem wykrywania częstotliwości podstawowych, za pomocą tej metody jest problematyczna. Niska rozdzielczość częstotliwości w tym zakresie, prowadzić może do trudności w dokładnym wykrywaniu dźwięków w niskim paśmie, takich jak basowe struny gitary [14].

2.5.2. Okna czasowe

Podczas wykonywania DFT lub FFT na sygnale zakłada się, że jest on periodyczny (okresowy). Jednak w rzeczywistości analizowany jest tylko jego skończony fragment, a nagłe obcięcia sygnału mogą powodować powstawanie nowych, nieistniejących

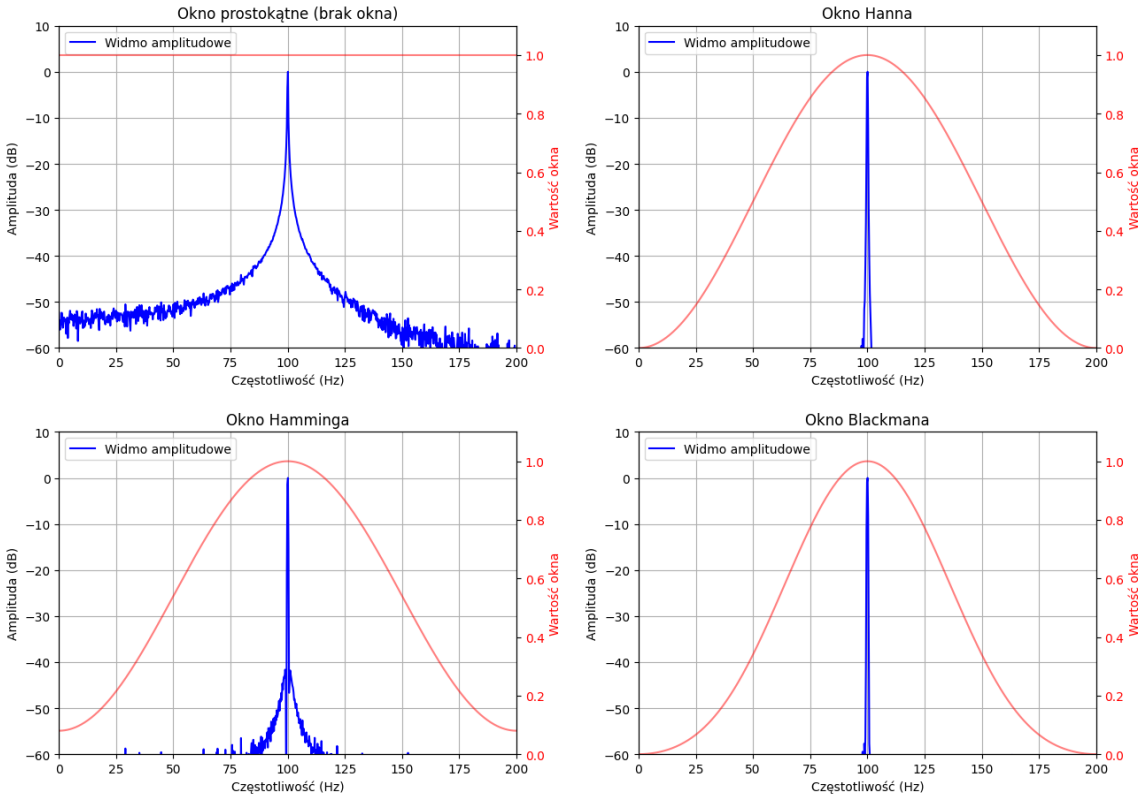
składowych częstotliwościowych, nazywanych wyciekami widma (ang. *spectral leakage*). W celu zredukowania tego efektu stosuje się okna czasowe (ang. *window functions*), czyli funkcje matematyczne, używane w analizie Fouriera [15].

Przy założeniu, że analizowany jest sygnał $u(n)$ w pewnym skończonym przedziale czasu o długości N, wynikiem tej analizy z użyciem funkcji okna $w(n)$ będzie sygnał $g(n)$ jak pokazano we wzorze (21).

$$g(n) = u(n) \cdot w(n), \quad -\infty < n < \infty \tag{21}$$

W zależności od postaci funkcji okna, widmo analizy $g(n)$ będzie różniło się od widma analizowanego sygnału $u(n)$. W kontekście analizy częstotliwościowej, pod kątem detekcji pojedynczych częstotliwości lub złożonych sygnałów takich jak akordy, wybór odpowiedniego okna jest kluczowy [16].

Na rysunku 2.5 przedstawiono wpływ funkcji okien czasowych z tabeli 2.6 na sygnał o częstotliwości podstawowej $f = 100\,Hz$ w postaci widm amplitudowych. Wykres w lewym górnym rogu pokazuje zastosowanie okna prostokątnego na sygnale, czyli funkcję równą jeden. Okno to, nie ma wpływu na sygnał, wykres został umieszczony jako punkt odniesienia dla pozostałych.



Rysunek 2.5. Porównanie wpływu funkcji wybranych okien czasowych na widmo amplitudowe dla częstotliwości sygnału $f = 100\,Hz$.

- Okno Hamminga (22) redukuje wyciek widma w porównaniu z oknem prostokątnym. Główna składowa częstotliwości pozostaje wyraźna, a pozostałe boczne prążki widma są zauważalnie tłumione. Ma stosunkowo wąską główną składową widma, przez co zapewnia dobrą dokładność lokalizacji częstotliwości.
- Funkcja Hanna (23) jest zbliżona do okna Hamminga, jednak tłumi wyciek widma w większym stopniu. Dodatkowo charakteryzuje się niższymi wartościami skrajnych prążków minimalizując zakłócenia.
- Okno Blackmana (24) zapewnia najsilniejsze tłumienie wycieku widma w porównaniu z pozostałymi analizowanymi oknami.

Tabela 2.6. Tabela wybranych funkcji okien czasowych

Okno czasowe	Funkcja okna
Okno Hamminga	$w(n) = 0.53836 - 0.46164\cos\left(\frac{2\pi n}{N-1}\right)$ (22)
Okno Hanna (Hanninga)	$w(n) = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right)$ (23)
Okno Blackmana	$w(n) = 0.42 - 0.5\cos\left(\frac{2\pi n}{N-1}\right) + 0.08\cos\left(\frac{4\pi n}{N-1}\right)$ (24)

W przypadku analizy pojedynczych dźwięków, częstotliwości fundamentalnych najlepiej sprawdzi się okno Hamminga, ze względu na dokładność lokalizacji częstotliwości. Natomiast do analizy złożonych sygnałów, zawierających wiele blisko położonych częstotliwości, lepszym wyborem będzie okno Hanna lub Blackmana, ponieważ zapewniają lepsze tłumienie zakłóceń bocznych.

2.5.3. HPS

Algorytm obliczania spektrów harmonicznych (ang. *Harmonic Product Spectrum*, HPS) zaprojektowany przez Nolla w 1969 roku jest jednym z najprostszych i najskuteczniejszych podejść do detekcji częstotliwości. Metoda ta, opiera się na analizie harmonicznych sygnału, co sprawia, że jest odpowiednim narzędziem do rozpoznawania wysokości dźwięku w muzyce.

Głównym zadaniem HPS jest wykrycie podstawowej częstotliwości odpowiadającej za dźwięk, co jest możliwe do osiągnięcia w szerokim zakresie warunków. Algorytm oblicza

wartości funkcji $Y_k(f_k)$ dla każdej próbki o indeksie k widma $X(f_k)$ o częstotliwości próbki f_k (14,19) w sposób przedstawiony we wzorze (25).

$$Y_k(f_k) = \prod_{r=1}^{R} |X_k(f_k \cdot r)|, \quad k = 0,1 \ldots \frac{N}{2} - 1 \tag{25}$$

Gdzie R – liczba rozważanych harmonicznych, N – liczba próbek sygnału. Częstotliwość widma, dla której wartość funkcji Y_k była największa to częstotliwość podstawowa (26).

$$f_0 = \arg\max_f Y(f) \tag{26}$$

Metoda HPS często wybiera częstotliwość dwa razy większą niż rzeczywista f_0, co powoduje błąd oktawowy. Sprawdzenie drugiej największej wybitności amplitudy poprawia dokładność HPS (27).

$$\text{Jeżeli } Y\left(\frac{f}{2}\right) > 0.2 \cdot Y(f), \text{ to wybierz } \frac{f}{2} \text{ jako } f_0 \tag{27}$$

Algorytm jest także wrażliwy na silne zakłócenia poza główną harmoniczną. Ze względu na niską rozdzielczość częstotliwościową dla niskich tonów, często stosuje się technikę Zero Padding (ZP) [17]. ZP polega na dodaniu zer na koniec sygnału przed wykonaniem FFT, w celu zwiększenia dokładności detekcji częstotliwości. Dla sygnału o długości N, zero padding dodaje $M - N$ zer, gdzie M to nowa długość sygnału będąca potęgą dwójki. W efekcie uzyskane widmo jest szersze w dziedzinie częstotliwości.

2.6. Analiza sygnału gitarowego

Sygnał gitarowy jest wynikiem drgających strun i powstałych przez nią fal dźwiękowych, które zostały zarejestrowane przez mikrofon, przetworniki gitary elektrycznej bądź inny audio rejestrator. Sygnał ten musi być odpowiednio przetworzony i przygotowany do dalszej analizy pod kątem wykrywania częstotliwości. W tym podrozdziale omówione zostaną zagadnienia związane z analizą dźwięku gitary.

Charakterystyka sygnału gitarowego jest złożona z częstotliwości podstawowych oraz harmonicznych. Te drugie nadają instrumentom gitarowym ich szczególne brzmienie. Każdy dźwięk zagrany na gitarze posiada swoją częstotliwość fundamentalną, która zależy od czynników, takich jak grubość, napięcie i długość struny. W wyniku wibracji struny powstają także częstotliwości harmoniczne tworzące złożony dźwięk. W zależności od siły oraz sposobu ataku struny, poprzez rękę grającą, sygnał dźwiękowy zmienia swoją

amplitudę. Oprócz pożądanych dźwięków, w sygnale mogą być zawarte także zakłócenia powstałe w wyniku stosowania konkretnych technik gitarowych, takich jak gra kostką.

Zakres częstotliwości dźwięków możliwych do uzyskania na gitarze zależy w głównej mierze od stroju otwartego instrumentu oraz od długości jego szyjki. Dla stroju standardowego w tonacji E2 (tabela 2.3.) dolny próg tego zakresu będzie wynosić od $82{,}4\ Hz$. Przyjmując, że na podstrunnicy gitary może znajdować od 20 do 24 progów, najwyższe częstotliwości fundamentalne do uzyskania w tym konkretnym przypadku wynoszą odpowiednio od $1046{,}50$ do $1318{,}51\ Hz$. Należy mieć na uwadze, że za pomocą technik artykulacyjnych takich jak bending, bądź elementów konstrukcyjnych gitar elektrycznych np., ruchomych mostków, z instrumentu można wydobyć jeszcze wyższe częstotliwości.

2.6.1. Detekcja częstotliwości

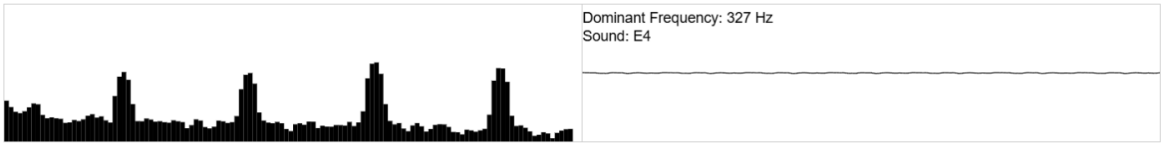
Aby skutecznie i efektywnie przeprowadzić detekcję częstotliwości podstawowej dźwięku gitary, należy odpowiednio przygotować sygnał pod analizę. Biorąc pod uwagę charakterystykę sygnału gitarowego, w szczególności zakres analizowanych częstotliwości, kluczowe jest odpowiednie dobranie wartości częstotliwości próbkowania zgodnie z twierdzeniem Nyquista-Shannona (28).

$$f_s \geq 2 \cdot f_{max} \tag{28}$$

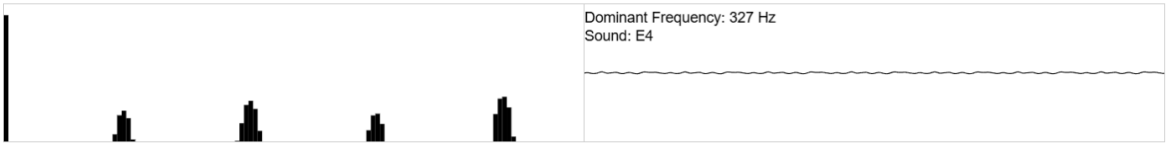
Maksymalną częstotliwością w analizowanym sygnale jest f_{max}. W praktyce stosuje się wyższe wartości, aby uniknąć efektu aliasingu i umożliwić lepszą filtrację. Typowe wartości stosowane w analizie audio to 8000, 16000, 44100 oraz $48000\ Hz$.

Sygnał dźwiękowy odebrany bezpośrednio ze źródła, bez odpowiedniej filtracji, może zawierać zakłócania w postaci szumu oraz częstotliwości nieistotne z punktu widzenia analizy. W celu oczyszczenia sygnału stosuje się filtry dolnoprzepustowe (LPF), które eliminują składowe częstotliwości powyżej interesującego pasma, oraz filtry górnoprzepustowe (HPF), które usuwają składowe niskich częstotliwości, takie jak szum wynikający z drgań strun. Filtr Butterwotha cechuje się maksymalnie płaską charakterystyką amplitudową w paśmie przepustowym. Oznacza to, że sygnał zachowuje jednolitą amplitudę aż do częstotliwości odcięcia, a po jej przekroczeniu tłumienie narasta w monotoniczny sposób. Kaskadowe połączenie dwóch filtrów LPF i HPF drugiego rzędu zapewnia tłumienie na poziomie $-24\ dB$ na oktawę, czyli redukcję amplitudy sygnału szesnastokrotnie na każde podwojenie częstotliwości. Taka konfiguracja filtru Butterwortha dobrze nadaje się do analizy częstotliwościowej sygnału, ponieważ minimalizuje zniekształcenia i usuwa niepożądane składowe.

Przed przystąpieniem do analizy sygnału w domenie częstotliwości, dobrą praktyką jest modyfikacja sygnału poprzez funkcję okna czasowego. Funkcja ta minimalizuje efekt wycieku widma, który powstaje przy obliczaniu FFT na skończonym fragmencie sygnału co przedstawiono na rysunkach 2.6 oraz 2.7.



Rysunek 2.6. Widmo amplitudowe dźwięku E4 bez okna czasowego



Rysunek 2.7. Widmo amplitudowe dźwięku E4 z oknem Blackmana

Na lewej stronie rysunku 2.6 przedstawiono wykres widma amplitudowego dźwięku E4 zagranego na pierwszej strunie gitary elektrycznej. Pierwsza wybitność od lewej strony odpowiada amplitudzie częstotliwości podstawowej, natomiast kolejne dotyczą jej harmonicznych. Efekt użycia okna Blackmana widnieje na następnym rysunku 2.7, gdzie widmo pozbawione jest wycieku, czyli nieznaczących częstotliwości. W obu przypadkach, analizowany jest fragment FFT, zawierający tylko częstotliwości w przedziale od $f_{min} = 50$ do $f_{max} = 1500\ Hz$. W tym celu określono pierwszy oraz ostatni indeks tablicy FFT za pomocą wzoru (29):

$$k_{min} = \left\lfloor \frac{f_{min} \cdot 2N}{f_s} \right\rfloor, \qquad k_{max} = \left\lceil \frac{f_{max} \cdot 2N}{f_s} \right\rceil \tag{29}$$

Prawą część rysunków stanowi wykres sygnału w dziedzinie czasu, zawierający informację o zidentyfikowanym dźwięku. Rozpoznanie częstotliwości podstawowych, wyłącznie na podstawie FFT i wysokości amplitud, może prowadzić do błędnego wskazania harmonicznych – drugiej w przypadku rysunku 2.6 lub trzeciej na rysunku 2.7. W celu poprawnego rozpoznawania częstotliwości podstawowych, szczególnie w obecności silnych harmonicznych, należy stosować algorytmy takie jak HPS, który wzmacnia częstotliwość fundamentalną poprzez analizę wielokrotnie przeskalowanych widm. Najlepsze rezultaty uzyskuje w podejściach hybrydowych, łączących metody dziedziny częstotliwości z funkcjami domeny czasowej, takimi jak YIN czy AMDF, co zostało zastosowane w tym przypadku.

2.6.2. Osadzenie dźwięku na tabulaturze

Proces osadzenia dźwięku na tabulaturze zaczyna się od przekształcenia rozpoznanej częstotliwości w dźwięk na skali muzycznej o określonej oktawie. Dopiero za pomocą uzyskanej w ten sposób notacji, można wskazać konkretne miejsca występowania tonu na podstrunnicy. Aby uzyskać grywalny zapis tabulatorowy należy uwzględnić czynniki takie jak punkt początkowy oraz optymalne rozmieszczenie dźwięków.

W celu określenia dźwięku, jaki odpowiada danej częstotliwości, zazwyczaj stosuje się system równomiernie temperowany, w którym oktawa dzieli się na dwanaście części. Cechuje się tym, że odstępy pomiędzy kolejnymi półtonami są jednakowe w skali logarytmicznej. Częstotliwość dźwięku f, względem dźwięku odniesienia f_0 w tym systemie wyraża się wzorem (30).

$$f = f_0 \cdot 2^{\frac{n}{12}} \tag{30}$$

Gdzie n to liczba półtonów pomiędzy dźwiękami f_0 oraz f. Standardowo przyjmuje się, że częstotliwość dla dźwięku A4 wynosi $440\,Hz$ i traktuje się ją jako częstotliwość odniesienia f_0. Z wykorzystaniem wzoru (31) uzyskuje się odstęp tonalny pomiędzy analizowanymi częstotliwościami.

$$n = round\left(12 \cdot log_2\left(\frac{f}{f_0}\right)\right) \tag{31}$$

Aby otrzymać dźwięk dla danej częstotliwości, należy przesunąć się o n tonów względem A4, tak jak przedstawiono w tabeli 2.7. Na dźwięk w tym systemie składa się również oktawa (32), która jest wynikiem dzielenia całkowitego odstępu tonalnego przez liczbę półtonów.

$$o = \left\lfloor \frac{n}{12} \right\rfloor \tag{32}$$

Tabela 2.7. Tabela odstępów tonalnych względem dźwięku A

Oktawa	**A**	**A#**	**B**	**C**	**C#**	**D**	**D#**	**E**	**F**	**F#**	**G**	**G#**
o-1	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1
o	0	1	2	3	4	5	6	7	8	9	10	11
o+1	12	13	14	15	16	17	18	19	20	21	22	23

W zależności od stroju gitary, dany dźwięk można zagrać na kilku różnych strunach (tabela 2.8). Przyjmując dźwięk ze stroju otwartego za punkt odniesienia i korzystając ze wzoru na odległość tonalną (31), można obliczyć wartość progu, na którym znajduje się szukany dźwięk dla danej struny. Ze względu na fizyczne ograniczenia instrumentu należy sprawdzić, czy uzyskany numer progu nie wykracza poza podstrunnicę – wartości

mniejsze od 0 lub większe od numeru ostatniego progu wykluczają występowanie dźwięku.

Tabela 2.8. Tabela dźwięków pierwszej oktawy dla stroju E-Standard

Struna \ Próg	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
6	E2	F2	F#2	G2	G#2	A2	A#2	B2	C3	C#3	D3	D#3
5	A2	A#2	B2	C3	C#3	D3	D#3	E3	F3	F#3	G3	G#3
4	D3	D#3	E3	F3	F#3	G3	G#3	A3	A#3	B3	C4	C#4
3	G3	G#3	A3	A#3	B3	C4	C#4	D4	D#4	E4	F4	F#4
2	B3	C4	C#4	D4	D#4	E4	F4	F#4	G4	G#4	A4	A#4
1	E4	F4	F#4	G4	G#4	A4	A#4	B4	C5	C#5	D5	D#5

Problem wyboru najlepszej pozycji dla danego dźwięku względem poprzedniego, można potraktować jako warstwowy graf skierowany, w którym wierzchołki (33) to możliwe pozycje dźwięków W_k, a krawędzie to koszt przejścia między nimi.

$$W_k = (s_k, n_k) \tag{33}$$

Gdzie s_k – numer struny, n_k – numer progu. Koszt przejścia pomiędzy dwoma kolejnymi pozycjami P_k oraz P_{k+1} można przedstawić za pomocą wzoru (34).

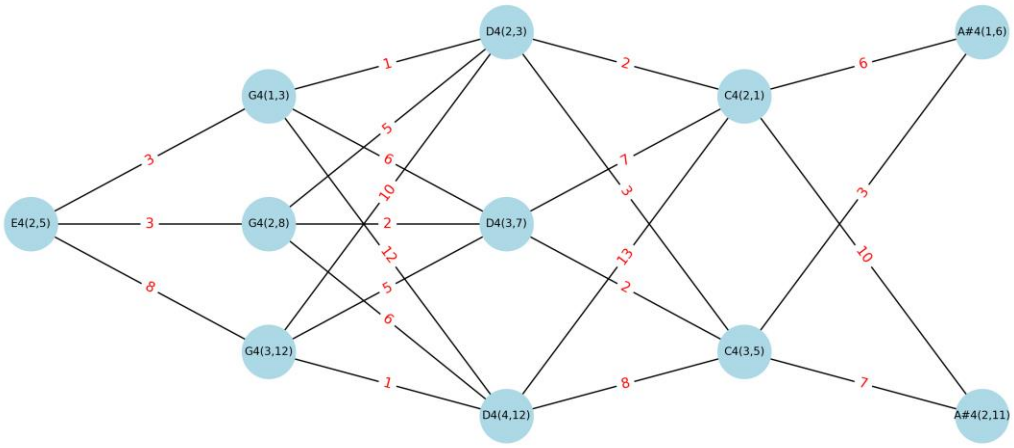
$$\mathrm{c}(W_k, W_{k+1}) = w_n \cdot |n_{k+1} - n_k| + w_s \cdot |s_{k+1} - s_k| + D(s_k, s_{k+1}, n_k, n_{k+1}) \tag{34}$$

Gdzie w_n – waga odległości progowej, w_s – waga zmiany struny, D – dodatkowa kara za trudniejsze przejścia. Graf swoją strukturą przypomina sieć neuronową – posiada warstwy wszystkich dźwięków z sekwencji, które zawierają wierzchołki będącymi możliwościami występowania dźwięku. Każdy wierzchołek jest skierowany na wszystkie wierzchołki warstwy następnej. W ten sposób graf opisuje wszystkie możliwości zanotowania całych kolejnych zestawów dźwięków z różnymi wagami przejść.

Problem wyboru najlepszej pozycji dźwięku, można rozwiązać za pomocą algorytmu do znajdowania najkrótszej drogi w grafie, oznaczającej optymalne przejście pomiędzy dźwiękami. Algorytm Dijkstry stosuje się, do znajdywania najkrótszej drogi (o najmniejszym koszcie) pomiędzy wierzchołkiem początkowym W_0 oraz wszystkimi innymi wierzchołkami. Wierzchołek początkowy inicjowany jest z wartością 0, podczas gdy pozostałym wierzchołkom przypisywane są nieskończoności. Następnie algorytm iteruje po krawędziach, oblicza koszt przejścia i aktualizuje odległość, jeżeli nowa ścieżka jest krótsza niż dotychczasowa. Operacje te są powtarzane do momentu, aż wszystkie wierzchołki zostaną odwiedzone a ich długości ścieżek obliczone [18]. W praktyce

wynikiem działania algorytmu Dijkstry mogą być wszystkie możliwe sekwencje wierzchołków z obliczonym kosztem przejścia.

Na rysunku 2.8 przedstawiono sekwencję dźwięków w stroju E-Standard w postaci skierowanego grafu. Wierzchołkowi początkowemu odpowiada dźwięk E4, zagrany na piątym progu drugiej struny. Kolejne punkty reprezentują możliwe pozycje dla dźwięków G4, D4, C4 oraz A#4. Linie łączące wierzchołki stanowią krawędzie z obliczonym kosztem przejścia, gdzie $w_n = w_s = D = 1$.



Rysunek 2.8. Przykładowy skierowany graf możliwych pozycji dla sekwencji dźwięków

W ogólnym przypadku ścieżka o najniższym koszcie $W_0(s_0, n_0) \rightarrow W_1(s_1, n_1) \rightarrow \cdots \rightarrow W_k(s_k, n_k)$ odpowiada kolejnym parom (s_k, n_k) stanowiącym współrzędne dźwięku na podstrunnicy. Dla przykładu na rysunku 2.8, algorytm Dijkstry wyznaczy drogę: $E4(2,5) \rightarrow G4(1,3) \rightarrow D4(2,3) \rightarrow C4(3,5) \rightarrow A\#4(1,6)$, jako najszybszą o łącznym koszcie ścieżki równym 10. Wynikową tabulaturę dla tej sekwencji dźwięków przedstawia rysunek 2.9.

E| ---------- ------ **3** ------ ---------- ---------- ------ **6** ------
B| ------ **5** ------ ---------- ------ **3** ------ ---------- ----------
G| ---------- ---------- ---------- ------ **5** ------ ----------
D| ---------- ---------- ---------- ---------- ----------
A| ---------- ---------- ---------- ---------- ----------
E| ---------- ---------- ---------- ---------- ----------

Rysunek 2.9. Tabulatura o optymalnym koszcie przejść

Rozdział 3

Wymagania i narzędzia

3.1. Wymagania funkcjonalne i niefunkcjonalne

W procesie projektowania aplikacji ważnym etapem jest szczegółowe zdefiniowanie wymagań, które zostaną wdrożone podczas implementacji. Dokładne określenie potrzeb zapewnia odpowiednie zaplanowanie struktury systemu, co wpływa na jego stabilność, skalowalność i bezpieczeństwo. Jasno postawione cele pozwalają uniknąć nieporozumień podczas implementacji, dostarczając końcowy produkt zgodnie oczekiwaniami użytkownika. W tym podrozdziale szczegółowo omówiono wymagania funkcjonalne i niefunkcjonalne dla aplikacji webowej TabComposer, będącej edytorem tabulatur gitarowych

3.1.1. Wymagania funkcjonalne

Funkcje i operacje oferowane przez aplikację są nazywane wymaganiami funkcjonalnymi. Opisują one konkretne działania i interakcje, które umożliwia system a użytkownik może wykonać. Zaliczają się do nich główne mechanizmy systemu dotyczące jego funkcjonalności. W przypadku edytora tabulatur gitarowych, do wymagań funkcjonalnych projektu należą rejestracja i logowanie użytkowników, treści generowane przez użytkowników oraz edytor tabulatur.

- **Logowanie i rejestracja**

Aplikacja TabComposer umożliwia tworzenie kont użytkowników za pomocą loginu (nazwy użytkownika), adresu e-mail oraz silnego hasła, poprzez formularz znajdujący się na stronie. Nazwy użytkowników oraz adresy e-mail w systemie są unikalne. Zasady tworzenia haseł obejmują minimalną długość 6 znaków, wymagana jest co najmniej jedna cyfra, znak niealfanumeryczny oraz wielka litera. W przypadku błędnie wprowadzonych

danych aplikacja informuje użytkownika o szczegółach, za pomocą komunikatu zamieszczonego w formularzu.

- **Treści generowane przez użytkowników**

Zalogowani użytkownicy mają pełny dostęp do tworzenia, edytowania, usuwania i przechowywania tabulatur w aplikacji. Każda tabulatura składa się z metadanych, zawierających informacje o tytule utworu i autorze, opcjonalny opis tabulatury oraz datę utworzenia projektu muzycznego. Do danych muzycznych należy strojenie instrumentu, liczba dostępnych progów i zestawienie dźwięków w systemie taktów reprezentowanych w strukturze JSON. Wartości rytmiczne nut obejmują wartości 1, 2, 4, 8, 16 oraz 32 (tabela 2.4). Każda tabulatura jest powiązana tylko z użytkownikiem, który ją stworzył.

- **Edytor tabulatury**

Najważniejszą funkcjonalnością aplikacji jest edytor tabulatur gitarowych. Pozwala użytkownikom na wizualizację tabulatury w postaci responsywnego i interaktywnego widoku. Umożliwia modyfikację lub usuwanie istniejących zapisów oraz dodawanie nowych w szerokiej skali. Wersja demonstracyjna edytora jest również dostępna dla niezalogowanych użytkowników.

Z edytorem zintegrowany jest pasek narzędzi zawierający dodatkowe ustawienia w trzech odrębnych zakładkach. Globalne ustawienia definiują wartości takie jak domyślne tempo, wartości metrum oraz długości nut i interwały pomiędzy nimi. Odtwarzacz tabulatury pozwala na monitorowanie postępów, oferując odsłuch napisanego utworu, natomiast analizator dźwięku umożliwia wspomaganie układania tabulatury poprzez analizę dźwięku instrumentu.

Nowy takt, może być dodany jako pusty, bądź jako kopia wybranego, istniejącego już taktu. Metrum oraz tempo określane są dla każdego taktu osobno i mogą być odpowiednio modyfikowane. Nuty umiejscawiane są na pierwszej możliwej pozycji danej struny taktu. Parametry nut, takie jak wartości progów, długości dźwięku, interwał przesunięcia oraz technika artykulacyjna podlegają modyfikacji. Edytor umożliwia również edycję metadanych tabulatury.

Wszystkie błędy, związane z nieprawidłowym wprowadzeniem danych są komunikowane w odpowiednich dla danej operacji miejscach. Błędy te obejmują w głównej mierze aspekty dotyczące poprawności zapisu rytmicznego oraz umiejscawiania dźwięków.

- **Odtwarzacz tabulatury**

Tabulatura może być odtwarzana w celu kontroli efektu dotychczasowej pracy bądź nauki napisanego utworu. W pasku narzędzi dostępne są przyciski służące do odtwarzanie piosenki od początku bądź od konkretnego taktu a także pauzowanie oraz stopowanie

utworu. Dodatkowo odtwarzacz obsługuje suwak prędkości odtwarzania kolejnych dźwięków. Aktualnie grane nuty są podświetlane na tabulaturze.

- **Analizator dźwięku**

Funkcja analizy dźwięku instrumentu umożliwia rozpoznawanie dźwięków granych w czasie rzeczywistym, które następnie nanoszone są na tabulaturę w postaci nut. W pasku narzędzi dostępne są przyciski służące do rozpoczęcia i przerwania nagrywania, a także do wyboru dostępnego mikrofonu, odsłuchu jakości dźwięku i włączenia efektów. Dodatkowo istnieje możliwość wyświetlenia wykresu widma amplitudowego analizowanego sygnału oraz wykresu sygnału w dziedzinie czasu.

3.1.2. Wymagania niefunkcjonalne

Cechy systemu, które nie są bezpośrednio związane z jego funkcjonalnością, a określają jakość działania aplikacji nazywa się wymaganiami niefunkcjonalnymi. Obejmują one aspekty wydajności, bezpieczeństwa, estetyki oraz użyteczności danego systemu. Dla aplikacji TabComposer kluczowymi wymaganiami niefunkcjonalnymi są: bezpieczeństwo danych, wydajność aplikacji, responsywny design, jednolity styl i wygląd, czytelność i prostota obsługi, system automatycznego zapisu postępów prac i architektura przystosowana do rozbudowy aplikacji.

- **Bezpieczeństwo danych**

Aplikacja przechowuje oraz ma dostęp do poufnych danych użytkowników, takich jak hasła, adresy e-mail oraz dane tabulatur. Z uwagi na wrażliwość tych informacji, system musi zapewnić odpowiednią ochronę. Dane podczas transmisji są szyfrowane z wykorzystaniem protokołu HTTPS. Ponadto, system zapewnia mechanizm uwierzytelniania i autoryzacji za pomocą tokenów JWT. Każda poufna operacja, taka jak logowanie, uzyskanie dostępu do danych tabulatur musi być autoryzowana za pomocą specjalnie wygenerowanego tokenu o określonym czasie działania. Hasła przechowywane w bazie danych są odpowiednio haszowane dzięki wykorzystaniu frameworka ASP.NET Identity.

- **Wydajność aplikacji**

System zapewnia interaktywny edytor tabulatury z wieloma funkcjami, w tym odtwarzanie i nagrywanie w czasie rzeczywistym. Aby zapewnić użytkownikowi najlepszą jakość korzystania z aplikacji, musi ona działać płynnie i responsywnie. Wymagania wydajności obejmują minimalizację latencji, czyli opóźnień przy przetwarzaniu dźwięku, a także optymalizację kodu i efektywne zarządzanie zasobami. W tym celu używane są funkcje asynchroniczne do ładowania danych oraz reaktywna biblioteka React.ts, pozwalająca na ponowne renderowanie tylko wymaganych części widoków aplikacji.

- **Responsywny design**

Aplikacja jest przystosowana do działania na wielu urządzeniach i ekranach z zróżnicowaną rozdzielczością poprzez elastyczny interfejs. Zastosowano techniki projektowania responsywnego z wykorzystaniem biblioteki React Bootstrap. Różne komponenty wyświetlane na ekranie, takie jak panel nawigacyjny, dopasowują się odpowiednio do szerokości ekranu zapewniając czytelność i łatwą obsługę aplikacji.

- **Jednolity styl i wygląd**

Wymagania w zakresie estetyki aplikacji obejmują spójność kolorystyczną wraz z układem graficznym całego interfejsu. Jednolity styl ułatwia nawigację oraz korzystanie z systemu.

- **Czytelność i prostota obsługi**

Szybkie opanowanie wszystkich funkcjonalności systemu może zapewnić, że użytkownik będzie chętniej i dłużej korzystał z aplikacji. Aplikacja jest intuicyjna i prosta w użyciu, układ interfejsu jest przejrzysty i logiczny, a przyciski są wyraźnie oznaczone. System jasno komunikuje błędy w przypadku nieprawidłowych operacji.

- **Mechanizmy automatycznego zapisu**

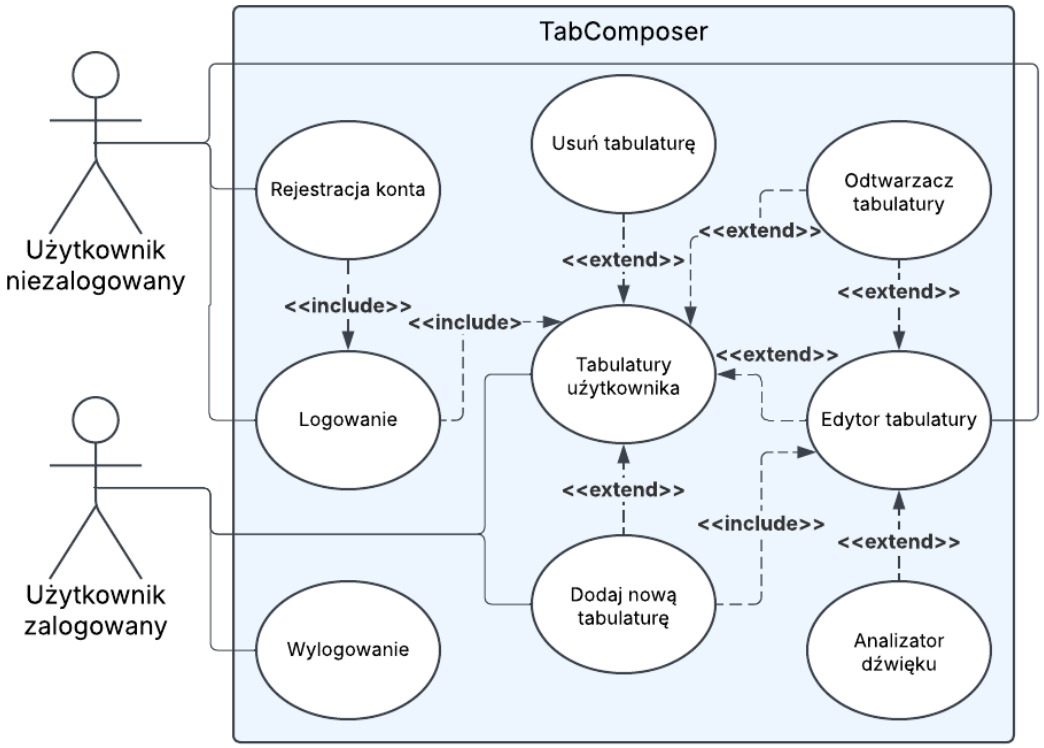
Aby zapobiec utracie postępów w pracy nad tabulaturą, aplikacja automatycznie zapisuje zmiany dokonywane w edytorze tabulatur. Obejmuje to mechanizm autosave, który w regularnych odstępach czasowych zapisuje wprowadzone zmiany do bazy danych. Dodatkowo użytkownik informowany jest o przebiegu zapisu.

- **Architektura przystosowana do rozbudowy**

Projekt ma predyspozycje do rozbudowy o nowe funkcjonalności. Jest przygotowany pod rozwój z uwagi na jego modularność i elastyczność. Warstwa frontendu jest oddzielona od backendu przy użyciu architektury REST API (ang. *Representational State Transfer Application Programming Interface*). System jest zaprojektowany w oparciu o wzorce projektowe takie jak kompozycja komponentów dla frontendu oraz architektura warstwowa dla backendu.

3.2. Przypadki użycia aplikacji

Różne interakcje z systemem modeluje się poprzez przypadki użycia (ang. *Use Cases*). Określona akcja wykonana przez aktora, np. użytkownika jest przypadkiem pojedynczego scenariusza, na który system odpowiada zgodnie z ustaloną logiką. Modelowanie przypadków użycia określa zbiór zachowań systemu i zależności pomiędzy jego aktorami. Przypadki użycia aplikacji TabComposer przedstawiają interakcje użytkowników z systemem w zakresie rejestracji oraz zarządzania tabulaturami, w tym edycji, odtwarzania i nagrywania. Diagram przypadków użycia w języku UML (ang. *Unified Modeling Language*) został przedstawiony na rysunku 3.1.



Rysunek 3.1. Diagram przypadków użycia aplikacji TabComposer

Użytkownicy oraz inne podmioty, takie jak zewnętrzne systemy, wchodzące w interakcję z aplikacją, nazywane są aktorami. Posiadają różne role i uprawnienia, które określają zakres działań wykonywanych w systemie. W aplikacji TabComposer aktorzy definiują podział funkcjonalności między użytkownikami a systemem.

- Niezalogowany użytkownik – aktor o ograniczonych możliwościach. Może przeglądać stronę główną aplikacji, skorzystać z edytora w trybie demonstracyjnym oraz założyć konto bądź zalogować się do systemu.
- Zalogowany użytkownik – aktor posiadający pełen dostęp do funkcji aplikacji, takich jak tworzenie, edycja, nagrywanie, odtwarzanie oraz usuwanie tabulatur.

3.3. Opis narzędzi i technologii

Aplikacja TabComposer została zaprojektowana i zaimplementowana z wykorzystaniem nowoczesnych narzędzi i technologii. W tym podrozdziale przedstawiono opis najważniejszych technologii użytych w projekcie.

- **Środowisko programistyczne**

Do pracy nad projektem użyto Visual Studio Community 2022, które stanowiło zintegrowane środowisko deweloperskie dla obu części aplikacji. Wybór ten umożliwił płynne zarządzanie projektami ASP.NET Core dla backendu oraz React z TypeScript na platformie Node.js dla frontendu. W celu zarządzania zależnościami wykorzystano menedżery pakietów, takie jak NuGet oraz npm. Dodatkowo, środowisko pozwoliło na wgląd do bazy danych za pomocą eksploratora SQL Server.

- **Backend**

Część backendowa aplikacji została opracowana w języku C# przy użyciu platformy ASP.NET Core. Z uwagi na potrzebę efektywnego zarządzania danymi, wykorzystano Entity Framework Core (EF Core), umożliwiający proste mapowanie obiektów na relacyjne tabele w bazie danych SQL Server. Innym frameworkiem użytym w aplikacji jest ASP.NET Identity, który służy do zarządzania użytkownikami. Jest to przydatne narzędzie w zakresie bezpieczeństwa – zapewnia haszowanie haseł oraz autoryzację za pomocą tokenów JWT. Backend działa jako REST API, wystawiające endpointy umożliwiające frontendowi wymianę danych z aplikacją w postaci plików JSON. Do testowania działania aplikacji wykorzystano narzędzie Swagger, umożliwiające wykonywanie żądań http.

- **Frontend**

Frontend aplikacji został zaimplementowany z użyciem React na platformie Node.js. Jako język programowania wybrano TypeScript, ze względu statyczne typowanie poprawiające czytelność kodu. Wybór ten ułatwił także zarządzanie złożonymi komponentami aplikacji. Projekt frontendowy został zoragnizowany w postaci aplikacji jednostronicowej (SPA) zapewniając płynne i dynamiczne interakcje użytkownika. Do zarządzania stanem aplikacji użyto biblioteki mobx, co znacznie ułatwiło monitorowanie zmian złożonych obiektów i natychmiastowe aktualizacje widoku. Do komunikacji z backendowym API wybrano bibliotekę Axios, umożliwiającą asynchroniczne wykonywanie żądań http. Natomiast za obsługę urządzeń wejściowych audio oraz analizę dźwięku odpowiada biblioteka Tone.js.

3.4. Metodyka pracy nad projektem

Praca nad projektem TabComposer była realizowana w sposób przyrostowy. Funkcje i elementy aplikacji były stopniowo implementowane i testowane na bieżąco. Projekt podzielono na kilka etapów, z których każdy skupiał się na konkretnych zadaniach. W miarę postępu prac, niektóre części systemu były odpowiednio modyfikowane i optymalizowane.

W ramach pierwszego etapu zaimplementowano część backendową aplikacji. Skonfigurowano serwis EF Core, z podejściem Code First co pozwoliło na łatwe utworzenie struktury bazy danych. Następnie wdrożono ASP.NET Identity wraz z mechanizmem autoryzacji użytkowników z wykorzystaniem JWT. Stworzono kontroler API służący do logowania i rejestracji użytkowników. Postępy testowano za pomocą narzędzia Swagger.

Drugi etap obejmował utworzenie i konfigurację części frontendowej aplikacji a także jej wymianę danych z backendem. Zaimplementowano mechanizmy logowania oraz rejestracji a także przechowywania danych użytkownika w przeglądarce. Po zakończeniu fazy testów integracji z API, przystąpiono do zamodelowania tabulatury i opracowania systemu do jej tworzenia oraz edycji. Porównano podejścia do zarządzania stanem komponentów, które oferowane były przez natywne rozwiązania React oraz zewnętrzną bibliotekę mobx opartą o wzorzec projektowy obserwatora. Następnie zaprojektowano serwisy serializacji danych tabulatury do formatu JSON po stronie frontendowej oraz backendowej aplikacji. Etap ten zakończył się obszernym badaniem i testowaniem działania oraz zapisywania tabulatur do bazy danych.

W dalszych etapach zaimplementowano mechanizm autosave, służący do cyklicznego zapisywania postępów. Poprawiono implementację logowania oraz przechowywania danych użytkownika. Kolejnym zadaniem było utworzenie odtwarzacza tabulatury oraz analizatora audio, który umożliwił zapis rozpoznanych nut na tabulaturze. W końcowej fazie dopracowano detale interfejsu użytkownika oraz wprowadzono drobne optymalizacje.

Rozdział 4

Specyfikacja zewnętrzna

4.1. Wymagania sprzętowe i programowe

Spełnienie określonych wymagań sprzętowych i programowych zapewni płynne i bezproblemowe działanie aplikacji TabComposer. Wymagania te dotyczą użytkowników końcowych korzystających z aplikacji przez przeglądarkę oraz administratorów lub deweloperów uruchamiających wersję lokalną.

- **Wymagania sprzętowe**

W tabeli 4.1 przedstawiono minimalne oraz zalecane wymagania sprzętowe dla urządzeń użytkowników końcowych, obejmujące procesor, ilość pamięci RAM, system operacyjny oraz wymagania dotyczące karty dźwiękowej.

Tabela 4.1. Wymagania sprzętowe

Komponent	Minimalne wymagania	Zalecane wymagania
Procesor (CPU)	2-rdzeniowy, 2.0 GHz	4-rdzeniowy, 3.0 GHz
Pamięć Ram	4 GB	8 GB
Karta dźwiękowa	Brak	Zintegrowana / ASIO
System operacyjny	Windows 10 / Linux	Windows 10 / Linux

Brak karty dźwiękowej w urządzeniu skutkuje brakiem dostępu do funkcjonalności aplikacji związanej z odtwarzaniem oraz przetwarzaniem dźwięku. Wymagane są również urządzenia odtwarzające i rejestrujące dźwięk, takie jak głośniki i mikrofon, które są niezbędne do prawidłowego działania niektórych części aplikacji.

- **Wymagania programowe**

Wymagania programowe niezbędne do poprawnego działania aplikacji przedstawione są w tabeli 4.2. Dla użytkownika końcowego kluczowym i jedynym zaleceniem jest dobór odpowiedniej przeglądarki, natomiast dla dewelopera są to wymagania dotyczące wersji platform deweloperskich.

Tabela 4.2. Wymagania programowe

Program	Zalecana wersja
Przeglądarka	Obsługująca Web Audio Api, wspierająca ESM (Chrome 90+, Firefox 90+, Edge 90+)
Platforma .NET	8.0.400
Baza danych	SQL Server 19
Platforma Node.js / menedżer npm	20.17.0 / 10.8.2

4.2. Instalacja i aktywacja

TabComposer działa jako aplikacja webowa, dlatego nie wymaga instalacji. Teoretycznie, aby skorzystać z programu w przyszłości, należało będzie odwiedzić odpowiedni adres internetowy, pod którym aplikacja zostanie uruchomiona. Na moment tworzenia niniejszej pracy, aplikacja nie jest jeszcze dostępna w sieci.

Istnieje jednak możliwość uzyskania wersji lokalnej aplikacji dla zaawansowanych użytkowników. W tym celu niezbędne jest spełnienie wszystkich wymagań programowych z tabeli 4.2. W pierwszej kolejności należy uzyskać kod źródłowy z repozytorium GitHub, odwiedzając stronę https://github.com/PDrzazgaGit/TabComposerApp, albo poprzez środowisko programistyczne Visual Studio. Następnie za pomocą konsoli należy zainstalować zależności i uruchomić projekt backendowy oraz projekt frontendowy.

```
C:\>     cd /ścieżka/do/katalogu/TabComposerApp
\TabComposerApp>      dotnet restore
\TabComposerApp>      dotnet build
\TabComposerApp>      dotnet run
\TabComposerApp>      cd TabComposerApp.client
\TabComposerApp.client>      npm install
\TabComposerApp.client>      npm start
```

Aby korzystać z pełnej funkcjonalności aplikacji TabComposer, użytkownik musi się zarejestrować, podając nazwę użytkownika, adres e-mail oraz hasło zgodne z polityką bezpieczeństwa.

4.3. Obsługa aplikacji

Aplikacja TabComposer została zaprojektowana z myślą o intuicyjnej i prostej obsłudze dla użytkowników.

4.3.1. Użytkownik niezalogowany

Funkcjonalność aplikacji dla użytkownika nieposiadającego konta bądź niezalogowanego jest ograniczona. Punktem otwarcia jest strona główna, zawierająca krótki opis projektu, w tym instrukcję obsługi aplikacji oraz wiadomość od autora. Na górze ekranu, pod paskiem adresowym przeglądarki znajduje się panel nawigacyjny, składający się z linków: „TabComposer", „Try Editor" oraz „Sign In". Pierwszy z nich przekierowuje na stronę główną projektu. Drugi otwiera wersję demonstracyjną edytora tabulatury, która cechuje się tym, że nie jest możliwe zapisanie postępu pracy ze względu na brak konta. Ostatni link wyświetla panel logowania.

4.3.2. Panel logowania i rejestracji

Przejście do okna logowania i rejestracji możliwe jest poprzez wybranie linku „Sign In" w panelu nawigacyjnym, lub dopisanie „/login" w pasku adresu przeglądarki. W przypadku braku wymaganej autoryzacji użytkownika, aplikacja przekieruje go na stronę logowania.

Interfejs logowania dostosowuje się do rozdzielczości ekranu. Na komputerach z szerokim ekranem panel logowania wyświetlany jest w formie panelu bocznego, wysuwanego z prawej strony. Natomiast na urządzeniach mobilnych oraz węższych ekranach formularz obejmuje cały ekran, zapewniając wygodniejszą obsługę.

Aby zalogować się do systemu, niezbędne jest podanie nazwy użytkownika oraz hasła. Istnieje również możliwość zapisania danych autoryzacyjnych w pamięci przeglądarki, przy zaznaczonej opcji „Remember me".

W celu rejestracji nowego użytkownika należy wybrać link „Sign up" w dolnej części panelu. Spowoduje to zmianę zawartości okna – pojawi się nowe pole adresu email a opcja zapamiętania użytkownika w przeglądarce zostanie ukryta. Zarówno nazwa użytkownika, jak i adres email muszą być unikalne w systemie.

Jeśli wprowadzone dane będą niepoprawne, na przykład błędne hasło lub powtarzająca się nazwa użytkownika, odpowiedni komunikat wyświetli się w formularzu. Pomyślne zalogowanie lub rejestracja skutkuje zalogowaniem użytkownika do systemu oraz przekierowaniem na kartę do zarządzania tabulaturami.

4.3.3. Użytkownik zalogowany

Użytkownik posiadający konto w systemie TabComposer ma przede wszystkim możliwość zapisu swoich tabulatur oraz większą swobodę edycji, ponieważ demonstracyjna edytora nie umożliwia wyboru dogodnego stroju otwartego. Panel nawigacyjny wygląda inaczej, zawiera linki „New Tablature", „My Tabs" oraz „Sign Out".

4.3.4. Zarządzanie tabulaturami

W zakładce „My Tabs” znajduje się strona do zarządzania tabulaturami użytkownika. Lista tabulatur składa się kafelków wyświetlanych jeden pod drugim. Każda pozycja zawiera tytuł, strój otwarty, liczbę taktów, opis oraz datę utworzenia tabulatury. Dla każdej tabulatury dostępne są trzy przyciski. Pierwszy, w kolorze jasnoszarym, służy do otwarcia tabulatury w trybie tylko do odczytu, który dodatkowo umożliwia na jej odtwarzanie. Drugi, o ciemno szarej barwie, otwiera edytor tabulatury. Ostatni, czerwony przycisk, służy do usunięcia tabulatury a wybranie tej opcji powoduje wyświetlenie okienka potwierdzenia operacji. Na końcu listy znajduje się dodatkowo zielony przycisk, umożliwiający dodanie nowej tabulatury.

4.3.5. Kreator tabulatury

Akcja dodania nowej tabulatury dostępna jest tylko dla zalogowanych użytkowników, z każdego miejsca w aplikacji poprzez panel nawigacyjny lub w zakładce „My Tabs”. Wybranie tej opcji powoduje wyświetlenie się okienka kreatora. Aby utworzyć tabulaturę, należy wprowadzić jej tytuł, liczbę progów, opis oraz strój otwarty spośród dostępnych, wskazanych w tabeli 4.3 strojów.

Tabela 4.3. Lista dostępnych strojów w aplikacji TabComposer

Nazwa stroju	**Dźwięki strun n…1**
E-Standard	E2 A2 D3 G3 B3 E4
Eb-Standard	D#2 G#2 C#3 F#3 A#3 D#4
E-Standard Bass (5-string / 4-string)	B0 / E1 A1 D2 G2
D-Standard	D2 G2 C3 F3 A3 D4
D-Standard Bass (5-string / 4-string)	A0 / D1 G1 C2 F2
Db-Standard	C#2 F#2 B2 E3 G#3 C#4
C-Standard	C2 F2 A#2 D#3 G3 C4
D-Drop	D2 A2 D3 G3 B3 E4
C-Drop	C2 G2 C3 F3 A3 D4
B7-Standard	B2 E2 A2 D3 G3 B3 E4

Zmiana stroju wybranego na tym etapie nie jest już później możliwa. Po udanym utworzeniu tabulatury użytkownik zostaje przekierowany do strony edytora.

4.3.6. Edytor tabulatury

Edytor tabulatury w wersji dla zalogowanych użytkowników wyróżnia się od jego wersji demonstracyjnej tylko obecnością mechanizmu autosave. Widok edytora składa się z trzech głównych elementów: edytora metadanych, obszaru edycji oraz paska narzędzi edytora.

- **Metadane tabulatury**

Kliknięcie w tytuł tabulatury powoduje wyświetlenie okienka edycji jej metadanych. W przeciwieństwie do okienka kreatora tabulatury, nie ma w nim możliwości ponownej zmiany stroju.

- **Pasek narzędzi edytora**

Pasek narzędzi edytora znajduje się poniżej obszaru edycji tabulatury. Jeżeli obszar ten zajmuje całą wysokość strony, pasek narzędzi umiejscawia się w pozycji absolutnej na dole ekranu. Dostępne są trzy zakładki: „Global Toolbar", „Tab Player" oraz „Tab Recorder".

- **Global Toolbar**

Ustawienia dla całej tabulatury znajdują się w pierwszej zakładce. Elementy wchodzące w skład narzędzi globalnych przedstawia tabela 4.4. Każdy nowy takt oraz nuta dodana do tabulatury będzie zawierała domyślne wartości, takie jakie zostały ustawione w „Global Toolbar".

Tabela 4.4. Lista elementów zakładki „Global Toolbar"

Nazwa elementu	Rodzaj elementu	**Opis**
Display	Pole numeryczne	Liczba taktów w wierszu
Tempo	Pole numeryczne	Globalne tempo taktów
Numerator	Pole numeryczne	Globalny numerator taktów
Denominator	Pole numeryczne	Globalny denominator taktów
Duration	Lista rozwijana	Globalny czas trwania nut
Interval	Lista rozwijana	Globalny krok przesunięcia nut
Shift on delete	Pole wyboru	Przesuwanie nut przy usunięciu
Saving… / Up to date	Przycisk / Informacja	Ręczny zapis postępów

- **Tab Player**

Zakładka „Tab Player" umożliwia odtwarzanie dźwięków zapisanych na tabulaturze. Jest jedyną dostępną opcją paska narzędzi dla odtwarzacza tabulatury w trybie tylko do odczytu. Tabela 4.6 przedstawia elementy oraz akcje dostępne w „Tab Player".

Tabela 4.5. Lista elementów zakładki „Tab Player"

Nazwa elementu	Rodzaj elementu	**Opis**
Pauza	Przycisk	Pauza odtwarzania tabulatury
Start	Przycisk	Odtwarzanie tabulatury
Stop	Przycisk	Przerwanie odtwarzania tabulatury
Start at measure	Pole numeryczne	Rozpoczęcie od wskazanego taktu
Speed	Suwak / etykieta	Zmiana prędkości odtwarzania

- **Tab Recorder**

Za pomocą trzeciej zakładki użytkownik może wykorzystywać analizator audio do automatycznego układania tabulatury. W tabeli 4.7 przedstawiono zawartość oraz funkcjonalność „Tab Recorder”.

Tabela 4.6. Lista elementów zakładki „Tab Recorder”

Nazwa elementu	Rodzaj elementu	**Opis**
Nagrywaj	Przycisk	Rozpoczęcie nagrywania
Stop	Przycisk	Przerwanie nagrywania
Monitor on / off	Przełącznik	Odsłuch mikrofonu
Show charts	Pole wyboru	Wyświetlanie wykresów
Domyślny …	Lista rozwijana	Wybór urzędzenia nagrywania
Refresh	Przycisk	Odświeżanie listy urządzeń
Effects on / off	Przełącznik	Efekty mikrofonu
Wykres lewy	Kanwa (canvas)	Widmo amplitudowe sygnału
Wykres prawy	Kanwa (canvas)	Sygnał w domenie czasu

Naciśnięcie przycisku nagrywania skutkuje włączeniem metronomu, ustawionego na wartości metrum oraz tempa zgodne z narzędziami globalnymi. Po odtworzeniu jednego taktu rozpoczyna się rejestrowanie dźwięku. Rozpoznane nuty są umieszczane na nowo dodanych taktach, a ich pozycje modyfikowane w celu uzyskania jak najlepszego ustawienia pod kątem ergonomii gry. Aby przerwać nagrywanie, należy wybrać przycisk stop.

Efekty służą do redukcji szumów mikrofonu oraz poprawy jakości analizy sygnału. Opcja ta jest domyślnie włączona. Wykres widma amplitudowego sygnału przedstawia zakres od 50 do 1500 Hz, czyli częstotliwości możliwe do osiągnięcia na większości gitar.

- **Dodawanie i modyfikacja taktu**

W edytorze funkcjonalność obszaru tabulatury obejmuje dodawanie nowych taktów, ich modyfikację oraz nanoszenie nowych nut. Takt może być utworzony jako pusty za pomocą przycisku „New Empty Measure” lub jako kopia istniejącego taktu poprzez wskazanie jego identyfikatora i wciśnięcie przycisku „Copy”. Przyciski te znajdują się zawsze za ostatnim taktem tabulatury. Nad siatką tabulatury, w obrębie każdego taktu, wyświetlane są informacje o jego indeksie, metrum oraz tempie. Kliknięcie w ten obszar spowoduje pojawienie się okienka modyfikacji wartości metrum, tempa oraz usunięcia taktu.

- **Dodawanie i modyfikacja nuty**

Aby dodać nową nutę, należy kliknąć w wybraną linię siatki tabulatury. Spowoduje to wyświetlenie się okienka wyboru nuty lub pauzy, z możliwością zmiany wartości rytmicznej, niezależnie od ustawień narzędzi globalnych. Każda nuta jest tworzona

z wartością progu wynoszącą zero i umieszczana automatycznie się na końcu linii. Kliknięcie w nowo utworzoną nutę również wyświetli okienko modyfikacji, którego elementy przedstawia tabela 4.7.

Tabela 4.7. Lista elementów okienka nuty oraz pauzy

Nazwa elementu	Rodzaj elementu	**Opis**
Edit Note / Pause	Etykieta	Informacja o wartości nuty / pauzy
Fret	Pole numeryczne	Numer progu nuty
Duration	Lista rozwijana	Wartość rytmiczna nuty / pauzy
None (domyślnie)	Lista rozwijana	Techniki artykulacyjne
Slide	Pole wyboru	Dodatkowa technika artykulacyjna
Link (kiedy Slide włączony)	Pole wybrou	Slide od poprzedniej nuty
Strzałka w lewo	Przycisk	Przesunięcie w lewo nuty / pauze
Strzałka w dół	Lista rozwijana	Wartość interwału przesunięcia
Strzałka w prawo	Przycisk	Przesunięcie w prawo nuty / pauze
Delete Note / Pause	Przycisk	Usunięcie nuty / pauzy

Przesunięcie nuty można również osiągnąć za pomocą mechanizmu przeciągnięcia. Nuta lub pauza zmieni swoje położenie o wartość, jaką wskazuje jej interwał. Aby zamienić miejscem nutę lub pauzę z inną sąsiadującą nutą, należy użyć dedykowanych przycisków.

- **Informowanie o błędach**

Wszystkie błędy wynikające z nieprawidłowego używania edytora są wyświetlane bezpośrednio pod elementami, które zawierają niepoprawne wartości. Zaliczają się do nich błędy dotyczące metadanych tabulatury, paska narzędzi oraz obszaru tabulatury a w tym okienek dodawania i modyfikacji nut oraz edycji taktów.

4.4. Administracja systemu

Administracja systemu obejmuje zarządzanie kontami użytkowników, bazą danych tabulatur oraz monitorowanie działania aplikacji. Ważnymi kwestiami są również zapewnienie bezpieczeństwa, regularne aktualizacje oraz kopie zapasowe danych.

W systemie TabComposer nie ma oddzielnego panelu administratora. Użytkownicy posiadają konta, w ramach których mogą jedynie tworzyć i zarządzać swoimi tabulaturami. Aplikacja nie posiada funkcjonalności, która pozwoliłaby na publiczne udostępnianie treści generowanych użytkowników, dlatego zapewnienie moderacji jest zbędne. Baza danych aplikacji przechowuje informacje o użytkownikach oraz tabulaturach. Okresowa optymalizacja oraz archiwizacja nieaktywnych danych zapewnią wydajność systemu. Wykrywanie ewentualnych błędów oraz problemów z wydajnością systemu można

osiągnąć poprzez monitorowanie logów serwera. W przypadku wykrycia błędów konieczne jest ich szybkie usunięcie. System został zaprojektowany z myślą o dalszym rozwoju. Zastosowane rozwiązania w architekturze aplikacji umożliwiają oddawanie funkcji poprzez rozbudowę API oraz interfejsu użytkownika. Aktualizacje powinny być przeprowadzane w czasie o najmniejszym natężeniu ruchu w systemie oraz w sposób zachowujący ciągłość jego działania.

4.5. Kwestie bezpieczeństwa

Aplikacja TabComposer zapewnia wysoki poziom bezpieczeństwa, kontrolując dostęp do zasobów. Wszystkie operacje prowadzące do zmian w bazie danych oraz udostępniające użytkownikom ich dane wymagają autoryzacji. Funkcjonalność ta kontrolowana jest za pomocą tokenów JSON Web Token (JWT), zapewniając bezpieczne i wydajne zarządzanie sesjami użytkowników.

Wszystkie dane użytkowników, w tym tabulatury i poufne informacje logowania przechowywane są w bazie danych. Hasła są haszowane przy użyciu algorytmów kryptograficznych stosowanych przez mechanizm ASP.NET Identity. W przypadku nieautoryzowanego dostępu do bazy danych, odczytanie haseł jest znacznie utrudnione. Dodatkowo obowiązuje polityka haseł – każde hasło musi składać się z co najmniej sześciu znaków, zawierać znak specjalny, cyfrę oraz wielką literę.

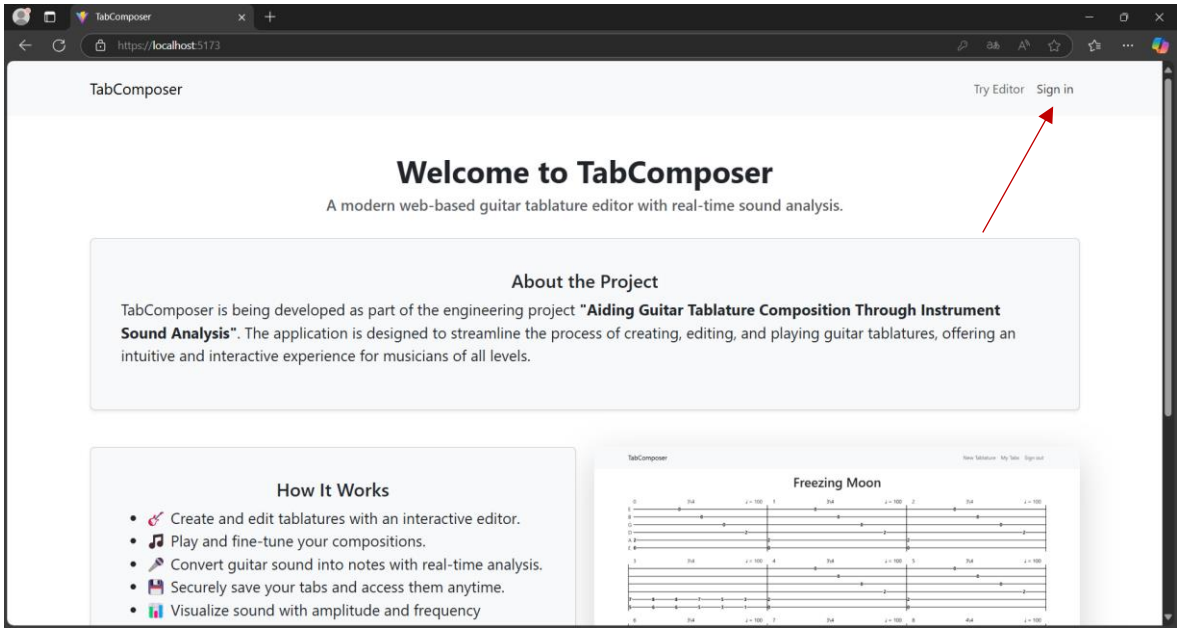
Informacje przesyłane pomiędzy użytkownikiem a aplikacją są walidowane zarówno po stronie klienta, jak i po stronie serwera. W przypadku wykrycia niepoprawności danych, żądanie zostaje odrzucone. Rozwiązanie to zapobiega atakom typu SQL Injection oraz Cross-Site Scripting. Dodatkowym środkiem ochrony jest szyfrowanie transmisji danych przy użyciu protokołu https.

4.6. Scenariusze z systemu

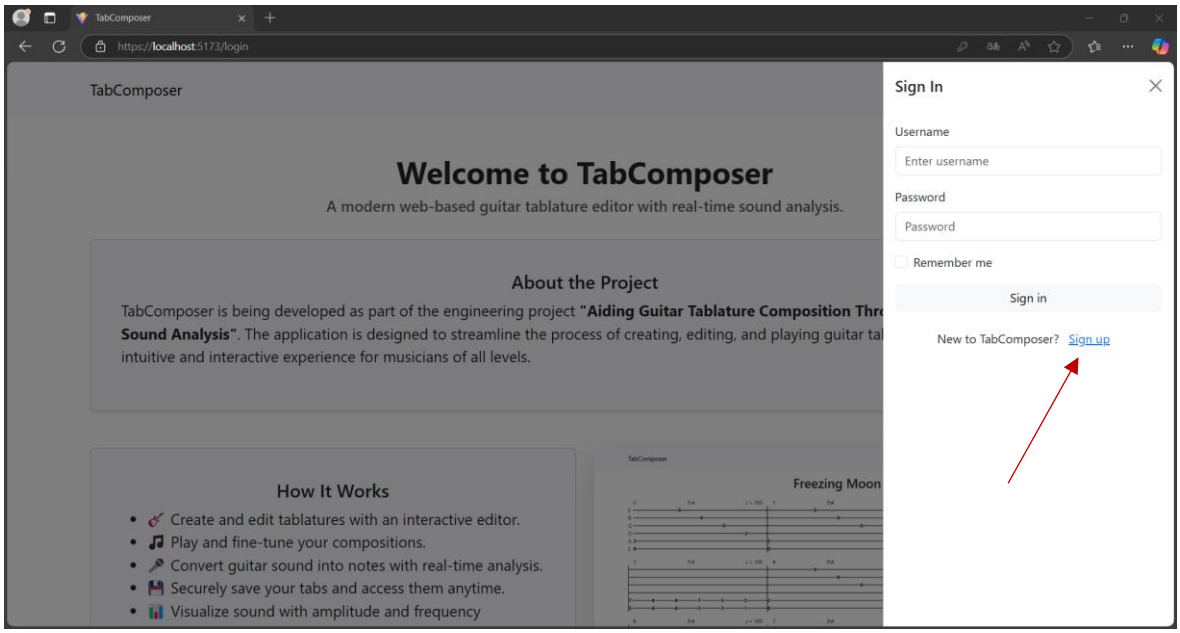
W tym podrozdziale przedstawiono kilka najważniejszych scenariuszy użytkowania systemu. Opisane przypadki obejmują proces rejestracji, tworzenia oraz edycji tabulatury, odtwarzania oraz analizy dźwięku. Scenariusze zostały zilustrowana zrzutami ekranu ukazującymi interakcje użytkownika z aplikacją.

4.6.1. Rejestracja nowego użytkownika

Przypadek ten obejmuje proces utworzenia konta przez nowego użytkownika. Na rysunku 4.1 przedstawiono stronę domową aplikacji a czerwoną strzałką zaznaczono link, który powoduje otwarcie panelu logowania. Zrzut ekranu 4.2 wskazuje w jaki sposób przejść do widoku rejestracji.

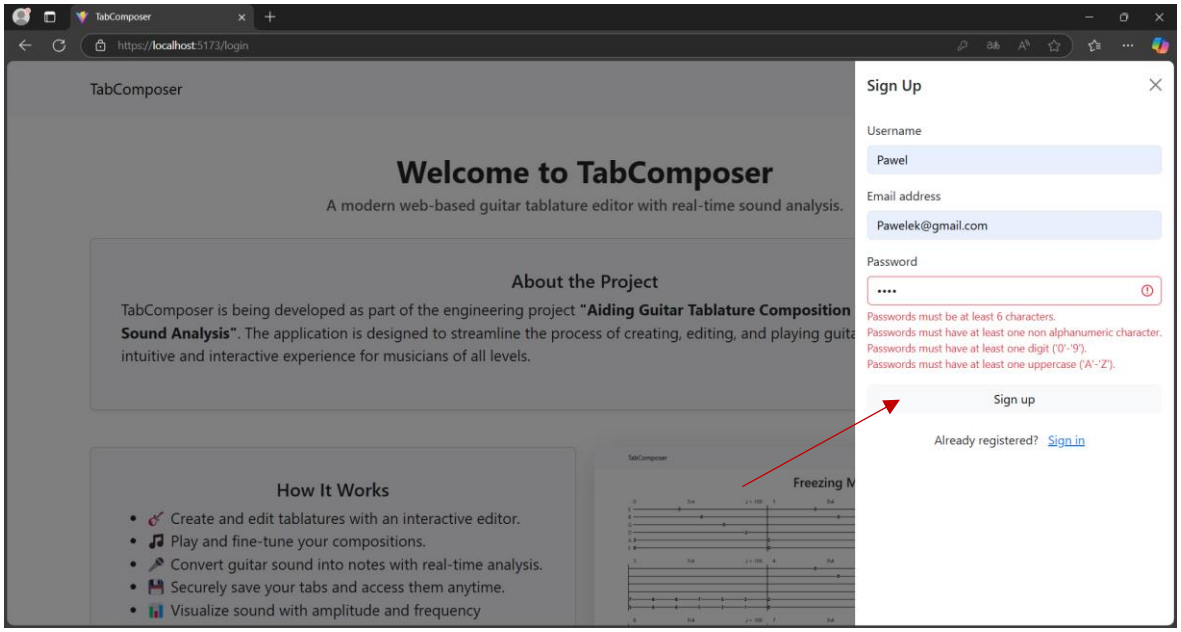


Rysunek 4.1. Widok strony domowej aplikacji TabComposer

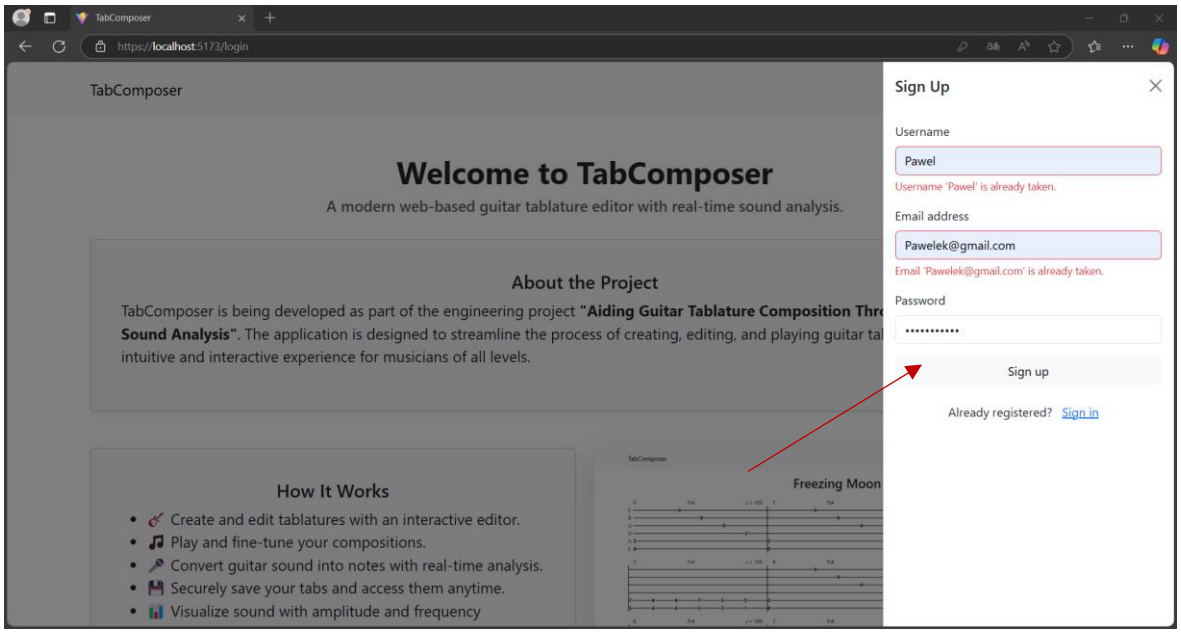


Rysunek 4.2. Widok panelu logowania aplikacji TabComposer

Kolejne zrzuty ekranu ilustrują zgłoszenie komunikatu o nieprawidłowych danych. W pierwszym przypadku, na rysunku 4.3, wprowadzone hasło nie jest zgodne z wymaganiami. Ilustracja 4.4 pokazuje, podana nazwa oraz adres e-mail już istnieją w systemie.

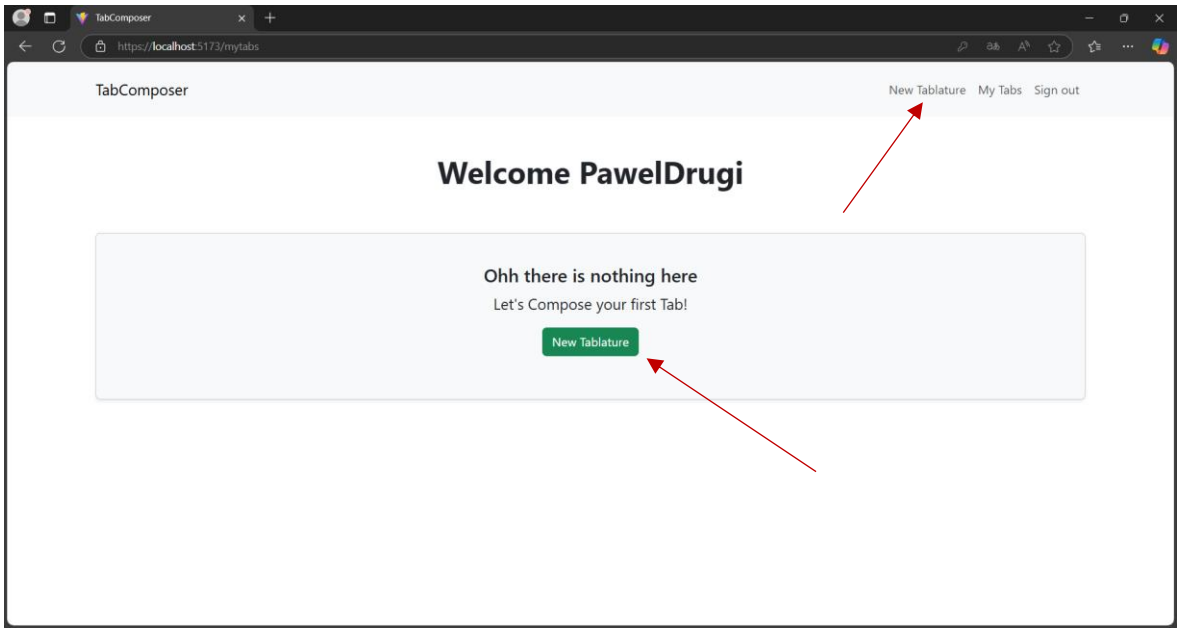


Rysunek 4.3. Hasło niezgodne z wymaganiami



Rysunek 4.4. Użytkownik o podanej nazwie już istnieje

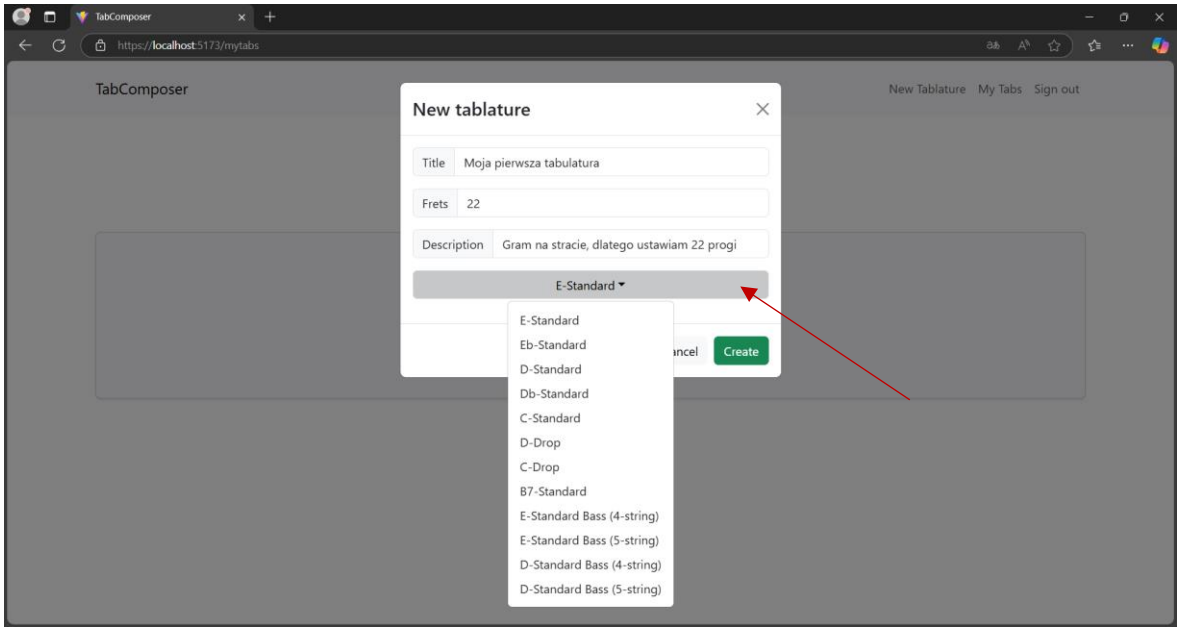
W kolejnym przypadku, po wprowadzeniu odpowiednich danych, konto zostało pomyślnie zarejestrowane, a użytkownik został przekierowany na stronę „My Tabs.



Rysunek 4.5. Widok okna „My Tabs” w aplikacji TabComposer

4.6.2. Utworzenie nowej tabulatury

Aplikacja TabComposer umożliwia utworzenie nowej tabulatury z dwóch dostępnych pozycji wskazanych na rysunku 4.5. Do wyboru jest link „New Tablature” lub przycisk o tej samej nazwie, które powodują wyświetlenie się okna kreatora przedstawionego na rysunku 4.6.



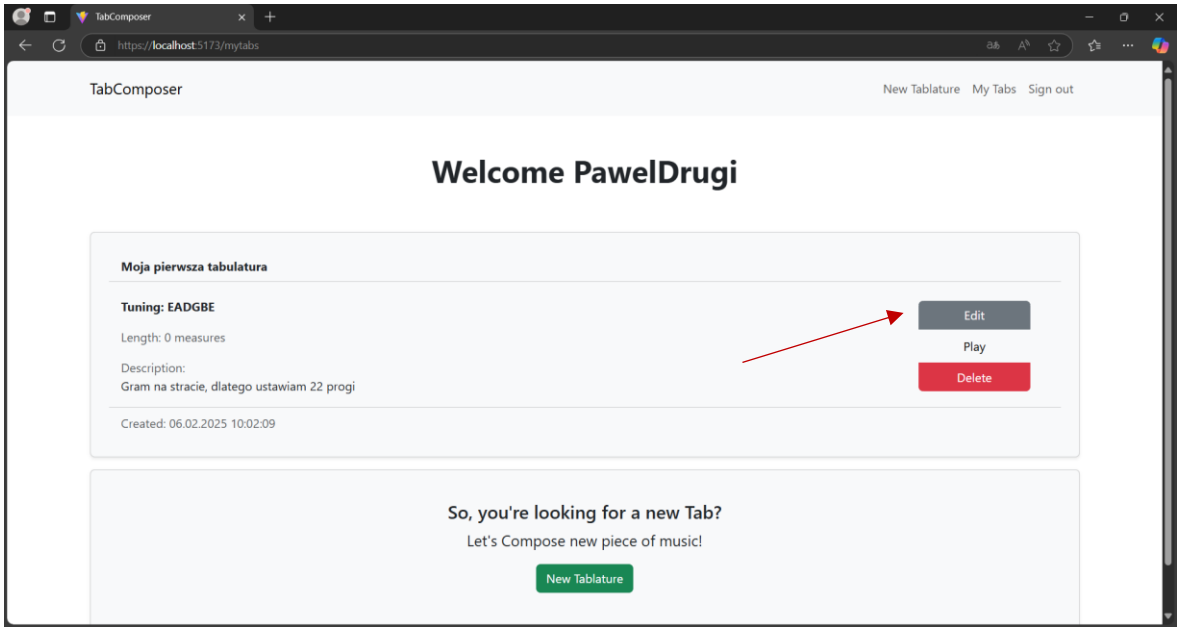
Rysunek 4.6. Widok okna kreatora tabulatur w aplikacji TabComposer

Kliknięcie przycisku „Create” powoduje utworzenie nowej tabulatury oraz przekierowanie do strony edytora. W przypadku próby utworzenia nowej tabulatury

z poziomu widoku okna edycji innej tabulatury, proces ten nie spowoduje przekierowania, ze względu na możliwość utraty wprowadzonych danych.

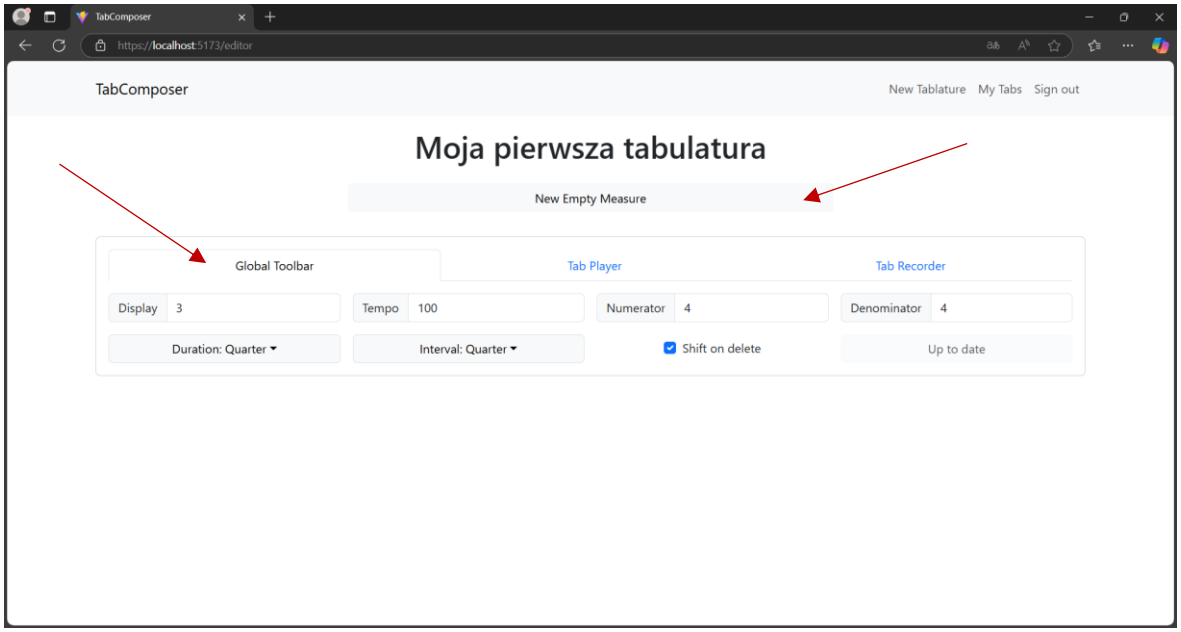
4.6.3. Edycja tabulatury

Z poziomu widoku tabulatur, możliwe jest sprawdzenie ich listy, zawierającej elementy przedstawione na zrzucie ekranu 4.7.



Rysunek 4.7. Widok okna „My Tabs” z listą tabulatur

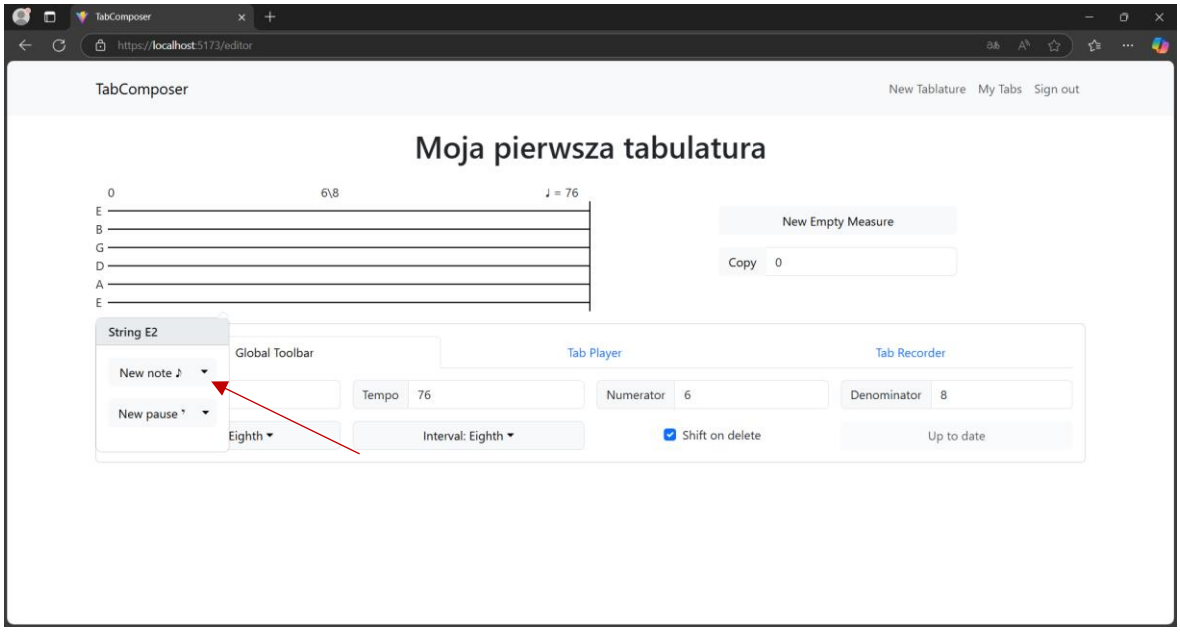
Aby przejść do edytora przedstawionego na rysunku 4.8 należy kliknąć przycisk z napisem „Edit”.



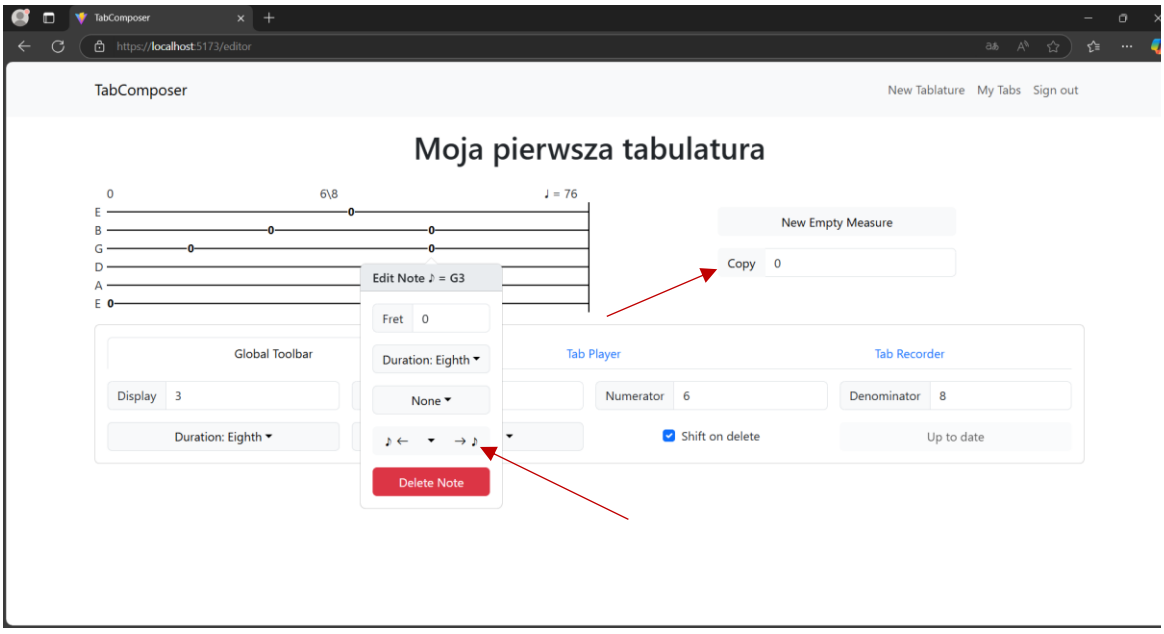
Rysunek 4.8. Widok edytora w aplikacji TabComposer

Przed przystąpieniem do tworzenia tabulatury, istnieje możliwość odpowiedniego dostosowania ustawień paska globalnych narzędzi „Global Toolbar”. Aby utworzyć takt, należy kliknąć przycisk „New Empty Measure”. Efekt podjętych działań przedstawia rysunek 4.9.

Dalsza edycja polega na dodawaniu nowych nut oraz edytowaniu ich właściwości. Proces tworzenia piosenki ilustrują zrzuty 4.9 oraz 4.10, które przedstawiają możliwości edytora, takie jak dodawanie nowej nuty, przesuwanie jej po strunie oraz kopiowanie taktu

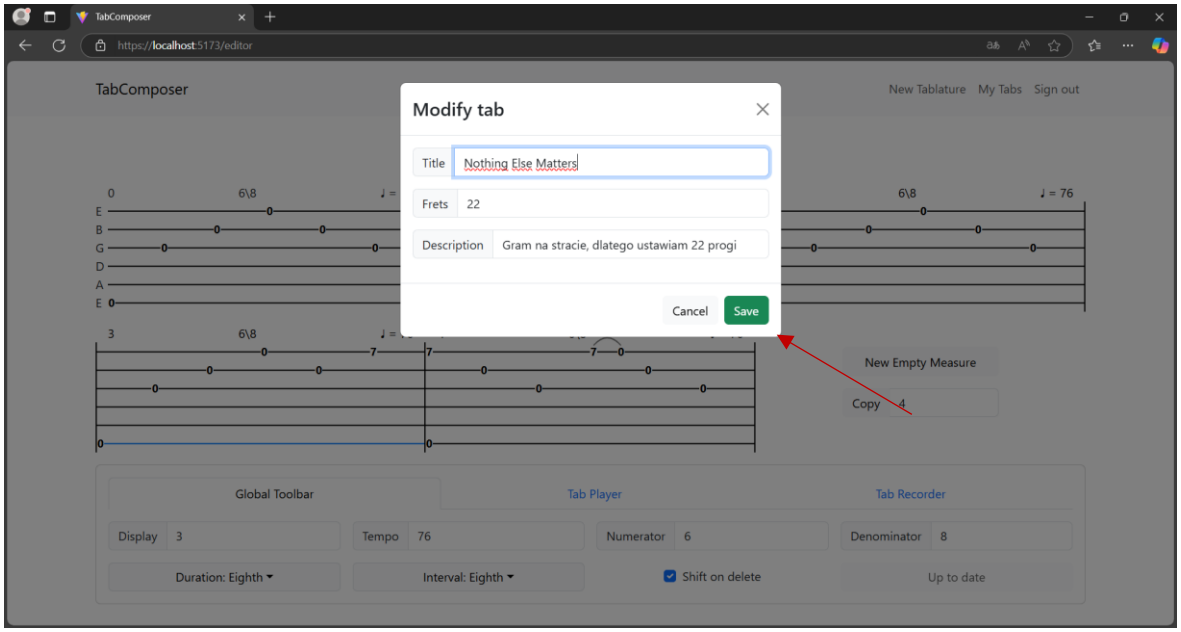


Rysunek 4.9. Dodawanie nowej nuty do taktu



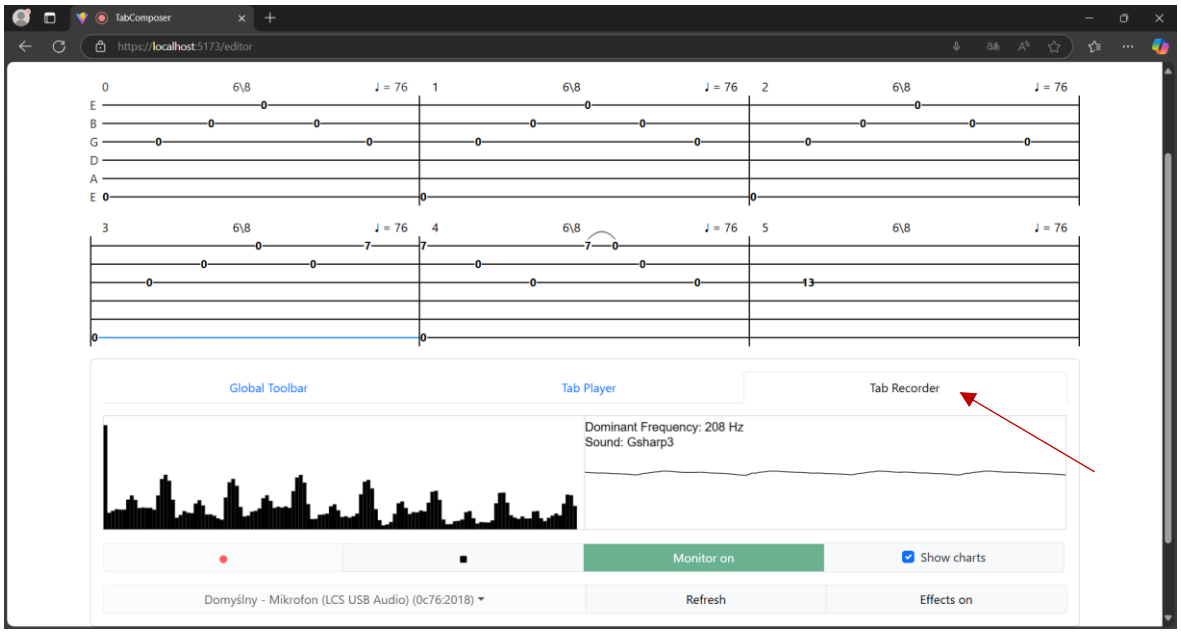
Rysunek 4.10. Przesuwanie nuty po strunie i kopiowanie taktu

Metadane tabulatury przedstawione na rysunku 4.11 mogą zostać zmienione poprzez kliknięcie w jej tytuł.



Rysunek 4.11. Zmiana metadanych tabulatury

Oprócz standardowego edytora, dostępny jest także analizator dźwięku, który przetwarza sygnał wejściowy na nuty tabulatury. Znajduje się on w zakładce „Tab Recorder” jak przedstawiono na zrzucie ekranu 4.12.



Rysunek 4.12. Widok paska narzędzi „Tab Recorder”

Rozdział 5

Specyfikacja wewnętrzna

5.1. Przedstawienie idei

Głównym celem niniejszej pracy jest zaprojektowanie oraz zaimplementowanie aplikacji webowej o nazwie TabComposer, będącej edytorem tabulatur gitarowych. Wyróżniającą się funkcjonalnością aplikacji jest wspomaganie układania tabulatury, poprzez analizę dźwięku instrumentu, umożliwiając automatyczne generowanie tabulatury na podstawie strumieniowanego dźwięku.

Ważnym aspektem aplikacji jest również balans pomiędzy prostotą oraz poprawnością zapisu w kontekście muzycznym, odnoszący się do łatwości użytkowania aplikacji. Przedstawienie tabulatury bazuje na jej tradycyjnej postaci z uwzględnieniem elementów standardowej notacji muzycznej, niezbędnych do analizy audio.

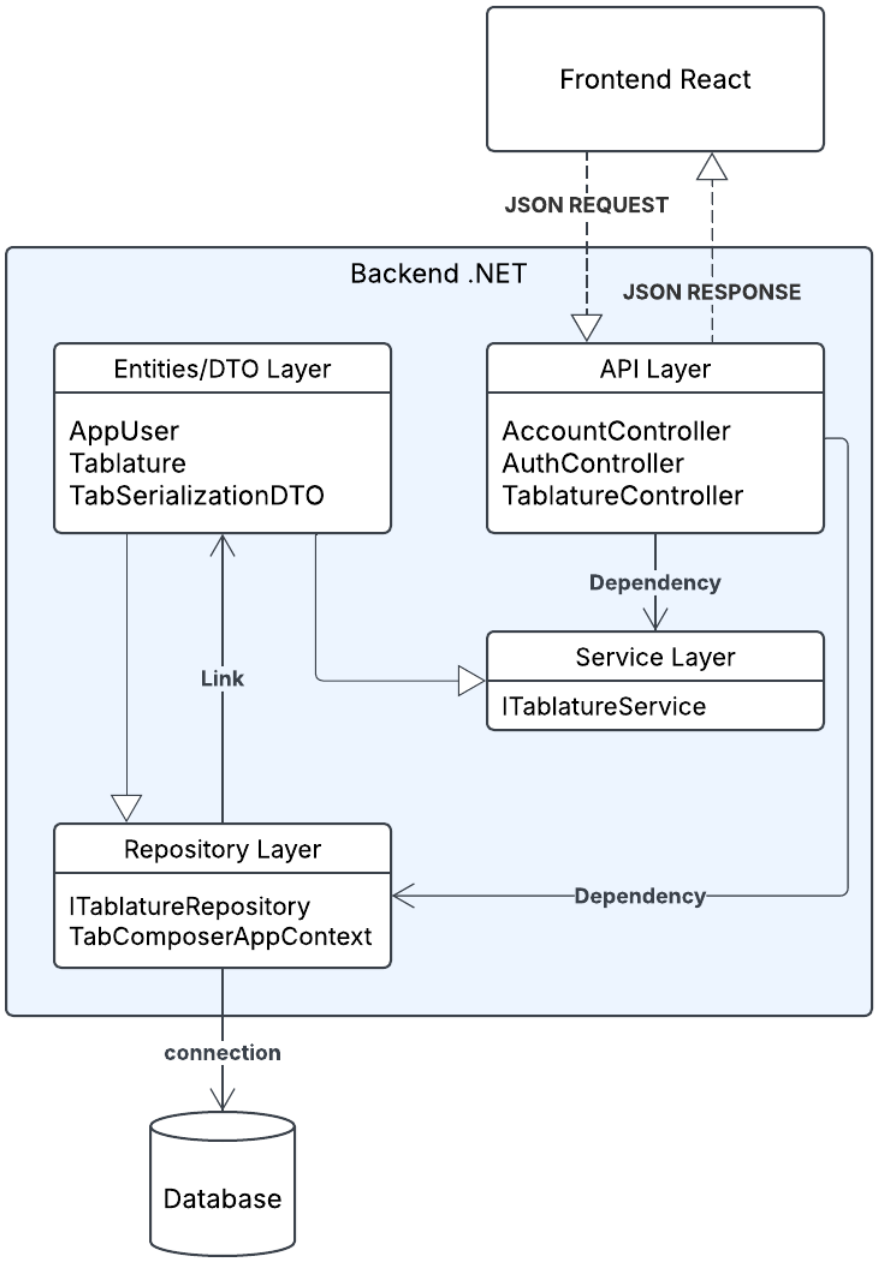
Następną ważną cechą systemu jest intuicyjny oraz interaktywny interfejs graficzny, ułatwiający korzystanie z aplikacji, szczególnie dla początkujących gitarzystów. Aplikacja posiada system kont użytkowników do przechowywania ich projektów. Architektura jest zaprojektowana z myślą o przyszłej rozbudowie funkcjonalnej aplikacji.

5.2. Architektura systemu

System zaprojektowano w oparciu o architekturę klient-serwer. Jest to model, w którym aplikacja dzieli się na dwie główne części: **backend** oraz **frontend**. Klient integruje się z warstwą prezentacji aplikacji i odpowiada za interakcję z użytkownikiem. Żądania klienta wysyłane są do serwera stanowiącego warstwę logiki biznesowej aplikacji. Serwer wykonuje określone operacje oraz zwraca odpowiedź.

5.2.1. Backend

Backend aplikacji został zrealizowany w technologii ASP.NET Core. Jest to nowoczesny, wydajny i dobrze wspierany framework do tworzenia aplikacji webowych. Oferuje gotowe mechanizmy do tworzenia RESTful API, pozwalając na łatwą komunikację z frontendem aplikacji. Posiada wbudowane mechanizmy bezpieczeństwa takie jak ASP.NET Identity oraz JWT. Zapewnia prostą integrację baz danych z systemem oraz mapowanie relacyjnych tabel na obiekty za pośrednictwem Entity Framework Core. Na rysunku 5.1 przedstawiono diagram architektury backendu aplikacji TabComposer.



Rysunek 5.1. Diagram komponentów architektury backendu TabComposer

Backend w aplikacji TabComposer realizuje **wzorzec warstwowy**, który pozwala na dobrą organizację kodu, separację odpowiedzialności oraz ułatwia rozwój systemu. Struktura ta składa się z warstwy prezentacji API, warstwy logiki biznesowej (ang. *services*), warstwy dostępu do danych (ang. *repository*) oraz warstwy modelu danych (ang. *entities*).

- **Warstwa prezentacji**

API odpowiada za obsługę żądań http i komunikację z frontendem. Składa się z kontrolerów, które przyjmują zapytania, walidują dane oraz delegują logikę do następnych warstw. *AccountController* odpowiada za pobieranie profilu użytkownika. Wyszukuje informacje na podstawie identyfikatora uzyskanego z JWT. Zwraca dane takie jak adres e-mail oraz nazwa użytkownika. *AuthController* zajmuje się uwierzytelnianiem użytkowników. Sprawdza, czy użytkownik jest autoryzowany, rejestruje nowych użytkowników w bazie oraz loguje użytkowników generując token JWT. *TablatureController* zarządza tabulaturami użytkowników. Współpracuje z warstwą dostępu do danych ITablatureRepository oraz z warstwą biznesową ITablatureService. Pobiera identyfikator użytkownika z tokenu JWT i waliduje poprawność danych tabulatury przed ich zapisaniem. Umożliwia operacje takie jak dodawanie, aktualizację, usuwanie a także udostępnianie tabulatur.

- **Warstwa logiki biznesowej**

Warstwa serwisów implementuje zasady działania aplikacji oraz przetwarza otrzymane dane. W aplikacji TabComposer za warstwę logiki biznesowej po stronie serwera odpowiada interfejs *ITablatureService*. Serwis ten umożliwia przekształcanie obiektów zawierających dane tabulatury na format JSON i odwrotnie. Dane są dodatkowo walidowane zapewniając spójność przechowywanych danych.

- **Warstwa repozytorium**

Warstwa systemu oparta na wzorcu repozytorium zajmuje się komunikacją z bazą danych. Wykonuje operacje **CRUD** (ang. *Create, Read, Update, Delete*) na bazie danych i pełni rolę abstrakcji, oddzielającej logikę dostępu do danych od logiki biznesowej. W projekcie rolę repozytorium pełni interfejs *ITablatureRepository* oraz klasa *TabComposerAppContext*. Klasa kontekstu odpowiada za mapowanie encji oraz wykonywanie operacji na danych, natomiast interfejs repozytorium definiuje konkretne metody asynchroniczne zapewniające dostęp do danych tabulatur.

- **Warstwa modelu danych**

Struktury danych używane w aplikacji definiowane są w warstwie modelu danych. Mogą być to **encje** lub **obiekty transferu danych** (ang. *Data Transfer Objects*, DTO).

W aplikacji TabComposer klasy *AppUser* oraz *Tablature* są encjami bazy danych, natomiast klasy pomocnicze serializacji tabulatury stanowią DTO.

W projekcie wykorzystano również **wzorzec wstrzykiwania zależności** (ang. *Depedency Injection*, DI). Umożliwia on elastyczne zarządzanie zależnościami pomiędzy komponentami aplikacji co ułatwia rozszerzanie systemu. W ramach tego wzorca serwis serializacji danych i repozytorium wstrzykiwane są do kontrolera tabulatury.

5.2.2. Frontend

Frontend aplikacji został zaimplementowany w technologii React z TypeScript. Jest to biblioteka dobrze nadająca się do budowy interaktywnych aplikacji webowych, umożliwiająca efektywne zarządzanie stanem oraz dynamiczną aktualizację interfejsu użytkownika bez konieczności przeładowania strony. *TypeScript*, w przeciwieństwie do JavaScript, jest językiem statycznie typowanym, co zwiększa czytelność kodu i poprawia kontrolę nad strukturą aplikacji.

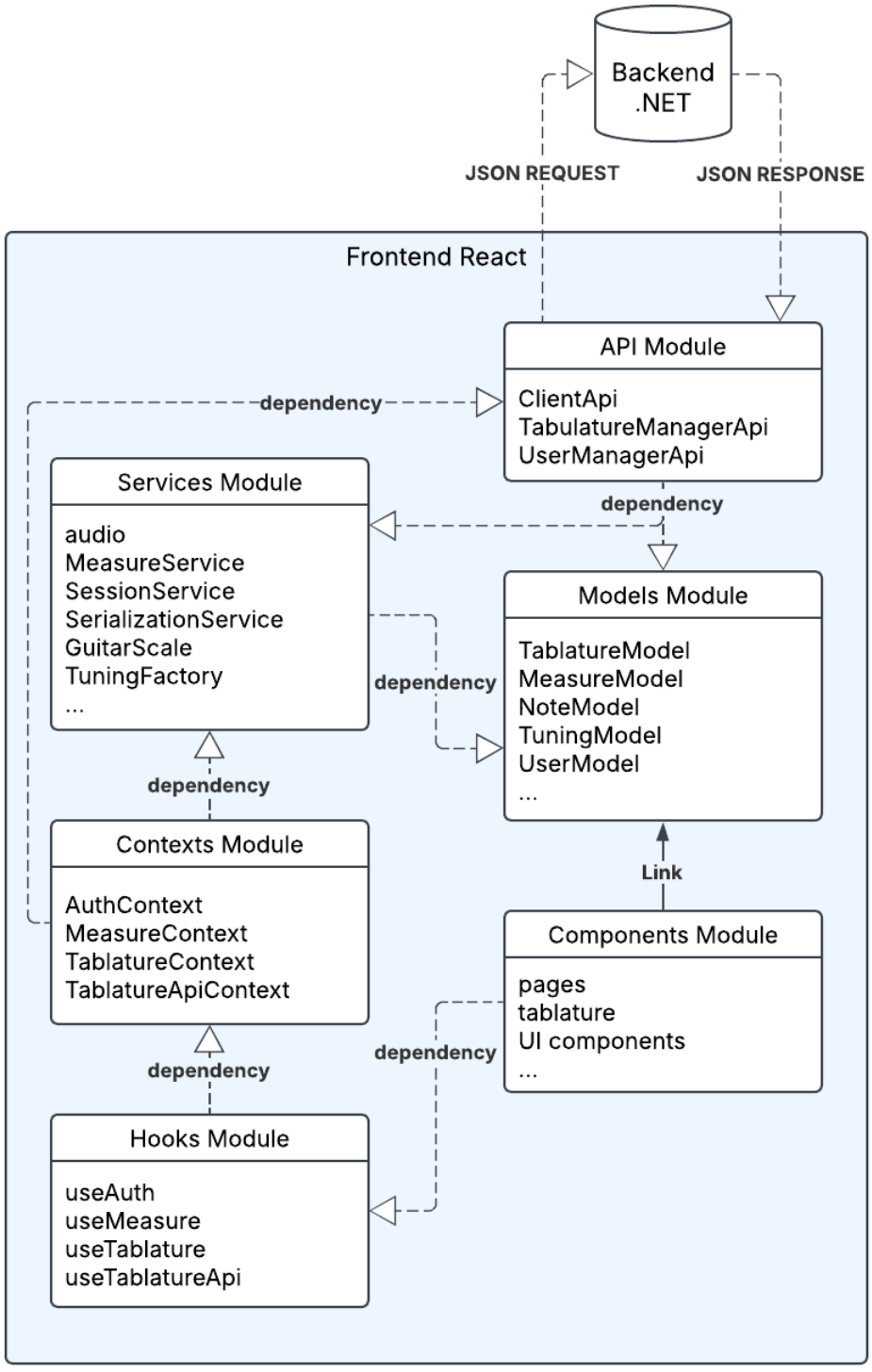
W aplikacji TabComposer frontend realizuje architekturę komponentową, zgodnie z filozofią React. Wykorzystuje wzorce projektowe takie jak **kompozycja** – elementy interfejsu użytkownika, **obserwator** – dynamiczne odświeżanie widoków, **singleton** – zapewnienie połączenia z backendem i urządzeniami audio, oraz **fabryka** do tworzenia różnych wersji struktur danych. Aplikacja została podzielona na funkcjonalne moduły w celu organizacji i skalowalności kodu. Rysunek 5.2 przedstawia diagram architektury frontendu aplikacji TabComposer i zależności między jego komponentami.

- **API**

Moduł **api** dostarcza klasy oraz ich interfejsy służące do komunikacji z backendem. Główną klasą jest singleton *ClientApi* pełniący funkcję centralnego klienta http. Klasy *UserManagerApi* oraz *TablatureManagerApi* używają tego singletonu do obsługi swoich żądań, związanych z zarządzaniem stanem użytkownika oraz operacjami na tabulaturach.

- **Context**

Mechanizm *useContext* w React umożliwia przekazywanie danych pomiędzy komponentami bez potrzeby używania *props*. Przechowuje globalny stan, który może być dostępny w dowolnym miejscu w obrębie jego zasięgu, zdefiniowanego przez *provider*. Moduł **context** zawiera deklaracje oraz implementacje kontekstów do obsługi obiektów z modułu api, a także modeli strukturalnych tabulatury i taktu.



Rysunek 5.2. Diagram modułów architektury frontendu TabComposer

- **Hooks**

Hooki w React są funkcjami pozwalającymi na zarządzanie stanem i innymi aspektami życia komponentów bez potrzeby tworzenia klas. Moduł **hooks** obejmuje funkcje uzyskiwania dostępu do kontekstów aplikacji.

- **Services**

Moduł **services** zawiera logikę biznesową, która musi zostać wykonywana na frontendzie ze względu na jego funkcjonalność. Obsługuje między innymi **serializacji** i **deserializację** danych tabulatury w formacie *JSON*, operacje na danych przechowywanych w pamięci przeglądarki, logikę działania taktów oraz fabryki obiektów modelu, takich jak *ITuning*, *Sound* oraz *Note*. Mechanizmy odpowiedzialne za operacje związane z przetwarzaniem dźwięku znajdują się w pod-module **audio**. Dotyczą obsługi odtwarzacza tabulatury, mikrofonu oraz analizatorów dźwięku.

- **Models**

Wzorzec kompozycji to podejście projektowe polegające na składaniu mniejszych, niezależnych obiektów w większe struktury, zamiast używać dziedziczenia. Moduł **models** zawiera główne modele strukturalne, wykorzystywane do budowania większych elementów logicznych w paradygmacie kompozycji.

- **Components**

Moduł **components** zawiera komponenty interfejsu użytkownika, w tym pod-moduły takie jak **pages** odpowiadające za poszczególne widoki, oraz **tablature** wyświetlające interaktywną tabulaturę.

5.3. Opis struktur i bazy danych

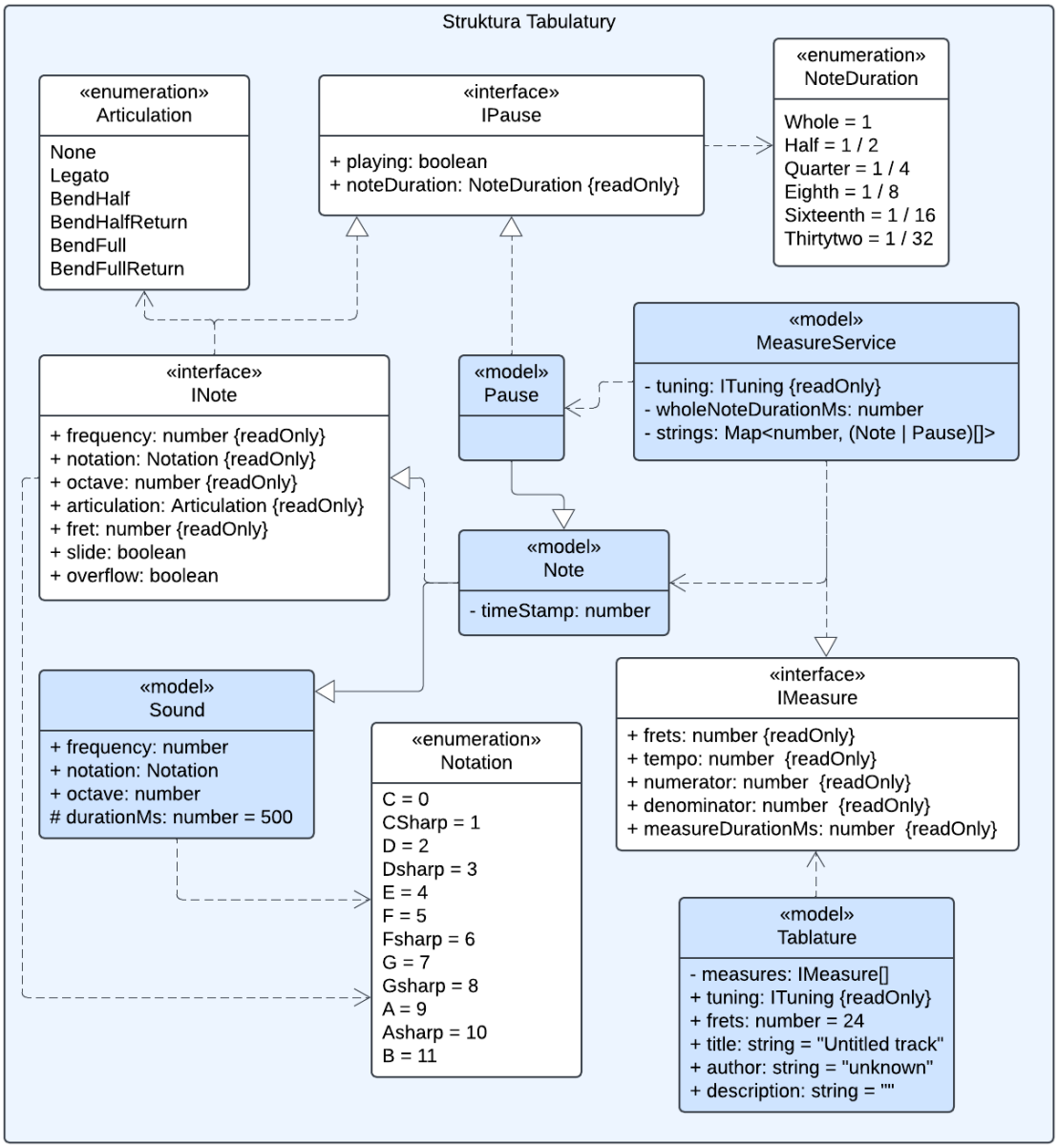
Jednym z najważniejszych aspektów podczas tworzenia aplikacji TabComposer było odpowiednie zaprojektowanie struktur danych reprezentujących tabulaturę. Z uwagi na specyfikę aplikacji zaszła konieczność podzielenia tych struktur na dwie kategorie, które stanowią **rozbudowane struktury edytora tabulatury** frontendu oraz **struktury służące do przesyłania informacji** pomiędzy klientem a serwerem, z zachowaniem spójności ich deklaracji. Dane tabulatur przesyłane są w formacie JSON i deserializowane do postaci obiektów DTO.

Informacje o użytkownikach i tabulaturach przechowywane są w relacyjnej bazie danych SQL Server. Wykorzystano technikę ORM (ang. *Object-Relational Mapping*) umożliwiającą pracę z danymi w sposób obiektowy, dzięki zastosowaniu **Entity Framework Core** w projekcie ASP.NET.

5.3.1. Struktury danych tabulatury

Aplikacja TabComposer wykorzystuje złożone struktury danych reprezentujące tabulaturę, w sposób analogiczny do podziału muzycznego. Tabulatura zbudowana jest

z taktów, zawierających struny z rozmieszonymi na nich nutami. Podejście, w którym bardziej złożone obiekty składają się z mniejszych komponentów realizuje m.in. **wzorzec kompozytu** – rozszerzanie funkcjonalności poprzez agregację zamiast dziedziczenia. Model ten zapewnia elastyczność i hermetyzację danych struktury (rysunek 5.3).



Rysunek 5.3. Struktura modelu tabulatury

Korzeń struktury edytora tabulatury zaczyna się od głównej klasy *Tablature* służącej do zarządzania danymi muzycznymi.

- Interfejs *ITuning* opisuje strój otwarty w jakim tworzona jest tabulatura. Odpowiedni obiekt implementujący ten interfejs uzyskuje się z wykorzystaniem klasy *TablatureFactory* realizującej wzorzec projektowy fabryki. Model stroju

jest wspólny dla całej struktury, ponieważ w rzeczywistości tabulatura może mieć tylko jedno strojenie.

- Metadane tabulatury przechowują informacje o tabulaturze takie jak jej tytuł, autor czy opis a także ilość progów dostępnych na podstrunnicy.
- Tablica obiektów implementujących interfejs *IMeasure* odnosi się do osi czasowej tabulatury.

Za obsługę wszystkich operacji wykonywanych na nutach i pauzach znajdujących się w tabulaturze odpowiada klasa *MeasureService*, która jednocześnie realizuje interfejs serwisowy *IMeasure*.

- Dane taktu, takie jak tempo i wartości metrum używane są do obliczania jego fizycznych właściwości, w tym całkowitego czasu trwania oraz długości pełnej nuty.
- Mapowanie numerów strun na listy nut lub pauz odzwierciedla strukturę siatki tabulatury.

Obiekty *Note* przechowują informacje o właściwościach muzycznych nut znajdujących się na strunach. Dziedziczą po klasie *Sound*, która definiuje obiekt zawierający fizyczne dane dźwięku.

- Do rzeczywistych właściwości nut należą częstotliwość, czas trwania w milisekundach i moment rozpoczęcia względem taktu.
- Dane muzyczne obejmują oktawę, notację muzyczną, wartość rytmiczną dźwięku, próg na tabulaturze, technikę artykulacyjną czy informacje o tym, że dźwięk jest właśnie odtwarzany.
- Klasa *Pause* reprezentuje pauzę – jest podobna do *Note*, jednak nie posiada niektórych właściwości nuty.
- Notacja muzyczna, wartości rytmiczne oraz techniki artykulacyjne definiowane są poprzez typy wyliczeniowe *Notation*, *NoteDuration* oraz *Articulation*.

W strukturze tabulatury dla edytora, znajduje się wiele informacji, które mogą być obliczone na podstawie innych danych. Przechowywanie ich w modelu ma na celu optymalizację wydajności aplikacji umożliwiając szybkie przetwarzanie danych bez konieczności każdorazowego obliczenia tych wartości. Jednakże dane te są zbędne przy przesyłaniu tabulatury na serwer, ponieważ mogą być zrekonstruowane po stronie klienta. W systemie zaimplementowano dwie odpowiadające sobie struktury danych, po stronie klienta i serwera, zawierające wyłącznie informacje niezbędne do odtworzenia tabulatury w edytorze. Usługi **serializacji** oraz **deserializacji**

przekształcają te złożone obiekty na format *JSON* oraz odwrotnie. Na rysunku 5.4 przedstawiono tabulaturę zawierającą jeden takt z nutami na piątej oraz szóstej strunie.


```
{"title":"Przykład",
"author":"PawelDrugi",
"frets":24,
"tuning":{
  "tuning":{
    "1":{"notation":2,"octave":4},
    "2":{"notation":9,"octave":3},
    "3":{"notation":5,"octave":3},
    "4":{"notation":0,"octave":3},
    "5":{"notation":7,"octave":2},
    "6":{"notation":2,"octave":2}}},
"measures":[
  {
    "tempo":100,
    "numerator":4,
    "denominator":4,
    "notes":{
      "1":[],
      "2":[],
      "3":[],
      "4":[],
      "5":[{"fret":10,"timeStamp":1800,"noteDuration":0.0625,"articulation":0}],
      "6":[{"fret":0,"timeStamp":0,"noteDuration":0.25,"articulation":0}]
      }
  }
],
"description":"Przykładowa Tabulatura w JSONIE"}
```


Rysunek 5.4. Tabulatura w formacie JSON

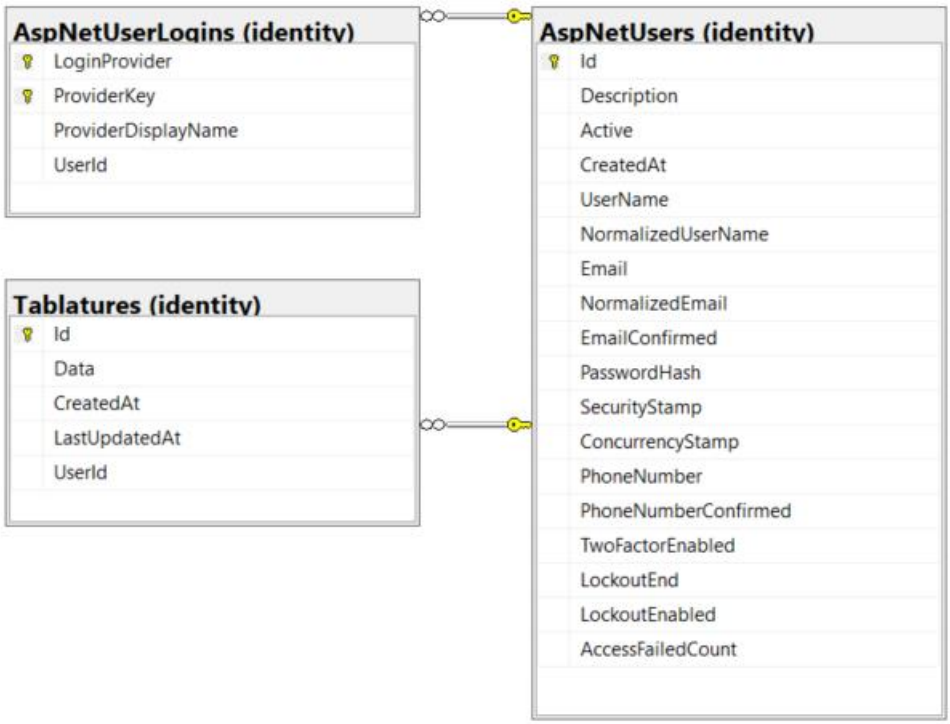
5.3.2. Baza danych

W aplikacji TabComposer zastosowano technikę mapowania obiektowo-relacyjnego pozwalającą na pracę z bazą danych w sposób obiektowy, bez konieczności ręcznego pisania złożonych zapytań SQL. Dzięki temu rozwiązaniu wszystkie operacje CRUD przeprowadzane są na obiektach w C# a EF Core tłumaczy je na odpowiednie kwerendy SQL. Aplikację zintegrowano z bazą danych SQL Server, a do jej projektowania wykorzystano podejście Code-First.

Code-First jest metodą pracy z bazą danych, polegającą na definicji modeli danych w kodzie źródłowym, na podstawie których EF Core generuje i synchronizuje jej strukturę automatycznie. Eliminuje to konieczność ręcznego tworzenia schematu bazy danych oraz zachowuje wysoką spójność między kodem aplikacji a jej strukturą. Zaletą tego rozwiązania jest możliwość elastycznego modelowania danych oraz łatwej rozbudowę bazy w miarę dodawania nowych funkcji. W tym celu wykorzystano mechanizm migracji, który pozwala na wersjonowanie zmian w schemacie bazy i automatycznie aktualizacje.

Struktura bazy danych aplikacji TabComposer jest relatywnie prosta. Składa się z kilku gotowych tabel mechanizmu Identity Framework oraz encji Tablature do przechowywania danych o tabulaturach. Identity Framework odpowiada za zarządzanie użytkownikami

dostarczając gotowe mechanizmy logowania, rejestracji, autoryzacji oraz przechowywania haseł. Dzięki temu rozwiązaniu aplikacja jest przygotowana do dalszego rozwoju i rozszerzania funkcjonalności. Na rysunku 5.5 przedstawiono wybrane tabele aplikacji.



Rysunek 5.5. Diagram bazy danych

Tabele AspNetUsers oraz AspNetUserLogins zostały wygenerowane automatycznie po integracji aplikacji z Identity. Tabela AspNetUsers powstała po zmapowaniu obiektu *AppUser*, który rozszerza standardowego użytkownika *IdentityUser* między innymi o pole *CreatedAt* oraz klucze obce do tabeli Tablatures. Tabela ta przechowuje dane tabulatur, takie jak data utworzenia, data ostatniej modyfikacji, identyfikator właściciela oraz strukturę tabulatury w formacie *JSON* przechowywaną w polu typu string. Tabele AspNetUsers i Tablature połączone są relacją **jeden do wielu**, która wskazuje, że użytkownik może mieć wiele tabel, ale każda tabela tylko jednego użytkownika. Dodatkowo wprowadzono regułę **OnDelete Cascade** polegającą na automatycznym usuwaniu tabulatur powiązanych z usuwanym użytkownikiem. Tabela AspNetUserLogins jest jedną z wielu jakie oferuje narzędzie Identity. Służy do przechowywania dodatkowych informacji dla użytkowników logujących się przez zewnętrznych dostawców, na przykład Google oraz Facebook. Dzięki obecności takich rozwiązań aplikacja jest przygotowana na dalszy rozwój i integrację z nowymi funkcjonalnościami.

5.4. Biblioteki

W projekcie wykorzystano szereg bibliotek rozszerzających funkcjonalność i usprawniających działanie aplikacji zarówno po stronie serwera jak i klienta. W tym podrozdziale omówiono najważniejsze biblioteki oraz ich zastosowanie w aplikacji.

5.4.1. Rozszerzenia backendu

- **Newtonsoft.Json** to biblioteka służąca do **serializacji** i **deserializacji** danych JSON. Pozwala na łatwe konwertowanie obiektów C# na JSON oraz odwrotnie. W aplikacji służy do walidowania danych tabulatury otrzymanych od klienta.
- Na serwerze wykorzystano także biblioteki **Entity Framework Core** oraz **Identity**, które zostały omówione w części 5.3.2 Baza danych.

5.4.2. Rozszerzenia frontendu

- **React** jest popularną biblioteką do budowy interfejsów użytkownika. Bazuje na reaktywnym podejściu, które pozwala na automatyczne aktualizowanie widoku w odpowiedzi na zmiany danych. W przeciwieństwie do frameworków takich jak Angular, koncentruje się na warstwie widoku, budując go w oparciu o komponenty funkcyjne oraz klasowe. Wszystkie widoki aplikacji TabComposer zostały zaprojektowane jako komponenty funkcyjne React.
- **Axios** to narzędzie do obsługi zapytań http. Umożliwia wysyłanie **żądań** (ang. *request*) do API oraz obsługę **odpowiedzi** (ang. *response*). W projekcie odpowiada za komunikację frontendu z serwerem, przetwarzanie zapytań oraz aktualizuje stan autoryzacji aplikacji za pomocą globalnych interceptorów.
- **Mobx** jest biblioteką służącą do **zarządzania stanem** aplikacji. Podobnie jak React opiera się na reaktywnym podejściu, jednak nie wymaga zmiany referencji obiektu do aktualizacji stanu. Wykorzystuje **obserwatory** do śledzenia reaktywnych zmiennych, umożliwiając odpowiedzi widoku na zmiany właściwości obiektów mutowalnych, co nie jest osiągalne w React bez tworzenia ich głębokich kopii (zmiany referencji). W aplikacji mobx obsługuje strukturę danych oraz widok tabulatury, pozwalając na natychmiastową aktualizację stanu nawet przy zmianie właściwości tego złożonego obiektu.
- **Tone.js** jest rozbudowanym narzędziem audio, które bazuje na Web Audio Api. Oferuje wiele funkcji, takich jak generowanie dźwięków, efekty oraz analiza sygnału. Biblioteka ta w projekcie odpowiada za odtwarzanie tabulatury,

rejestrowanie dźwięku oraz **analizę sygnałową** w dziedzinie czasu (waveform) oraz w domenie częstotliwości (FFT).

- **PitchFinder** jest biblioteką wykorzystywaną do detekcji częstotliwości dźwięku. Oferuje wydajne implementacje algorytmów takich jak AMDF, YIN czy Dynamic Wavelet.

5.5. Przegląd klas

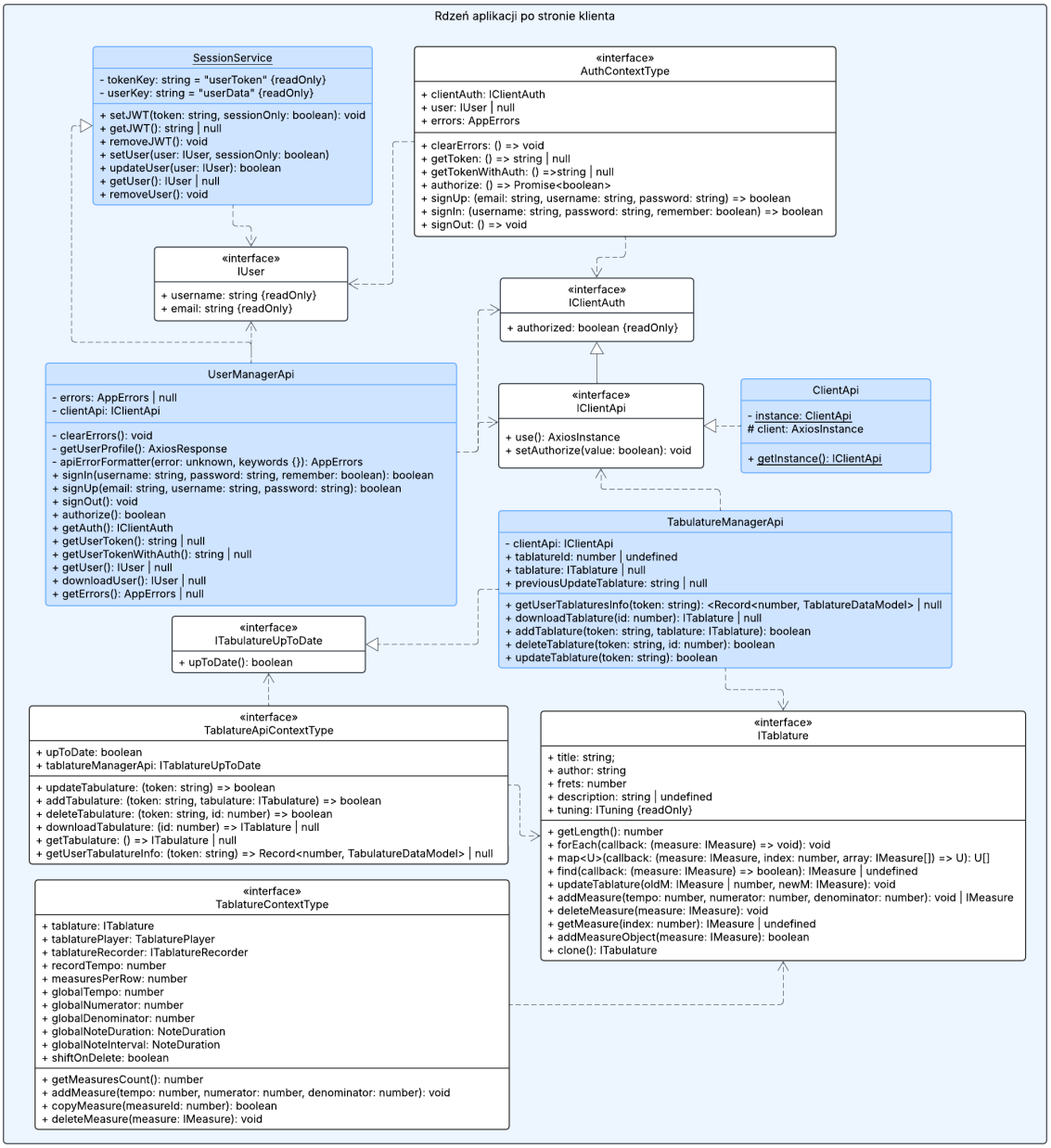
W projekcie TabComposer główną część funkcjonalności stanowią komponenty frontendowe odpowiedzialne za zarządzanie użytkownikami i tabulaturami, obsługę sesji oraz interakcję z API. Istotną rolę pełni również moduł analizy dźwięku, wykorzystywany do ekstrakcji informacji muzycznych z sygnału audio. W tym podrozdziale omówione zostaną najważniejsze klasy frontendu obejmujące rdzeń aplikacji oraz moduł audio.

5.5.1. Rdzeń aplikacji

Za główną funkcjonalność aplikacji odpowiadają moduły api, context, models oraz services, które dzielą odpowiedzialność za realizację poszczególnych operacji. Diagram najważniejszych klas przedstawiony jest na rysunku 5.6.

- *ClientApi* to główna klasa singleton odpowiedzialna za obsługę żądań http w aplikacji. Wykorzystuje w tym celu bibliotekę Axios i używa globalnych interceptorów do ustawiania statusu autoryzacji klienta. Pole *authorized* jest obserwowane przez mechanizmy biblioteki mobx, co pozwala na natychmiastową reakcję aplikacji przy zmianie tego stanu.
- *UserManagerApi* służy do zarządzania użytkownikami w aplikacji. Używa instancji *ClientApi* do komunikacji z backendem. Obsługuje logowanie (*singIn*), rejestrację (*signUp***)**, wylogowanie (*signOut***)** oraz pobieranie informacji o użytkowniku (*downloadUser*) w asynchroniczny sposób. Wykorzystuje klasę *SessionService* do operacji na danych użytkowników zapisanych w sesji lub w pamięci lokalnej przeglądarki.
- *TablatureManagerApi* jest klasą umożliwiającą dostęp do danych tabulatur z serwera oraz ich modyfikację poprzez asynchroniczne metody (*downloadTablature***,** *addTablature***,** *updateTablature***,** *deleteTablature* korzystając w tym celu z instancji *ClientApi*. Dodatkowo przechowuje obserwowalny obiekt *Tablature*, którego modyfikacje wpływają natychmiastowo na widok aplikacji

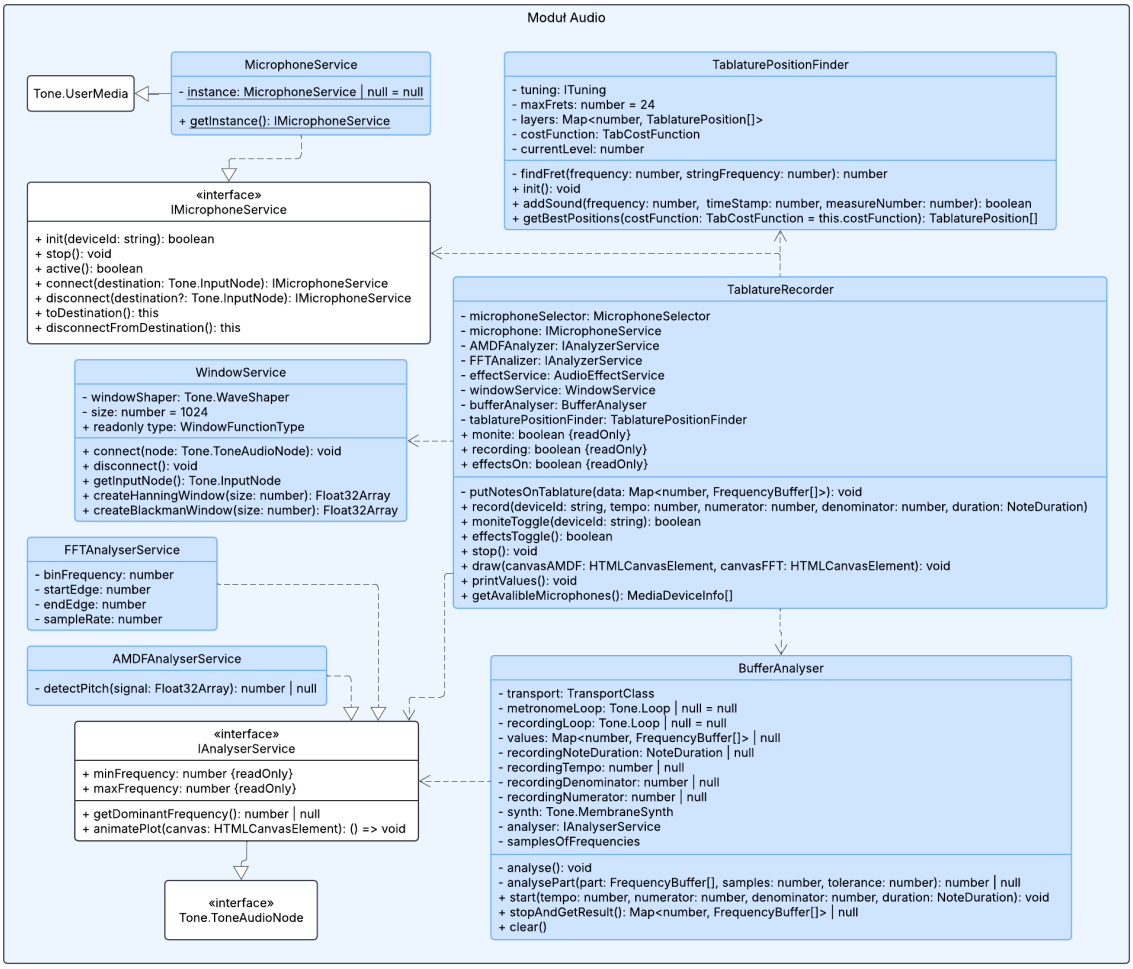
- Interfejsy takie jak *AuthContextType*, *TablatureApiContextType* oraz *TablatureContextType* definiują konteksty zarządzania funkcjonalnością aplikacji w widokach React za pomocą mechanizmu useContext.



Rysunek 5.6. Rdzeń aplikacji TabComposer

5.5.2. Moduł audio

W aplikacji TabComposer wszystkie operacje na sygnale dźwiękowym, takie jak rejestrowanie dźwięku, przetwarzanie danych, analiza i interpretacja wyników wykonywane są w module audio. Diagram najważniejszych komponentów do pracy z sygnałem jest przedstawiony na rysunku 5.7.



Rysunek 5.7. Moduł Audio aplikacji TabComposer

- *MicrophoneService* to klasa singleton służąca do obsługi wejścia mikrofonowego w aplikacji. Rozszerza klasę *Tone.UserMedia* dzięki czemu może być łączona w łańcuchy przetwarzania dźwięku. W aplikacji stanowi punkt wejściowy do analizy audio.
- *IAnalyserService* jest interfejsem deklarującym metody dla różnych implementacji analizatorów dźwięku. Dziedziczy metody interfejsu *Tone.ToneAudioNode* uzyskując definicje metod służących do przetwarzania sygnału.
- *AMDFAnalyserService* oraz *FFTAnalyserService* są klasami realizującymi *IAnalyserService* oraz rozszerzającymi klasę *Tone.Analyser* o funkcjonalność detekcji częstotliwości fundamentalnej analizowanego sygnału. Dodatkowo posiadają metody do graficznej reprezentacji analizy. Pierwsza z nich, wykorzystuje bibliotekę PitchFinder i wykonuje algorytm **AMDF** na sygnale w dziedzinie czasu. Druga klasa korzysta z wbudowanego **FFT** w Tone.js i analizuje widmo amplitudowe sygnału w domenie częstotliwości za pomocą wzorów (20, 29) tak jak przedstawiono na rysunku 5.8.

```
12    this.binFrequency = this.sampleRate / (this.size * 2);
13    this.startEdge = Math.ceil(this.minFrequency / this.binFrequency);
14    this.endEdge = Math.floor(this.maxFrequency / this.binFrequency);
.
.
.
18    getDominantFrequency(): number | null {
19      const magnitudes = this.getValue() as Float32Array;
20
21      let maxIndex = this.startEdge;
22      let maxValue = magnitudes[this.startEdge];
23
24      for (let i = this.startEdge; i <= this.endEdge; i++) {
25        if (magnitudes[i] > maxValue) {
26          maxValue = magnitudes[i];
27          maxIndex = i;
28        }
29      }
30
31      const frequency = (maxIndex * this.sampleRate) / (this.size * 2);
32
33      return maxValue > -Infinity ? frequency : null;
34    }
```

Rysunek 5.8 Wyznaczenie dominującej częstotliwości z FFT w kodzie

- *BufferAnalyser* jest klasą analizującą próbki częstotliwości podstawowej, które wyznaczono podczas rejestrowania dźwięku. Odpowiada za wybranie najbardziej **stabilnej częstotliwości** występującej w danym przedziale czasu, wyznaczonym przez tempo i metrum taktu oraz spodziewaną wartość rytmiczną nuty.
- *TablaturePositionFinder* stanowi implementację szczególnego **grafu skierowanego**, którego wierzchołki układają się w kolejne warstwy, przypominając swoją strukturą schemat jednokierunkowej sieci neuronowej (rysunek 2.8). Każda z warstw zawiera wszystkie możliwe pozycje występowania jednego dźwięku na podstrunnicy. Metoda *addSound* dodaje kombinacje numerów strun i progów pod jakimi znajduje się określona częstotliwość. W tym celu wykorzystuje wzór (31) wyznaczając odległość tonalną pomiędzy częstotliwościami otwartej struny a rozpatrywanego dźwięku (rysunek 5.9). Pomiędzy wierzchołkami sąsiednich warstw obliczany jest koszt przejścia w zależności od numerów strun oraz progów (34). Uproszczony algorytm Djikstry *getBestPositions* znajduje najlepsze rozmieszczenie dźwięków na gryfie.
- *WindowService* to klasa zajmująca się generowaniem okien czasowych dla analizy dźwięku w domenie częstotliwości. Wykorzystuje klasę *Tone.WaveShaper* do modyfikacji strumienia sygnału. Umożliwia użycie okna prostokątnego, okna Hanna oraz okna Blackmana.

- *TablatureRecorder* jest główną klasą zarządzającą procesami nagrywania, przetwarzania, analizowania oraz umieszczania dźwięków na tabulaturze. Łączy klasy mikrofonu, efektów dźwiękowych i analizatorów w jeden łańcuch przetwarzania sygnału.

```
26   private findFret(frequency: number, stringFrequency: number): number {
27     return Math.round(12 * Math.log2(frequency / stringFrequency));
28   }
29
30   public addSound(
31     frequency: number,
32     timeStamp: number,
33     measureNumber: number
34   ): boolean {
35     this.layers.set(this.currentLevel, []);
36     const currentLayer = this.layers.get(this.currentLevel);
37     this.tuning.forEach((string: number, openSound: Sound) => {
38       const fret = this.findFret(frequency, openSound.frequency);
39       if (fret >= 0 && fret <= this.maxFrets) {
40         currentLayer?.push(
41           new TablaturePosition(
42             fret,
43             Number(string),
44             timeStamp,
45             measureNumber
46           )
47         )
48       }
49     })
50     if (currentLayer?.length === 0)
51       return false;
52     this.currentLevel++;
53     return true;
54   }
```

Rysunek 5.9 Dodawanie warstwy pozycji nut w kodzie

Rozdział 6

Weryfikacja i walidacja

6.1. Sposoby testowania

Rozwój aplikacji TabComposer odbywał się na zasadach modelu iteracyjno-przyrostowego. W miarę postępów prac projektowane i implementowane były nowe funkcjonalności, które testowano odrębnie od istniejących modułów. Weryfikacja i walidacja dotyczyła testów jednostkowych, integracyjnych oraz systemowych.

- **Testy jednostkowe** polegały na ocenie działania metod oraz pojedynczych klas projektu. Przeprowadzano je ręcznie poprzez uruchamianie aplikacji i analizowanie danych wyjściowych. Badano odporność na błędy oraz weryfikowano działanie logiki dla różnych wartości brzegowych.
- **Testy integracyjne** sprowadzały się do łączenia zweryfikowanych modułów w większe grupy testowe. Sprawdzano poprawność komunikacji między ich interfejsami a także warstwami systemu. W szczególności skupiono się na prawidłowej obsłudze błędów otrzymywanych od innych komponentów aplikacji, a także na poprawności przesyłanych danych pomiędzy klientem a serwerem.
- **Testy systemowe** obejmowały całościowe sprawdzanie działania systemu dla różnych scenariuszy użycia. W głównej mierze oceniano odporność aplikacji na błędne dane wprowadzane przez użytkownika.

Każdy moduł aplikacji został przetestowany pod kątem jego funkcjonalności oraz integracji z innymi komponentami systemu w ograniczonym zakresie, dla określonych przypadków testowych.

6.2. Organizacja i zakres testowania

Pierwsza iteracja aplikacji obejmowała zaprojektowanie oraz zaimplementowanie backendu systemu. Głównym celem było wdrożenie i skonfigurowanie usług bazodanowych, serwisów odpowiedzialnych za obsługę kont użytkowników oraz ich warstwy prezentacji. Za narzędzie do testowania tych funkcjonalności posłużył Swagger, umożliwiający wykonanie zapytań do API aplikacji. Sprawdzono działanie mechanizmów logowania, rejestracji, otrzymywania danych użytkownika oraz autoryzacji tokenem JWT.

W drugiej iteracji utworzono szkielet frontendu aplikacji, zawierający jej wstępny podział funkcjonalny. Zaimplementowano obsługę logowania i rejestracji oraz mechanizm lokalnego przechowywania informacji o użytkowniku. Testy jednostkowe dotyczyły modułów zapisujących dane w pamięci przeglądarki oraz w kontekście użytkownika aplikacji. Testy integracyjne sprawdzały poprawność komunikacji frontendu z backendem oraz reakcje aplikacji na otrzymywane informacje.

Trzecia iteracja dotyczyła wdrożenia edytora tabulatury do klienta systemu. W jej trakcie utworzono wiele pojedynczych serwisów wykonujących określone akcje oraz modeli danych dla widoku aplikacji. W pierwszej fazie testowano każdą nową funkcjonalność w indywidualny sposób. Następnie sprawdzano, jak poszczególne elementy aplikacji oddziałują ze sobą. Ostatnim etapem była ocena całokształtu działania edytora tabulatury.

W ramach czwartej iteracji utworzono serwisy serializacji oraz deserializacji danych tabulatury zarówno po stronie klienta oraz serwera. Utworzono warstwę repozytorium backendu oraz skonfigurowano nowe endpointy warstwy prezentacji dla danych tabulatur. Ze względu na konieczność spójności danych przechowywanych w bazie z strukturami obecnymi na frontendzie, dużą uwagę poświęcono testom jednostkowym klas konwertujących DTO. Następnie skupiono się na testach integracyjnych, które weryfikowały poprawność danych przesyłanych pomiędzy warstwami systemu. Wysyłano zapytania do API zawierające dane tabulatur z poprawnymi oraz błędnymi strukturami w formacie JSON. Sprawdzano także zachowania systemu na brak wymaganej autoryzacji, przy żądaniach takich jak uzyskanie dostępu do danych konkretnego użytkownika.

W następnych iteracjach wdrożono moduł audio aplikacji. W pierwszej kolejności testowano jego działanie w sytuacjach wyjątkowych, takich jak brak dostępnego mikrofonu bądź odtwarzacza dźwięku. Kluczową fazą tworzenia całego systemu było weryfikowanie wyników analizy sygnałowej dla różnych metod uzyskiwania częstotliwości fundamentalnej. Konsekwencją tych testów był wybór metody **AMDF** jako sposobu detekcji dźwięku w aplikacji.

6.3. Usunięte błędy

Podczas testowania systemu zidentyfikowano oraz usunięto szereg błędów dotyczących różnych modułów aplikacji. Zdecydowana większość z nich została wykryta na etapie testów integracyjnych.

- **Niepotrzebne rendery komponentów interfejsu użytkownika**. W trakcie testowania interfejsu edytora tabulatury zaobserwowano, że komunikaty wysyłane do konsoli w przeglądarce pojawiają się kilkukrotnie. Użycie komponentu *<StrictMode>* jako nadrzędnego dla całej struktury projektu powinno wymusić jedynie podwójny render komponentów do celów testowych. Błędy w logice zarządzania stanem, powodujące zbędne odświeżanie komponentów, zostały usunięte a kod zoptymalizowano.
- **Błędy zarządzania stanem złożonych obiektów**. Podczas testowania edytora tabulatury zauważono, że mimo zmian zachodzących w modelu, widok nie odświeżał się poprawnie. Błąd ten wynikał z niezmienianej referencji obiektu *Tablature*. Początkowo obrano podejście polegające na tworzeniu głębokich kopii wszystkich obiektów tej struktury i zmienianiu referencji w komponentach React. Jednakże dla dużych struktur rozwiązanie to nie było optymalne. Ostatecznie wykorzystano bibliotekę mobx oraz mechanizmy obserwatorów do prawidłowej aktualizacji stanu.
- **Błędy DTO**. W fazie integracji frontendu z backendem dotyczącej przesyłania danych tabulatur zaobserwowano, że zapisują się niepoprawnie w bazie danych. Błąd ten wynikał z braku deklaracji niektórych właściwości DTO na serwerze jako nullowalnych. Inne błędy napotkane w tej iteracji dotyczyły literówek nazw pól DTO oraz faktu, że backend traktował klucze JSON jako czułe na wielkość liter (ang. *case-sensitive*).
- **Błędy modułu audio**. W trakcie testów integracyjnych funkcjonalności dotyczącej przetwarzania dźwięku zwrócono uwagę, że klasa dostarczająca efekty dźwiękowe, takie jak bramka szumów oraz filtry przepustowe nie ma wpływu na sygnał. Powodem była niepoprawna kolejność elementów w łańcuchu przetwarzania sygnału. Innym podobnym błędem było nieprawidłowe umieszczenie klasy okna czasowego w łańcuchu, co powodowało błędne wyniki analizatora AMDF.

Rozdział 7

Podsumowanie i wnioski

7.1. Uzyskane wyniki

Aplikację TabComposer udało się zaimplementować w stopniu zadowalającym. Wszystkie minimalne wymagania dotyczące systemu zostały spełnione a główny cel pracy – „Wspomaganie tworzenia tabulatur gitarowych z wykorzystaniem analizy dźwięku instrumentu" – osiągnięto z powodzeniem. Zgodnie z początkowymi założeniami wdrożono system kont użytkowników, edytor tabulatury z możliwością odtwarzania piosenek oraz moduł analizy dźwięku. Aplikacja jest prosta w obsłudze, posiada jednolity styl oraz responsywny design. Interfejs użytkownika działa płynnie i wydajnie, nawet podczas przetwarzania sygnału audio. System dba o bezpieczeństwo danych użytkowników i automatycznie zapisuje postępy ich pracy.

- **System kont użytkowników** pozwala na dostęp do przechowywanych tabulatur w bazie danych aplikacji. Umożliwia różne operacje na tabulaturach takie jak tworzenie, modyfikację, odtwarzanie oraz usuwanie tabulatur.
- **Edytor tabulatury** dostępny jest dla użytkowników zalogowanych i niezalogowanych. Udostępnia szereg operacji na tabulaturze związanych z obsługą taktów, nut oraz metadanych tabulatury. Posiada pasek narzędzi z zakładką ułatwiającą pracę z aplikacją poprzez definiowanie domyślnych ustawień dla nowych elementów tabulatury.
- **Odtwarzacz tabulatury** występuje jako zakładka w pasku narzędzi. Służy do odtwarzania dźwięków umieszczonych na tabulaturze. Możliwe jest dostosowanie prędkości odtwarzania a także wskazanie taktu, od którego zacznie się odgrywanie tabulatury. Dodatkowo, proces ten można wstrzymać lub anulować. Odtwarzacz jest również umieszczony w widoku tabulatury tylko do odczytu.
- **Analizator dźwięku** dostępny jest w ostatniej zakładce paska narzędzi. Rejestrowanie dźwięku rozpoczyna się po odegraniu jednego taktu przez

- metronom. W zależności od metrum oraz długości nut ustawionych jako domyślne, analizator dopasowuje stabilne częstotliwości do odpowiednich ram czasowych taktu, wskazując ich początek jako czas początkowy nuty. Inną funkcjonalnością analizatora jest rysowanie wizualizacji sygnału w domenie czasu oraz częstotliwości. Użytkownik ma możliwość wybrać urządzenie wejściowe z listy dostępnych mikrofonów.
- **Mechanizm autosave** dba, aby minimalizować ryzyko utraty pracy poprzez przypadkowe zamknięcie aplikacji bądź edytora. Co każde pięć sekund od momentu wykrycia zmian, tabulatura jest automatycznie aktualizowana.

7.2. Napotkane problemy

Podczas projektowania i implementacji aplikacji napotkano wyzwania teoretyczne oraz technologiczne, związane z zastosowaniem nieznanych dotąd technologii i bibliotek, a także z wyborem odpowiedniego podejścia i jego realizacją. Mimo to każdy z problemów udało się rozwiązać i żaden z nich nie utrudnił realizacji projektu w znaczący sposób.

Jednym z głównych założeń aplikacji TabComposer jest wydajny i dynamiczny interfejs użytkownika. Wykorzystanie React w projekcie zapewniło odpowiedni balans pomiędzy poziomem skomplikowania użycia a celem do osiągnięcia. Na przestrzeni pierwszych dni realizacji frontendu, poznawano funkcje tej biblioteki a także składnię i możliwości języka TypeScript. Wykryte błędy związane z zarządzaniem stanem aplikacji wynikały z niewystarczającego poziomu poznania tej technologii. Niemniej jednak, dzięki takiemu podejściu stopniowo zgłębiano fundamentalne aspekty React i stale optymalizowano kod, uzyskując satysfakcjonujące wyniki.

Podczas implementowania modułu audio aplikacji największym wyzwaniem okazało się znalezienie odpowiedniej metody analizy sygnału, która umożliwiłaby wykrywanie jego częstotliwości fundamentalnej. Początkowo skupiono się na analizie widma amplitudowego uzyskiwanego za pomocą FFT. Jednakże metoda ta nie działała dobrze dla częstotliwości z zakresu od 50 do 200 Hz, a uzyskiwane wyniki często były kolejnymi harmonicznymi częstotliwości podstawowej.

7.3. Rozwój aplikacji

Obecny stan produkcyjny aplikacji TabComposer pozostawia kilka różnych obszarów systemu otwartych do dalszego rozwoju. Architekturę warstwową systemu wraz z elastycznymi bibliotekami, takimi jak EF Core czy Identity, wybrano z myślą o rozbudowie funkcjonalnej systemu w przyszłości. Docelowo aplikacja TabComposer ma stać się wielofunkcyjną platformą społecznościową dla gitarzystów.

- **Normalizacja bazy danych**. Baza danych aplikacji TabComposer przechowuje dane tabulatury w postaci ciągu znaków. Zapisywanie tabulatur jako obiektów JSON do jednej tabeli jest relatywnie prostym rozwiązaniem, wystarczającym do funkcjonowania systemu. Jednakże, sposób ten przy wzroście ilości rekordów przechowywanych w bazie może prowadzić do redundancji danych i negatywnie wpływać skalowalność systemu. W ramach pierwszego etapu rozbudowy systemu planowana jest refaktoryzacja i powiększenie bazy danych o dodatkowe tabele odpowiadające strukturze tabulatury. Wyeliminuje to nadmiarowość a rozmiar danych przesyłanych pomiędzy klientem a serwerem zostanie zmniejszony.
- **Publiczny dostęp do tabulatur oraz profile użytkowników**. W początkowym zamyśle aplikacja TabComposer miała zostać wyposażona w wyszukiwarkę tabulatur udostępnionych przez użytkowników. Mieliby oni możliwość oznaczania swoich utworów jako publiczne – każdy użytkownik posiadałby dostęp do czytania oraz odtwarzania takiej tabulatury. Rozważano także wprowadzenie systemu rankingu, ocen, komentarzy oraz ról użytkowników, jednakże ze względu na możliwości projektowe zrezygnowano z implementacji tych funkcji, które zostają do wdrożenia w przyszłości.
- **Analiza z wykorzystaniem uczenia maszynowego**. W aplikacji TabComposer zdecydowano się na typowo analityczne podejście do problemu analizy dźwięku, które polega na badaniu różnych parametrów danego sygnału. Alternatywną metodą generowania tabulatury na podstawie dźwięku gitary jest wykorzystanie algorytmów uczenia maszynowego. Pierwotnie planowano wybrać i wytrenować odpowiedni model matematyczny, jednakże metoda ta wymaga przygotowania bardzo dużych zbiorów danych treningowych, co pochłonęłoby zbyt wiele czasu. Niemniej jednak dobrze wyuczony algorytm mógłby wzbogacić funkcjonalność aplikacji o wykrywanie tempa, długości granych nut czy nawet metrum, dlatego rozwiązanie to jest planowane do wdrożenia w przyszłości.

Bibliografia

[1] Erich M. von Hornbostel, Curt Sachs. Systematik der Musikinstrumente. *Zeitschrift für Ethnologie*, 553-590, 1914.

[2] Denyer Ralph. *The Guitar Handbook: A Unique Source Book for the Guitar Player*. Knopf, Londyn, 1992.

[3] https: How to read tab | Songsterr Tabs with Rhythm (dostęp:29.09.2024)

[4] Alan V. Oppenheim, Alan S. Willsky. *Signals and Systems*. Prentice Hall, Upper Saddle River, 1997.

[5] https: Sygnały i przebiegi cyfrowe, i analogowe | Części elektroniczne. Dystrybutor i sklep online - Transfer Multisort Elektronik Polska (dostęp:07.11.2024)

[6] https: Zero crossing - Wikipedia (dostęp:09.11.2024)

[7] https: Algorithmic frequency/pitch detection series - Part 1: Making a simple pitch tracker using Zero Crossing Rate | by Rishikesh Daoo | The Seeker's Project | Medium (dostęp:10.11.2024)

[8] https: Pitch Detection Methods (dostęp:17.11.2024)

[9] Alain de Cheveigne, Hideki Kawahara. YIN, a fundamental frequency estimator for speech and music. The Journal of the Acoustical Society of America, 111(5), 1900–1911, 2002.

[10] Myron J. Ross, Harry L. Shaffer, Andrew Cohen, Richard Freudberg, Harold J. Manley. Average Magnitude Difference Function Pitch Extractor. IEEE Transactions On Acoustics, Speech, And Signal Processing. Vol. ASSP-22, No. 5, 1974.

[11] Tan, Li, Montri Karnjanadecha. Pitch detection algorithm: autocorrelation method and AMDF. *Proceedings of the 3rd international symposium on communications and information technology*. Vol. 2. 2003.

[12] James W. Cooley, John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. Mathematics *of Computation*, 19(90): 297-301. 1965.

[13] Heckbert, Paul. "Fourier transforms and the fast Fourier transform (FFT) algorithm. *Computer Graphics* (1995): 15-463. 1995.

[14] https: Fast Fourier transform - Wikipedia (dostęp:22.12.2024)

[15] https: Spectral leakage - Wikipedia (dostęp:05.01.2025)

[16] https: Window function - Wikipedia (dostęp:06.01.2025)

[17] de la Cuadra, Patricio, et al. *Efficient Pitch Detection Techniques for Interactive Music*. Center for Computer Research in Music and Acoustics, Stanford University, n.d.

[18] Javaid, Adeel. Understanding Dijkstra's algorithm. *Available at SSRN 2340905*, 2013.

Dodatki

Spis skrótów i symboli

TAB	skrótowy zapis tabulatury, także symbol
ZCR	zero crossing rate
AMDF	średnia różnica magnitudy (ang. *Average Magnitude Differential Function)*
FFT	szybka transformata Fouriera (ang. *Fast Fourier Transform*)
DFT	dyskretna transformata Fouriera (ang. *Discrete Fourier Transform*)
HPS	algorytm wykywania wysokości dźwięku (ang. *Harmonic Product Spectrum*)
ZP	dodanie zer na koniec sygnału (ang. *Zero Padding*)
LPF	filtr dolnoprzepustowy (ang. *Low Pass Filter*)
HPF	filtr górnoprzepustowy (ang. *High Pass Filter*)
RESTAPI	interfejs komunikacji pomiędzy aplikacjami (ang. *Representational State Transfer Application Programming Interface*)
EF Core	framework zarządzania bazą danych (ang. *Entity Framework Core*)
SPA	aplikacja jednostronnicowa (ang. *Single Page Application*)
JWT	token autoryzacji (ang. *JSON Web Token*)
CRUD	operacje na bazie danych (ang. *Create, Read, Update, Delete*)
DTO	obiekty transferu danych (ang. *Data Transfer Objets*)
DI	wstrzykiwanie zależności (ang. *Dependency Injection*)
ORM	mapowanie obiektowo-relacyjne (ang. *Object-Relational Mapping*)
UML	język modelowania wizualnego (ang. *Unified Modeling Language*)

Lista dodatkowych plików, uzupełniających tekst pracy

W systemie, do pracy dołączono dodatkowe pliki zawierające:

- źródła programu,
- film pokazujący działanie opracowanego oprogramowania.

Spis rysunków

Spis tablic