Uniwersytet Mikołaja Kopernika w Toruniu

Wydział Matematyki i Informatyki

Daniel Krzykowski

nr albumu: 308308

Informatyka

Praca inżynierska

Aplikacja do obróbki warstw ścieżek dźwiękowych

Promotor

Dr Marta Jadwiga Burzańska

Toruń 2023/2024

Spis treści

[Wstęp 2](#_Toc165485008)

[Cel projektu 2](#_Toc165485009)

[Użyte technologie 3](#_Toc165485010)

[Rozpoznanie z konkurencją 4](#_Toc165485011)

[Prezentacja elementów aplikacji 10](#_Toc165485012)

[Interfejs Użytkownika 10](#_Toc165485013)

[Navbar 11](#_Toc165485014)

[Sidebar 12](#_Toc165485015)

[Multitrack 13](#_Toc165485016)

[Struktura projektu 17](#_Toc165485017)

[Implementacja najważniejszych funkcjonalności 20](#_Toc165485018)

[Testowanie i naprawa błędów 20](#_Toc165485019)

[Przebieg pracy 20](#_Toc165485020)

[Podsumowanie 20](#_Toc165485021)

[Bibliografia 21](#_Toc165485022)

# Wstęp

Zastanawiając się nad tematem do mojej pracy inżynierskiej, zauważyłem, że aplikacje open source do prostej obróbki audio na komputery takie jak Audacity są strasznie ubogie w możliwości oraz mało intuicyjne dla przeciętnego użytkownika. Pomyślałem, więc że dobrym rozwiązaniem będzie stworzenie bardziej przyjaznego dla przeciętnego użytkownika edytora ścieżek audio, który ma możliwości prostej obróbki dźwięków na wielu warstwach.

Praca, którą wykonałem będzie skupiać się na stworzeniu przeglądarkowej aplikacji DAW z możliwością łatwej i interaktywnej korekcji ścieżek audio. Przeznaczenie aplikacji będzie opierać się głównie na ludziach, chcących robić „remiksy” (1) oraz „mashupy” (2). Aplikacja będzie posiadać możliwości takie jak: wydłużanie/skracanie ścieżek, zwalnianie i przyspieszanie ich w czasie oraz planowana jest możliwość automatyzacji głośności danych ścieżek w prosty sposób. Dodatkiem do aplikacji ma być biblioteka audio, która może importować pliki audio.

# Cel projektu

Celem tego projektu było stworzenie aplikacji webowej, która będzie dawała możliwość użytkownikowi przetwarzanie i obróbkę ścieżek audio dając podstawowe narzędzia do edycji takich ścieżek na wielu warstwach.

Aby zrealizować ten projekt wykorzystałem w pewnym stopniu plugin do biblioteki wavesurfer.js (3), który tworzy nam Multitrack (4) wizualizujący ścieżki audio na wielu warstwach, synchronizuje wszystkie ścieżki, aby móc je odtwarzać w tym samym czasie oraz umożliwia zaznaczanie ich regionu dzięki pluginowi region (5). Dodaje on także automatyzacje głośności na całej ścieżce z innego pluginu od wavesurfer.js (envelope) (6).

Plugin Multitrack jest pluginem, który został napisany przez 5 osobowy zespół i został udostępniony na Github na licencji BSD-3-Clause (7) co oznacza, że projekt ten może być modyfikowany i wykorzystywany pod warunkiem spełnienia 3 warunków opisanych w tej licencji. Projekt ten nie ma bezpłatnego wsparcia a proponowanie nowych funkcji można kierować pod adres głównego autora tego pluginu. Plugin ten zawiera również prostą dokumentację co ułatwia zrozumienie struktury i funkcji tego programu.

W celu lepszej kontroli nad tym pluginem i rozbudowania go o większe funkcjonalności w moim projekcie, skopiowałem pliki z folderu /src (8) projektu wavesurfer-multitrack oraz wykorzystałem przykład ze strony wavesurfer.xyz (3).

# Użyte technologie

Technologie, jakich użyłem do stworzenia mojej aplikacji oraz funkcjonalności większości operacji na ścieżkach audio były między innymi:

* [JavaScript (ES13)](https://pl.wikipedia.org/wiki/JavaScript) (9)
* [React](https://react.dev/) (10)
* [TypeScript](https://www.typescriptlang.org/) (11)
* [Web Audio API (v1.0)](https://github.com/WebAudio/web-audio-api) (12)
* [wavesurfer.js](https://github.com/katspaugh/wavesurfer.js) (13)
* [Bootstrap](https://getbootstrap.com/docs/5.3/getting-started/introduction/) (14)
* [Zustand](https://github.com/pmndrs/zustand) (15)
* [Framer-motion](https://www.framer.com/motion/) (16)
* [audiobuffer-to-wav](https://github.com/Experience-Monks/audiobuffer-to-wav) (17)
* [lamejs](https://github.com/zhuker/lamejs) (18)

# Rozpoznanie z konkurencją

Pierwszym krokiem, jaki podjąłem w realizowaniu mojej pracy było znalezienie i zrozumienie otoczenia konkurencyjnego, z którego zbierałem informację na temat rozwiązań użytych przed dane firmy i osoby, które stworzyły produkty z zakresu tworzenia muzyki. W ramach tego procesu, postarałem się zidentyfikować, trendy, rozwiązania oraz mocne i słabe strony owych produktów, aby móc zacząć myśleć nad pomysłami i rozwiązaniami dla mojego projektu.

W pierwszej kolejności przy szukaniu informacji o konkurencji skupiłem się na tym, aby znaleźć konkurencję, która tworzy programy DAW *(ang. Digital Audio Workstation), czyli programy komputerowe przeznaczone do pracy z dźwiękiem.* (19)Na rynku jest wiele takich programów, ale postanowiłem, że skupię się w ramach projektu na programach DAW, które opierają swoje działanie o przeglądarki internetowe. Dlatego też z góry odrzuciłem programy DAW przeznaczone na komputery PC oraz urządzenia mobilne.

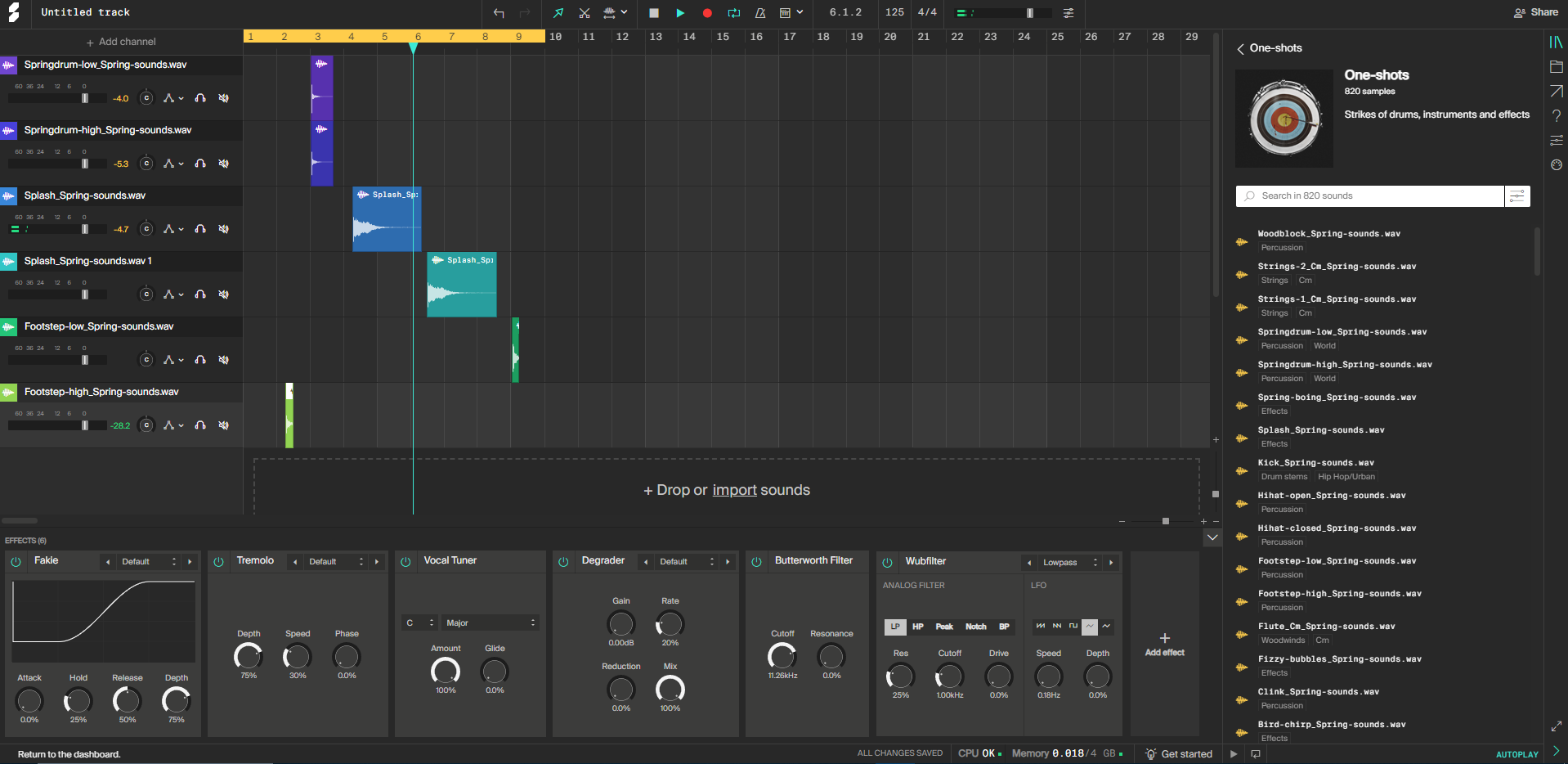
Ważnym elementem analizy konkurencji było zgromadzenie danych dotyczących dostępności informacji. Stworzyłem, więc listę programów, które będziemy analizować pod kątem owej dostępności informacji.

Wśród nich mamy kilka programów przeglądarkowych DAW:

* [BandLab](https://www.bandlab.com/) (20)
* [Audiotool](https://studio.audiotool.com/) (21)
* [Soundation](https://soundation.com/) (22)
* [Amped Studio](https://ampedstudio.com/) (23)
* [Soundtrap](https://www.soundtrap.com/) (24)
* [SkyTracks](https://skytracks.io/) (25)
* [GridSound](https://github.com/gridsound/daw) (26)



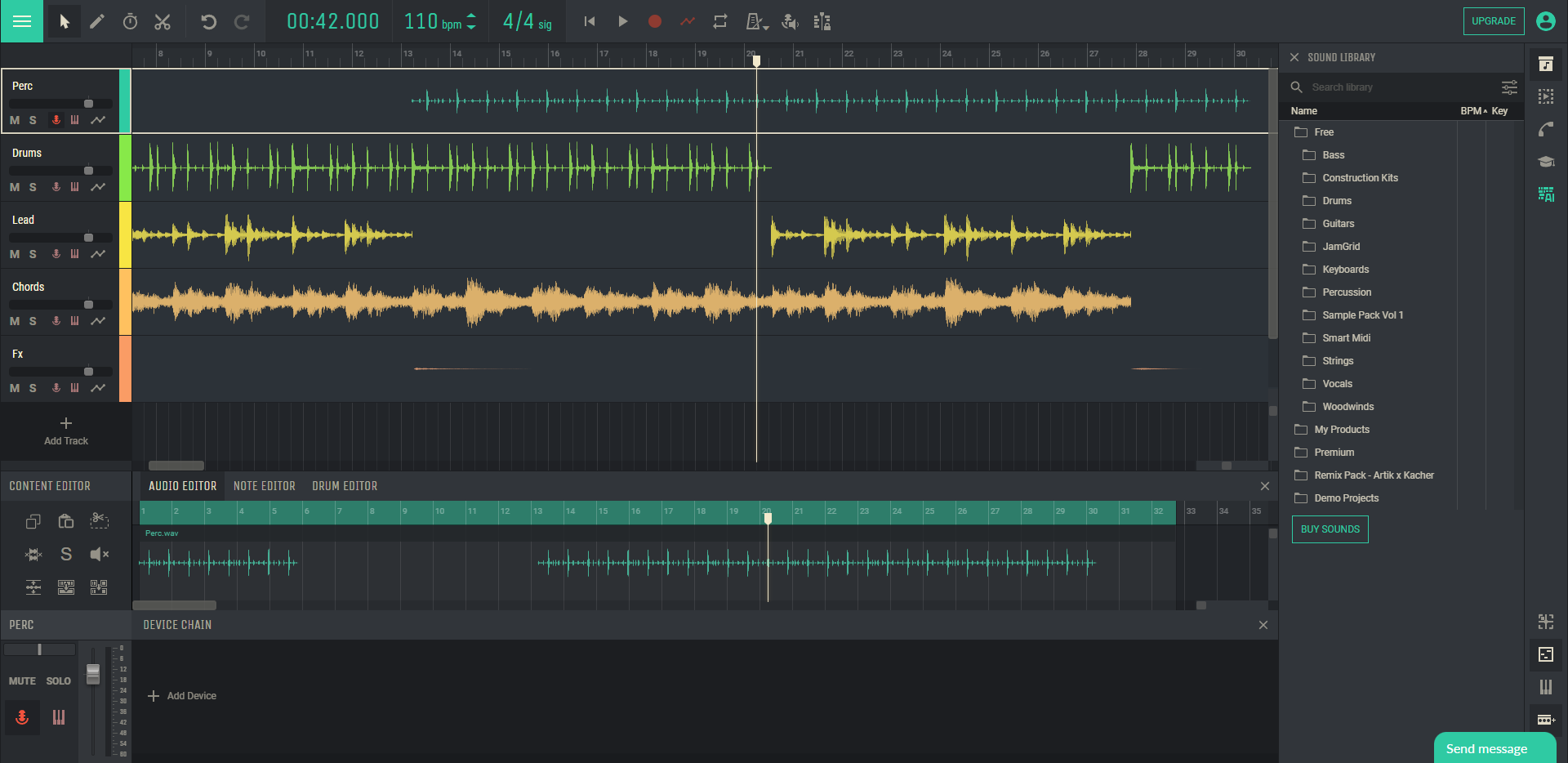
Interfejs graficzny programu Bandlab



Interfejs graficzny programu Soundation



Interfejs graficzny programu Audiotool



Interfejs graficzny programu Amped Studio

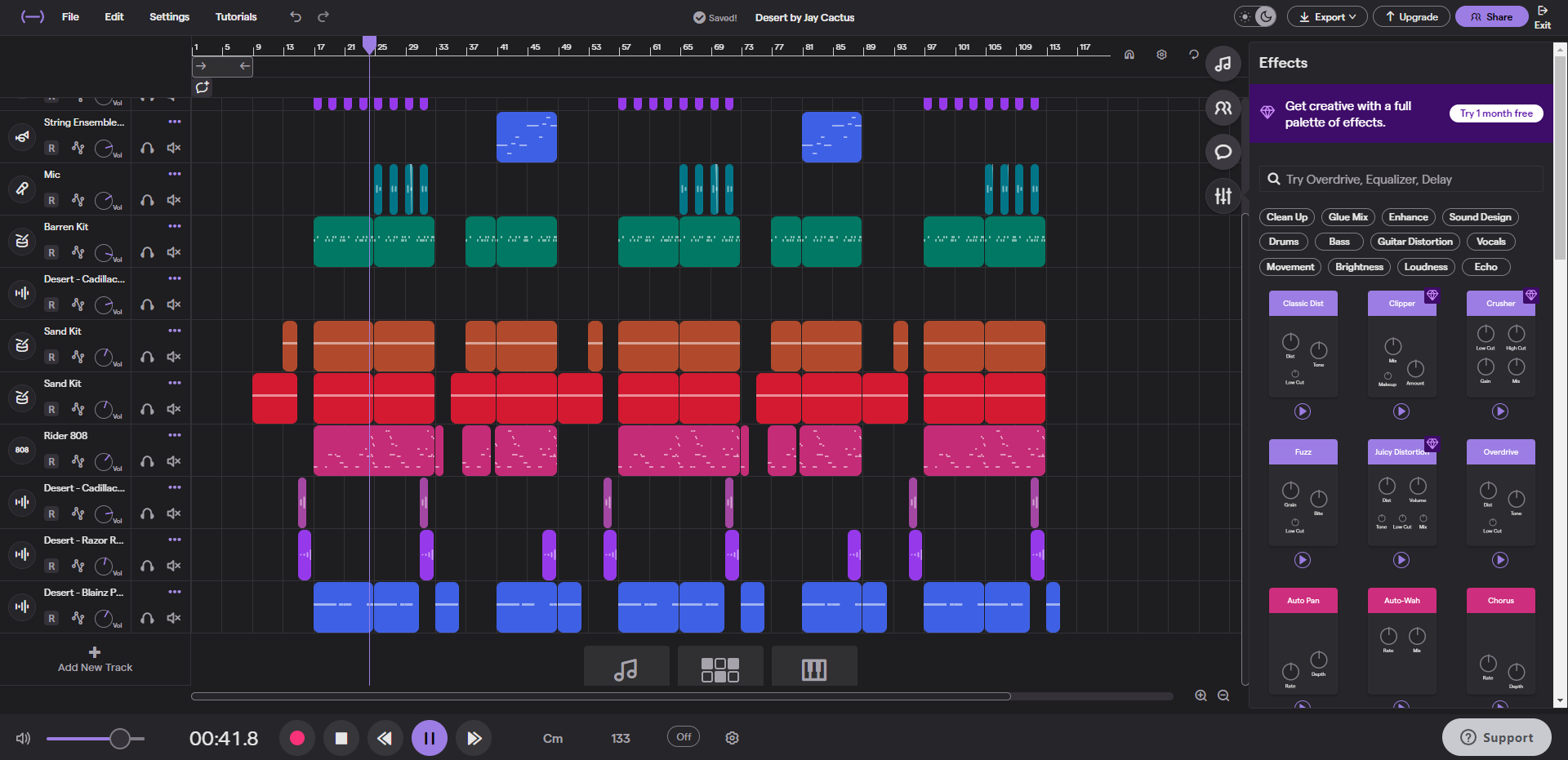
Programy takie jak BandLab, Audiotool, Soundation czy Amped Studio umożliwiają głównie darmowe korzystanie z programów. Audiotool również posiada instrukcje do korzystania z ich aplikacji, którą zamieścili na stronie GitHub. Niestety programy te nie udostępniają żadnych informacji na temat zastosowanych technologii oraz nie udostępniają publicznie kodów ani API swoich produktów.

Warto wspomnieć, że BandLab wyróżnia się opcją stworzenia projektu poprzez „[SongStarter](https://www.bandlab.com/songstarter)” (27), która generuje przez wsparcie sztucznej inteligencji gotowy szkic projektu muzycznego pod dany gatunek. W odróżnieniu od Amped Studio, które też posiada opcje generowania projektu poprzez AI aplikacja Bandlab generuje użytkownikowi warstwy zapisane w [MIDI](https://pl.wikipedia.org/wiki/MIDI) (28) a nie wygenerowane już czyste ścieżki audio instrumentów. Jest to całkiem przydatna funkcjonalność dla artystów chcących zyskać inspiracje przy tworzeniu muzyki. Bandlab oferuje również automatyczne usługi [masteringu](https://muzyczny.pl/portal/mastering-w-produkcji-muzycznej/) (29), które pozwalają użytkownikom szybko i profesjonalnie zakończyć swoje utwory bez potrzeby korzystania z zewnętrznego oprogramowania lub specjalistycznych umiejętności. Platforma zapewnia także nieograniczoną pamięć w chmurze na projekty muzyczne oraz integrację społecznościową, która pozwala na publikowanie utworów i interakcję z innymi twórcami.

Bandlab i Soundstation zawierają współpracę w czasie rzeczywistym, ale Soundstation zawiera dodatkowe funkcjonalności, takie jak czat i wideo w studiu. Soundstation oferuje również wysokiej jakości bibliotekę dźwięków. Platforma ta umożliwia również nagrywanie na żywo i oferuje zasoby edukacyjne, wspierające rozwój umiejętności producentów muzycznych.

Aplikacja Audiotool charakteryzuje się modułowym interfejsem, który pozwala na dowolne łączenie instrumentów i efektów, a także dostępem do rozległej biblioteki sampli *(dźwięk zapisany w postaci cyfrowej, przekształcany w samplerze i używany do tworzenia muzyki)* (30) i predefiniowanych dźwięków.

Amped Studio za to wyróżnia się funkcją Hybrid Technology, która pozwala na używanie zarówno tradycyjnych, jak i webowych instrumentów VST *(ang. Virtual Studio Technology)* (31) w środowisku online.



Interfejs graficzny programu Soundtrap

Program Soundtrap oferuje darmowe korzystanie z ich programu oraz możliwość podłączenia się (w opcji dla deweloperów) do ich własnego API co jest wyróżniającą się funkcjonalnością na tle innych. Inne funkcjonalności, które oferuje nie wyróżniają się na tle Bandlaba czy Soundstation. Również posiada on możliwość współpracy w czasie rzeczywistym nad projektem, obszerną bibliotekę audio, automatyczny mastering (29) oraz edukacyjne zasoby do rozwijania swoich umiejętności muzycznych.



Interfejs graficzny programu SkyTracks

Program SkyTracks poza częściowo darmową opcją korzystania z programu oferuje również informacje o technologiach użytych do stworzenia tego programu. Informacje te można znaleźć w podstronie firmy w footerze [About us](https://skytracks.io/about-us/) (25). Program ten oferuje podobne możliwości co poprzednicy ale warto wspomnieć o tym, że posiada system wersjonowania projektów, co może być przydatne jeśli chcemy zachować różne wersje projektu, co może ułatwić eksperymentowanie z różnymi aranżacjami i ustawieniami bez ryzyka utraty wcześniejszych prac. SkyTracks oferuje także zintegrowany system komentarzy i notatek bezpośrednio w interfejsie użytkownika, co umożliwia członkom zespołu zostawianie uwag i sugestii na temat konkretnych ścieżek czy fragmentów utworu. Do funkcjonalności, które mogą wyróżniać się w SkyTracks można zaliczyć opcje personalizacji interfejsu czy możliwość automatyzacji procesów miksowania (32) i masteringu (29).



Interfejs graficzny programu GridSound

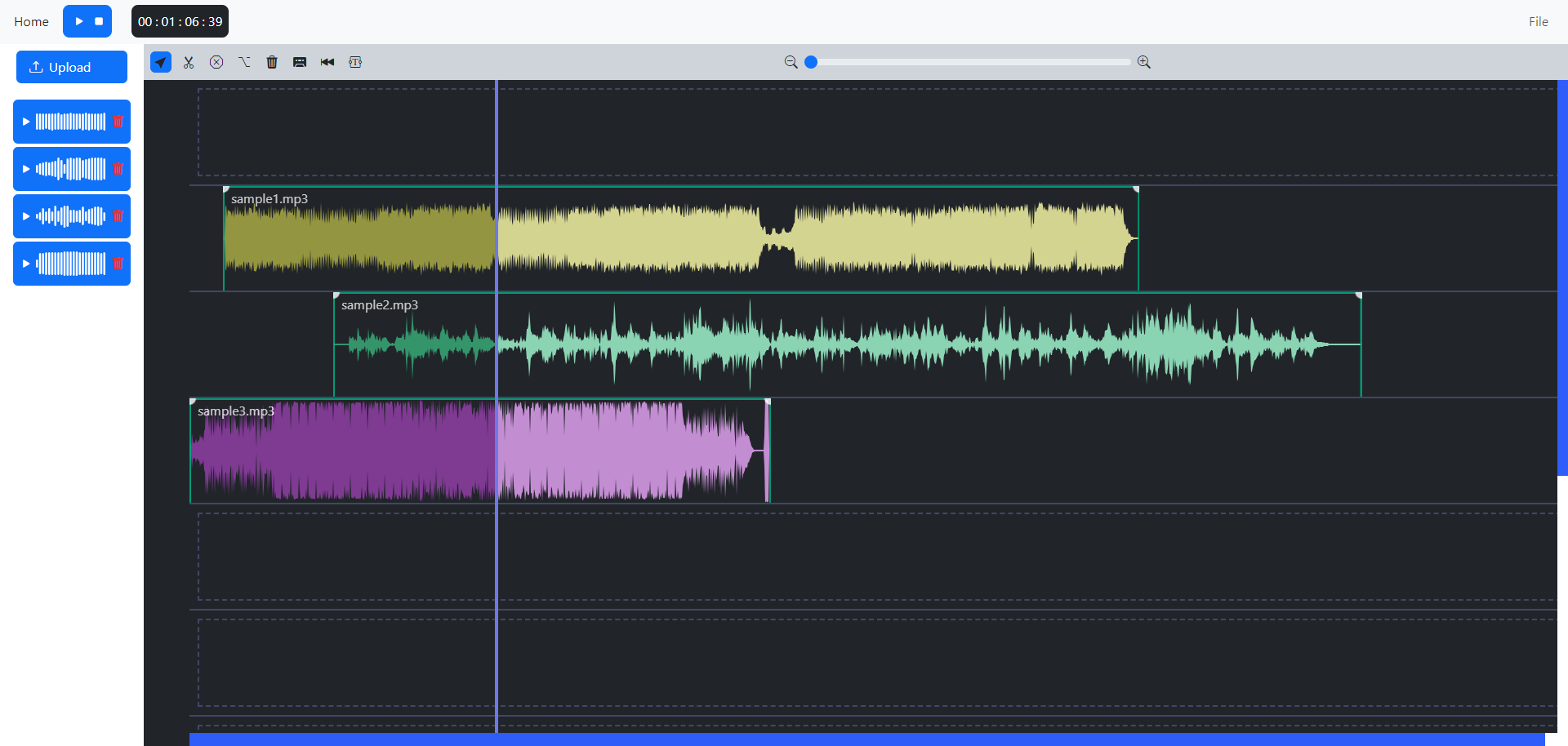
Ostatnim programem, który okazał się najlepszym pod względem dostępu do kodu jest GridSound. Jest to projekt typu open source, który został zamieszczony na GitHubie. Jest to projekt na licencji [AGPL-3.0 license](https://github.com/gridsound/daw/blob/main/LICENSE) (33) prowadzony przez 5 osobowy zespół. Aktywność, jaka przypada na wspieranie projektu, jeśli chodzi o aktualność programu jest w miarę dobra. Nowe wersje są wprowadzane, co parę miesięcy. Projekt ten wydaje się najlepszym wyborem, jeśli chodzi o rozpoznanie w tym, jakie rozwiązania są używane oraz jak wygląda architektura dla takiego typu projektów. Nie wyróżnia się niczym na tle innych programów do obróbki dźwięku, ale jego kod jest ogólnie dostępny co pomoże mi w zrozumieniu jak osiągnąć niektóre wymagane efekty w projekcie, który chcę zrealizować.

Porównując owe programy z moim projektem aplikacji można zauważyć, że jest to o wiele uboższa wersja programu DAW. Jest to spowodowane tym, że program ten w zamyśle jest przeznaczony do prostej obróbki audio. Tworzenie kompozycji instrumentalnych czy zaawansowane projekty muzyczne nie są celem tego programu. Ma on służyć prostej obróbce audio z możliwością nakładania wielu warstw ścieżek.

# Prezentacja elementów aplikacji

Aplikacja, którą stworzyłem, jest edytorem audio przeznaczonym do pracy z wieloma warstwami dźwiękowymi. Nadałem jej nazwę „Remuz.io”. Umożliwia ona tworzenie kompozycji muzycznych oraz dźwiękowych, a także oferuje możliwość przechowywania dźwięków załadowanych przez użytkownika w trakcie sesji. W dalszej części przedstawię i wyjaśnię działanie poszczególnych komponentów interfejsu użytkownika tej aplikacji.

## Interfejs Użytkownika



Interfejs graficzny mojej aplikacji Remuz.io

Aplikacja posiada interfejs użytkownika, który dzieli się na kilka głównych komponentów: navbar, sidebar oraz edytor multitrack z towarzyszącym toolbarem. Całość ma służyć wysyłaniu swoich plików audio do tej aplikacji, możliwość wybierania dźwięku, który nas aktualnie interesuje i który chcemy przetworzyć, możliwość prostej nawigacji i kontroli naszej ścieżki audio w czasie oraz eksportowaniu naszego projektu w formacie MP3 (34) o danym bitrate (35) lub w czystym formacie WAV (36).

## Navbar

Navbar to pasek, który jest paskiem przewodnim w aplikacji. Jest on umieszczony na górze ekranu i służy do podstawowej kontroli nad odtwarzaniem wszystkich ścieżek audio oraz eksportu naszego projektu. Posiada on również przycisk „Home” do powrotu strony startowej naszej aplikacji.



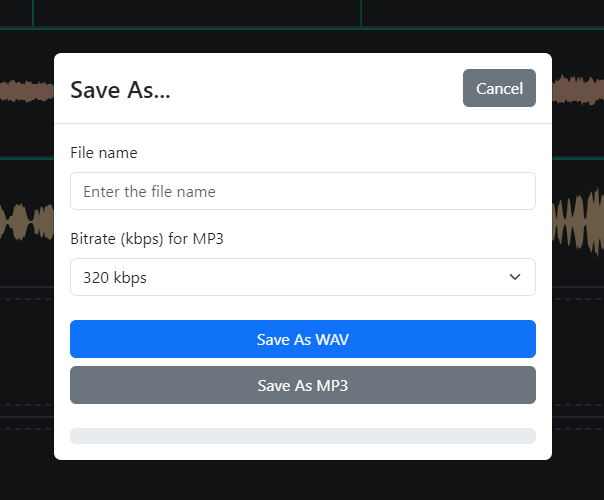
Fragmenty paska Navbar mojej aplikacji

Użytkownik znajdzie tutaj 2 przyciski do:

* Odtwarzania i zatrzymywania muzyki – przycisk ten umożliwia uruchomienie lub wstrzymanie odtwarzanych ścieżek.
* Stopowania muzyki – przycisk ten resetuje odtwarzanie ścieżki do początku i zatrzymuje nasze ścieżki.

Pasek ten posiada wyświetlacz aktualnego czasu, który wyświetla aktualny czas odtwarzania, co jest niezwykle przydatne podczas precyzyjnej edycji i synchronizacji dźwięków. Wyświetla on kolejno godziny, minuty, sekundy oraz milisekundy czasu aktualnej pozycji naszego odtwarzacza ścieżek.

Na pasku mamy również przycisk „File”, który służy do eksportowania naszego projektu jako plik MP3 bądź WAV. Po kliknięciu na przycisk wyświetla nam się takie oto okno:

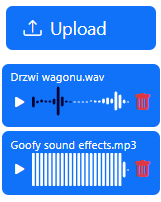


Okno zapisu projektu w formie audio w mojej aplikacji

Użytkownik ma możliwość zapisania swojego projektu jako plik WAV lub MP3 pod dowolną nazwą, którą sam określi. Jeśli nazwa nie zostanie podana, projekt zostanie automatycznie zapisany jako „Project\_RemuzIO”. Dla formatu MP3 (34) dostępna jest opcja wyboru bitrate audio (35), co umożliwia kontrolę nad poziomem kompresji tego formatu. Zapis w formacie WAV (36) jest szybszy niż w formacie MP3, ponieważ nie wymaga kompresji połączonej ścieżki. Postęp renderowania utworu można śledzić na pasku postępu znajdującym się na dole okna. Po zakończeniu renderowania, plik może zostać zapisany w folderze pobranych plików lub w zależności od przeglądarki, w wybranym przez użytkownika folderze.

## Sidebar

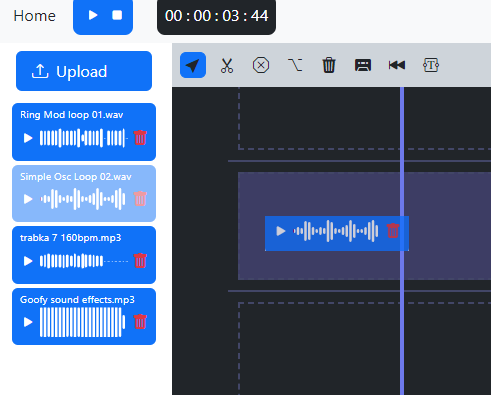
Sidebar to boczny pasek, pełniący funkcję biblioteki dźwięków przeznaczonych do przechowywania w celu ich późniejszego wyboru jako ścieżek do edycji. Pasek ten znajduje się po lewej stronie interfejsu i umożliwia zarządzanie utworami, które mają zostać włączone do projektu.



Załadowane dźwięki audio na pasku Sidebar

Przycisk „Upload” znajdujący się na górze tego komponentu umożliwia importowanie plików audio z lokalnego dysku i ich integrację z aktualnym projektem. Importowanie obsługuje przesyłanie kilku ścieżek naraz, co ułatwia pracę i skraca czas, szczególnie gdy użytkownik chce załadować wiele dźwięków z jednego folderu.

Po załadowaniu plików użytkownik ma możliwość odtworzenia każdej ścieżki, aby sprawdzić, jak brzmi w danym fragmencie, klikając na zwizualizowaną falę dźwięku. Można zatrzymać i wznowić audio za pomocą przycisku po lewej stronie z ikoną białego trójkąta odtwarzacza bądź pauzy, w zależności od tego, czy audio jest odtwarzane, czy nie. Każdy plik audio w swoim okienku posiada nazwę, co ułatwia szybką identyfikację. Ponadto, każdy odtwarzacz ma również opcję usunięcia audio z biblioteki dźwięków, wyświetlaną w formie czerwonej ikonki kosza.



Przedstawienie możliwości przeciągania dźwięku z biblioteki dźwięków na wybraną ścieżkę edytora audio Multitrack

Jednak najważniejszą funkcją jest możliwość przeciągnięcia wybranego dźwięku na wybraną warstwę audio w komponencie Multitrack. Użytkownik musi przytrzymać lewy przycisk myszy na bloku audio i przeciągnąć go na wybraną warstwę, która podświetli się na niebiesko, co umożliwi załadowanie dźwięku na tej warstwie. Możliwe jest również załadowanie dźwięku na warstwę, która już zawiera plik audio, w takim przypadku mechanizm działania jest taki sam, z tą różnicą, że wizualnie nie zostanie zaznaczony obszar na warstwie przeznaczony do upuszczenia audio. Dzięki tej funkcjonalności będzie można skutecznie manipulować wybranymi ścieżkami w komponencie Multitrack, korzystając z narzędzi, które szczegółowo opiszę w dalszej części tekstu.

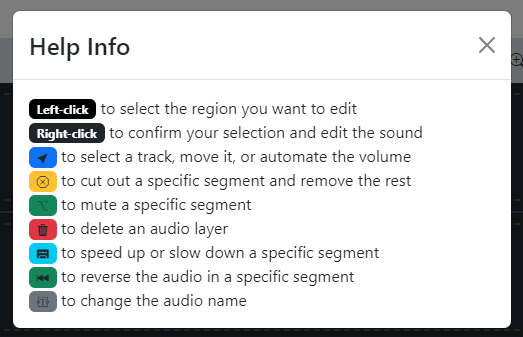
## Multitrack

Multitrack, czyli najważniejsza część naszej aplikacji to komponent, który pozwala nam na edytowanie naszych ścieżek oraz łatwe zarządzanie nimi w czasie. Komponent ten zawiera swój pasek narzędzi zwany jako Toolbar.



Pasek narzędzi (Toolbar) dla komponentu Multitrack

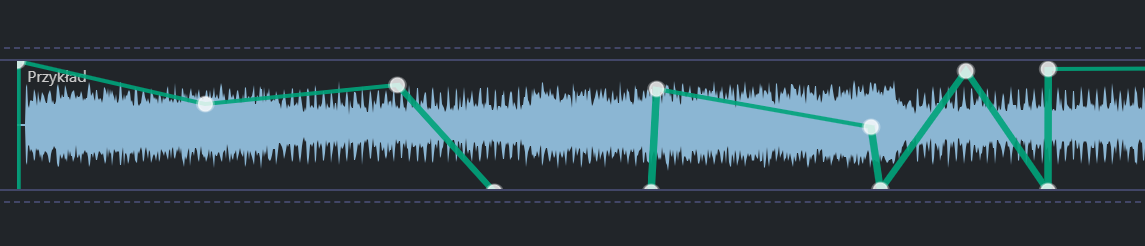
Na pasku po lewej stronie znajdują się narzędzia, które pozwalają wybrać interesujący nas tryb edycji. Po środku umieszczony jest suwak, który służy do oddalania lub przybliżania widoku, w zależności od tego, jak szczegółowo chcemy przejrzeć nasze warstwy ścieżek audio. Po prawej stronie znajduje się ikona znaku zapytania, która po kliknięciu wyświetla okno z wyjaśnieniami dotyczącymi używania narzędzi oraz ich przeznaczenia.



Okno z informacjami pomocniczymi do narzędzi w komponencie Multitrack mojej aplikacji

Przejdźmy teraz do tego jak dane narzędzia wyglądają w mojej aplikacji.

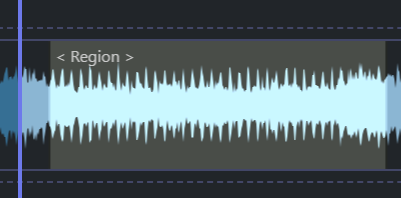
Pierwszym narzędziem będzie „Select Tool”. Po jego wybraniu uzyskamy możliwość przeciągania ścieżek audio w czasie prawym przyciskiem myszy oraz będziemy mogli zautomatyzować głośność danej ścieżki, tworząc punkty automatyzacji za pomocą kliknięcia lewym przyciskiem myszy. Automatyzacja głośności polega na dynamicznym zmienianiu poziomu głośności ścieżki za pomocą krzywej, która, poprzez umieszczone punkty, określa, jak głośno ma grać utwór w danym momencie. Termin ogólnej automatyzacji miksowania w muzyce jest bardziej wyjaśniony w niniejszym artykule (37). Poniższe zdjęcie pokazuje, jak wygląda przykładowa automatyzacja głośności ścieżki za pomocą tego narzędzia.



Przedstawienie działania funkcji automatyzacji głośności ścieżki audio w mojej aplikacji

Jak widzimy, im wyżej znajduje się dany punkt automatyzacji, tym głośniejszy jest dźwięk w danym momencie, a im niżej, tym cichszy.

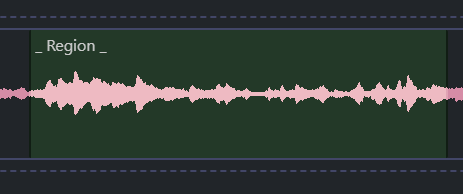
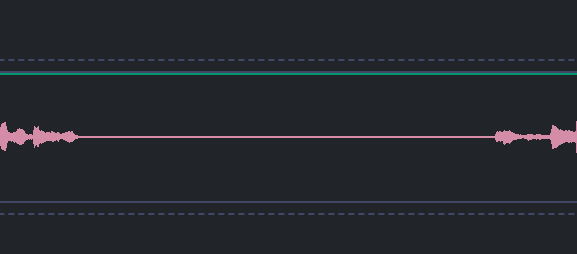
Drugim narzędziem jest „Cut Fragment Tool”. Jego użycie pozwala na wycięcie wybranego fragmentu ścieżki, a po zaakceptowaniu zmian, na warstwie pozostaje tylko zaznaczony wcześniej fragment audio. Po kliknięciu lewym przyciskiem myszy na warstwę, pole zaznaczenia pojawia się na początku ścieżki i umożliwia pełną kontrolę nad procesem edycji. Nasza warstwa w tym momencie blokuje przesuwanie ścieżki. Możemy wtedy przesuwać to pole oraz rozciągać je, przytrzymując jego początek lub koniec. Aby zatwierdzić zmiany i dokonać edycji, wystarczy kliknąć prawym przyciskiem myszy. Następnie ścieżka jest przetwarzana, co skutkuje wygenerowaniem zmodyfikowanej wersji.

Ścieżka w trakcie zaznaczania fragmentu audio i po przetworzeniu

Trzecim narzędziem na naszym pasku jest „Delete Fragment Tool”. Obsługa tego narzędzia jest stosunkowo prosta i intuicyjna. Aby usunąć niepotrzebny fragment ścieżki, należy go najpierw zaznaczyć, korzystając z tego samego sposobu, co w przypadku poprzedniego narzędzia. Po dokonaniu wyboru odpowiedniego segmentu, wystarczy kliknąć prawym przyciskiem myszy. W ten sposób wybrany fragment zostanie usunięty z naszej ścieżki.

Czwartym narzędziem jakie posiada moja aplikacja jest „Mute Fragment Volume Tool”. Narzędzie to umożliwia wyciszanie wybranych fragmentów audio na wybranej warstwie. Użytkownik zaznacza fragment, który ma zostać wyciszony, a następnie klikając prawy przycisk myszy zatwierdza zmiany, co skutkuje wizualną modyfikacją amplitudy dźwięku na ścieżce, pokazując, że dany segment został przetworzony i wyciszony.

Wybrany segment do wyciszenia i przetworzony po zatwierdzeniu

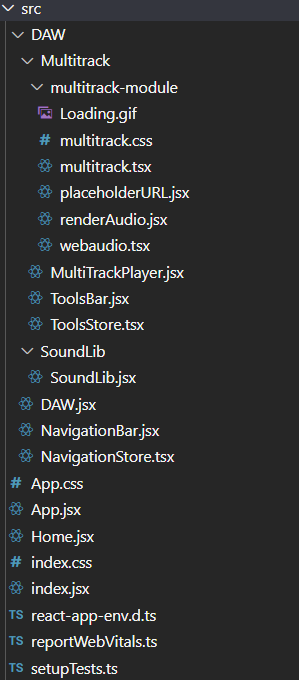
Piątym narzędziem jakie posiada moja aplikacja jest narzędzie „Delete Track Tool”. Służy ono do usuwania danej ścieżki audio z danej warstwy. Wystarczy kliknąć na daną ścieżkę i pojawi nam się okno z zapytaniem czy chcemy usunąć daną ścieżkę. Po zatwierdzeniu wybrana ścieżka jest usuwana. Efekt cichej kreski jest niestety spowodowany ograniczeniami implementacji samego pluginu Multitrack. Na szczęście poza wizualnym aspektem efekt ten nie wpływa na inne dźwięki i można traktować go jako puste pole, na które możemy również załadować inne dźwięki.

Szóste narzędzie do edycji ścieżek, „Change Speed Fragment Track Tool”, umożliwia przyspieszanie lub zwalnianie tempa wybranych fragmentów audio na danej warstwie. Zaznaczanie fragmentu funkcjonuje podobnie jak w przypadku poprzednich narzędzi, lecz po kliknięciu prawym przyciskiem myszy pojawia się okno z suwakiem, umożliwiające wybór prędkości fragmentu. Prędkość można regulować od 0,5x do 2x. Po ustawieniu żądanej prędkości, użytkownik może zapisać zmiany za pomocą przycisku „Save changes” lub anulować je, klikając „Close”. Po zatwierdzeniu, fragment zostanie odpowiednio zwolniony lub przyspieszony. Aby na przykład skrócić długość audio czterokrotnie, należy dwukrotnie zastosować zwolnienie z prędkością 0,5x na ten sam fragment.

Siódmym i ostatnim narzędziem do manipulacji dźwiękiem jest „Reverse Fragment Track Tool”. Narzędzie to umożliwia uzyskanie efektu odwróconego wstecz audio dla wybranego fragmentu. Obsługa tego narzędzia jest identyczna jak w przypadku poprzednich, gdzie użytkownik zaznaczał fragment, a następnie zatwierdzał zmiany. Po zatwierdzeniu otrzymujemy przetworzone audio z odwróconą ścieżką dźwiękową w wybranym fragmencie.

Ostatnie, ósme narzędzie na naszym pasku to „Change Text Track Tool”. Narzędzie to służy do zmiany nazwy ścieżki na danej warstwie, co ułatwia identyfikację ścieżek, szczególnie gdy na kilku warstwach znajdują się podobne ze względu na kształt fali dźwięku. Obsługa narzędzia jest prosta, wystarczy kliknąć lewym przyciskiem myszy na wybraną ścieżkę, co spowoduje wyświetlenie okna do zmiany nazwy. Po zatwierdzeniu wprowadzonych zmian, tekst nazwy ścieżki zostanie zaktualizowany do nowo wpisanej wartości.

# Struktura projektu



Przedstawienie struktury projektu w mojej aplikacji webowej

Aplikacja została stworzona w oparciu o szablonowy projekt React z wykorzystaniem TypeScriptu. Wynikało to z faktu, że zaimportowany przeze mnie plugin, Multitrack, został napisany w tym języku i wymagał uruchomienia go w projekcie tego typu. Opcja ta pozwoliła mi zaoszczędzić czas, który musiałbym poświęcić na przekształcenie całego pluginu na JavaScript. Wobec tego postanowiłem stworzyć nowy projekt z nastawieniem na TypeScript. Pomijając pliki szablonowe, skupmy się na plikach i folderach, które zostały przeze mnie utworzone. Plik App.jsx zawiera trasy z react-router-dom (38), które umożliwiają nawigację po aplikacji. Nawigacja po aplikacji jest prosta, strona startowa o ścieżce „/” to komponent Home, a strona, do której przechodzimy przez komponent Home, to komponent DAW z ścieżką „/app”.

Komponent Home zawiera przycisk „Start”, który przenosi nas do aplikacji i ładuje komponent DAW. Lokalizacja pliku komponentu DAW znajduje się w folderze o tej samej nazwie, który zawiera także pliki takie jak NavigationBar.jsx i NavigationStore.jsx oraz foldery jak SoundLib czy Multitrack. Komponent DAW jest odpowiedzialny za ogólne działanie naszej aplikacji, ponieważ zawiera wszystkie inne komponenty, które odpowiadają za poszczególne funkcjonalności aplikacji. Plik DAW.jsx korzysta z takich komponentów jak NavigationBar, SoundLib, ToolsBar czy MultiTrackPlayer.

Komponent NavigationBar zarządza paskiem nawigacyjnym (*[Navbar](#_Navbar)*), którego szczegółową obsługę omówiłem w oddzielnym podrozdziale. Dodatkowo, dla tego komponentu stworzony został plik NavigationStore.jsx, który za pomocą biblioteki Zustand zajmuje się przekazywaniem i przechowywaniem zmiennych oraz funkcji między komponentami. Szczegółowe działanie tej biblioteki w kontekście mojej aplikacji wyjaśnię w rozdziale poświęconym implementacji kodu.

Komponent SoundLib, umieszczony w folderze o tej samej nazwie, odpowiada za pasek boczny ([*Sidebar*](#_Sidebar)), który służy jako biblioteka dźwięków. W komponencie SoundLib przechowywane są funkcje i elementy niezbędne do zarządzania biblioteką dźwięków oraz przenoszenia dźwięków do komponentu MultiTrackPlayer, który jest kolejnym ważnym elementem naszej aplikacji.

Plik MultiTrackPlayer.jsx znajduje się w folderze Multitrack razem z ToolsBar.jsx i ToolsStore.jsx. Plik ToolsBar.jsx odpowiada za działanie komponentu paska narzędzi (Toolbar), natomiast ToolsStore.jsx służy do przekazywania i przechowywania zmiennych oraz funkcji między komponentem ToolsBar a komponentem MultiTrackPlayer. Komunikacja ta odbywa się za pomocą biblioteki Zustand. MultiTrackPlayer.jsx jest odpowiedzialny za inicjalizację instancji [Multitrack](#_Multitrack) oraz za koordynację wszystkich funkcjonalności Multitracka z innymi komponentami, w tym z komponentami w folderze o nazwie multitrack-module, co zapewnia ich współpracę.

Folder multitrack-module, który znajduje się wewnątrz folderu Multitrack, zawiera pliki odpowiedzialne za działanie komponentu MultiTrackPlayer. Komponenty te odpowiadają w szczególności za przetwarzanie wszystkich warstw ścieżek dźwiękowych oraz za ogólne funkcjonowanie pluginu Multitrack. Dwa kluczowe pliki w tym folderze to multitrack.tsx i webaudio.tsx, które wstępnie były napisane w języku TypeScript przez autora pluginu Multitrack a później zostały zmodyfikowane przeze mnie w celach rozbudowy funkcjonalności tego pluginu. Plik multitrack.tsx zawiera klasę Multitrack, która obejmuje metody, zmienne i funkcje odpowiedzialne za działanie całego komponentu [Multitrack](#_Multitrack) z poziomu wewnętrznego. Jest on powiązany z plikiem webaudio.tsx, ponieważ plik ten zawiera klasę WebAudioPlayer, odpowiedzialną za operacje na dźwięku, w tym jego modyfikacje.

Oprócz tych dwóch, folder zawiera również pliki takie jak renderAudio.jsx, placeholderURL.jsx, multitrack.css i Loading.gif. Plik multitrack.css oraz obraz Loading.gif odpowiadają za styl komponentu Multitrack, a dokładniej za styl paska przewijania i efekt ładowania ścieżek na poszczególnych warstwach. Plik renderAudio.jsx odpowiada za funkcje renderowania projektu w aplikacji, natomiast placeholderURL.jsx służy jako pomocniczy plik do wczytywania przygotowanego około 5 minutowego dźwięku mp3 zapisanym jako base64 do ładowania pustych warstw w komponencie Multitrack.

# Implementacja najważniejszych funkcjonalności

Niniejszy rozdział jest poświęcony szczegółowemu opisowi procesu implementacji kluczowych funkcjonalności mojej aplikacji. Implementacja tych funkcjonalności jest sercem naszego systemu, gdyż to one decydują o unikalności i użyteczności finalnego produktu. Celem tego rozdziału jest nie tylko przedstawienie technicznych aspektów implementacji, ale również opisanie podejścia do problemów, które napotkałem podczas wdrażania tych funkcjonalności.

Zacznę od przedstawienia funkcjonalności, od których rozpocząłem pracę nad aplikacją, a na końcu opiszę implementację tych funkcjonalności, które realizowałem jako ostatnie.

## Biblioteka dźwięków

Pierwszą funkcjonalnością, którą omówię, jest biblioteka dźwięków, znana również jako komponent SoundLib. Jest on zaimplementowany w pliku SoundLib.jsx mojego projektu. Plik ten zawiera dwie funkcje zwracające komponenty. Pierwszą z nich jest funkcja SoundLib bez parametrów, a drugą funkcja SoundItem, która przyjmuje parametry takie jak *sound, currentFileID, isPlaying, togglePlayback, removeSound*. Parametry te zostaną dokładniej wyjaśnione przy omawianiu funkcji SoundItem.

Zacznijmy od przyjrzenia się funkcji SoundLib, która służy do wyświetlania wszystkich bloków audio przeznaczonych do odtwarzania w bibliotece. Komponent ten korzysta z kilku kluczowych zmiennych stanu zarządzanych przez hooki *useState* z biblioteki React (10), które są niezbędne do zarządzania odtwarzaniem dźwięku i jego właściwościami. Poniżej przedstawiono listing kodu, który prezentuje inicjalizację tych zmiennych:

const [currentFileID, setCurrentFileID] = useState(null);

const [isPlaying, setIsPlaying] = useState(false);

const [sounds, setSounds] = useState([]);

Pierwszą zmienną jest stan aktualnego ID pliku, który użytkownik wybrał do odtworzenia w bibliotece. Drugą zmienną jest stan informujący czy dana ścieżka jest aktualnie odtwarzana. Trzecią zmienną stanu jest lista dźwięków, które obecnie znajdują się w bibliotece dźwięków.

Zobaczmy teraz, jak działa hook *useEffect*, który jest wywoływany przy zmianie zmiennej *sounds*. Za jego pomocą tworzę wizualizacje fal dźwiękowych za pomocą biblioteki wavesurfer.js. Dla każdego dźwięku nadawany jest odpowiedni wygląd fali dźwięku.

useEffect(() => {

sounds.forEach((sound) => {

if (!sound.wavesurfer) {

const wavesurfer = WaveSurfer.create({

container: `#waveform-container-${sound.id}`,

waveColor: 'white',

progressColor: '#000235',

height: 30,

barWidth: 2.5,

barRadius: 3,

barGap: 0,

normalize: false,

barMinHeight: 5,

cursorWidth: 0,

interact: true,

});

wavesurfer.load(sound.audioSrc).then(() => { console.log("Sound " + sound.name + " loaded" ) })

.catch(() => { console.log("Sound" + sound.name + "not loaded" )});

wavesurfer.on('finish', () => {

wavesurfer.stop(); // zatrzymuje i resetuje na początek

setCurrentFileID(null); // resetuje obecnie odtwarzany indeks

setIsPlaying(false); // ustawia stan odtwarzania na false

});

setSounds(prevSounds => prevSounds.map(s =>

s.id === sound.id ? { ...s, wavesurfer: wavesurfer } : s

));

}

});

// Funkcja czyszcząca

return () => {

sounds.forEach((sound) => {

sound.wavesurfer && sound.wavesurfer.destroy();

});

};

}, [sounds]);

Do przypisania zmiennej obiektu *wavesurfer* używam funkcji *WaveSurfer.create(...)*,która zawiera parametry określające wygląd mojej wizualizacji. Warto zaznaczyć, że parametr *container* musi wskazywać identyfikator diva, który ma reprezentować odpowiedni dźwięk. Jest to wymóg niezbędny do poprawnego działania wizualizacji. W tym celu przekazujemy zmienną sound.id, którą dopisujemy do identyfikatora diva o id „waveform-container-”. Spośród ważniejszych parametrów w *WaveSurfer.create* warto wspomnieć również o interact, który ustawiamy na true, aby umożliwić przewijanie i ogólną interakcję z naszym odtwarzaczem dźwięku poza klikaniem przycisku startu i pauzy. Po przypisaniu, wywoływana jest funkcja *wavesurfer.load*, która ładuje dźwięk z adresu *sound.audioSrc* do obiektu *wavesurfer*. Po załadowaniu wyświetlany jest komunikat konsolowy informujący o załadowaniu dźwięku. Dodatkowo zastosowany jest listener ‘finish’ dla obiektu *wavesurfer*, który po zakończeniu odtwarzania zatrzymuje dźwięk, resetuje *currentFileID* oraz ustawia *isPlaying* na *false*. Co najważniejsze, w wywołaniu *useEffect* ustawiany jest nowy dźwięk za pomocą funkcji setSounds, gdzie obiekt wavesurfer jest przypisywany do odpowiedniego dźwięku. Używana jest również funkcja czyszcząca, która niszczy obiekt wavesurfer, jeśli ten jest odmontowywany. Jest to niezbędne do zapobiegania wyciekom pamięci.

Po omówieniu działania hooka *useEffect* w tej funkcji, przejdźmy do omówienia implementacji funkcji dodawania i usuwania dźwięków. Zacznijmy od przyjrzenia się elementowi input, który aktywowany jest poprzez kliknięcie przycisku „Upload”, wykorzystując metodę *htmlFor*:

<input

type="file"

accept="audio/mpeg, audio/wav"

onChange={(e) => {

const files = Array.from(e.target.files);

CheckAndAddSound(files)

}}

style={{ display: 'none' }}

id="uploadInput"

multiple

/>

Ustawiłem element ten tak, aby akceptował pliki audio w formatach MPEG i WAV, co określiłem za pomocą atrybutu *accept="audio/mpeg, audio/wav".* Dodatkowo, zaimplementowałem funkcję, która jest wywoływana każdorazowo, gdy użytkownik wybierze pliki. Ta funkcja pobiera wybrane pliki, konwertuje je na tablicę i przekazuje do funkcji *CheckAndAddSound*, która zarządza dodawaniem dźwięków do aplikacji. Aby zachować estetykę interfejsu, ukryłem element input poprzez ustawienie jego stylu na display: '*none*'. Dzięki temu, użytkownik może aktywować wybór plików poprzez kliknięcie innego elementu interfejsu, który z kolei wywołuje kliknięcie na ukryty element input. Na potrzeby identyfikacji dla komponentu z przyciskiem „Upload” nadałem temu elementowi identyfikator *uploadInput*, a atrybut *multiple* pozwala użytkownikowi wybrać więcej niż jeden plik jednocześnie.

Przejdźmy teraz do funkcji sprawdzającej nasze ładowane pliki czyli do *CheckAndAddSound.* W funkcji tej, która przyjmuje tablicę plików jako argument, głównym celem jest weryfikacja dopuszczalności typów plików i ich dodawanie do stanu aplikacji, zarządzającego dźwiękami. Na początku funkcji zdefiniowana jest tablica *allowedTypes*, zawierająca typy plików, które są akceptowane — w tym przypadku są to formaty audio MPEG i WAV. Następnie, używając metody *filter* na przekazanej tablicy plików, funkcja eliminuje te pliki, które nie odpowiadają żadnemu z dopuszczalnych typów, wykorzystując metodę *includes* do sprawdzenia, czy dany typ pliku znajduje się w tablicy *allowedTypes*. Ostatecznie, przefiltrowane pliki, czyli te spełniające kryteria dopuszczalnych typów, są dodawane do stanu dźwięków za pomocą funkcji *addSounds*.

Przejdźmy do funkcji *addSounds.* Funkcja ta jest funkcją asynchroniczną, która operuje na tablicy plików audio, zwaną *newFiles*. Celem tej funkcji jest przetworzenie przekazanych plików i dodanie ich do stanu zarządzającego dźwiękami w aplikacji. W pierwszym kroku funkcja ta mapuje tablicę *newFiles* na tablicę obietnic, które są rozwiązane, gdy tylko metadane każdego z plików audio zostaną załadowane. Dla każdego pliku audio tworzony jest nowy obiekt Audio, a metody *onloadedmetadata* i *onerror* są ustawiane jako funkcje obsługujące odpowiednio załadowanie metadanych i błędy. Jeśli metadane zostaną pomyślnie załadowane, obietnica jest rozwiązana i zwraca obiekt reprezentujący dźwięk. W przypadku wystąpienia błędu, obietnica jest odrzucana, a komunikat o błędzie jest zwracany. Funkcja *Promise.all* jest używana do synchronicznego oczekiwania na załadowanie wszystkich plików audio. Po zakończeniu procesu ładowania, *addSounds* aktualizuje stan *sounds*, dodając nowe dźwięki do już istniejących. W tym celu używana jest funkcja aktualizatora stanu, która przyjmuje obecny stan jako argument i zwraca nowy stan, będący połączeniem poprzednich i nowo dodanych dźwięków. Ta operacja jest realizowana za pomocą operatora rozproszenia, co pozwala na proste i efektywne połączenie dwóch tablic dźwięków w jedną zaktualizowaną listę.

Wiedząc już jak działa funkcja dodawania dźwięków, przejdźmy do funkcji usuwania danego dźwięku po identyfikatorze. Funkcja ta jest wywoływana po kliknięciu ikonki kosza na danym odtwarzaczu ścieżki audio. Funkcja o nazwie *removeSound* służy do usuwania dźwięku o określonym ID z listy dźwięków. Przyjmuje ona jako argument *idToRemove*, identyfikator dźwięku, który ma zostać usunięty. Na początku funkcja wyszukuje w tablicy *sounds* czyli dźwięk odpowiadający danemu ID. Jeżeli taki dźwięk zostanie znaleziony i jest skojarzony z obiektem *wavesurfer*, następuje zniszczenie tego obiektu, co jest kluczowe dla zapobiegania wyciekom pamięci. Na koniec, funkcja aktualizuje stan *sounds*, stosując metodę *filter*, aby usunąć wybrany dźwięk z tablicy, tworząc nową listę dźwięków, które nie zawierają już tego usuniętego ID.

Funkcja *togglePlayback* jest funkcją, która służy jako funkcja odpowiedzialna za kontrolowanie startowania i pauzowania wybranego odtwarzacza ścieżki audio. Przyjmuje *id* jako argument. Ta funkcja ma na celu kontrolowanie odtwarzania dźwięku o danym ID. Najpierw znajduje dźwięk o danym ID w tablicy *sounds*. Jeśli dźwięk nie jest znaleziony lub nie ma skojarzonego obiektu *wavesurfer*, wyświetla komunikat w konsoli i kończy działanie. Jeśli dźwięk jest aktualnie odtwarzany, zatrzymuje odtwarzanie i ustawia *isPlaying* na *false*. W przeciwnym razie zatrzymuje odtwarzanie wszystkich innych dźwięków, a następnie rozpoczyna odtwarzanie wybranego dźwięku. Aktualizuje również *currentFileID* na ID wybranego dźwięku i ustawia *isPlaying* na *true*.

Wszystkie te zmienne z komponentu SoundLib przekazuje do mapowanego komponentu SoundItem, który jako key przyjmuje *sound.id* oraz jako argumenty dostaje funkcje *removeSound* i *togglePlayback.* Stworzyłem komponent SoundItem jako oddzielny komponent, aby w wygodniejszy sposób przekazywać zmienne oraz aby kod wydawał się bardziej schludny. W komponencie tym używam obiektu showSound, który definiuje dwa stany mojej animacji *hidden* i *visible*. Animacja ta jest tworzona przez bibliotekę framer-motion. Każdy stan tej animacji ma określoną wartość *opacity*. Stan *visible* zawiera również obiekt *transition*, który definiuje czas trwania i łagodzenie animacji. Bardziej szczegółowy opis możliwości tej biblioteki można znaleźć w jej dokumentacji (16).

Do przeciągania naszych dźwięków użyłem hooka *useDrag* z biblioteki *react-dnd* (39)*.* Biblioteka ta służy do tworzenia elementów, które możemy łapać, przeciągać i opuszczać. Hook *useDrag* jest używany do umożliwienia przeciągania elementu. Typ elementu jest ustawiony na *sound*, a *item* jest obiektem zawierającym informacje o dźwięku. Typ elementu jest ustawiony akurat na *sound* aby zapobiec sytuacjom, gdy drugi hook *useDrop* w komponencie od MultiTrackPlayer miałby przyjmować jakieś inne elementy niż tylko element *sound*. Funkcja *collect* jest używana do zbierania informacji o stanie przeciągania, takich jak *isDragging*, który jest prawdziwy, gdy element jest przeciągany.

Znając już działanie naszej biblioteki dźwięków, przejdę teraz do omówienia sposobu, w jaki zmodyfikowałem funkcjonalności pluginu Multitrack, który został napisany w podstawowej, nierozwiniętej wersji przez wcześniej opisany zespół (7).

## Modyfikacja działania pluginu Multitrack

Zacznijmy od sposobu, w jaki podszedłem do tego problemu. Dysponując pluginem Multitrack, który został napisany całkowicie w TypeScript i tworzył elementy w czystym JavaScript, musiałem ustalić, która część kodu będzie odpowiedzialna za oddziaływanie na same warstwy audio, a która za komunikację między komponentami, głównie z komponentem ToolsBar. Postanowiłem, że część odpowiedzialna za działanie i ogólne przetwarzanie dźwięku w naszym komponencie MultiTrack znajdzie się w plikach *multitrack.tsx* oraz *webaudio.tsx*. Plik *webaudio.tsx* odpowiada za operacje na buforach audio i obsługę dźwięku przy pomocy biblioteki Web-Audio-API (12). Po zapoznaniu się z działaniem funkcji w obu tych plikach, postanowiłem dodać pierwsze nowe funkcje, które były mi potrzebne do rozszerzenia możliwości tego pluginu. Najpierw dodałem funkcję *stop*, która działa bardzo podobnie do już zaimplementowanej funkcji *pause*, ale dodatkowo ustawia zmienną *currentTime* na 0 oraz pozycję kursora aktualnie odtwarzanego momentu na 0, używając funkcji *updatePosition*.

Kolejną zmianą, którą wprowadziłem, była wymiana obiektu *placeholderURL*. Z uwagi na to, że długość ścieżki zapisanej w formacie base64 była dla mnie zbyt krótka, zdecydowałem się stworzyć za pomocą prostego skryptu w Pythonie plik audio w tym samym formacie, który nie wydaje dźwięku i jest 5-minutową ciszą. Importowanie tego pliku jako zwykłego tekstu czy odczyt bez rozszerzenia było problematyczne. Dlatego, upraszczając przekazanie wartości z tego pliku, wkleiłem jego zawartość do zmiennej typu string w oddzielnym pliku placeholderURL.jsx, który zwraca ten ciąg znaków. To rozwiązanie wydało mi się najlepsze i najprostsze do estetycznego przekazania tego zapisu do obiektu *PLACEHOLDER\_TRACK*, biorąc pod uwagę, że zawiera on bardzo dużo znaków, a dynamiczne pobieranie takiego zapisu nie miało sensu, gdyż jest to obiekt statyczny, niezmieniający swojej zawartości.

Następnym krokiem, który chciałem podjąć, było rozszerzenie funkcji dodawania i usuwania ścieżek. Chciałem, aby można było dodawać nowe ścieżki oraz usuwać je, tak aby użytkownik nie martwił się o brak dostępnych warstw w większych projektach. Niestety sposób, w jaki renderują się dane warstwy oraz ich manipulacja wymagałby gruntownej przebudowy całej implementacji działania pluginu Multitrack. Problemem okazało się stałe zainicjalizowanie liczby warstw w funkcji konstruującej klasę *Multitrack* zawartej w multitrack.tsx oraz brak możliwości stosunkowo dynamicznego odświeżania instancji komponentu Multitrack, ponieważ działa ona częściowo w środowisku React, a częściowo w czystym JavaScript. Wiedząc, jakie przeszkody mogłem napotkać, postanowiłem przyjąć inne podejście. Z racji, że jest to program do prostej obróbki audio, zdecydowałem się zainicjalizować 10 warstw, które niestety będą stanowić ograniczenie dla większych projektów. Z powodu ograniczeń w implementacji pluginu musiałem zrezygnować z możliwości rozszerzenia funkcjonalności mojego Multitracka. Dlatego dodawanie nowych ścieżek audio odbywa się bez dodawania nowych warstw, a sama implementacja tego, jak będzie dodawana nowa ścieżka, polega na upuszczeniu ścieżki audio na daną warstwę. W tym celu zaimplementowałem w pliku MultiTrackPlayer.jsx funkcję, która korzysta również z biblioteki *react-dnd* i nasłuchuje za pomocą hooka *useDrop*, który plik został upuszczony i na jakim indeksie warstwy. Po usprawnieniu funkcjonalności dodawania nowych ścieżek audio na warstwy postanowiłem dodać funkcję, która będzie usuwać ścieżkę audio z danej warstwy. Funkcja *removeTrack* działa niemal identycznie jak funkcja *addTrack*, z tą różnicą, że podczas usuwania ścieżki w funkcji inicjalizacji dźwięków *initAudio*, podajemy obiekt *PLACEHOLDER\_TRACK*, który załaduje 5-minutową ścieżkę ciszy. Dodatkowo, funkcja ta przyjmuje jako argument jedynie id naszej warstwy ścieżki, co wymaga stworzenia prostego obiektu typu *TrackOptions*, który zastąpi bieżącą wizualizację ścieżki audio. W tym celu tworzę obiekt *trackAdd*, który zawiera *id* oraz *position*. Id jest wybierane z tablicy aktualnych ścieżek, wskazując na tę z konkretnie usuwanym indeksem, a *position* ustalam na 0. Dzięki temu będziemy mogli przeprowadzić wymianę ścieżki audio na 5-minutową ciszę. Jedynym wizualnym minusem tego rozwiązania jest to, że wizualizacja fali dźwiękowej renderuje prostą kreskę dźwięku, co jest spowodowane koniecznością posiadania jakiegokolwiek dźwięku w obiekcie wavesurfer, który renderuje fale dźwiękowe po zainicjalizowaniu.

Kolejnym etapem, była modyfikacja tworzenia niektórych elementów multitracka w tym najbardziej warstw i samego kontenera. Kontenerowi dodałem classname „tracks-container” a warstwom klasę „track” oraz dodałem dla każdej tej klasy atrybut „track-id”, który przyjmuje wartość aktualnego id danej warstwy. Wprowadzenie takich zmian było wymagane ze względu na lepszą identyfikację elementów po stronie pliku MultiTrackPlayer.jsx oraz w celach precyzyjnego odwoływania się do danych elementów w innych częściach kodu, na przykład przy modyfikacji danej ścieżki, bądź jej wyboru.

W pliku Multitrack.tsx

Dodałem funkcje stop, dodałem funkcje removeTrack, dodałem parę linijek przy tworzeniu elementów multitracka, dodanie opcji zaznaczania markerem pola, które nas interesuje, dodanie funkcji EditSegment i processAudioPlayer do obróbki audio, oraz dodanie funkcji renderMultiTrackAudio do renderowania Audio, poprawienie niektórych funkcji sprawdzających przy odtwarzaniu audio, aby program nie wywoływał odtwarzania w trakcie przetwarzania dźwięku.

## Przekazywanie danych między komponentami

## Pasek narzędzi

## Okno renderowania

# 

# Testowanie i naprawa błędów

# Przebieg pracy

# Podsumowanie

# Bibliografia

1. [Online] https://pl.wikipedia.org/wiki/Remiks.

2. [Online] https://pl.wikipedia.org/wiki/Mashup\_(muzyka).

3. [Online] https://wavesurfer.xyz/examples/?multitrack.js.

4. [Online] https://en.wikipedia.org/wiki/Multitrack\_recording#Software.

5. [Online] https://wavesurfer.xyz/examples/?regions.js.

6. [Online] https://wavesurfer.xyz/examples/?envelope.js.

7. [Online] https://github.com/katspaugh/wavesurfer-multitrack?tab=BSD-3-Clause-1-ov-file.

8. [Online] https://github.com/katspaugh/wavesurfer-multitrack/tree/main/src.

9. [Online] https://pl.wikipedia.org/wiki/JavaScript.

10. [Online] 2023. https://react.dev/.

11. [Online] https://www.typescriptlang.org/.

12. [Online] 05 07 2023. https://github.com/WebAudio/web-audio-api.

13. [Online] 05 07 2023. https://github.com/katspaugh/wavesurfer.js.

14. [Online] https://getbootstrap.com/docs/5.3/getting-started/introduction/.

15. [Online] https://github.com/pmndrs/zustand.

16. [Online] https://www.framer.com/motion/.

17. [Online] https://github.com/Experience-Monks/audiobuffer-to-wav.

18. [Online] https://github.com/zhuker/lamejs.

19. [Online] 29 Marzec 2019. https://muzyczny.pl/portal/czym-jest-oprogramowanie-daw-i-do-czego-sluzy/.

20. [Online] 05 07 2023. https://www.bandlab.com/.

21. [Online] 05 07 2023. https://studio.audiotool.com/.

22. [Online] 05 07 2023. https://soundation.com/.

23. [Online] 05 07 2023. https://ampedstudio.com/.

24. [Online] 05 07 2023. https://www.soundtrap.com/.

25. [Online] 05 07 2023. https://skytracks.io/about-us/.

26. [Online] 05 07 2023. https://github.com/gridsound/daw.

27. [Online] https://www.bandlab.com/songstarter.

28. [Online] https://pl.wikipedia.org/wiki/MIDI.

29. [Online] https://muzyczny.pl/portal/mastering-w-produkcji-muzycznej/.

30. [Online] https://sjp.pwn.pl/slowniki/sampel.

31. [Online] https://pl.wikipedia.org/wiki/VST.

32. [Online] https://pl.wikipedia.org/wiki/Miksowanie.

33. [Online] 2007. https://github.com/gridsound/daw/blob/main/LICENSE.

34. [Online] https://pl.wikipedia.org/wiki/MP3.

35. [Online] https://emastered.com/blog/what-is-bitrate-in-music.

36. [Online] https://pl.wikipedia.org/wiki/WAV.

37. [Online] https://www.izotope.com/en/learn/what-is-mix-automation.html.

38. [Online] https://github.com/remix-run/react-router.

39. [Online] 05 07 2023. https://skytracks.io/.

40. [Online] https://www.bandlab.com/songstarter.