



**WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA**
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: INFORMATYKA

Specjalność: Technologie internetowe i mobilne

Kacper Nowak
Nr albumu studenta w67262

Music Player

Prowadzący: mgr inż. Ewa Żesławska

Praca projektowa programowanie obiektowe C#

Rzeszów 2023

Spis treści

Wstęp	4
1 Opis założeń projektu	5
1.1 Cele projektu	5
1.2 Założenia projektu	5
1.3 Wymagania funkcjonalne i нефункционалне	6
2 Opis struktury projektu	7
2.1 Kluczowe Technologie i Narzędzia w Projekcie	7
2.2 Narzędzia i Rozszerzenia NuGet	8
2.3 Minimalne wymagania systemowe	9
2.4 Zaprojektowana Struktura i Opis Techniczny	9
2.5 Zarządzanie danymi w aplikacji	17
3 Harmonogram realizacji projektu	19
3.1 Repozytorium i System Kontroli Wersji	20
3.2 Problemy i Trudności	20
4 Prezentacja warstwy użytkowej projektu	21
5 Podsumowanie	28
Bibliografia	29
Spis rysunków	30
Spis tablic	31

Wstęp

W dobie cyfrowej rewolucji muzycznej, gdzie dostęp do ulubionych utworów stał się niemal nieograniczony, nasze codzienne życie coraz częściej opiera się na dźwiękach płynących z różnorodnych urządzeń. W tym dynamicznie rozwijającym się świecie, Projekt "Music Player" wyróżnia się swoim ambitnym celem – stworzeniem nowoczesnej, wieloplatformowej aplikacji do odtwarzania muzyki, która harmonijnie połączy w sobie funkcjonalność, bezpieczeństwo i estetykę.

Rozdział 1

Opis założeń projektu

1.1 Cele projektu

Projekt "Music Player" ma na celu stworzenie nowoczesnej aplikacji do odtwarzania muzyki, która będzie funkcjonować zarówno na systemach Windows, jak i Linux. Kluczowym zadaniem jest zapewnienie spójnego i intuicyjnego interfejsu użytkownika, zrealizowanego przy użyciu frameworka Avalonia, co pozwoli na zachowanie jednolitej estetyki i funkcjonalności na różnych platformach. Aplikacja zostanie napisana w języku C# z wykorzystaniem środowiska .NET 6, co zagwarantuje jej nowoczesność oraz wysoką wydajność.

Istotną częścią projektu jest wdrożenie systemu logowania, który nie tylko zwiększy bezpieczeństwo i prywatność użytkowników, ale także umożliwi personalizację aplikacji. Dzięki temu użytkownicy będą mogli lepiej zarządzać swoimi playlistami i preferencjami muzycznymi.

Projekt zakłada również opracowanie skutecznego systemu zarządzania danymi baz danych SQL. To umożliwi efektywne przechowywanie informacji o utworach muzycznych, artystach i playlistach. Aplikacja będzie zawierać pełny zestaw funkcji CRUD, co pozwoli użytkownikom na tworzenie, czytanie, aktualizowanie i usuwanie danych w prosty i intuicyjny sposób. Zarządzanie tymi funkcjami zostanie zrealizowane przez starannie zaprojektowaną hierarchię klas, co zapewni logiczną organizację i łatwość w rozwijaniu aplikacji. Znaczącą częścią projektu będzie również rozbudowana obsługa wyjątków oraz mechanizmy walidacji danych, które zagwarantują stabilność aplikacji oraz poprawność przetwarzanych informacji.

Podsumowując, głównym celem projektu "Music Player" jest stworzenie użytecznej, bezpiecznej i przyjaznej dla użytkownika aplikacji do odtwarzania muzyki, która będzie dostosowana do potrzeb użytkowników na różnych platformach, jednocześnie oferując zaawansowane opcje zarządzania zawartością muzyczną.

1.2 Założenia projektu

- **Wieloplatformowość:** Aplikacja będzie kompatybilna zarówno z systemami Windows, jak i Linux.
- **Technologie:** Wykorzystanie języka programowania C# oraz frameworka .NET 6.
- **Interfejs Użytkownika:** Stworzenie interfejsu graficznego z użyciem Avalonia, zapewniającego spójność i intuicyjność obsługi na różnych platformach.
- **Baza Danych:** Początkowe użycie plików tekstowych jako bazy danych, z możliwością rozbudowy do systemu bazodanowego takiego jak SQL.
- **Zarządzanie Danymi:** Implementacja funkcjonalności CRUD (Create, Read, Update, Delete)
- **Walidacja Danych:** Weryfikacja poprawności danych wejściowych.
- **Obsługa Wyjątków:** Zapewnienie mechanizmów do obsługi błędów i wyjątków w aplikacji.

- Rozszerzalność: Możliwość rozbudowy aplikacji o dodatkowe funkcje, takie jak importowanie i eksportowanie danych z plików csv.
- Repozytorium Kodu: Utrzymanie i zarządzanie kodem źródłowym w publicznym repozytorium, np. na GitHub.

1.3 Wymagania funkcjonalne i нефункционалне

Wymagania funkcjonalne:

- Wieloplatformowość: Aplikacja będzie kompatybilna zarówno z systemami Windows, jak i Linux.
- Technologie: Wykorzystanie języka programowania C# oraz frameworka .NET 6.
- Interfejs Użytkownika: Stworzenie interfejsu graficznego z użyciem Avalonia, zapewniającego spójność i intuicyjność obsługi na różnych platformach.
- Baza Danych: Aplikacja zapewni zaawansowane możliwości zarządzania danymi w bazie SQL, umożliwiając efektywną organizację, aktualizację oraz optymalizację przechowywanych informacji.
- Zarządzanie Danymi: Implementacja funkcjonalności CRUD (Create, Read, Update, Delete)
- Walidacja Danych: Weryfikacja poprawności danych wejściowych.
- Obsługa Wyjątków: Zapewnienie mechanizmów do obsługi błędów i wyjątków w aplikacji.
- Rozszerzalność: Możliwość rozbudowy aplikacji o dodatkowe funkcje, takie jak importowanie i eksportowanie danych z plików csv.
- Repozytorium Kodu: Utrzymanie i zarządzanie kodem źródłowym w publicznym repozytorium, np. na GitHub.
- Migracja danych: Aplikacja umożliwi efektywną migrację danych z istniejących baz SQL do nowego systemu, wspierając bezproblemowy transfer danych.

Wymagania нефункционалне:

- Wydajność: Aplikacja powinna szybko reagować na działania użytkownika i efektywnie zarządzać zasobami systemu.
- Kompatybilność: Działanie na różnych systemach operacyjnych (Windows, Linux) bez konieczności modyfikacji.
- Bezpieczeństwo: Zabezpieczenie danych użytkownika i systemu przed nieautoryzowanym dostępem.
- Skalowalność: Możliwość rozbudowy aplikacji o nowe funkcje bez wpływu na istniejącą funkcjonalność.
- Łatwość Użytkowania: Intuicyjny i prosty w obsłudze interfejs użytkownika.
- Stabilność: Aplikacja powinna być stabilna i wolna od krytycznych błędów.
- Dostępność: Łatwość dostępu do aplikacji i jej funkcji dla różnych grup użytkowników.
- Dokumentacja: Zapewnienie kompletnej i zrozumiałej dokumentacji użytkownika i technicznej.

Rozdział 2

Opis struktury projektu

Projekt ten został zrealizowany z wykorzystaniem nowoczesnych technologii programistycznych. Głównym językiem programowania jest C#, znany ze swojej wszechstronności i wydajności, co czyni go idealnym wyborem dla aplikacji multimedialnych. Aplikacja korzysta z frameworka Entity Framework do zarządzania bazą danych, co pozwala na efektywne i bezproblemowe przechowywanie oraz dostęp do danych użytkownika. Interfejs użytkownika został zbudowany z wykorzystaniem Avalonia, nowoczesnego frameworka GUI, który zapewnia płynne i atrakcyjne wizualnie doświadczenie użytkownika.

2.1 Kluczowe Technologie i Narzędzia w Projekcie

- **.NET 6:** Jako kompleksowe środowisko uruchomieniowe i framework programistyczny, .NET 6 oferuje wydajność, bezpieczeństwo i wsparcie dla wielu platform, co jest niezbędne dla nowoczesnych aplikacji. Umożliwia ono tworzenie wydajnej i skalowalnej aplikacji, zapewniając jednocześnie wsparcie dla najnowszych standardów i praktyk programistycznych.
- **Avalonia :** Jest frameworkiem do tworzenia interfejsów użytkownika, który umożliwia tworzenie aplikacji działających na wielu platformach. W projekcie "Music Player", Avalonia jest wykorzystywana do stworzenia atrakcyjnego i responsywnego interfejsu użytkownika, który jest spójny na różnych systemach operacyjnych. Dzięki swojej elastyczności i wsparciu dla XAML, Avalonia idealnie nadaje się do budowy nowoczesnych aplikacji multimedialnych, takich jak "Music Player", oferując zaawansowane funkcje GUI, takie jak animacje, style i szablony.
- **Repozytorium kodu Github:** Jest to platforma do hostingu kodu źródłowego, która wykorzystuje system kontroli wersji Git. Umożliwia przechowywanie, zarządzanie i śledzenie zmian w kodzie projektów.
- **GitHub Actions:** Jest to narzędzie CI/CD (Continuous Integration/Continuous Deployment) zintegrowane z GitHubem, które umożliwia automatyzację różnych etapów rozwoju oprogramowania, takich jak testowanie, budowanie i wdrażanie aplikacji. W projekcie "Music Player", GitHub Actions jest używany do automatycznego budowania projektu, co jest widoczne w pliku `dotnet-build.yml`. Ten plik definiuje zadania, które są wykonywane automatycznie przy każdym pushu lub pull requestie, zapewniając ciągłą integrację i sprawdzanie jakości kodu.
- **Baza Danych MySQL:** Jest to popularny system zarządzania relacyjnymi bazami danych (RDBMS). Charakteryzuje się otwartym źródłem i jest szeroko stosowany w aplikacjach internetowych. W projekcie "Music Player", MySQL używany jest do przechowywania danych użytkowników, playlist, utworów i plików muzycznych. Jest to wybór odpowiedni dla aplikacji wymagających niezawodnego, skalowalnego i łatwego w zarządzaniu systemu bazodanowego.
- **Docker i Docker Compose:** Używane są do zarządzania lokalnym środowiskiem serwera SQL, co ułatwia testowanie i rozwój aplikacji. Docker pozwala na szybkie uruchomienie izolowanego

środowiska MySQL, a Docker Compose upraszcza konfigurację i zarządzanie wieloma kontenerami. Ta kombinacja narzędzi zapewnia wydajne i elastyczne środowisko deweloperskie, które jest kluczowe dla płynnego rozwoju projektu.

- **JetBrains DataGrid:** Jest to kontrolka interfejsu użytkownika umożliwiająca efektywne wyświetlanie i manipulowanie danymi w formie tabeli. W "Music Player" DataGrid jest wykorzystywany do prezentacji i zarządzania informacjami, takimi jak listy utworów czy dane użytkowników od strony administratora.
- **JetBrains Rider:** Jest zaawansowanym zintegrowanym środowiskiem programistycznym (IDE) dla platformy .NET, które oferuje bogate funkcje, takie jak refaktoryzacja, inteligentne podpowiedzi kodu i efektywne debugowanie.
- **Visual Studio:** Z kolei, zostało użyte do wygenerowania diagramu klas, co pomaga wizualizować strukturę i zależności między różnymi komponentami aplikacji. Użycie Visual Studio do tworzenia diagramów klas jest przydatne w celu lepszego zrozumienia architektury projektu.

2.2 Narzędzia i Rozszerzenia NuGet

Projekt "Music Player" wykorzystuje szereg rozszerzeń NuGet, które znacząco przyczyniają się do jego funkcjonalności i wydajności. Poniżej przedstawiam szczegółowy opis każdego z tych pakietów:

1. **Avalonia:** (W tym: Avalonia.Desktop, Avalonia.Themes.Fluent i Avalonia.Fonts.Inter) Są to główne pakiety frameworka GUI używany w projekcie. Avalonia umożliwia tworzenie atrakcyjnych, platformowo niezależnych interfejsów użytkownika. Jest to kluczowy element, który pozwala aplikacji działać na różnych systemach operacyjnych.
2. **CsvHelper:** Biblioteka ta ułatwia operacje na plikach CSV, takie jak ich czytanie i zapisywanie. Jest to przydatne w kontekście eksportowania lub importowania danych, np. list utworów.
3. **Microsoft.EntityFrameworkCore:** Jest to framework ORM (Object-Relational Mapping) od Microsoftu, który umożliwia efektywne zarządzanie bazą danych w sposób obiektowy. Jest kluczowy dla operacji CRUD (Create, Read, Update, Delete) w projekcie.
4. **Microsoft.EntityFrameworkCore.Tools:** Zestaw narzędzi wspierających Entity Framework Core, w tym migracje bazy danych, co jest istotne dla utrzymania i aktualizacji struktury bazy danych.
5. **NAudio:** Jest to biblioteka do obsługi dźwięku, która pozwala na odtwarzanie, nagrywanie, edycję i konwersję plików audio. Jest kluczowa dla funkcjonalności odtwarzacza muzyki w projekcie.
6. **Pomelo.EntityFrameworkCore.MySql:** Ten pakiet umożliwia wykorzystanie MySQL jako bazy danych w połączeniu z Entity Framework Core, co zapewnia wsparcie dla popularnego systemu zarządzania bazą danych MySQL.

Każdy z tych pakietów odgrywa istotną rolę w funkcjonowaniu aplikacji "Music Player", zapewniając niezbędne narzędzia i funkcje, które przyczyniają się do jej wydajności, stabilności i atrakcyjności użytkowej.

2.3 Minimalne wymagania systemowe

Wymagania minimalne dla systemu Windows:

1. **System Operacyjny:** Windows 10.
2. **Procesor:** Intel Core i3 (8. generacji) lub nowszy / AMD Ryzen 3 (2. generacji) lub nowszy, z co najmniej 4 rdzeniami.
3. **Pamięć RAM:** Minimum 6 GB.
4. **Miejsce na Dysku:** Minimum 500 MB wolnego miejsca na dysku twardym.
5. **.NET Runtime:** .NET Core 6.0 lub nowszy.
6. **Dźwięk:** Karta dźwiękowa kompatybilna z systemem operacyjnym.
7. **Karta Graficzna:** Wsparcie dla DirectX 9 lub nowszego z WDDM 1.0 driver.

Wymagania minimalne dla systemu Linux:

1. **System Operacyjny:** Ubuntu 16.04 lub nowsza, Fedora 29 lub nowsza, Debian 9 lub nowsza, lub inne kompatybilne dystrybucje.
2. **Procesor:** Intel Core i3 (8. generacji) lub nowszy / AMD Ryzen 3 (2. generacji) lub nowszy, z co najmniej 4 rdzeniami.
3. **Pamięć RAM:** Minimum 4 GB.
4. **Miejsce na Dysku:** Minimum 500 MB wolnego miejsca.
5. **.NET Runtime:** .NET Core 6.0 lub nowszy.
6. **Dźwięk:** Karta dźwiękowa kompatybilna z systemem operacyjnym.
7. **Biblioteki:** libgtk-3-0, libwebkit2gtk-4.0-37, libasound2 (dla Ubuntu/Debian), libX11, libXtst, libXScrnSaver, libsecret-1-0, libc6, libgcc1, libgdiplus (dla Fedora).

2.4 Zaprojektowana Struktura i Opis Techniczny

Projekt "Music Player" reprezentuje zaawansowane podejście do programowania aplikacji, łącząc w sobie najlepsze praktyki i wzorce projektowe. Składający się z 48 klas, w tym 4 interfejsów i 2 enumów, projekt ten jest doskonałym przykładem efektywnego wykorzystania architektury opartej na luźno powiązanych klasach, co jest możliwe dzięki zastosowaniu Dependency Injection (DI) oraz wzorca Singleton.

Diagram klas projektu



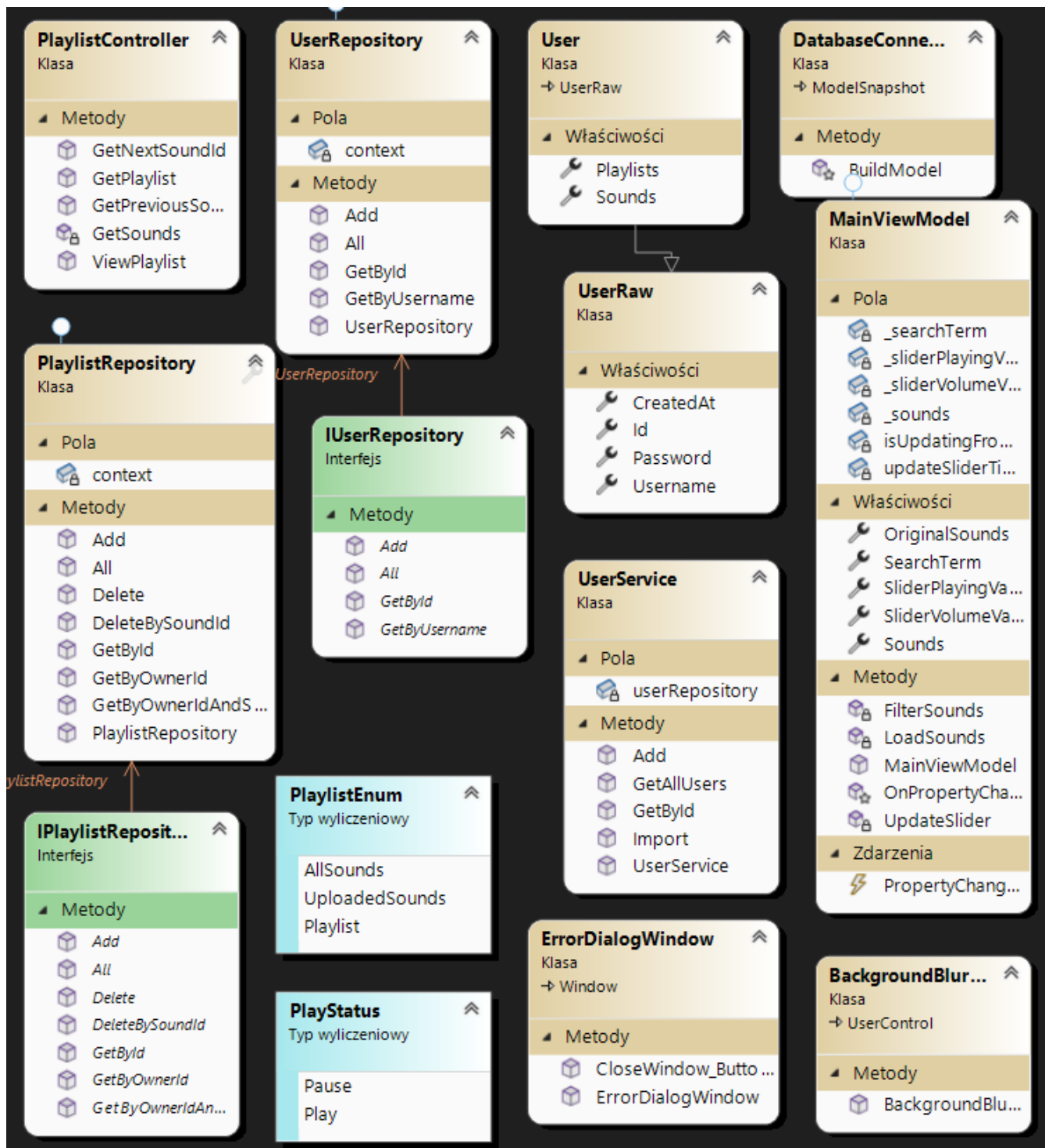
Rysunek 2.1: Diagram klas część 1

Na przedstawionym diagramie widzimy tylko część klas projektu (jedną z czterech części całego diagramu). Rozmieszczenie tych klas nie jest bezpośrednio związane, gdyż obejmuje zarówno klasy repozytorium bazy danych, serwisy, pomocnicze klasy do migracji takie jak `SoundsTable`, jak i klasy tworzące interfejs użytkownika GUI, na przykład `AddSoundWindow`. Interesujące może być zastosowanie dziedziczenia klasy `SoundRaw` w klasie `Sound`. Rozwiązanie to jest stosowane tylko w wybranych przypadkach. Klasa `Sound` reprezentuje tabelę `Sounds`, o której będzie mowa później, ale definiuje również obiekty takie jak `File` oraz `User`, które są używane w relacjach między tabelami w bazie danych. Klasa `SoundRaw` definiuje wyłącznie kolumny w bazie (nie uwzględnia relacji), co jest potrzebne w kontekście importu i eksportu danych do formatu CSV.

Kolejnym interesującym aspektem jest zastosowanie wzorca `Dependency Injection` do wstrzyki-

wania klasy repozytorium `SoundRepository` w konstruktorze serwisu `SoundService`. Interfejs `ISoundRepository` definiuje metody, które nasze `SoundRepository` implementuje, a następnie instancja tego repozytorium jest wstrzykiwana do serwisu.

Klasa `SoundService` oferuje wszechstronne zarządzanie danymi, w tym operacje CRUD, walidację danych, import i eksport, a także obsługę błędów. Podobnie klasa `PlaylistService` umożliwia wykonanie tych samych operacji, jednak nie zajmuje się walidacją danych, gdyż nie operuje na danych wprowadzonych przez użytkownika, co sprawia, że jej obsługa jest nieco prostsza. Kolejnym kluczowym elementem przedstawionego diagramu jest klasa `AudioPlayer` – to ona przetwarza, odtwarza i zarządza dźwiękiem w aplikacji. Dysponuje metodami takimi jak `GetTrackLengthInSeconds()`, która zwraca długość aktualnie odtwarzanego utworu w sekundach, `SetVolume()` do regulacji głośności, a także `Play()`, `Stop()` i `Pause()` do kontrolowania odtwarzania utworu.



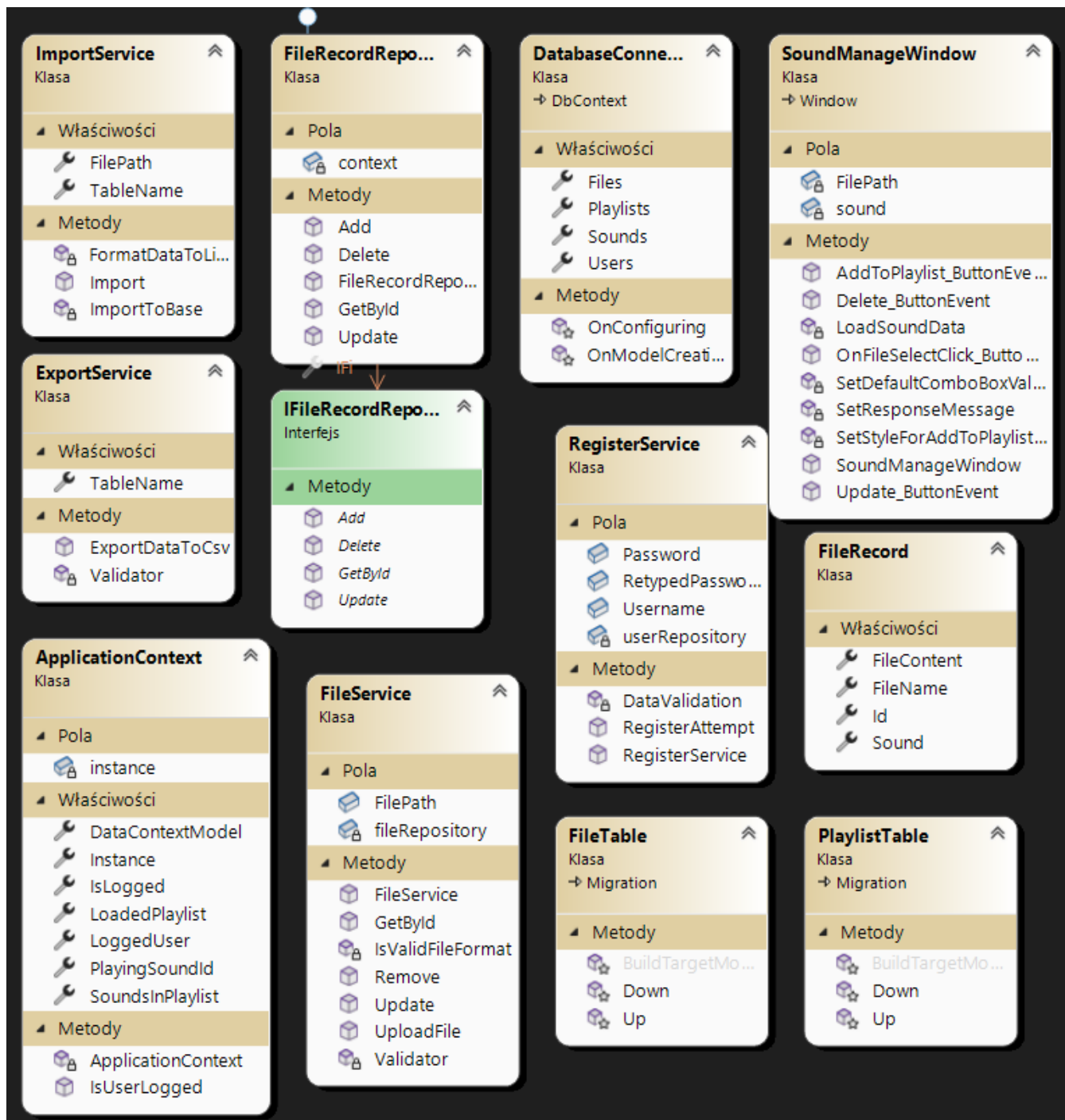
Rysunek 2.2: Diagram klas część 2

Na kolejnym, drugim już diagramie, główną uwagę przykuwa klasa `MainViewModel`, która odpowiada za zarządzanie danymi między interfejsem użytkownika (UI) a logiką aplikacji. Odpowiada ona za odtwarzanie, wyszukiwanie, przewijanie, regulację głośności oraz kontrolowanie utworów. Opis reszty klas przedstawiono w punktach:

1. **PlaylistController:** Klasa kontrolera zarządzająca playlistami. Posiada metody takie jak `getNextSoundId`, `getPlaylist`, `getPreviousSoundId`, `getSounds`, `viewPlaylist`, które służą do nawigacji i zarządzania listami odtwarzania w interfejsie użytkownika.
2. **UserRepository:** Repozytorium obsługujące operacje na danych użytkowników. Zawiera pola i metody niezbędne do realizacji operacji CRUD na użytkownikach, takie jak `Add`, `All`, `Delete`,

GetById, GetByUsername.

3. **IUserRepository:** Interfejs definiujący metody dostępu do danych użytkowników, których implementacja jest wymagana w `UserRepository`.
4. **User i UserRaw:** `User` definiuje tablice `Users` w bazie danych z relacją do tablic `Playlists` i `Sounds`. `UserRaw` reprezentuje surowe dane użytkownika, takie jak `Id`, `Password`, `Username`.
5. **UserService:** Serwis zapewniający logikę biznesową związaną z użytkownikami. Posiada pola i metody, takie jak `Add`, `GetAllUsers`, `GetById`, `Import`, które pozwalają na zarządzanie danymi użytkowników.
6. **DatabaseConnectionContextModelSnapshot:** Klasa `ModelSnapshot`, która zarządza wersjonowaniem modelu bazy danych..
7. **PlaylistRepository i IPlaylistRepository:** Analogicznie do `UserRepository`, te klasy zarządzają operacjami na playlistach, z metodami do wykonania operacji CRUD.
8. **PlaylistEnum i PlayStatus:** Typy wyliczeniowe, które definiują zbiór wartości, jakie mogą przyjąć playlisty (`AllSounds`, `UploadedSounds`, `Playlist`) oraz statusy odtwarzania (`Pause`, `Play`).
9. **ErrorDialogWindow:** Klasa okna dialogowego służąca do wyświetlania komunikatów o błędach, posiada metodę taką jak `CloseWindow_ButtonClick`.
10. **BackgroundBlur:** Klasa `UserControl`, która służy do dodawania efektu rozmycia tła w interfejsie użytkownika.

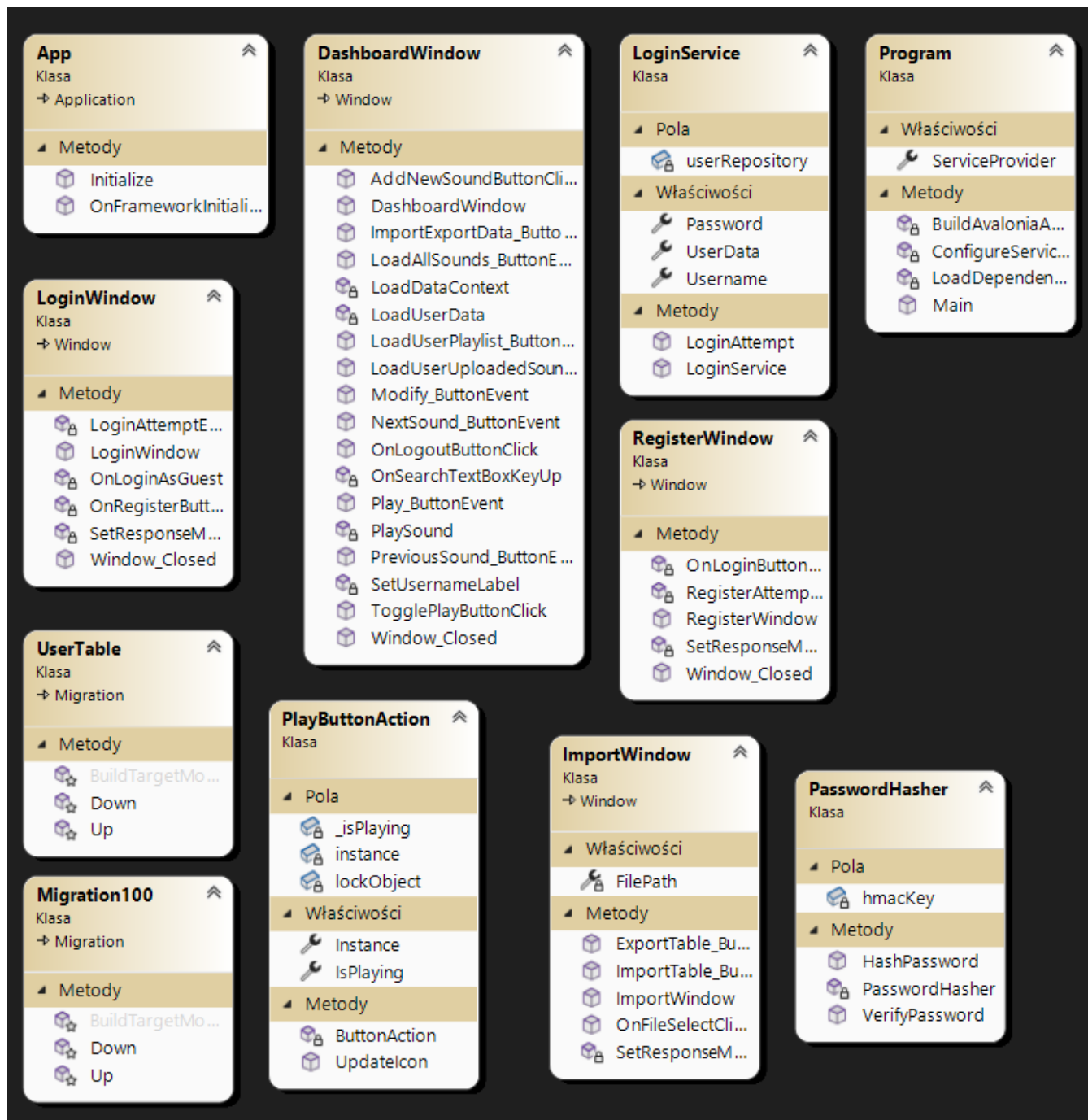


Rysunek 2.3: Diagram klas część 3

Przedostatnia część diagramu, czyli trzecia, prezentuje nam klasę `ApplicationContext`. Klasa ta służy jako centralny punkt zarządzania stanem aplikacji i zawiera informacje o zalogowanym użytkowniku, załadowanej playliście i odtwarzanym dźwięku. Wykorzystuje ona wzorec projektowy `Singleton`, który zapewnia stworzenie tylko jednej instancji klasy, co pozwala efektywnie przechowywać stan aplikacji. Opis reszty klas:

1. **ImportService**: Klasa służąca do importowania danych. Posiada właściwości takie jak `FilePath` i `TableName`, a także metody `FormatDataToList`, `Import`, `ImportToBase`, które pozwalają na przetwarzanie i załadowanie danych do bazy danych.
2. **FileRecordRepository**: Repozytorium obsługujące operacje na rekordach plików. Zawiera standardowe metody CRUD: `Add`, `Delete`, `GetById`, `Update`.

3. **IFileRecordRepository:** Interfejs definiujący kontrakt dla `FileRecordRepository`, zapewniając oddzielenie implementacji od abstrakcji.
4. **ExportService:** Klasa odpowiedzialna za eksport danych. Posiada metody takie jak `ExportDataToCsv` oraz `Validator`, które pozwalają na weryfikację i eksport danych do formatu CSV.
5. **DatabaseConnection:** Klasa z rozszerzeniem `DbContext`, używana przez Entity Framework do konfiguracji połączenia z bazą danych oraz definiowania modelu bazy danych za pomocą metod `OnConfiguring` i `OnModelCreating`.
6. **SoundManageWindow:** Klasa reprezentująca okno zarządzania dźwiękiem w interfejsie użytkownika. Zawiera metody do dodawania dźwięków do playlisty, usuwania, ładowania danych dźwiękowych, aktualizacji i innych interakcji z użytkownikiem.
7. **RegisterService:** Serwis obsługujący rejestrację użytkowników. Wykorzystuje `userRepository` do dostępu do danych użytkowników i zawiera metody `DataValidation`, `RegisterAttempt` do walidacji i próby rejestracji.
8. **FileService:** Serwis do zarządzania plikami, z metodami takimi jak `GetById`, `IsValidFileFormat`, `Remove`, `Update`, `UploadFile`, `Validator`.
9. **FileRecord:** Klasa reprezentująca rekord pliku w tabeli `Files` w bazie danych MySQL z właściwościami takimi jak `FileContent`, `FileName`, `Id`, `Sound`.
10. **FileTable i PlaylistTable:** Klasy migracji, które są częścią Entity Framework Migrations i zawierają metody `Up` i `Down` do aktualizacji schematu bazy danych.



Rysunek 2.4: Diagram klas część 3

Już ostatnia część diagramu prezentuje klasy takie jak **Program** (odpowiedzialna za uruchomienie aplikacji), **App** (inicjalizująca interfejs użytkownika w Avalonii) oraz **DashboardWindow**. **DashboardWindow** to klasa reprezentująca główne okno interfejsu użytkownika (Dashboard), przez które użytkownik może zarządzać aplikacją. Zawiera metody do zarządzania dźwiękiem, danymi oraz playlistami, takie jak **AddNewSoundButtonClickEvent**, **LoadData**, **LoadUserData**, **TogglePlayButtonClick** i inne, co umożliwia interaktywną obsługę różnorodnych funkcji aplikacji. Opis pozostałych klas:

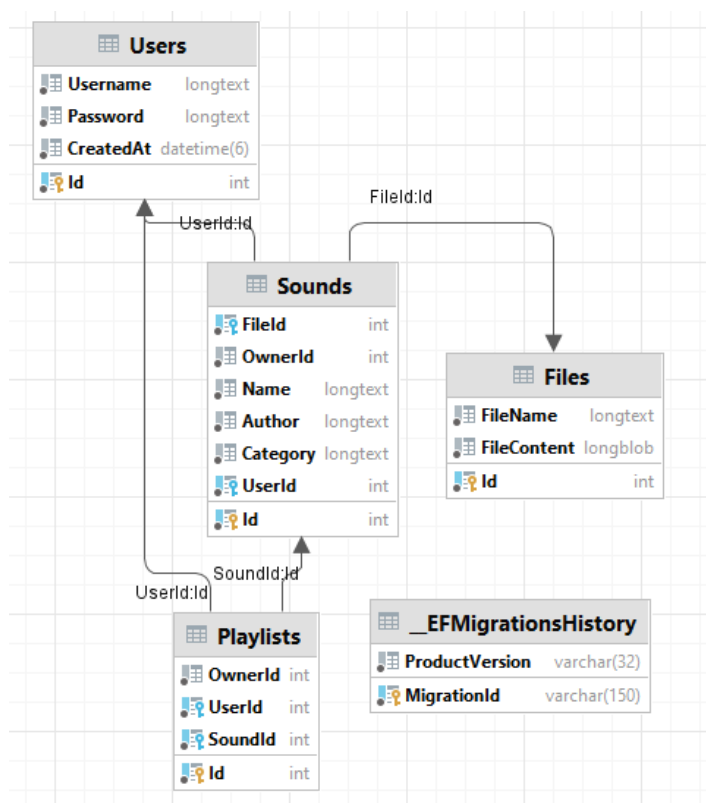
1. **LoginService**: Serwis odpowiedzialny za proces logowania. Używa **userRepository** do dostępu do danych użytkownika i zawiera metody takie jak **LoginAttempt** do obsługi weryfikacji danych logowania.
2. **RegisterWindow**: Okno do rejestracji nowych użytkowników. Posiada metody takie jak

OnLoginButtonClick, RegisterAttempt, co pozwala na obsługę zdarzeń związanych z procesem rejestracji.

3. **UserTable i Migration100:** Klasy reprezentujące migracje bazy danych, używane do aktualizacji schematu bazy danych, z metodami Up i Down.
4. **PlayButtonAction:** Klasa, która zarządza stanem przycisku odtwarzania. Posiada pola określające stan odtwarzania `_isPlaying` i metody takie jak `ButtonAction`, `UpdateIcon` co wskazuje na możliwość zmiany stanu przycisku odtwarzania. Dodatkowo, klasa ta jest tworzona jako pojedyncza instancja zgodnie z wzorcem projektowym Singleton.
5. **ImportWindow:** Okno do importowania danych do aplikacji. Zawiera metody `ExportTableButtonClickEvent`, `OnFileSelectClick`, które pozwalają na interakcję z użytkownikiem podczas importowania danych z plików.
6. **PasswordHasher:** Klasa zawierająca metody do hashowania i weryfikacji haseł, takie jak `HashPassword`, `VerifyPassword`, co jest istotne dla bezpieczeństwa danych użytkowników.

2.5 Zarządzanie danymi w aplikacji

Schemat bazy danych MySQL wykorzystywanej przez aplikację:



Rysunek 2.5: Schema bazy danych

Na załączonym schemacie bazy danych aplikacji widoczne są cztery kluczowe tabele, które współgrają ze sobą, tworząc kompleksowy system zarządzania danymi:

1. **Użytkownicy (Users):** Serce systemu stanowi tabela Users, przechowująca niezbędne informacje o użytkownikach, takie jak nazwa użytkownika (Username), hasło (Password) zahashowane przy pomocy algorytmu HMACSHA512, data utworzenia konta (CreatedAt) oraz uni-

kalny identyfikator (`Id`). Ta tabela jest fundamentem dla autentykacji oraz zarządzania profilami użytkowników.

2. **Dźwięki (Sounds):** Tabela `Sounds` służy jako repozytorium metadanych dla plików dźwiękowych w aplikacji. Każdy rekord dźwięku zawiera odniesienie do pliku (`FileId`), identyfikator właściciela (`OwnerId`), nazwę (`Name`), autora (`Author`), kategorię (`Category`) oraz unikalny identyfikator (`Id`). Powiązanie z tabelą `Users` za pomocą `UserId` umożliwia śledzenie, który użytkownik przesłał dany dźwięk, ułatwiając zarządzanie treścią i prawami własności.
3. **Pliki (Files):** W tabeli `Files` przechowywane są właściwe dane plików dźwiękowych, takie jak zawartość pliku (`FileContent`) i nazwa pliku (`FileName`), każdy z unikalnym identyfikatorem (`Id`). Ta tabela działa jako magazyn danych binarnych, co pozwala na oddzielenie fizycznych plików od ich opisowych metadanych przechowywanych w tabeli `Sounds`.
4. **Playlisty (Playlists):** Użytkownicy mogą tworzyć spersonalizowane playlisty za pośrednictwem tabeli `Playlists`, która zawiera odniesienia do ich twórców (`OwnerId`) oraz przypisane dźwięki (`SoundId`). Każda playlista posiada swój unikalny identyfikator (`Id`), co umożliwia łatwe odnajdywanie i zarządzanie kolekcjami dźwięków.
5. **Historia Migracji EF (EFMigrationsHistory):** Specjalna tabela `_EFMigrationsHistory` jest wykorzystywana przez Entity Framework do przechowywania historii migracji, które są niezbędne do aktualizacji i utrzymania integralności schematu bazy danych.

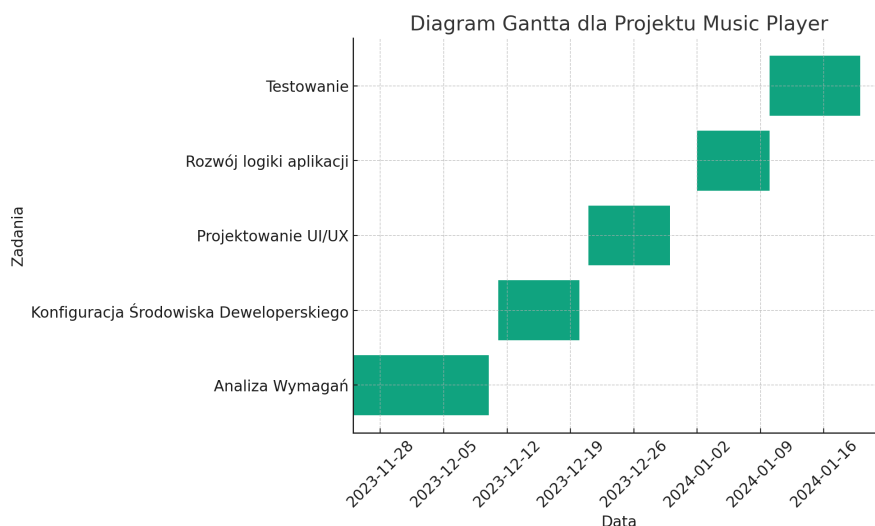
Wszystkie te tabele łączą się w spójny system, który umożliwia dynamiczne zarządzanie danymi użytkowników, ich twórczością oraz preferencjami. Dzięki starannie przemyślanej architekturze i relacjom między tabelami, aplikacja jest w stanie zapewnić efektywne i bezproblemowe doświadczenia dla użytkownika, od logowania i rejestracji, przez przesyłanie i odtwarzanie dźwięków, aż po tworzenie i użytkowanie playlist.

Aplikacja umożliwia również import i eksport danych w formacie CSV, jednakże jest to ograniczone, ponieważ operacje te są problematyczne dla tabeli `'Files'`. Zawiera ona bardzo duże rekordy, które obejmują całe pliki piosenek. W związku z tym, w obecnej wersji aplikacji, import i eksport tabeli `Files` nie jest dostępny. Import innych tabel, takich jak `'Users'`, `'Playlists'` i `'Sounds'`, funkcjonuje natomiast bez zarzutu.

Rozdział 3

Harmonogram realizacji projektu

W tym rozdziale przedstawiony jest harmonogram realizacji projektu aplikacji muzycznej, "Music Player", opracowanej w języku C#. Harmonogram ten jest graficznie reprezentowany za pomocą diagramu Gantta:



Rysunek 3.1: Schema bazy danych

Szczegółowy opis zadań dla projektu:

1. Analiza Wymagań (25.11.2023 - 10.12.2023)

Czas na zrozumienie i zebranie kluczowych funkcjonalności wymaganych przez aplikację. Proces ten obejmuje przygotowanie listy narzędzi potrzebnych do wykonania projektu, analizę konkurencyjnych aplikacji. Jest to fundament, na którym opiera się cały projekt.

2. Konfiguracja Środowiska Deweloperskiego (11.12.2023 - 20.12.2023)

Ustanowienie odpowiedniego środowiska deweloperskiego jest kluczowe dla produktywności. W tym okresie zostało skonfigurowane IDE, zależności, docker, docker-compose, biblioteki oraz repozytorium Git, aby zapewnić płynny start i ciągłość prac nad projektem.

3. Projektowanie UI/UX: (21.12.2023 - 30.12.2023)

Projekt interfejsu użytkownika (UI) oraz doświadczenia użytkownika (UX) ma bezpośredni wpływ na odbiór aplikacji przez użytkowników. W tym czasie budowany był cały główny interfejs UI by już na gotowej bazie okodować całą logikę aplikacji.

4. **Rozwój logiki aplikacji:** (02.1.2024 - 09.1.2024)

Kluczowy etap rozwoju projektu, gdzie fokus położony był na implementację zaprojektowanego interfejsu użytkownika z właściwymi mechanizmami działania aplikacji. Ten okres pracy obejmował kodowanie głównych funkcji aplikacji, takich jak odtwarzanie muzyki, zarządzanie playlistami.

5. **Testowanie** (10.01.2024 - 20.01.2024)

Testowanie jest niezbędnym etapem w cyklu życia oprogramowania. W tym okresie aplikacja była testowana manualnie i poprawiana na bieżąco. W testowaniu pomagał również wcześniej stworzony skrypt CI/CD, który przy każdym zatwierdzeniu zmian automatycznie sprawdzał, czy aplikacja kompiluje się poprawnie.

3.1 **Repozytorium i System Kontroli Wersji**

Projekt jest rozwijany z użyciem systemu kontroli wersji Git, co pozwala na efektywną współpracę i śledzenie historii zmian. Repozytorium projektu znajduje się na GitHub pod adresem: <https://github.com/byko-dev/music-player>. Projekt będzie zamieszczony pod danym linkiem do dnia 31.03.2025. Dzięki wykorzystaniu Git, możliwe było efektywne zarządzanie kodem źródłowym.

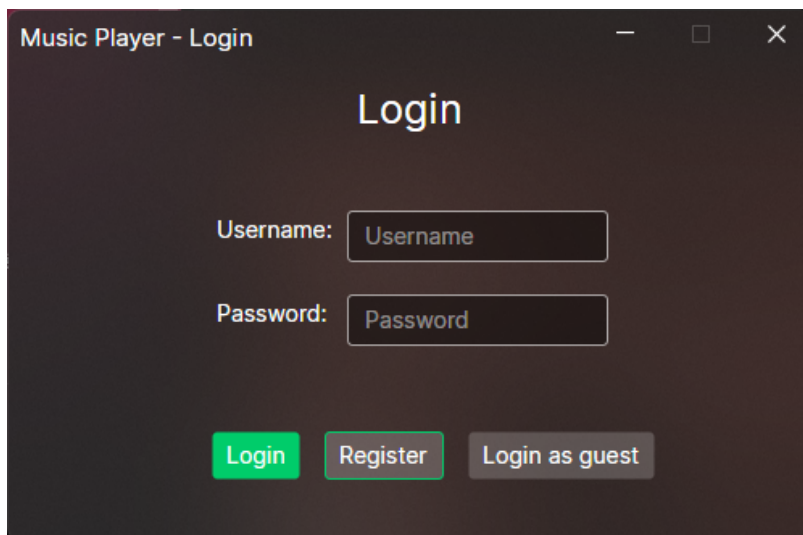
3.2 **Problemy i Trudności**

W trakcie realizacji projektu Music Player napotkano różnorodne wyzwania. Jednym z nich było zapewnienie wysokiej jakości interfejsu użytkownika, co wymagało dokładnego planowania i testowania. Największym wyzwaniem okazało się stworzenie systemu do importu i eksportu danych z aplikacji. Zaplanowane wcześniej relacje w tabelach nie ułatwiały zadania, ponieważ program eksportował niechciane i często błędne dane, a podczas importu trudno było zachować relacje między tabelami. Dostyc szybko okazało się konieczne odejście od importu/eksportu tabeli Files, ponieważ była to operacja wymagająca i zasobożerna. Rozwiązaniem problemu okazało się stworzenie obiektów z prefixem 'Raw', które nie implementują relacji tabelarycznych.

Rozdział 4

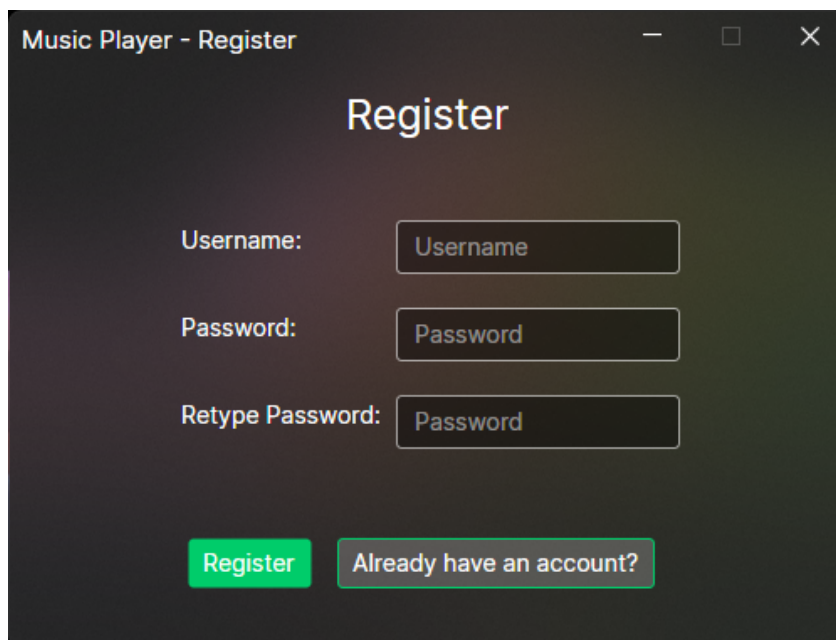
Prezentacja warstwy użytkowej projektu

Projekt "Music Player" został zaprojektowany z myślą o zapewnieniu intuicyjnego, łatwego w obsłudze i estetycznie przyjemnego interfejsu użytkownika. W tej części dokumentacji szczegółowo omówimy poszczególne komponenty interfejsu, w tym layout, nawigację, funkcje dostępne dla użytkownika oraz ogólną koncepcję i filozofię stojącą za projektowaniem UI/UX.



Rysunek 4.1: Formularz logowania

Formularz logowania w aplikacji "Music Player" stanowi pierwszy punkt interakcji użytkownika z aplikacją, łącząc w sobie elegancję i prostotę. Tło formularza jest częściowo przezroczyste dzięki zastosowaniu efektu rozmycia, co sprawia, że prezentuje się ono za każdym razem inaczej, w zależności od tła znajdującego się pod naszą aplikacją. Pola "Username" i "Password" są intuicyjnie umieszczone w centralnym punkcie okna, co pozwala na szybkie i wygodne wprowadzenie danych przez użytkownika. Trzy przyciski akcji – "Login", "Register" oraz "Login as guest" – są wyraźnie oznaczone i zróżnicowane kolorystycznie, co nie tylko dodaje dynamiki wizualnej, ale także prowadzi użytkownika przez proces logowania krok po kroku.



Music Player - Register

Register

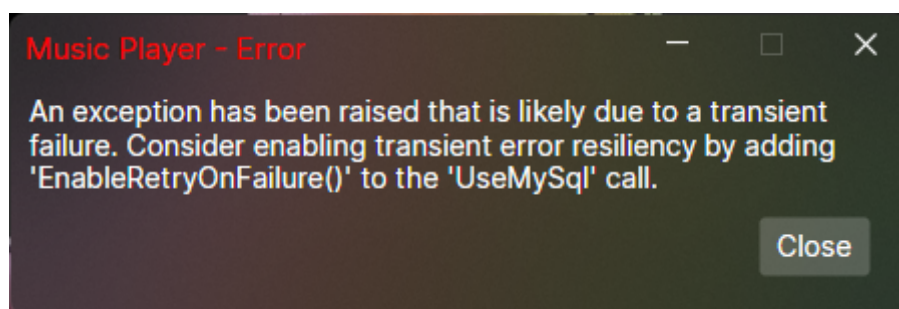
Username:

Password:

Retype Password:

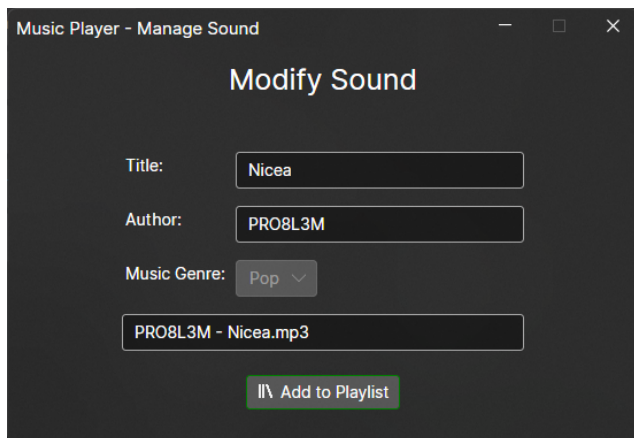
Rysunek 4.2: Formularz rejestracji

Ekran rejestracji w aplikacji ma podobną konstrukcję do wcześniej wspomnianego ekranu logowania. Składa się z trzech głównych pól: 'Username', 'Password' oraz 'Retype Password', które są wyraźnie oznaczone i logicznie rozmieszczone, ułatwiając w ten sposób proces zakładania nowego konta.

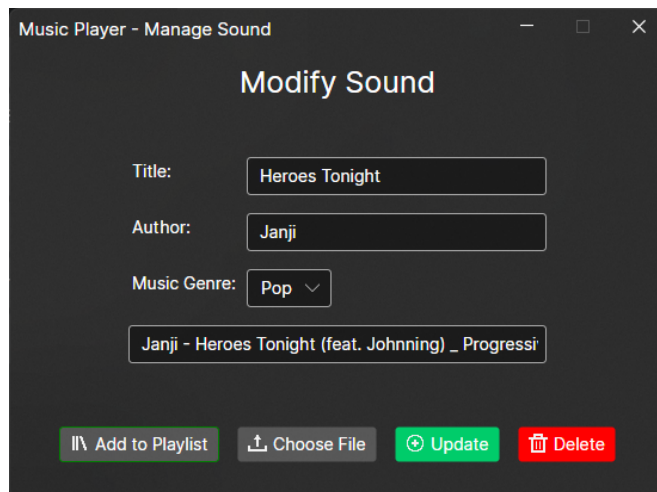


Rysunek 4.3: Okno błędów

Okno błędu w aplikacji "Music Player" zostało zaprojektowane w celu przekazania ważnych informacji w sposób jasny i zwięzły. Treść okna błędu informuje o wyjątku, który został zgłoszony z powodu chwilowego problemu i sugeruje rozwiązanie techniczne, które może pomóc w jego rozwiązaniu. Przycisk "Close" znajdujący się na dole okna jest wyraźnie zaznaczony i pozwala użytkownikowi szybko zamknąć komunikat błędu. Całość prezentuje się profesjonalnie i ma na celu ułatwienie szybkiego zidentyfikowania i rozwiązania problemu.



Rysunek 4.4: Okno edycji utworu - okno użytkownika



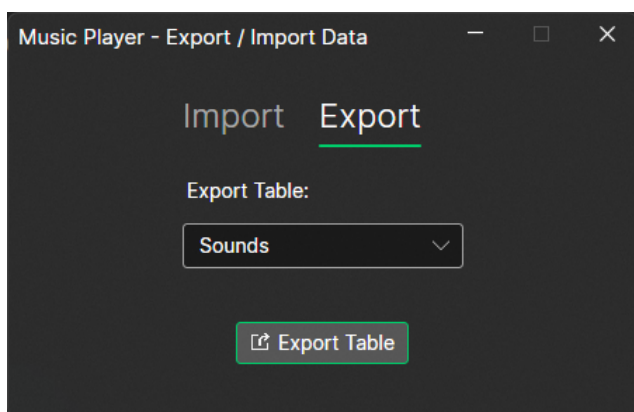
Rysunek 4.5: Okno edycji utworu - okno właściciela

Okna "Modify Sound" w aplikacji "Music Player" przedstawiają interfejs użytkownika umożliwiający zarządzanie informacjami o piosenkach. Są one dostosowane do różnych poziomów dostępu użytkownika, w zależności od tego, czy użytkownik dodał daną piosenkę do aplikacji.

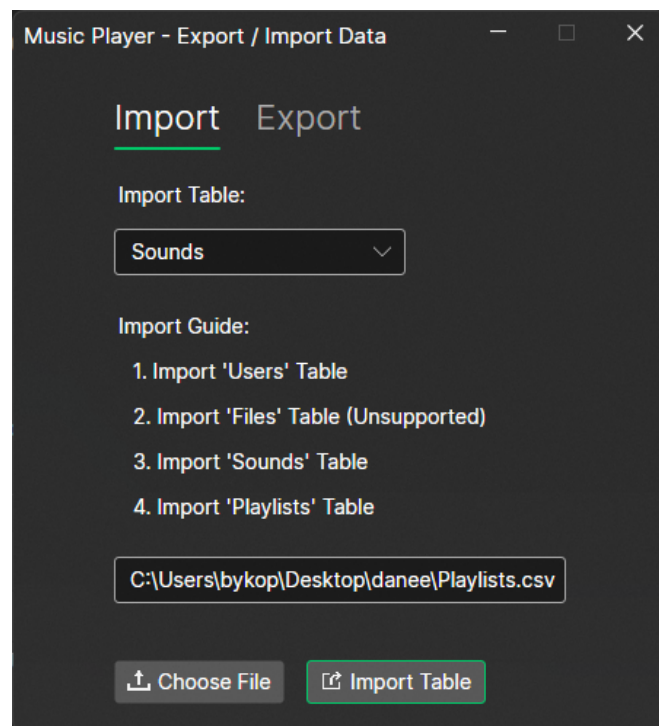
Pierwsze okno jest przeznaczone dla użytkownika, który nie ma możliwości edycji informacji o piosence, ponieważ nie jest jej właścicielem. Prezentuje podstawowe informacje takie jak tytuł utworu ("Nicea"), autora ("PRO8L3M") oraz gatunek muzyczny ("Pop"). Na dole okna znajduje się przycisk "Add to Playlist", który pozwala na dodanie utworu do osobistej playlisty. Całość prezentuje się czysto i profesjonalnie, z zachowaniem spójnej kolorystyki i stylu.

Drugie okno jest dostępne dla użytkownika, który jest właścicielem piosenki i dlatego ma możliwość jej edycji. Użytkownik może zmieniać tytuł ("Heroes Tonight"), autora ("Janji") oraz gatunek muzyczny z dostępnego menu rozwijanego. W tym oknie dodane są dodatkowe opcje: przycisk "Choose File" umożliwia zmianę pliku muzycznego, "Update" służy do zapisania zmian, a "Delete" pozwala na usunięcie utworu z bazy danych aplikacji. Dodatkowo, utwór można dodać do playlisty za pomocą przycisku "Add to Playlist". Przyciski akcji są intuicyjnie zaprojektowane z użyciem symboli i kolorów, gdzie zielony symbolizuje akcję aktualizacji, a czerwony - usunięcie.

Oba okna są zaprojektowane z myślą o przejrzystości i łatwości użytkowania, przy jednoczesnym zachowaniu funkcjonalności i estetycznego wyglądu zgodnego z ogólnym motywem aplikacji.



Rysunek 4.6: Okno eksportu danych

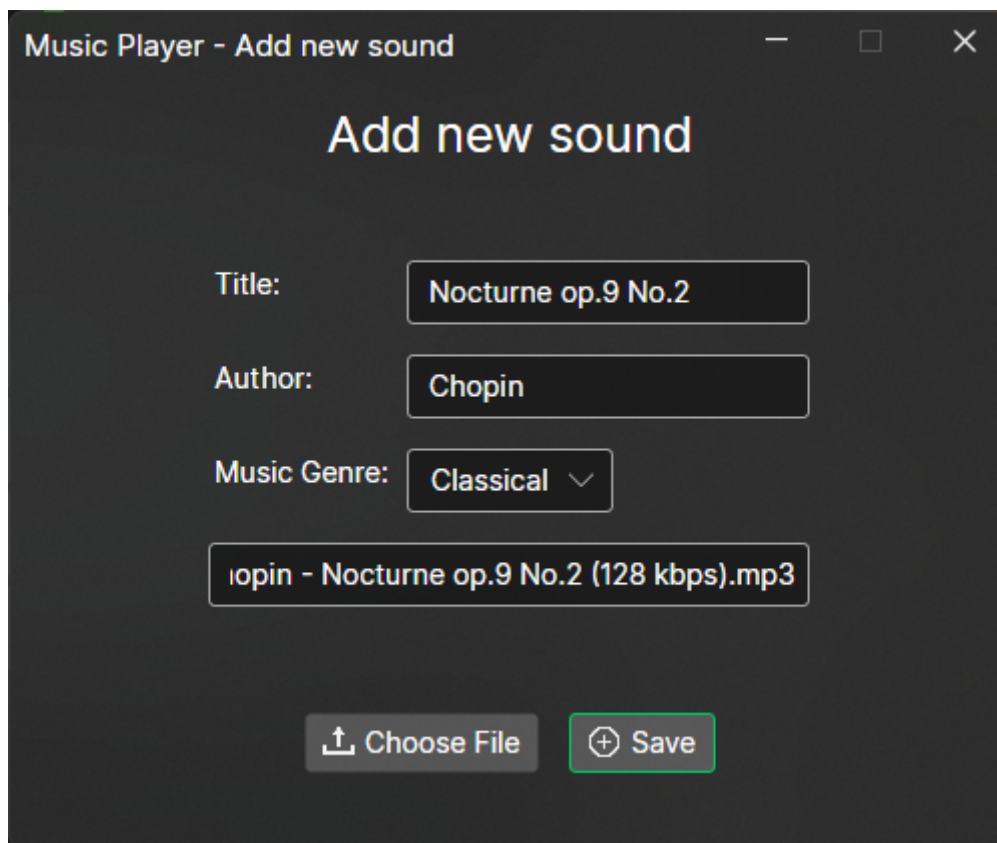


Rysunek 4.7: Okno importu danych

Okna importu i eksportu danych aplikacji "Music Player" są przejrzyste i funkcjonalne, umożliwiając użytkownikom łatwą migrację ich danych w formacie CSV, co jest standardem dla list utworów, playlist i informacji o użytkownikach.

Okno eksportu danych: W oknie eksportu użytkownik ma możliwość wyeksportowania wybranej tabeli danych do pliku CSV. Proces jest prosty i intuicyjny – wystarczy wybrać interesującą tabelę z rozwijanego menu (np. "Sounds"), a następnie kliknąć przycisk "Export Table". To wyzwoli automatyczne pobieranie pliku CSV zawierającego wszystkie dane z wybranej tabeli. Ta funkcja jest szczególnie przydatna przy tworzeniu kopii zapasowych lub przenoszeniu danych między różnymi instancjami aplikacji.

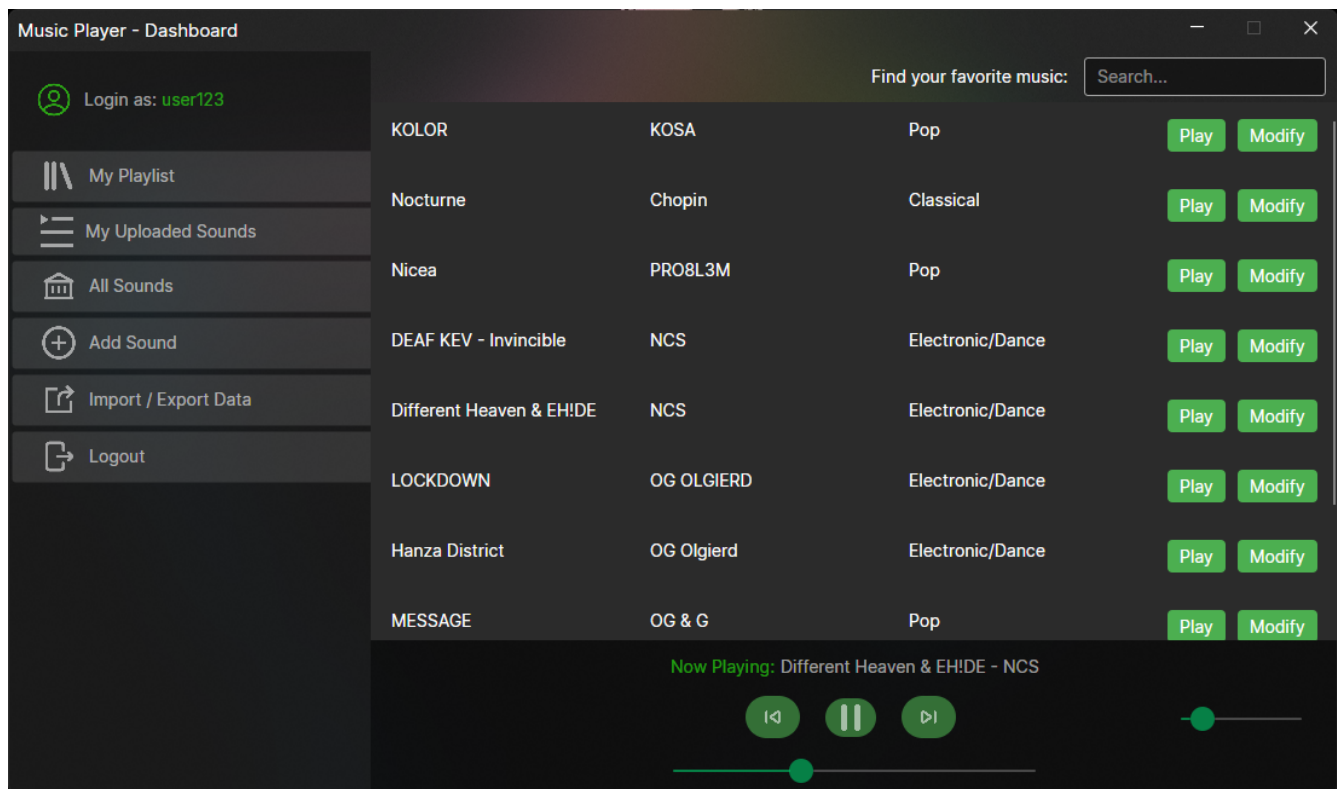
Okno eksportu danych: Okno importu danych oferuje użytkownikom możliwość załadowania danych z pliku CSV do aplikacji. Użytkownik może wybrać typ danych do importu (np. "Sounds", "Users", "Playlists") z rozwijanego menu. Pod menu znajduje się przewodnik po importowaniu, który krok po kroku informuje o sposobie importowania danych z różnych tabel. Aby zaimportować dane, użytkownik musi najpierw wybrać odpowiedni plik CSV z dysku, używając przycisku "Choose File", a następnie inicjować import przez kliknięcie "Import Table". Jest to krytyczna funkcja dla użytkowników, którzy chcą przywrócić dane lub zintegrować nowe kolekcje do swojej biblioteki.



Rysunek 4.8: Formularz dodawania utworów

Okno "Add new sound" w aplikacji "Music Player" jest przeznaczone do łatwego dodawania nowych utworów do biblioteki użytkownika. Użytkownicy mogą wprowadzić metadane dla każdego nowego dźwięku, takie jak tytuł utworu, autora oraz gatunek muzyczny z rozwijanego menu, gdzie można wybrać np. Classical, Pop, Rock, Jazz. Te pola pozwolą na lepszą organizację i wyszukiwanie w przyszłości.

Pole na dole, gdzie wyświetlana jest nazwa pliku, pokazuje wybrany utwór do dodania, który użytkownik załadował przez przycisk "Choose File". Po wybraniu pliku, utwór może być dodany do biblioteki aplikacji za pomocą przycisku "Save", który jest wyraźnie widoczny i łatwy do zlokalizowania. Proces dodawania jest intuicyjny i zaprojektowany tak, aby maksymalnie uproszczać zarządzanie kolekcją muzyki użytkownika.



Rysunek 4.9: Główne okno aplikacji

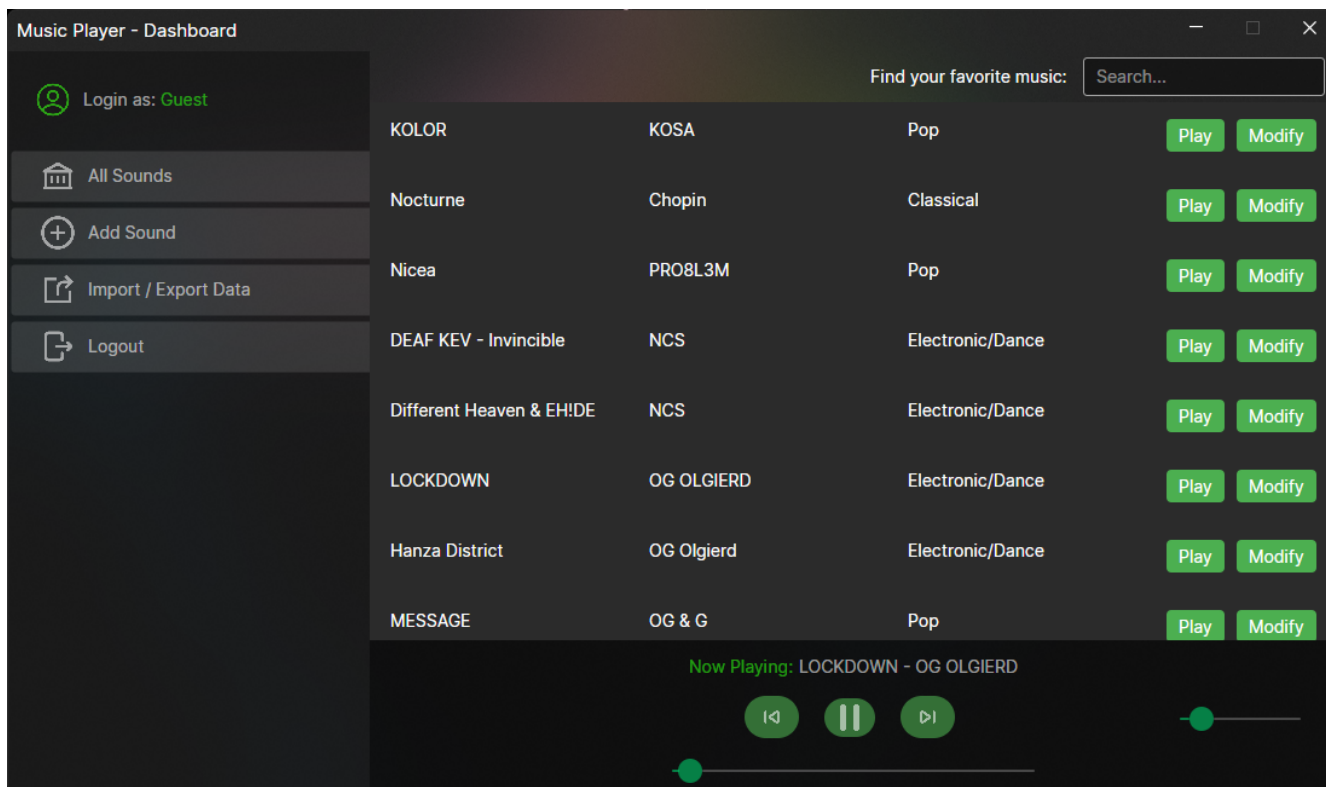
Główne okno aplikacji "Music Player" to centrum dowodzenia dla użytkownika, gdzie może zarządzać swoją muzyczną biblioteką i odtwarzaniem. Na lewym pasku bocznym znajduje się intuicyjne menu nawigacyjne z ikonami, które umożliwiają szybki dostęp do różnych sekcji aplikacji, takich jak "My Playlist", "My Uploaded Sounds", "All Sounds", a także opcje do dodawania nowych dźwięków i importu/eksportu danych. Na dole znajduje się opcja "Logout", która pozwala użytkownikowi bezpiecznie wylogować się z aplikacji.

W centralnej części okna znajduje się lista utworów z ich tytułami, autorami i gatunkami muzycznymi. Każdy wpis na liście ma przypisane przyciski "Play" i "Modify", które pozwalają odpowiednio na odtwarzanie wybranego utworu oraz jego edycję – ta druga opcja jest dostępna tylko dla utworów dodanych przez zalogowanego użytkownika.

Na górze tego panelu znajduje się pasek wyszukiwania, który umożliwia szybkie znalezienie ulubionej muzyki według nazwy utworu, wykonawcy lub gatunku.

Dolna część ekranu to dedykowany pasek odtwarzacza z podstawowymi przyciskami sterowania odtwarzaniem (play, pause, skip), a także pasek postępu, który wizualnie pokazuje postęp aktualnie odtwarzanego utworu. Ponadto wyświetlana jest nazwa aktualnie odtwarzanego utworu, co pozwala użytkownikowi na szybką identyfikację odtwarzanej kompozycji.

Całość interfejsu utrzymana jest w ciemnej kolorystyce, co jest przyjemne dla oka i nie męczy wzroku podczas dłuższego korzystania z aplikacji w różnych warunkach oświetleniowych.



Rysunek 4.10: Główne okno aplikacji - widok gościa

W porównaniu do wcześniejszego okna dashboardu dla zalogowanego użytkownika, obecne okno dla gościa prezentuje się bardzo podobnie, z tą różnicą, że nie zawiera opcji "My Playlist" ani "My Uploaded Sounds". To wskazuje na ograniczoną funkcjonalność dostępną dla użytkowników, którzy nie są zalogowani lub korzystają z aplikacji jako goście.

Status logowania na górze okna zmienia się z konkretnego identyfikatora użytkownika (np. user123) na ogólne oznaczenie "Guest", co podkreśla, że użytkownik nie jest obecnie zalogowany i korzysta z aplikacji w ograniczonym zakresie. Użytkownik gość ma dostęp do ogólnej listy utworów w sekcji "All Sounds", może przeszukiwać dostępną muzykę i korzystać z funkcji odtwarzania, ale nie posiada możliwości modyfikacji listy utworów ani zarządzania własnymi dźwiękami. Może również wyeksportować/importować dane i wylogować się z systemu, co wskazuje na pewne podstawowe interakcje, które są dostępne dla każdego, kto korzysta z aplikacji.

Rozdział 5

Podsumowanie

Projekt Music Player, który rozpoczął się od ambitnych założeń, nie tylko spełnił pierwotne oczekiwania, ale także znacznie je przekroczył. W miarę rozwoju projektu, udało się zaimplementować szereg dodatkowych elementów, które podniosły wartość aplikacji. Dzięki wprowadzeniu konfiguracji GitHub Actions, proces wdrażania i ciągłej integracji stał się bardziej efektywny i automatyczny, a zastosowanie docker-compose pozwoliło na sprawną konfigurację środowiska deweloperskiego.

Podczas fazy developmentu, opracowano i zintegrowano solidną bazę danych SQL, co było znaczącym krokiem naprzód w zakresie zarządzania i dostępności danych. Ta zmiana zapewniła lepszą skalowalność aplikacji i otworzyła drzwi do dalszego rozszerzenia funkcjonalności.

Patrząc w przyszłość, plany rozwojowe aplikacji Music Player obejmują wprowadzenie opcji tworzenia spersonalizowanych playlist przez użytkowników, co uczyni aplikację jeszcze bardziej interaktywną i personalną. Ponadto, rozważa się możliwość wyboru między różnymi bazami danych z muzyką, dając użytkownikom jeszcze szerszy dostęp do różnorodnych utworów i gatunków muzycznych. Prace nad rozbudową opcji konfiguracji interfejsu użytkownika oraz funkcji aplikacji są także wizją, która może przekształcić Music Player w jeszcze bardziej zaawansowane narzędzie, dostosowane do potrzeb i preferencji szerokiej grupy odbiorców. Znaczący nacisk położono na estetykę UI, co przełożyło się na przyjemność użytkowania i pozytywne doświadczenia w interakcji z aplikacją. Staranne projektowanie UI, zgodne z najnowszymi trendami w designie, sprawia, że aplikacja wyróżnia się na tle innych odtwarzaczy muzycznych, oferując intuicyjność oraz wizualny komfort. Nowoczesny interfejs użytkownika, opracowany z myślą o zapewnieniu najlepszych wrażeń, jest jednym z kluczowych osiągnięć projektu i stanowi solidną platformę dla planowanych rozszerzeń funkcjonalności Music Player'a.

Podsumowując, realizacja projektu Music Player przeszła długą drogę od początkowych założeń po wdrożenie praktycznych i zaawansowanych rozwiązań. Z sukcesem zbudowano podstawy, które nie tylko zapewniają stabilną i funkcjonalną aplikację, ale także stwarzają mocne fundamenty pod przyszłą ekspansję i innowacje.

Bibliografia

- [1] Jacek Matulewski, *C#: lekcje programowania: praktyczna nauka programowania dla platform .NET i .NET Core*, Helion, Gliwice 2021.
- [2] Joseph Albahari, Eric Johannessen, *C# 8.0 w pigułce*, Helion, Gliwice, 2021.
- [3] R. S. Miles, *C#: zacznij programować!*, Helion, Gliwice, 2020.
- [4] Włodzimierz Gajda, *Git : rozproszony system kontroli wersji*, Helion, Gliwice, 2013.
- [5] A. Stellman, J. Greene, *C#. Rusz głową!*, Helion, Gliwice, 2022.

Spis rysunków

2.1	Diagram klas część 1	10
2.2	Diagram klas część 2	12
2.3	Diagram klas część 3	14
2.4	Diagram klas część 3	16
2.5	Schema bazy danych	17
3.1	Schema bazy danych	19
4.1	Formularz logowania	21
4.2	Formularz rejestracji	22
4.3	Okno błędów	22
4.4	Okno edycji utworu - okno użytkownika	23
4.5	Okno edycji utworu - okno właściciela	23
4.6	Okno eksportu danych	24
4.7	Okno importu danych	24
4.8	Formularz dodawania utworów	25
4.9	Główne okno aplikacji	26
4.10	Główne okno aplikacji - widok gościa	27

Spis tabel