MusicPlayer

Skład zespołu:

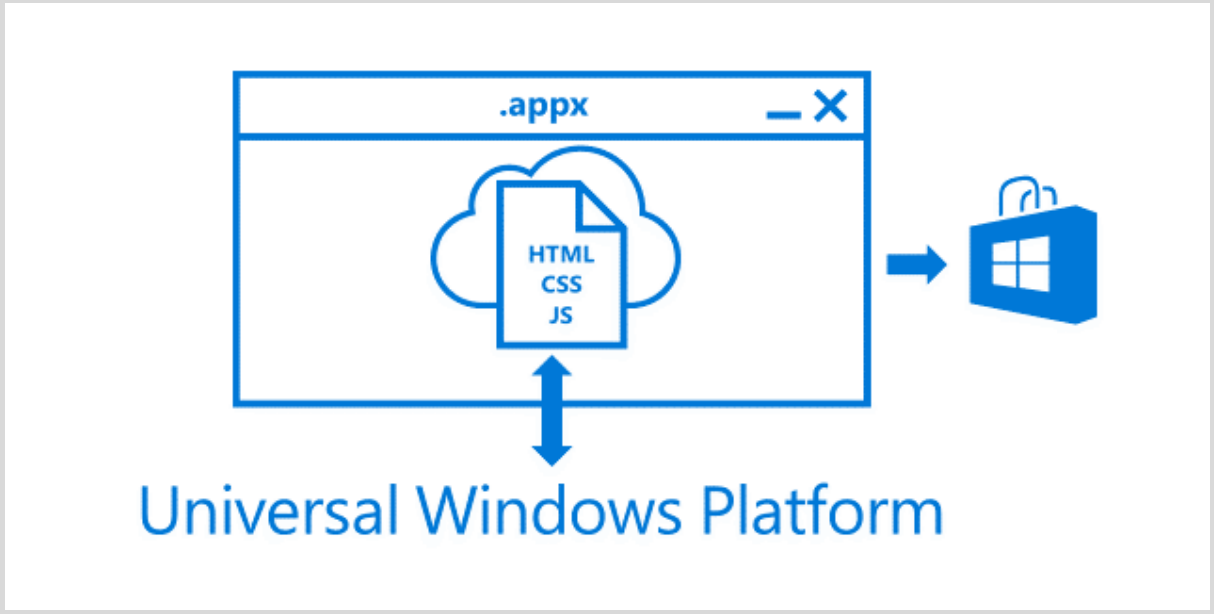
- Magdalena Kalisz
- Adam Bajguz
- Michał Kierzkowski

Opis projektu

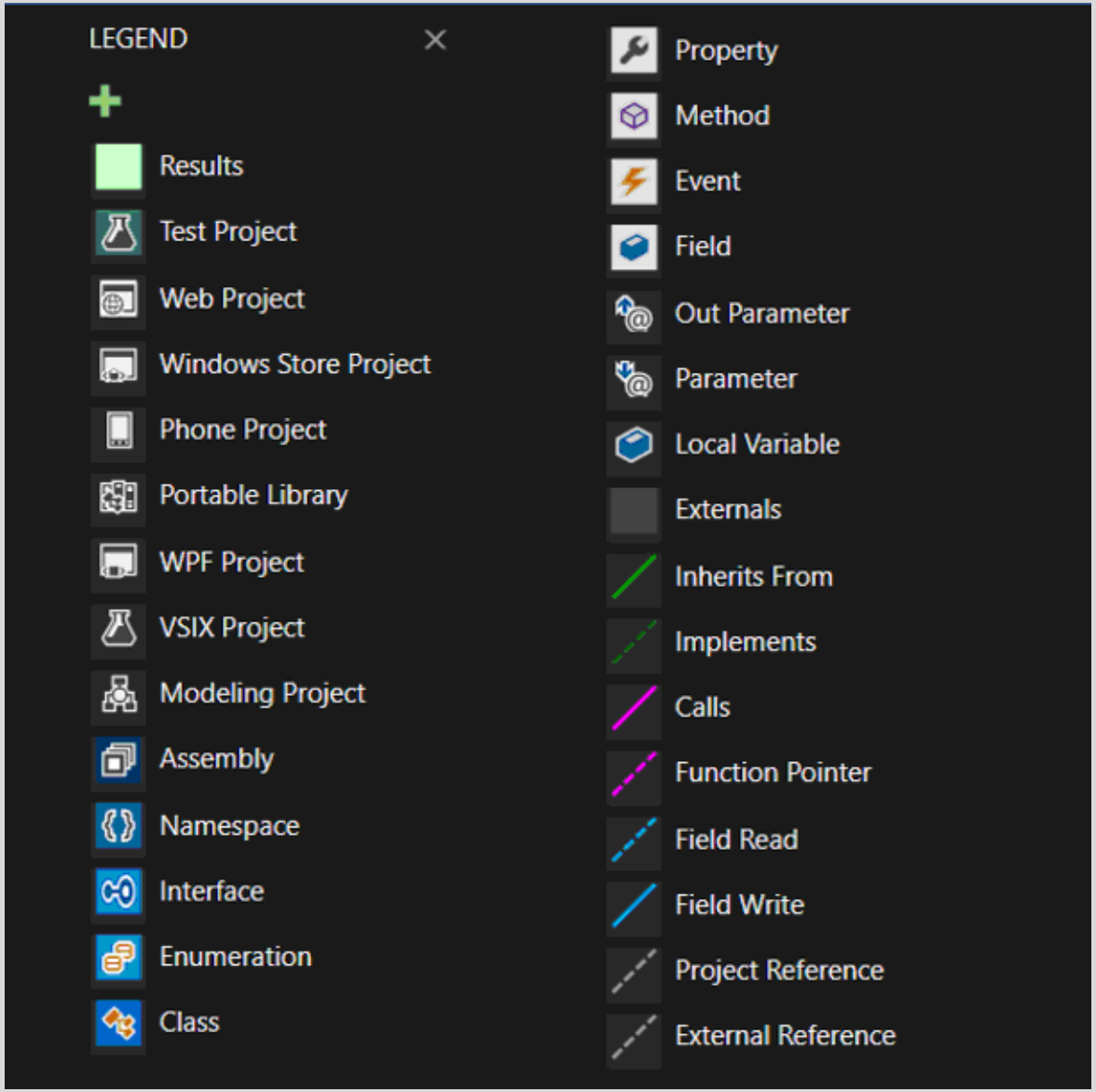
Odtwarzacz muzyczny na komputer PC z systemem Windows dla jednego użytkownika działający jako aplikacja Universal Windows Platform:

- obsługa podstawowych formatów plików dźwiękowych (przynajmniej WAV i MP3);
- playlisty odtwarzanych plików z możliwością edycji (dodawanie, usuwanie, zmiana kolejności);
- zapis playlist i ich eksport (do formatu XML i/lub JSON);
- biblioteka utworów z prezentacją w widokach względem nazwy artysty, jego albumów i ścieżek w albumie;
- możliwość sortowania widoków biblioteki przynajmniej po nazwie, roku wydania, długości ścieżki;
- grupowanie utworów w albumy (jeden utwór może znajdować się w kilku albumach);
- każdy utwór ma mieć przypisany dokładnie jeden gatunek (lista gatunków jest ustalana przez użytkownika, tzn. użytkownik dodaje, usuwa i edytuje dostępne gatunki);
- każdy utwór ma mieć możliwość dodania własnej grafiki, jeśli jej nie ma wyświetlana jest domyślna grafika zapisana w aplikacji lub wyświetlana jest okładka albumu o ile istnieje;
- każdy utwór w albumie ma przypisany numer ścieżki;
- przypisywanie albumu do artysty (artysta może posiadać wiele albumów);
- przypisywanie artysty do zespołu (artysta może być tylko w jednym zespole);
- album, utwór oraz artysta mogą posiadać dokładnie jedno zdjęcie/okładkę;
- oprócz playlist powinna istnieć również kolejka odtwarzania zawierająca wszystkie utwory do odtworzenia;
- użytkownik ma mieć możliwość dodania albumu/playlisty lub pojedynczego utworu do kolejki odtwarzania.

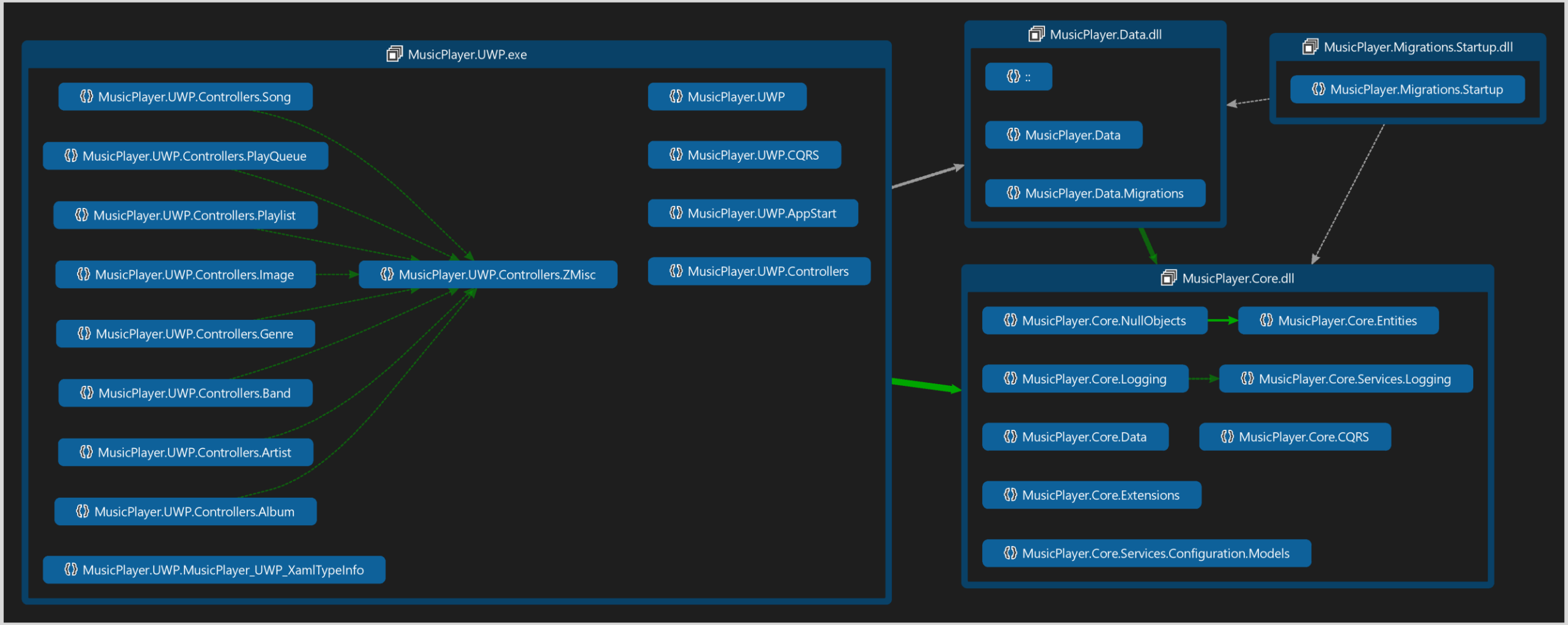
Wykorzystane technologie



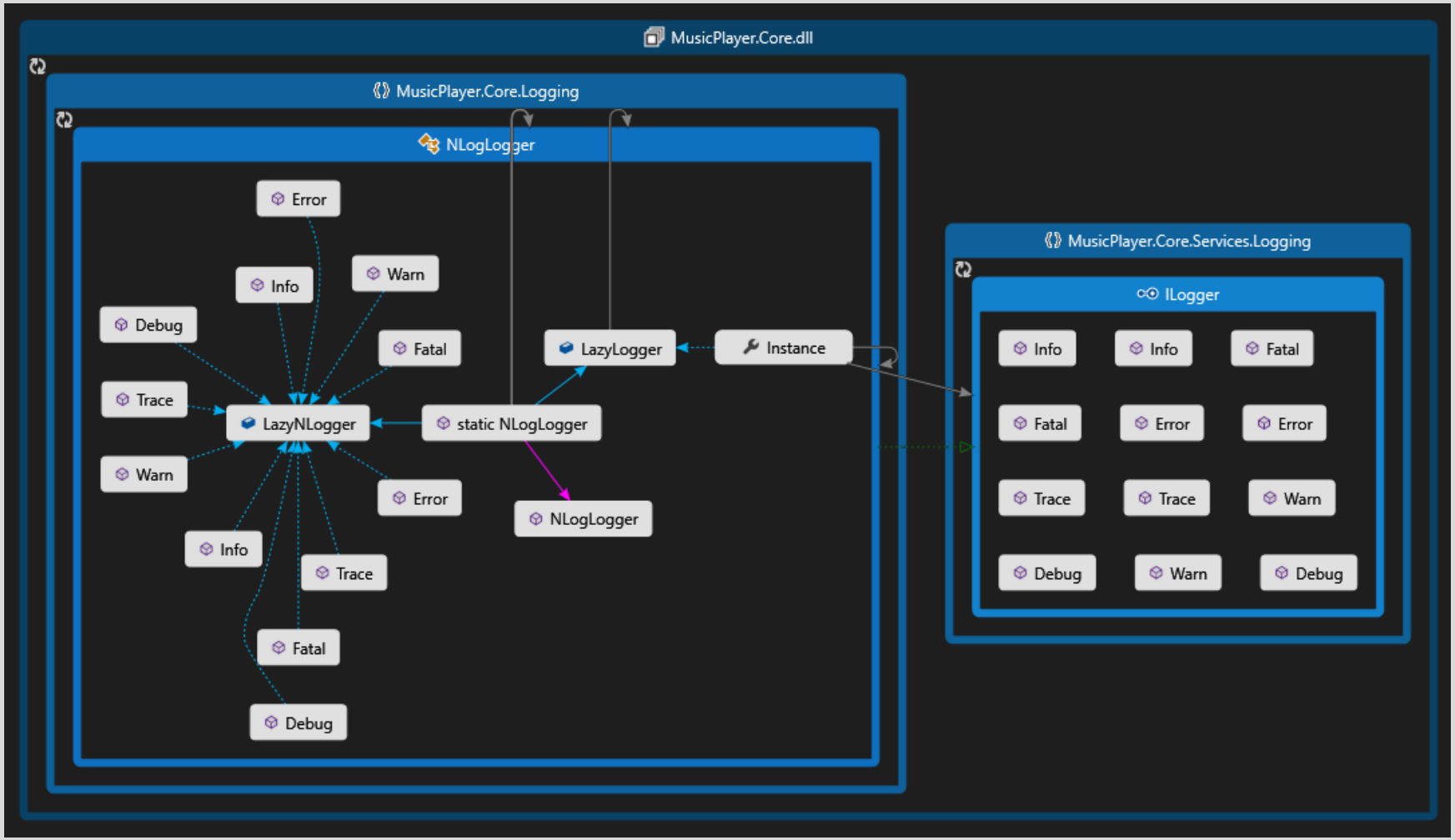
LEGENDA OZNACZEŃ NA DIAGRAMACH



Zależności stworzonych bibliotek w projekcie

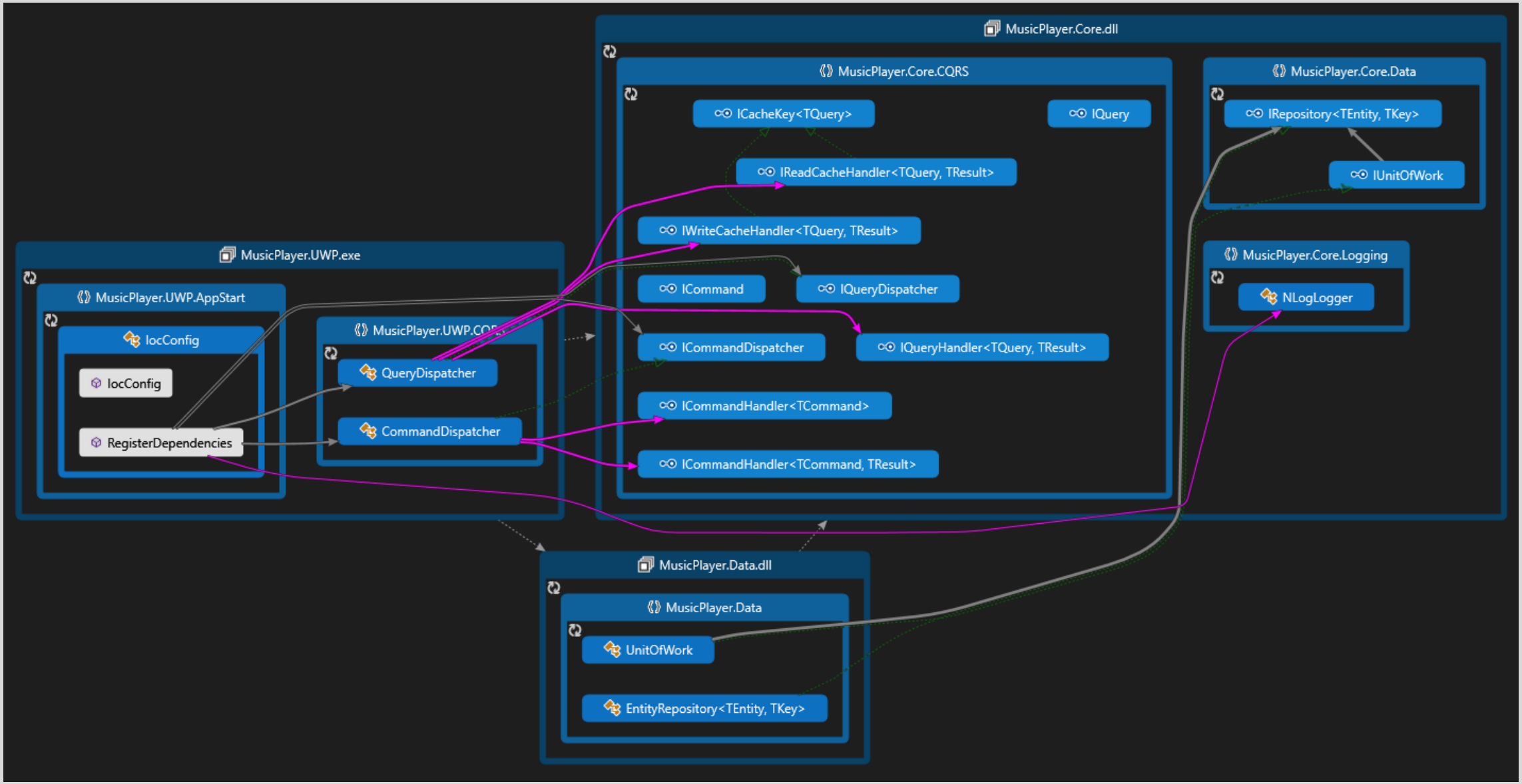


Wzorzec #1 (kreacyjny): Singleton



Singleton – kreacyjny wzorzec projektowy, użyty w celu jest ograniczenia możliwości tworzenia obiektów klasy NLogLogger do jednej instancji. NLogLogger jest klasą służąco do rejestrowania w bazie zachowań, operacji i ich rezultatów.

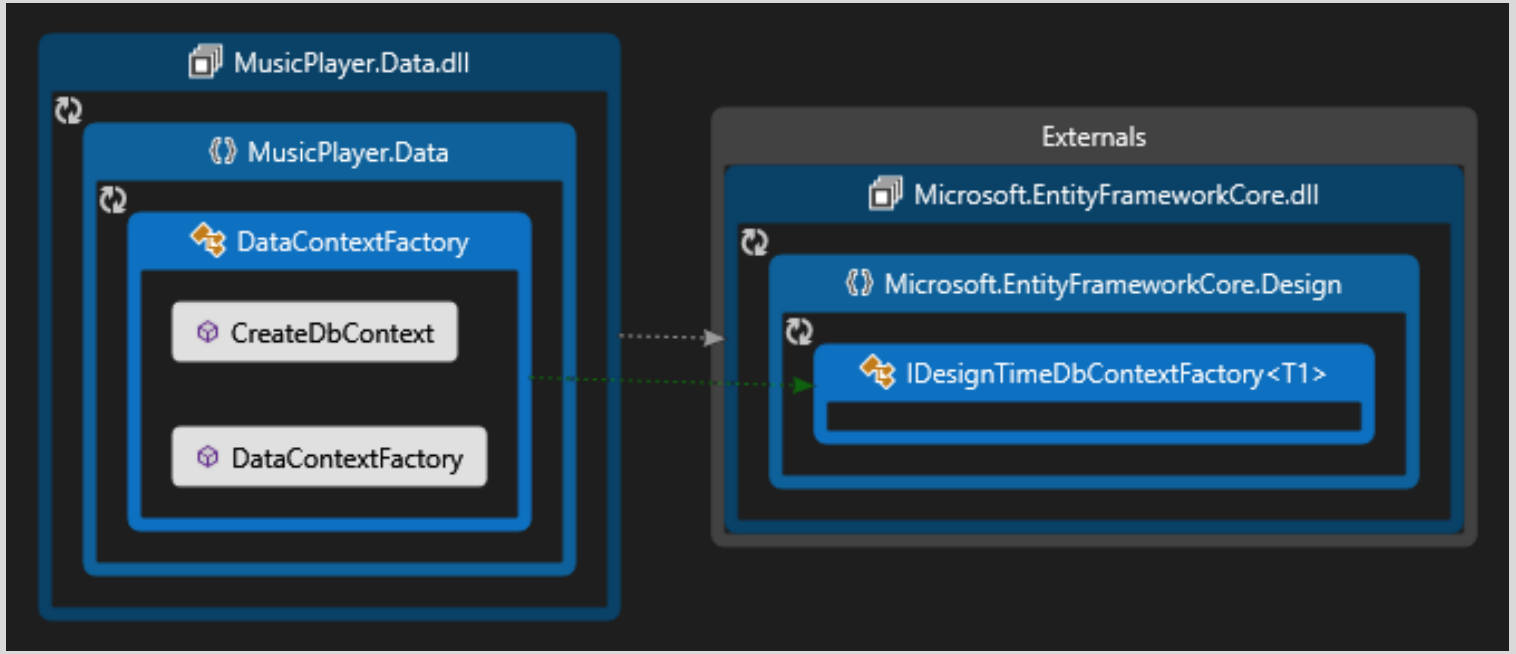
Wzorzec #2 (kreacyjny): Dependency injection



Wzorzec #2 (kreacyjny): Dependency injection

Dependency Injection (Wstrzykiwanie zależności) – wzorzec projektowy i wzorzec architektury oprogramowania zastosowany w projekcie w celu usunięcia bezpośrednich zależności pomiędzy komponentami na rzecz architektury typu plug-in. Polegać on będzie na przekazywaniu utworzonych instancji obiektów udostępniających swoje metody i właściwości obiektom, które z nich korzystają (np. jako parametry konstruktora). Wzorzec ten stanowi alternatywę do podejścia, gdzie obiekty tworzą instancję obiektów, z których korzystają np. we własnym konstruktorze. Dzięki takiemu podejściu kod tworzonej aplikacji opartej na Entitiy Framework będzie prostszy, bardziej zrozumiały i łatwiejszy do testowania.

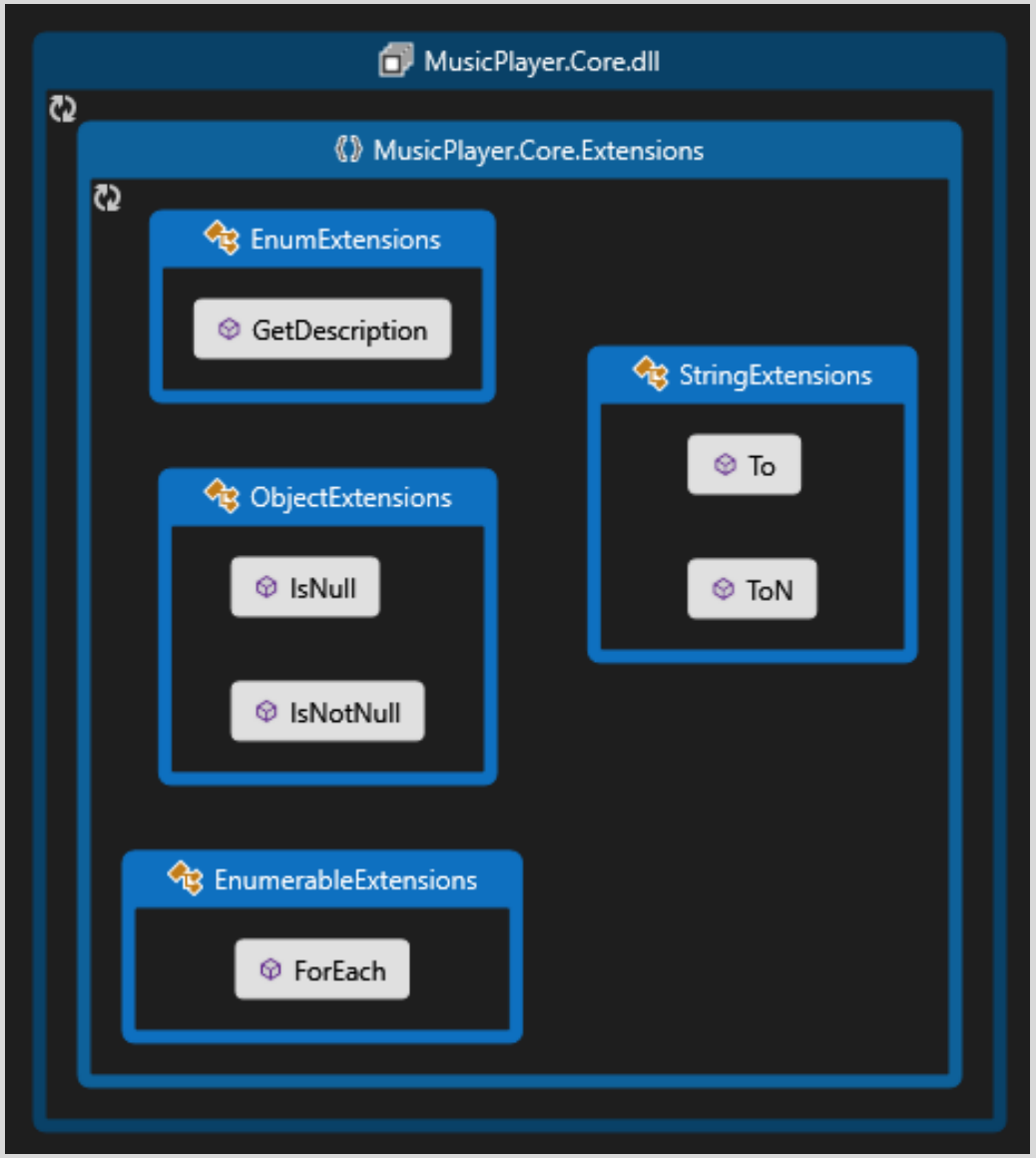
Wzorzec #3 (kreacyjny): Factory



Niektóre polecenia Entitiy Framework Core Tools (na przykład migracje poleceń) wymagają pochodnej DbContext wystąpienia, które ma zostać utworzone w czasie projektowania, aby można było zbierać szczegółowe informacje o aplikacji typy jednostek i sposobu mapowania ich na schemat bazy danych.

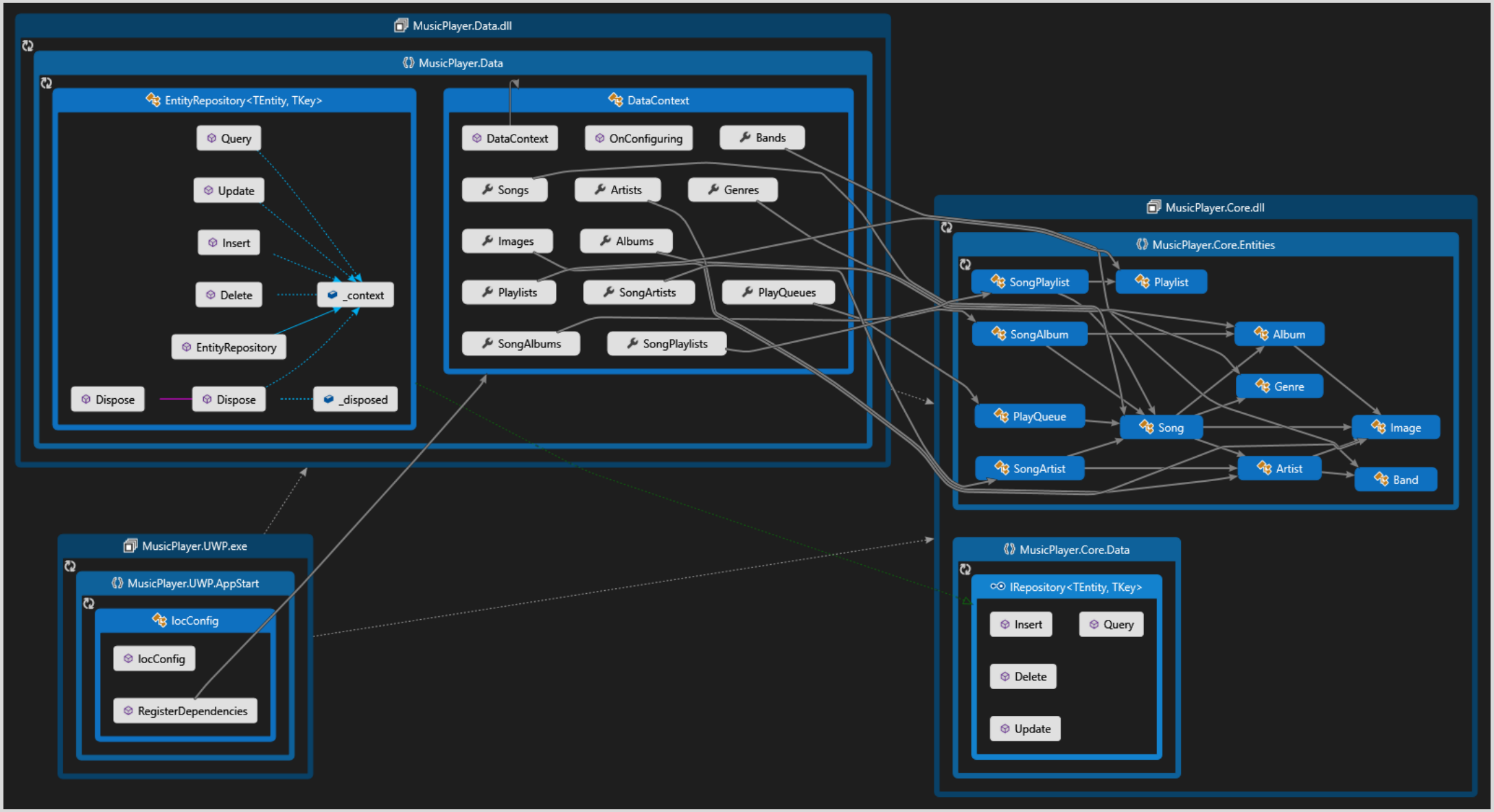
Narzędzie EF Core Tools można poinformować o sposobie tworzenia usługi DbContext implementując interfejs IDesignTimeDbContextFactory<TContext>. Podsumowując wzorzez Factory (Fabryki) użyty zostanie w celu realizacji połączenia z bazą podczas tworzenia migracji.

Wzorzec #4 (strukturalny): Extension Object



Wzorzec Extension Object został użyty w celu rozszerzenia istniejących klas, np. Object o dodatkowe funkcjonalności (metody). Zwykle klasa jest rozszerzana poprzez podklasę i dodawanie metod do klasy pochodnej. Obiekt rozszerzenia zapewnia rozszerzalność bez podklas.

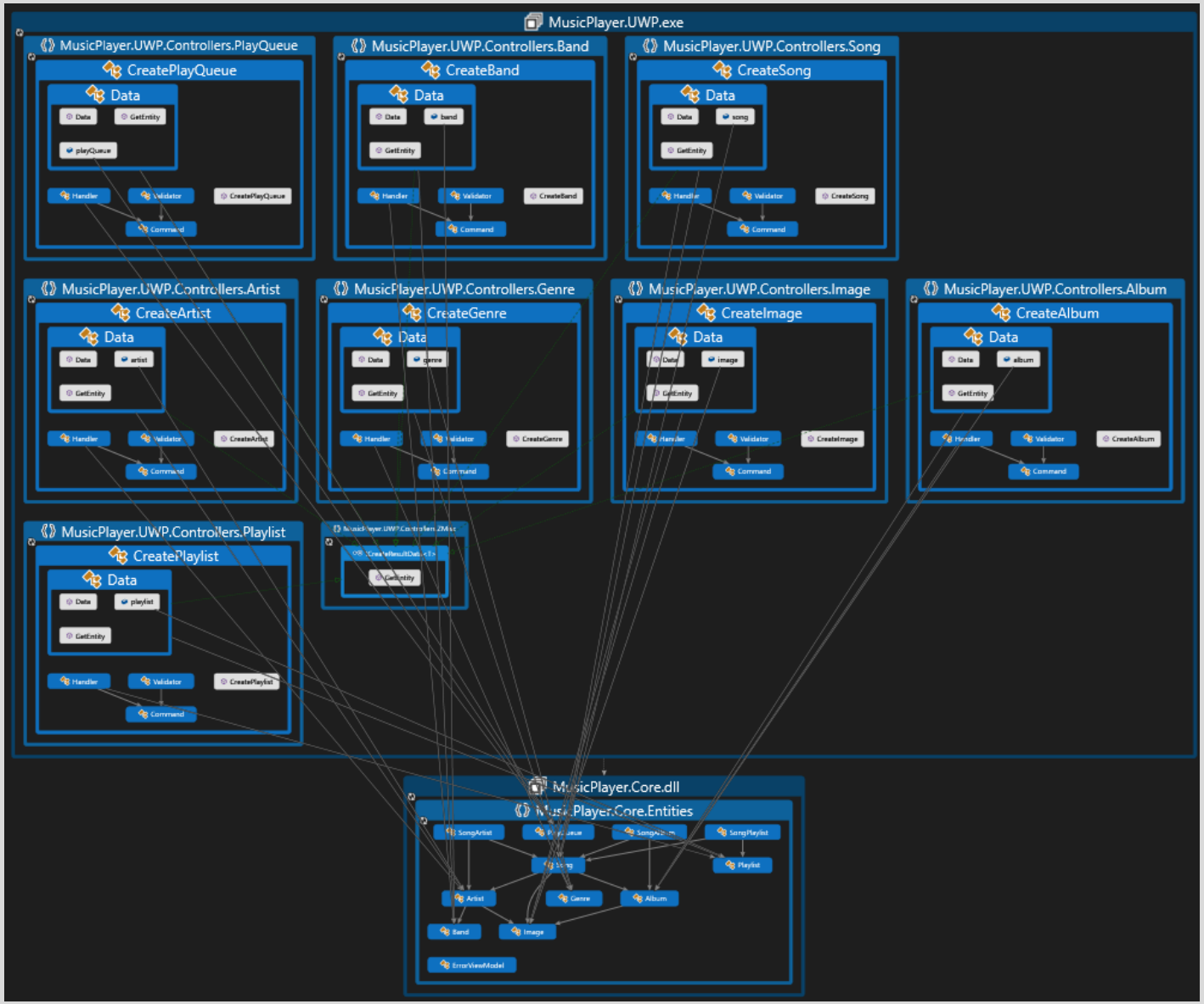
Wzorzec #5 (strukturalny): Repository



Wzorzec #5 (strukturalny): Repository

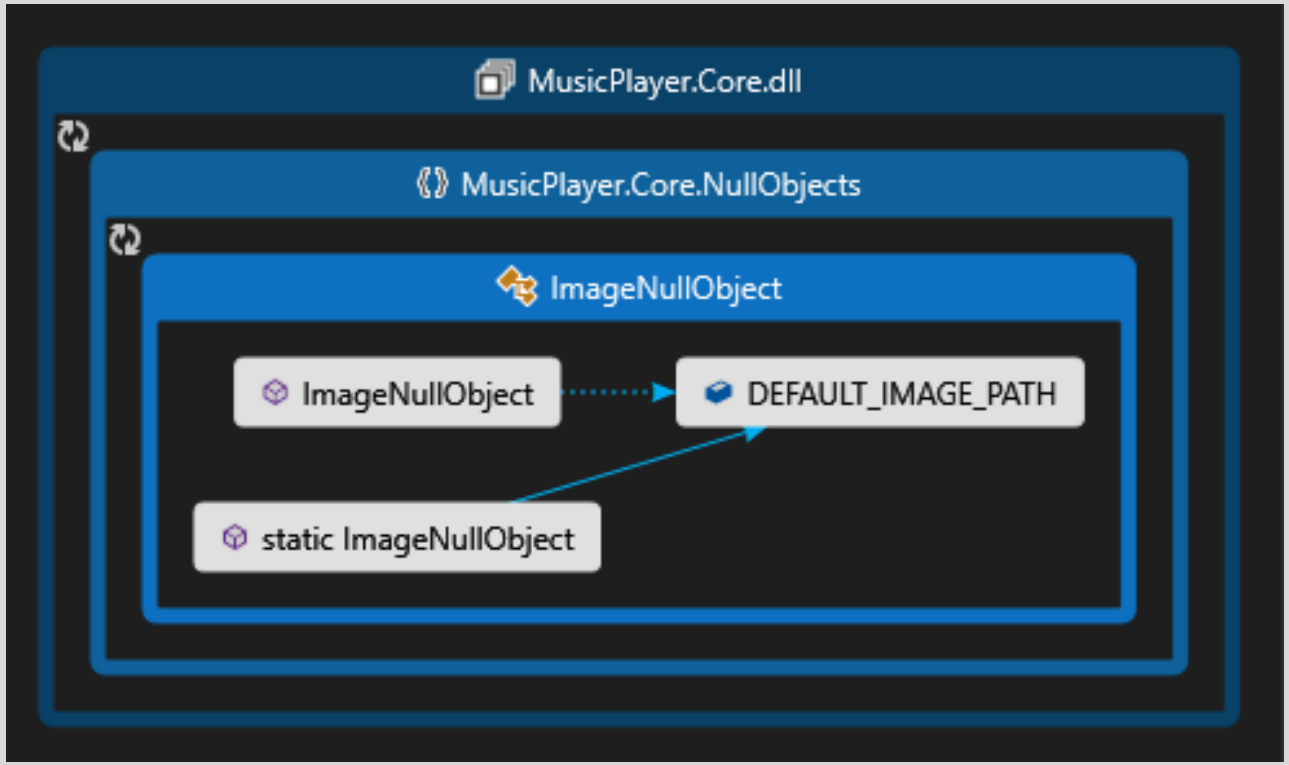
Repozytoria to klasy lub komponenty, które zawierając będą logikę wymaganą do uzyskania dostępu do źródeł danych. Scentralizują wspólną funkcjonalność dostępu do danych, zapewniając lepszą konserwowalność i oddzielenie infrastruktury lub technologii wykorzystywanej do uzyskiwania dostępu do baz danych z warstwy modelu domeny. Z uwagi na zastosowanie Mapowania Obiektowego Relacji (ORM), jakim jest Entity Framework, kod, który musi zostać zaimplementowany, jest uproszczony, dzięki LINQ i silnemu pisaniu. Pozwala to skoncentrować się na logice trwałości danych, a nie na dostępie do danych.

Wzorzec #6 (strukturalny): Private class data



Wzorzec Private class data rozwiązuje w projekcie problem jaki klasa może mieć, dotyczący ochrony stanu obiektu, w którym nie można zadeklarować finału. Działanie tego wzorca polega na usunięciu ekspozycji danych przez zabezpieczenie jej w klasie utrzymującej stan danych. W rezultacie oddzielamy dane od tych metod, które go używają, a tym samym tworzymy kolejną warstwę separacji od kontrolerów, którzy mają relacje danymi tworzonego obiektu. W skrócie, wzorzec umożliwi w kontrolerach inicjowanie danych należących do klas modeli.

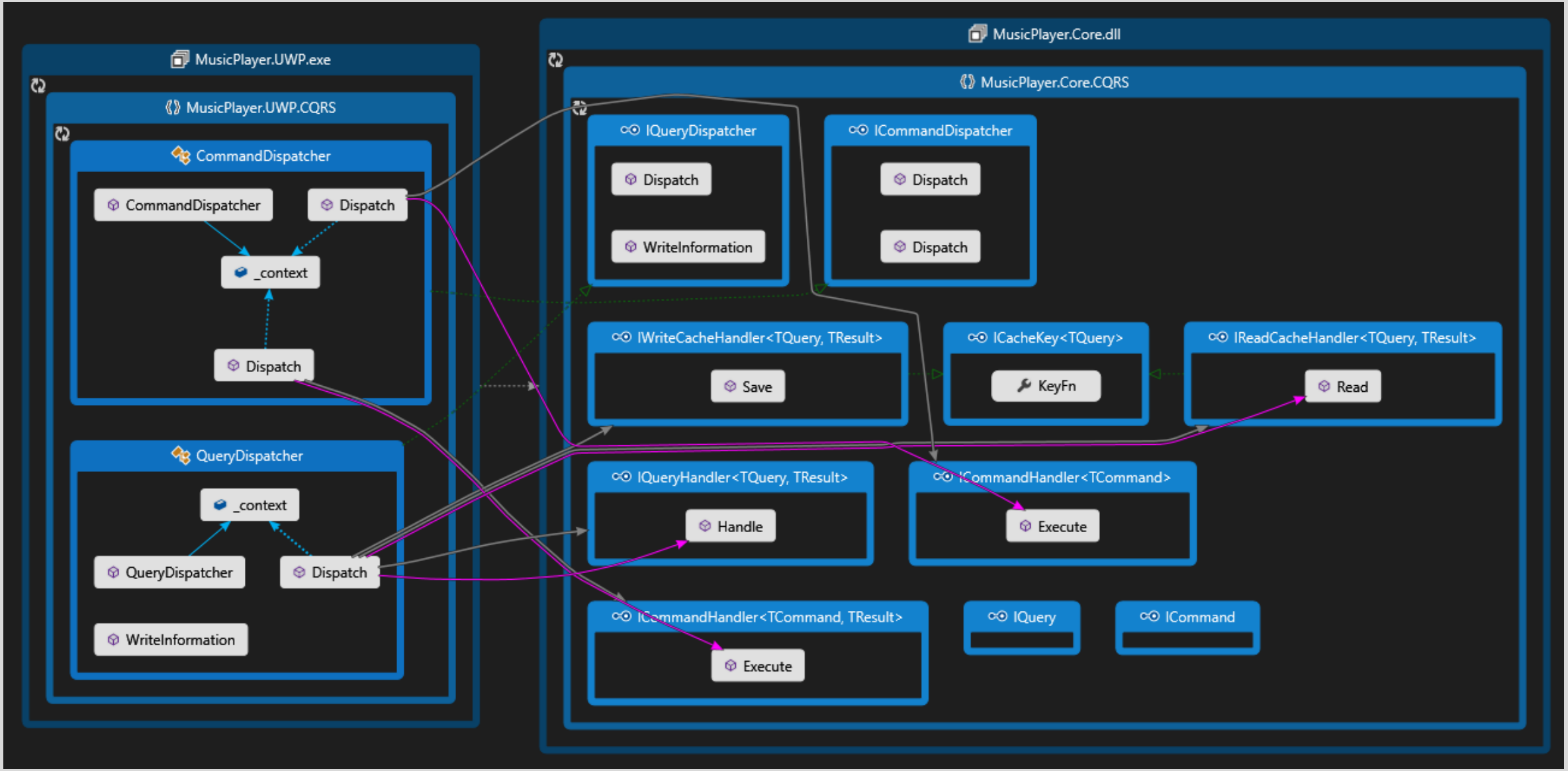
Wzorzec #7 (czynnościowy): Null Object



Null Object (Pusty obiekt) to czynnościowy wzorzec projektowy, użyty w celu realizacji braku obiektu – Zdjecia poprzez dostarczenie alternatywy, która oferuje domyślnie działanie puste, czyli niewykonujące żadnych operacji.

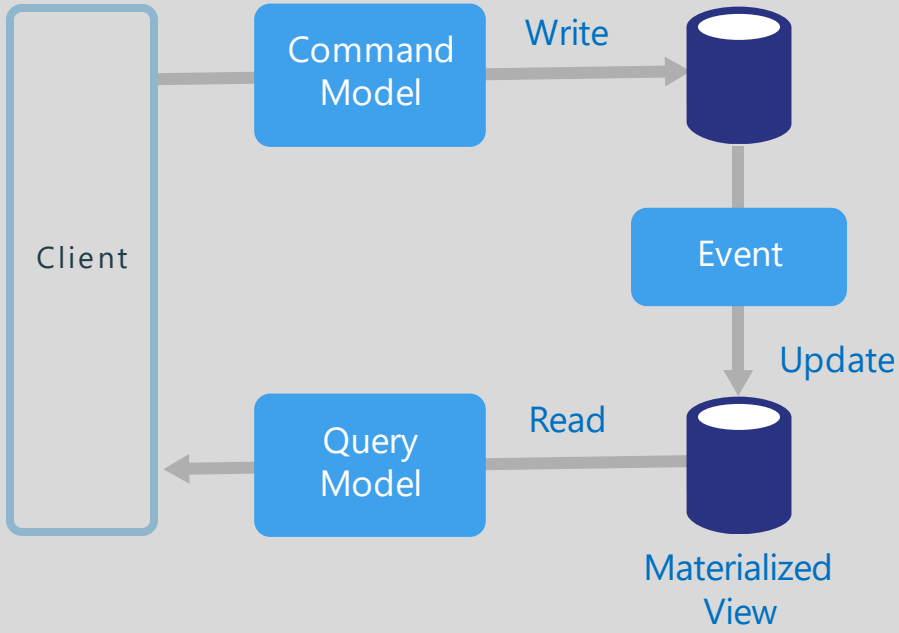
Wzorzec pusty obiekt umożliwi uniknięcie sprawdzenia, czy wartość jest różna od null przy zachowaniu zasad pełnej obiektowości (polimorfizm, abstrakcja, enkapsulacja).

Wzorzec #8 (czynnościowy): CQRS



Wzorzec #8 (czynnościowy): CQRS

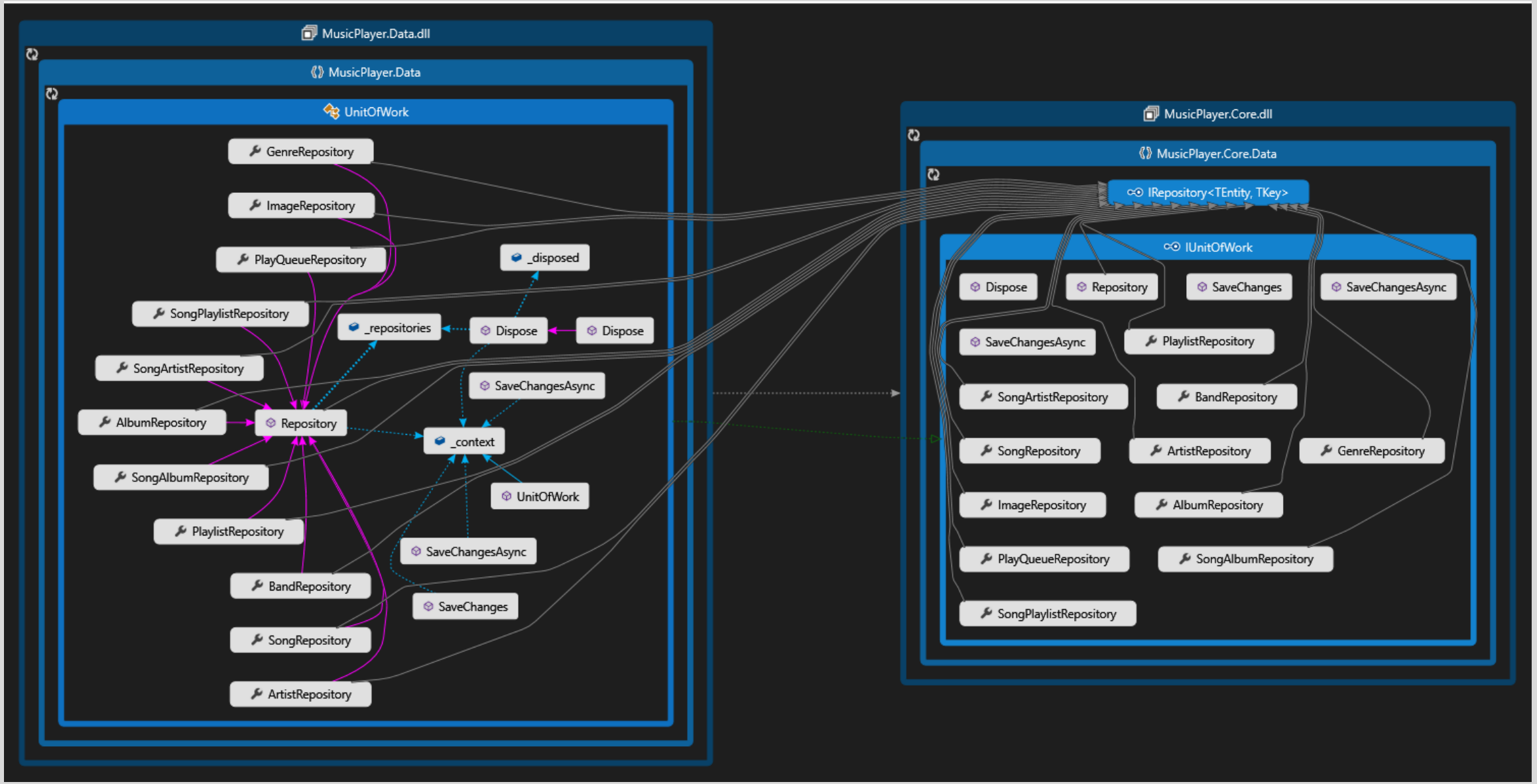
W tradycyjnych architekturach ten sam model danych jest używany do wysyłania zapytań do bazy danych i aktualizowania jej. Jest to proste i dobrze się sprawdza w przypadku podstawowych operacji CRUD. Jednak w naszej aplikacji, która jest bardziej złożona metoda została uznana za niewygodną i nieczytelną. Wówczas, mapowanie obiektu stało by się skomplikowane. Po stronie zapisu model może wdrażać złożoną walidację i logikę biznesową. W efekcie można uzyskać zbyt skomplikowany model, który wykonuje zbyt dużo działań.



Podejście CQRS rozwiązuje wymienione wyżej problemy, rozdzielając odczyty i zapisy na osobne modele przy użyciu poleceń do aktualizacji danych i zapytań do odczytu danych.

- Polecenia będą oparte na zadaniach, a nie skoncentrowane na danych. Dzięki temu będą one być umieszczane w kolejce do przetworzenia asynchronicznego, a nie przetwarzane synchronicznie.
- Zapytania nigdy nie modyfikują bazy danych. Zapytanie zwraca obiekt DTO, który nie hermetyzuje żadnej wiedzy domeny.

Wzorzec #9: Unit of Work

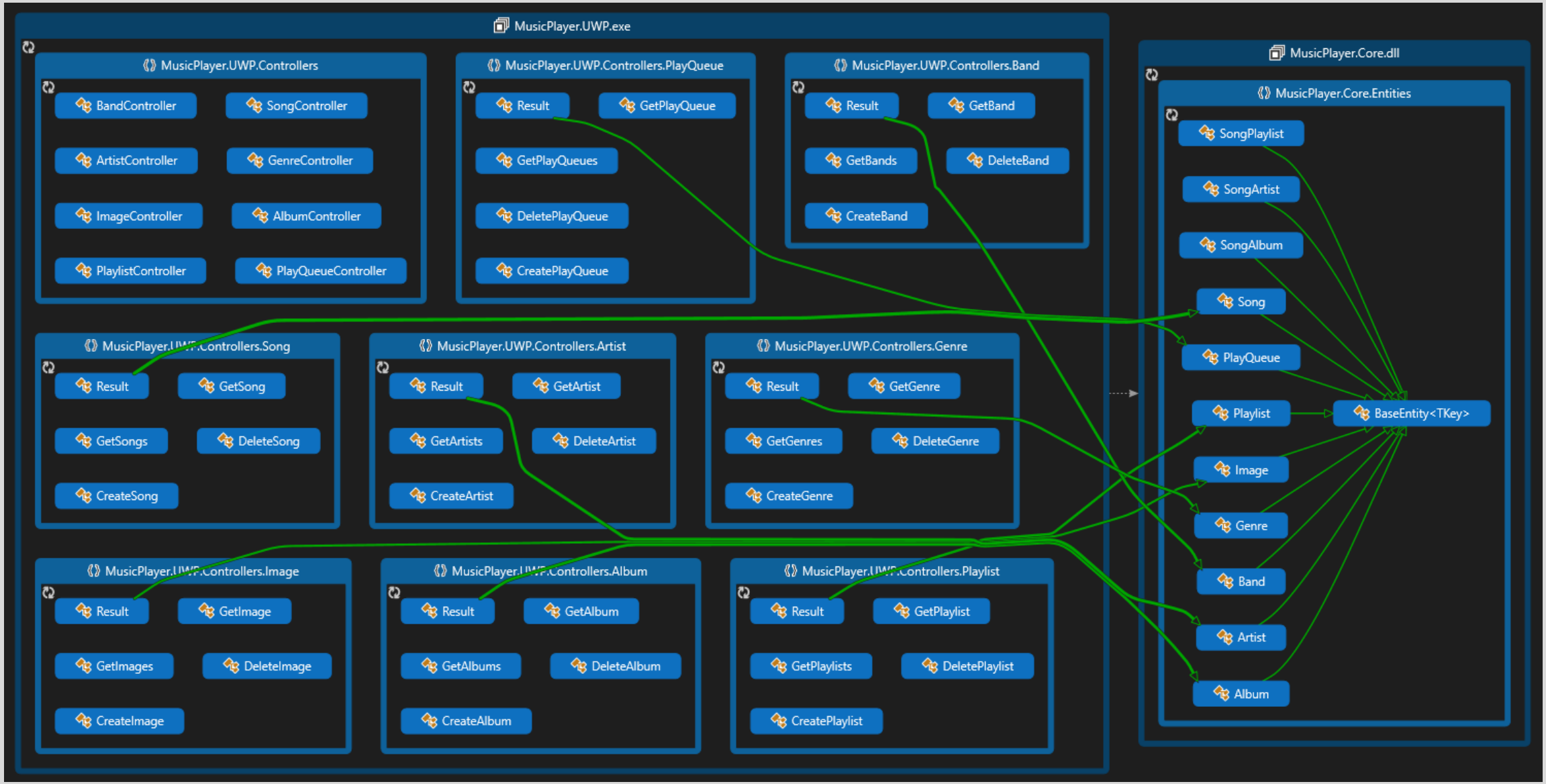


Wzorzec #9: Unit of Work

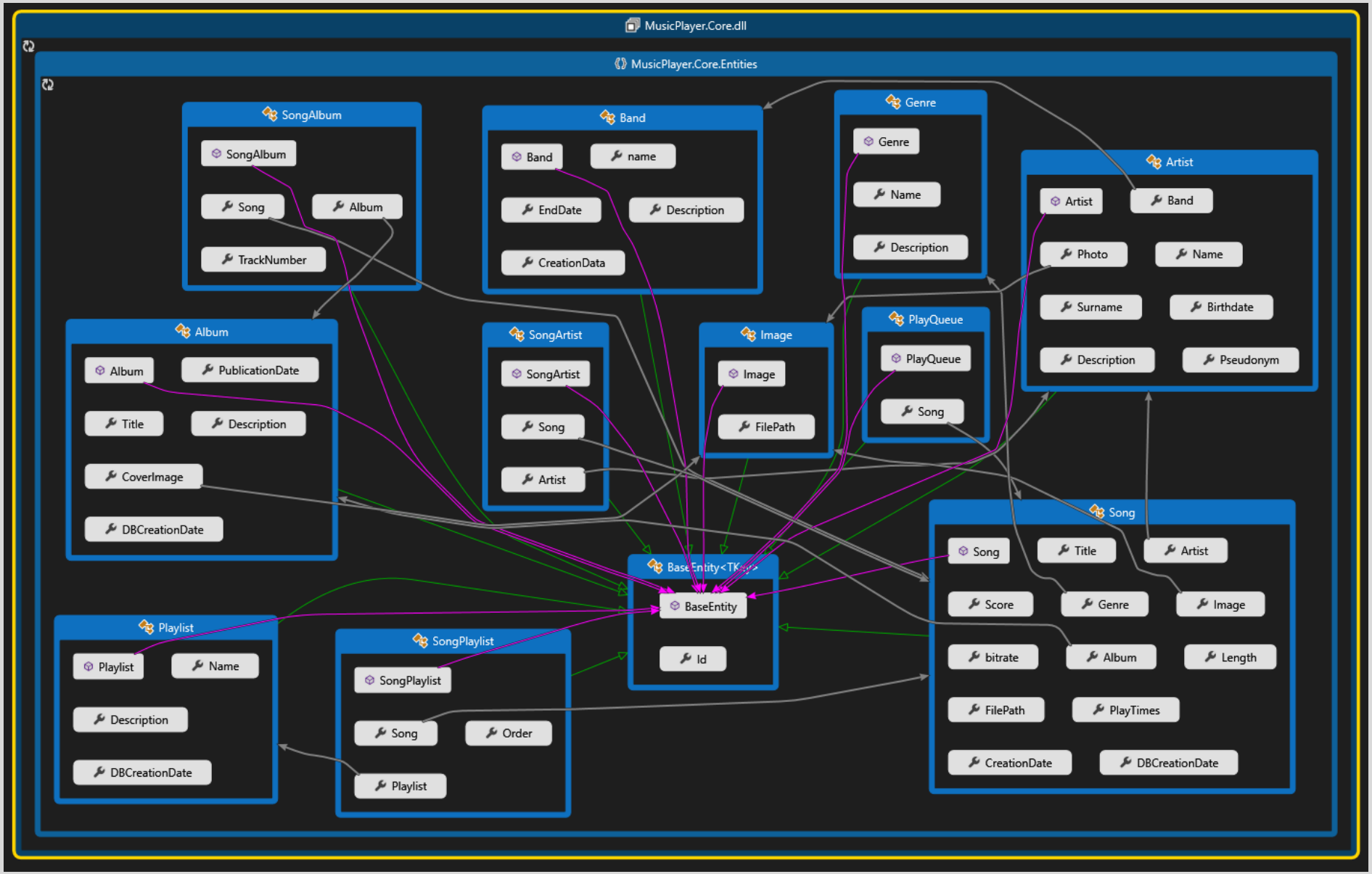
Unit of Work jest określana jako pojedyncza transakcja, która obejmuje wiele operacji wstawiania, aktualizowania lub usuwania. Mówiąc najprościej, oznacza to, że dla określonego działania użytkownika, takiego jak rejestracja na stronie internetowej, wszystkie operacje wstawiania, aktualizacji i usuwania są obsługiwane w ramach pojedynczej transakcji. Jest to bardziej wydajne niż obsługa wielu transakcji baz danych w sposób oparty na sieci.

Te wielokrotne operacje utrwalania wykonywane są później w pojedynczej akcji, gdy kod z warstwy aplikacji zarządza nimi wydając polecenia. Decyzja o wprowadzeniu zmian w pamięci do rzeczywistej bazy danych jest zwykle oparta na schemacie Unit of Work.

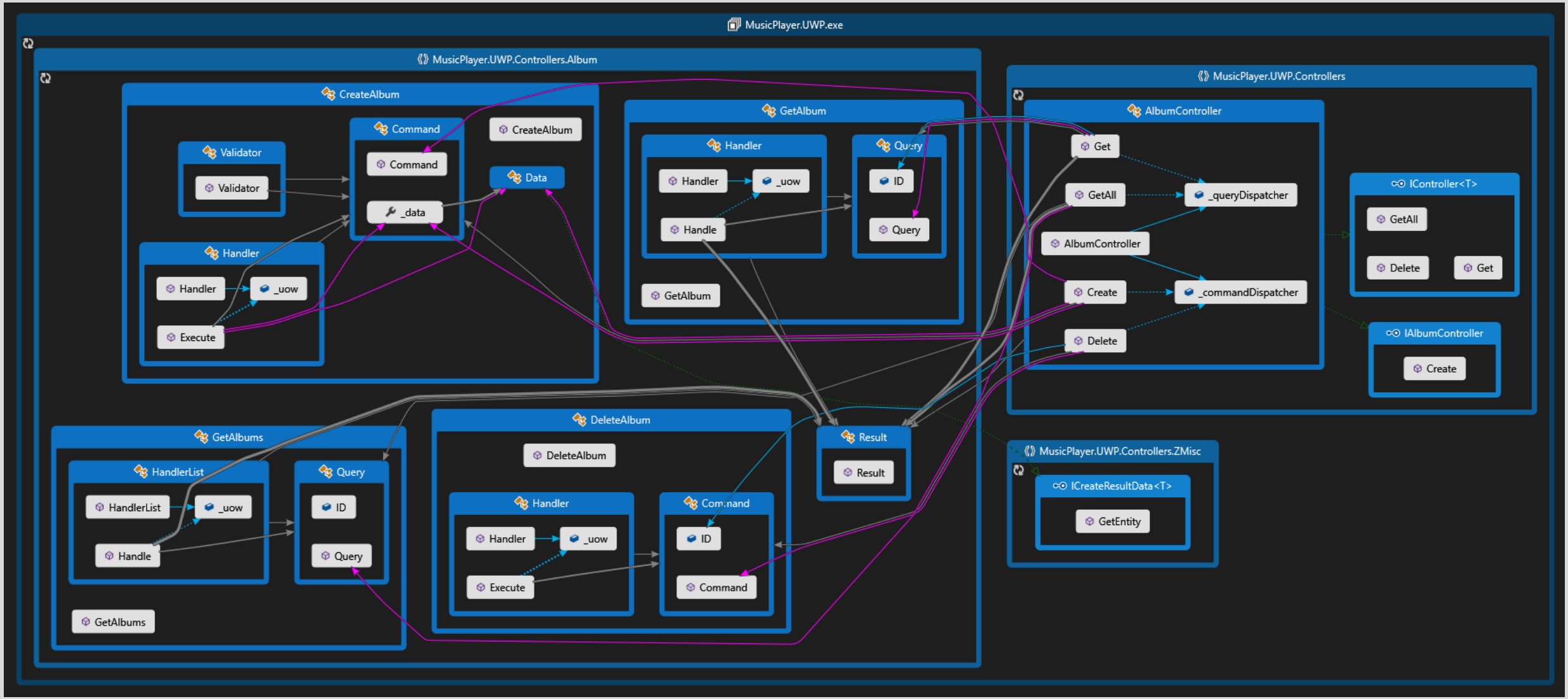
Wzorzec #10 (architekturalny): MVP - struktura



Wzorzec # 10 (architekturalny): MVP - Modele

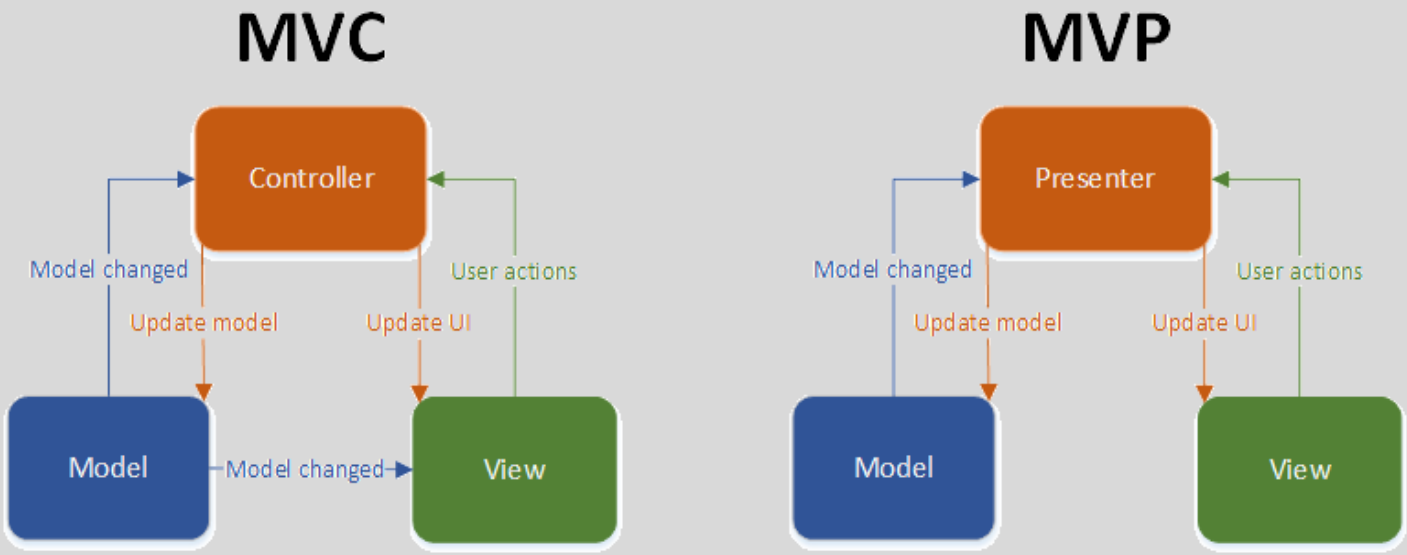


Wzorzec # 10 (architekturalny): MVP – Przykładowy kontroler / prezenter Albumu



Wzorzec #10 (architekturalny): MVP

Model-View-Presenter to wzorzec powstały na bazie wzorca MVC (Model-View-Controller). We wzorcu MVP prezenter jest tym samym, czym kontroler we wzorcu MVC z jedną mała różnicą, w prezenterze zawiera się logika biznesowa.



Dane nie są przekazywane bezpośrednio z modelu do widoku jak to ma miejsce w MVC. Prezener wysyła zapytanie do modelu, model zwraca dane do prezentera, prezenter przetwarza otrzymane dane i przekazuje do widoku.

W projekcie na chwilę obecną zastosowano nazewnictwo Prezentera jako Kontroler.

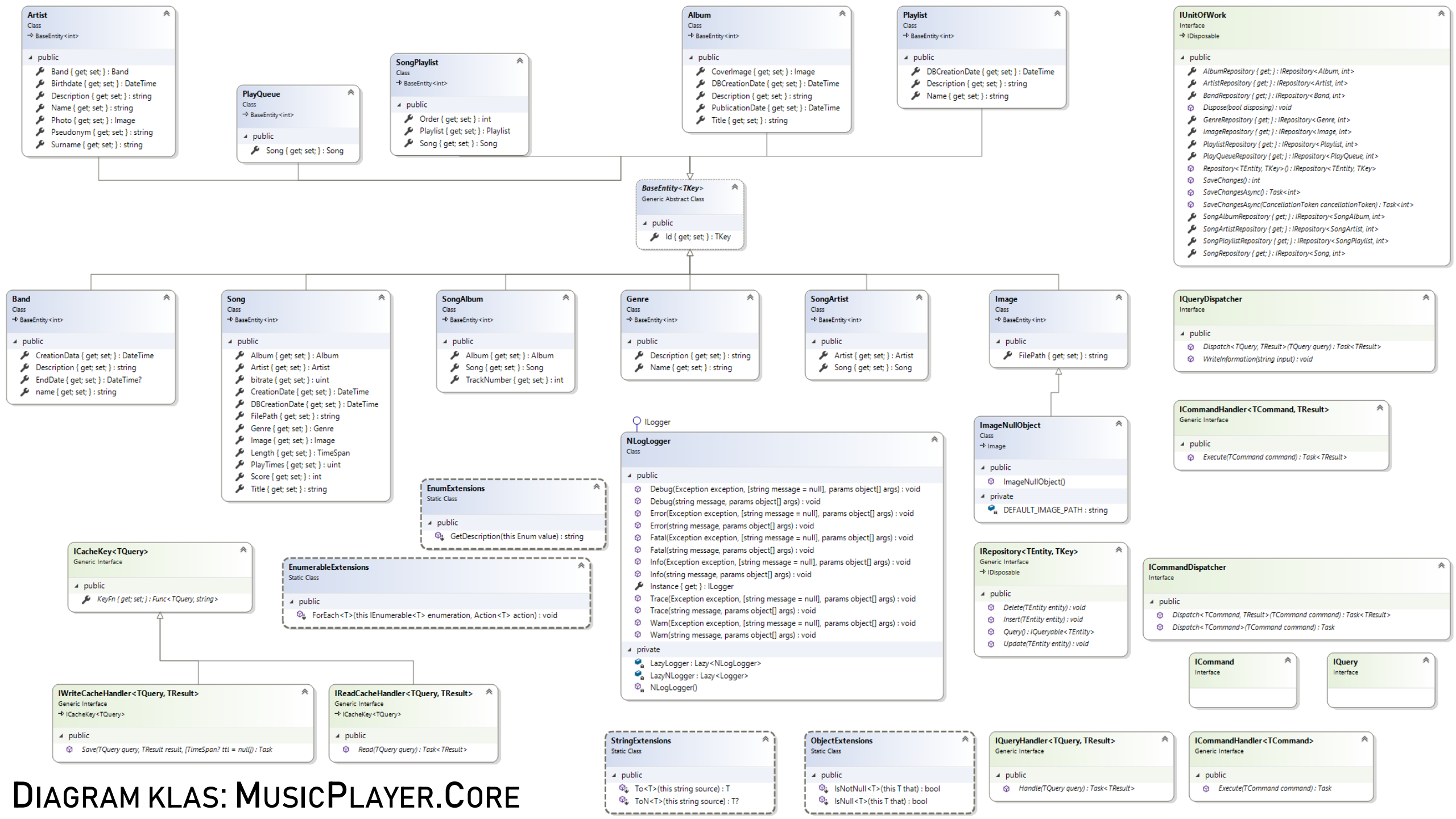


DIAGRAM KLAS: MUSICPLAYER.CORE

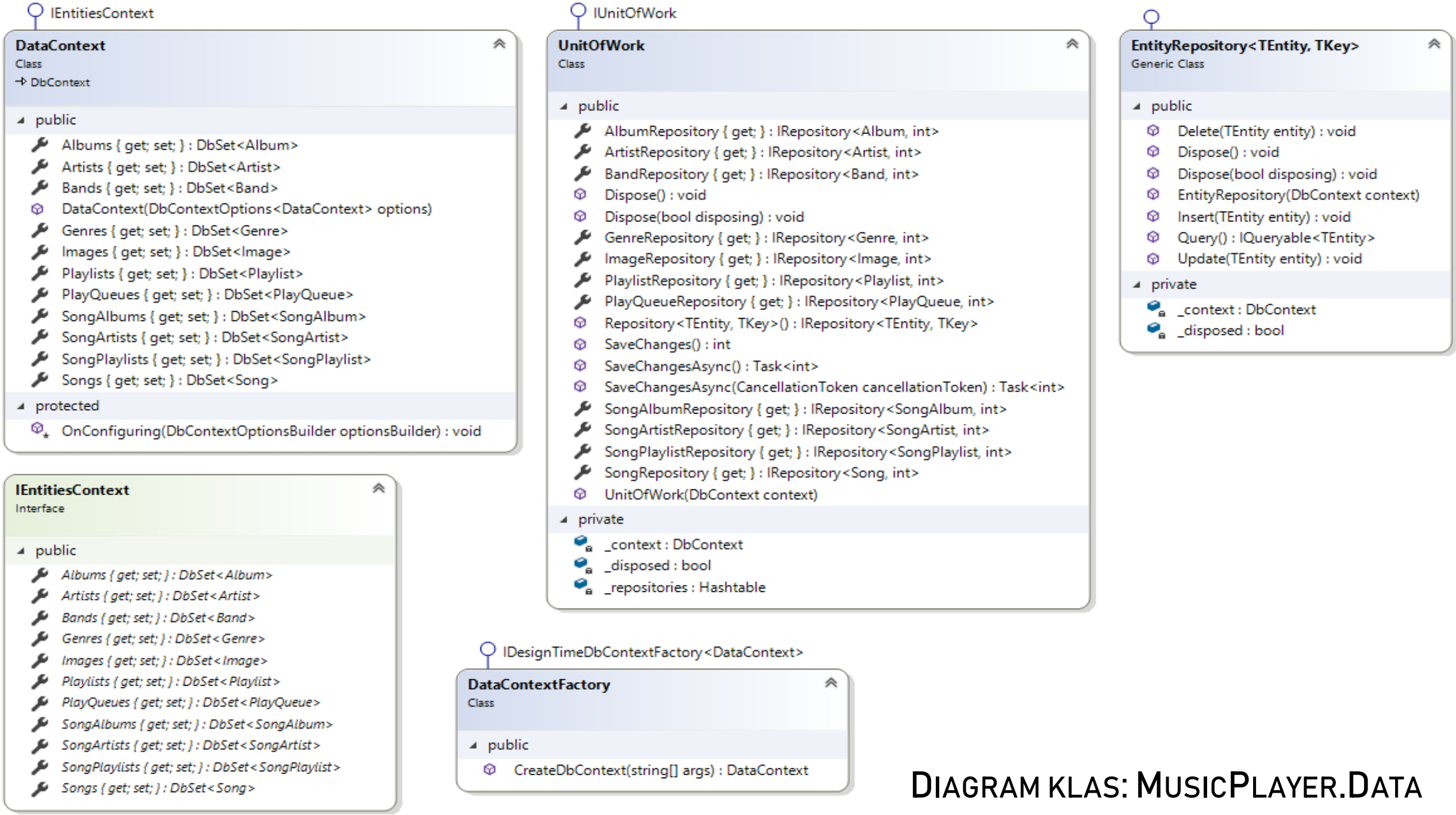


Diagram klas: MusicPlayer.Data

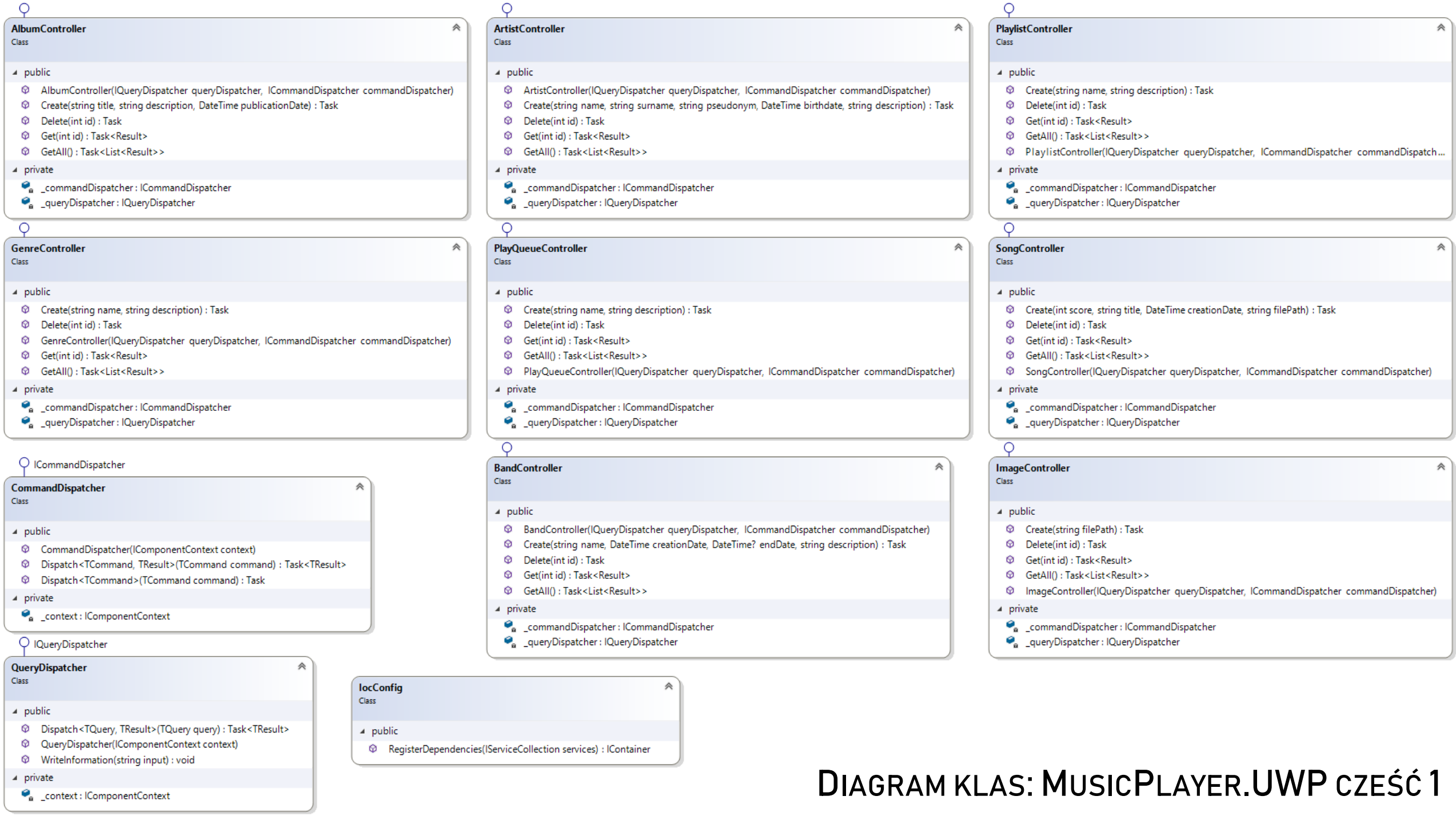


Diagram klas: MusicPlayer.UWP część 1

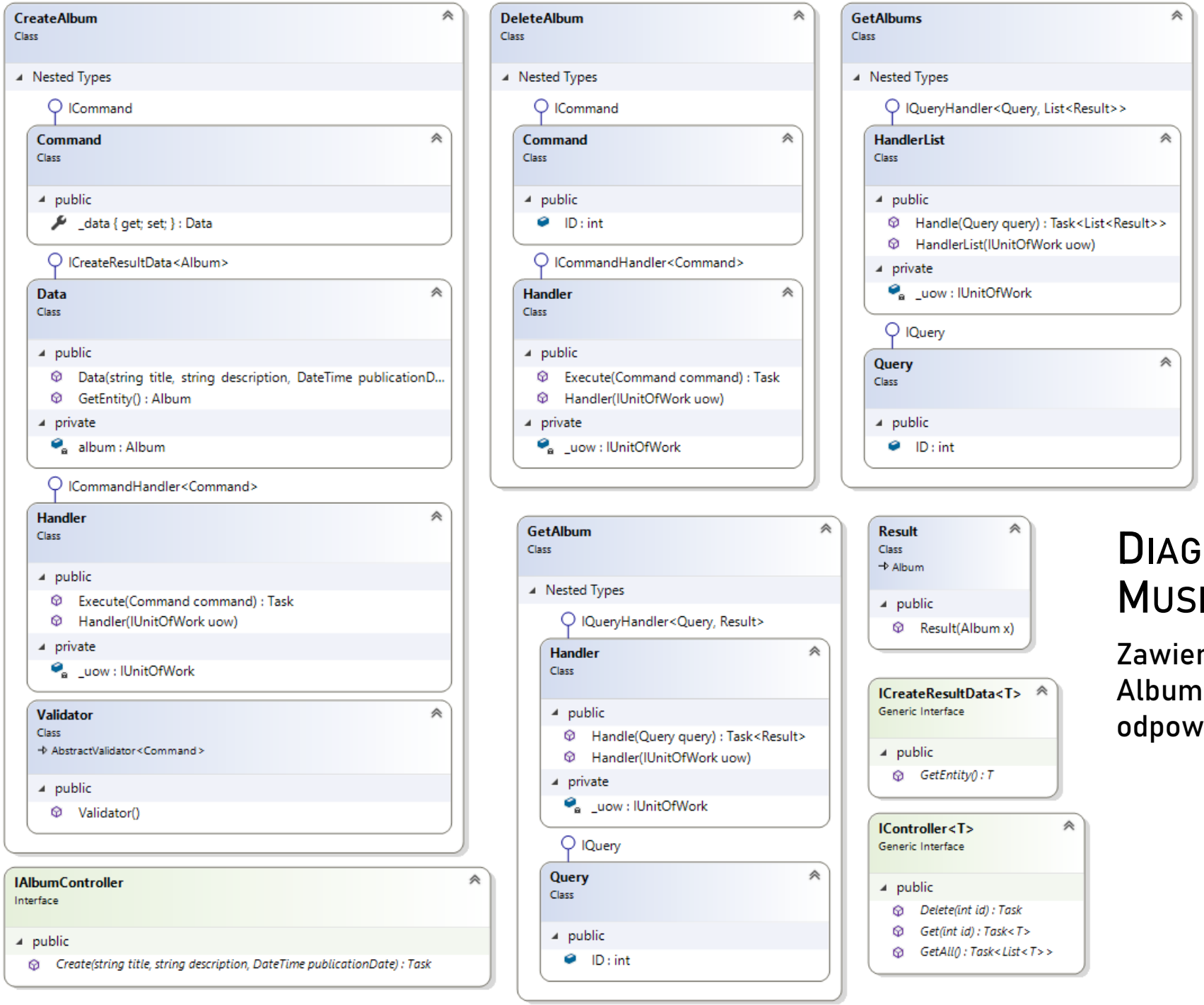


Diagram klas: MusicPlayer.UWP część 2

Zawierający interfejsy i Klasy związane z Albumem. Pozostałym modelom bazy odpowiadają zbliżone kontrolery.

Statystyki:
Łączna liczba klas w projekcie > 168
Łączna liczba interfejsów w projekcie > 24