**Źródło danych aplikacji YourHikes**

**Autor: MateuszRzewnicki**

**Grupa: Z712**

**Nr indeksu: 8138**

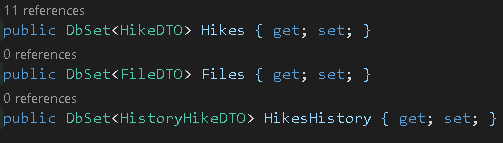
Wstęp

W projektowaniu i implementacji źródła danych wykorzystano podejście „Code First” które wykorzystując rozszerzenie *EntityFrameworkCore* umożliwia generowanie kodu bazy danych z poziomu kodu źródłowego programu gdzie modelujemy tabele za pomocą klas. Ponadto implementuje system kontroli wersji dla bazy danych. Tabela kontroli wersji znajduje się w bazie danych ale nie została ona zawarta na poniższym diagramie ponieważ nie realizuje ona projektowanej logiki systemu ani założeń biznesowych.

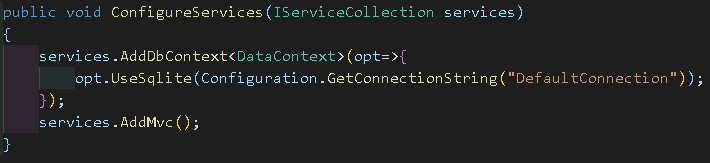
System składa się z trzech głównych tabel ich szczegółowy opis wraz z atrybutami znajduje się w dalszej części dokumentu. Plik z całym wygenerowanym kodem dodany jest do dokumentacji jako załącznik, znaleźć go można w tym samym folderze co czytany właśnie dokument pod nazwą: *hikes.db.sql.* Wykorzystany dostawca to *SQLite* w wariancie lekkiej bazy plikowej.

### Połączenie

Połączenie do bazy danych realizowane jest za pośrednictwem klasy *DataContext* która przechowuje właściwości dostępowe do tabeli o typie *DbSet*.

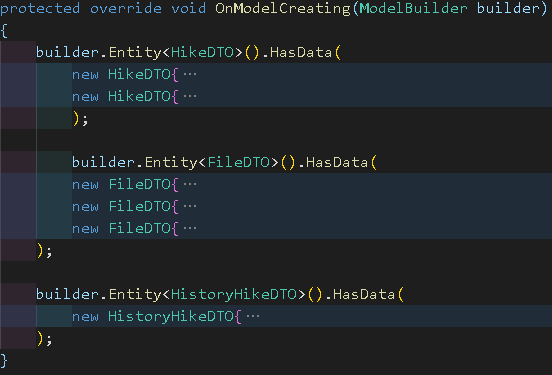


Połączenie inicjalizowane jest na etapie startu aplikacji w klasy *StartUp* na etapie składania konfiguracji usług co umożliwia szersze wykorzystanie użytych migracji do zadbania by aplikacja zawsze pracowała na najnowszej migracji. Co ważne aplikacja sama zadba o stworzenie bazy danych jeśli jej plik został usunięty. Aplikacja stworzy cała strukturę i wstrzyknie dane.



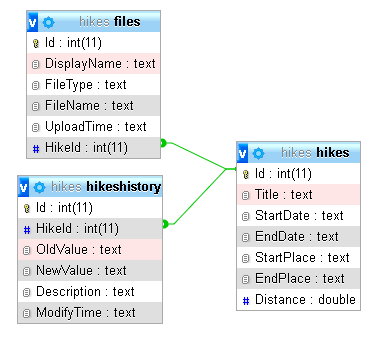
Jeśli aktualna wersje bazy danych różni się od tej kodzie źródłowym aplikacji, wywoływana jest aktualizacja jej struktury.

Ponadto poprzez nadpisanie metody *onModelCreating*, możliwe jest wstrzykiwanie danych wraz ze startem aplikacji:



### Klasy

Diagram bazy danych:



W kodzie źródłowym aplikacji powyższe tabele definiowane są za pomocą klas. W systemie przewidziano dla nich trzy klasy: *FileDTO*, *HikeDTO* i *HistoryHikeDTO*.

### Klasa FileDTO

Reprezentuje ona pliki które będą dołączane do wędrówki. Przewidziana została na rozróżnianie typów plików tak by od jednej tabeli mogły być wstawiane zarówno zdjęcia jak i pliki video. Zawiera ona w sobie 6 pól rzutowanych na tabele w bazie danych oraz jedno pole referencji:

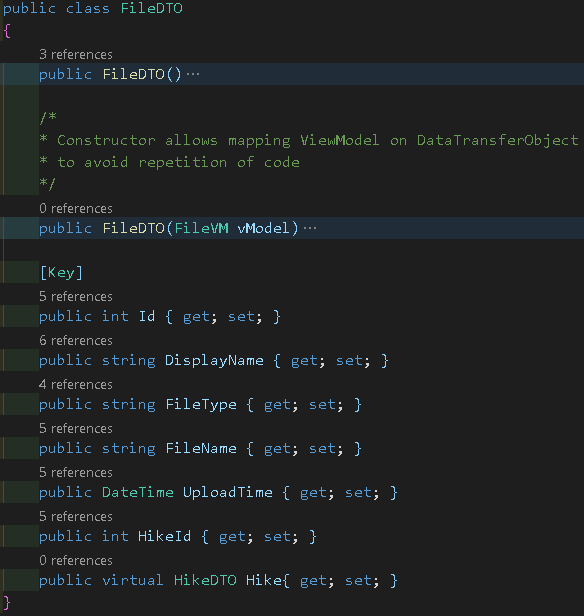
Pola:

1. Id – reprezentuje unikalny klucz rekordu w tabeli o typie liczbowym
2. DisplayName – reprezentuję wyświetlaną nazwę pliku w systemie o typie teksowym
3. FileType – reprezentuje ona typ pliku który przechowuje rekord o typie tekstowym
4. FileName – reprezentuje nazwę pliku która umożliwia odwołanie się do pliku w katalogu wędrówki na serwerze o typie tekstowym
5. UploadTime – reprezentuje czas wgrania pliku do systemu o typie daty i godziny
6. HikeId – reprezentuje powiązany rekord wędrówki o typie liczbowym

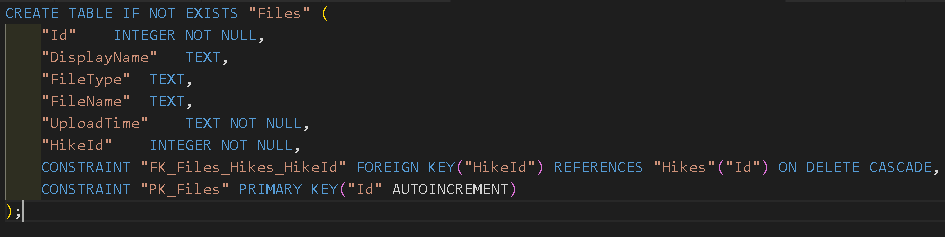
Referencje:

1. HikeDTO – reprezentuje powiązanie między rekordem pliku a wędrówki

Kod klasy FileDTO:



Przykład wygenerowanego kodu dla klasy FileDTO:



### Klasa HikeDTO

Reprezentuje ona wędrówki które będą tworzone w systemie. Przewidziana została tak by możliwe było dodawanie do rekordu wędrówki załączników przez klasę *FileDTO*. Zawiera ona w sobie 7 pól rzutowanych na tabele w bazie danych oraz jedną właściwość reprezentującą powiązanie:

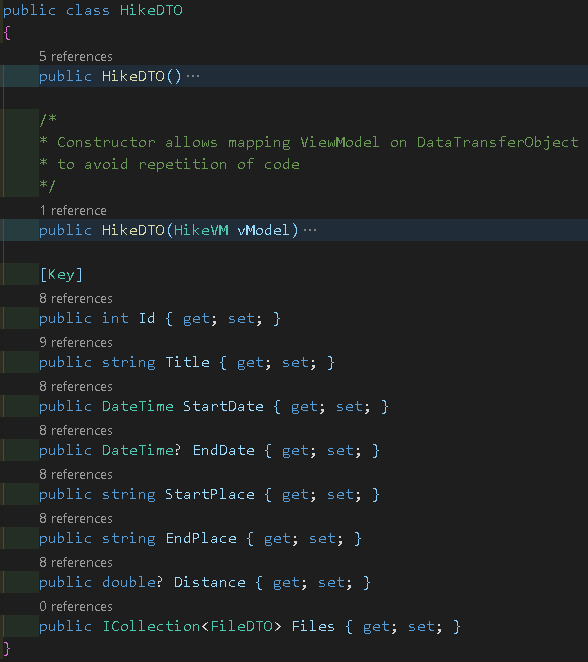
Pola:

1. Id - reprezentuje unikalny klucz rekordu w tabeli o typie liczbowym
2. Title – reprezentuje tytuł wędrówki o typie tekstowym
3. StartDate – reprezentuje planowaną datę i godzinne rozpoczęcia wędrówki o typie daty i czasu
4. EndDate - reprezentuje planowaną datę i godzinę zakończenia wędrówki o typie daty i czasu
5. StartPlace - reprezentuje planowane miejsce rozpoczęcia wędrówki o typie tekstowym
6. EndPlace - reprezentuje planowane miejsce zakończneia wędrówki o typie tekstowym
7. Distance – reprezentuje przebyty dystans o typie zmienno-przecinkowym

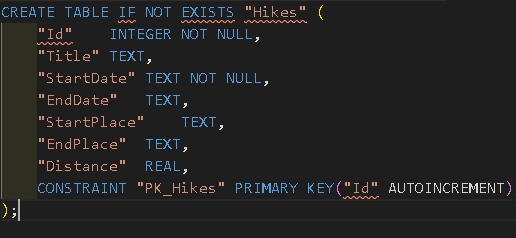
Referencje:

1. Files – reprezentuje powiązanie z kolekcją plików załączonych do wędrówki

Kod klasy HikeDTO:



Przykład wygenerowanego kodu dla klasy HikeDTO:



### Klasa HistoryHikeDTO

Reprezentuje ona zmiany w rekordzie wędrówki. Przewidziana została tak by niezależnie od modyfikowanego pola mogła przechować dane o zmianie. Zawiera ona w sobie 6 pól rzutowanych na tabele w bazie danych oraz jedną właściwość reprezentującą powiązanie:

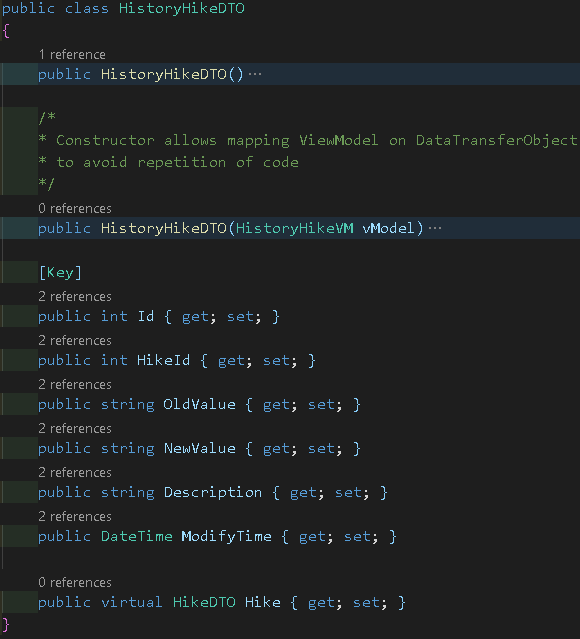
Pola:

1. Id - reprezentuje unikalny klucz rekordu w tabeli o typie liczbowym
2. HikeId - reprezentuje identyfikator wędrówki do którego wpis o zmianie należy, pole o typie liczbowym
3. OldValue – reprezentuje poprzednią wartość pola o typie tekstowym
4. NewValue - reprezentuje nową wartość pola o typie tekstowym
5. Description – reprezentuje informacje modyfikowanym polu o typie tekstowym
6. ModifyTime – reprezentuje datę i czas modyfikacji rekordu

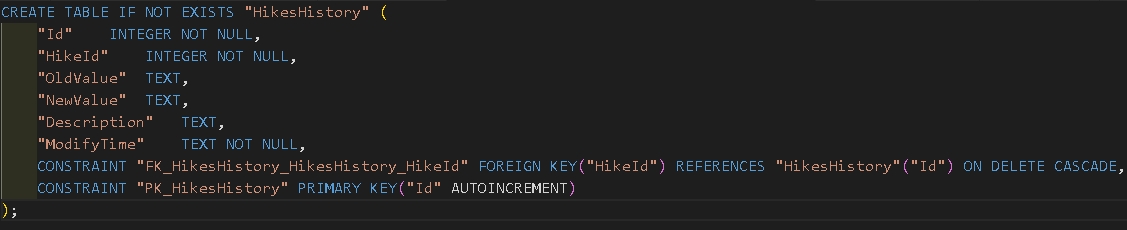
Referencje:

1. Hike – reprezentuje powiązanie między rekordem historii wędrówki a wedrówką

Kod klasy HistoryHikeDTO:



Przykład wygenerowanego kodu dla klasy HistoryHikeDTO:



### Dodatkowe atrybuty

#### Leniwe inicjowanie

Można zauważyć że niektóre właściwości zawierają słowa kluczowe „virtual”, są one implementacją wzorca projektowego Lazy initialization (z ang. Leniwe inicjowanie) polegającym na opóźnianiu tworzenia obiektów do czasu pierwszego zapotrzebowania na ten obiekt. Przy większej ilości modeli znacząco przyśpiesza to funkcjonowanie systemu.

#### Klasy DTO i VM

Sufix DTO jest skrótem od *Data Transfer Object* (z ang Obiekt Transferu Danych). Wszystkie klasy w systemie o tym sufiksie reprezentują modele tabeli oraz odzwierciedlają model rekordu pobranego z systemu z tej tabeli. Nie ma to wpływu na wykonywaną logikę, jest to jedynie jawna deklaracja przeznaczenia klasy po przez zachowanie odpowiedniej konwencji nazewniczej co ułatwia organizację oraz zarządzanie projektem. Odpowiednie nomenklatury umożliwiają unikać błędów w komunikacji oraz wykrywać je developerowi na etapie implementacji.

Sufix VM jest skrótem od *View Model* (z ang. Model Widoku) i reprezentuje jedynie modele danych oczekiwane przez widok w systemie, ich ciało może się różnić od klas DTO pomimo zawierania tej samej nazwy przed sufixem.

Dzięki zachowanej nomenklaturze mamy jawne rozróżnienie na dane które mogą być wyświetlane użytkownikowi oraz te które powinny być chronione, ponadto deweloper na etapie refaktoryzacji lub wyszukiwania błędów może szybko odnaleźć potencjalny błąd.

#### Rzutowanie w konstruktorze

W konstruktorach klas DTO zauważalna jest możliwość utworzenia obiektu rzutując obiekt klasy modelu widoku. Dzięki temu zabiegowi nie powtarza się kod przy próbie inicjowania tworzenia obiektów z interfejsów użytkownika. Potencjalne błędy są łatwiejsze do wyszukiwania przez posiadaną wiedze o fakcie rzutowania danych na siebie tylko w jednym miejscu a nie zależnie od kontekstu. Nie wpływa to na implementacje bazy danych.