Programowanie wizualne

opracował: Wojciech Frohmberg

Lab 6

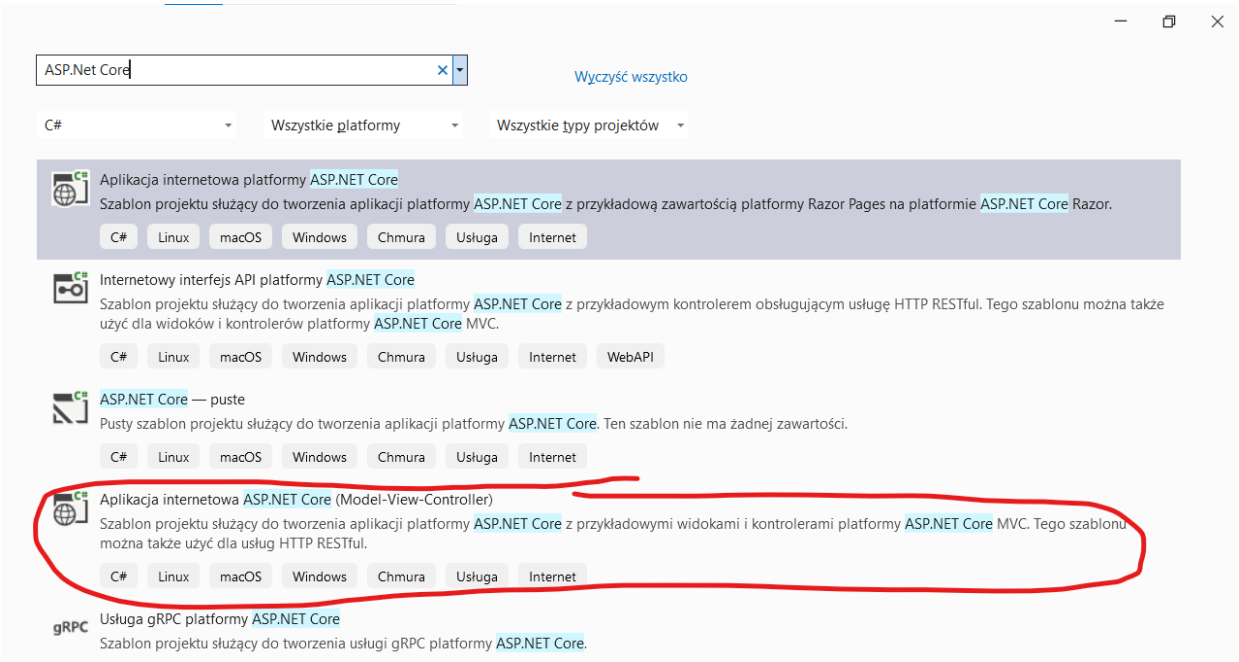
Zagadnienia do opanowania:

- Projekt MVC
- Entity Framework
- Razor

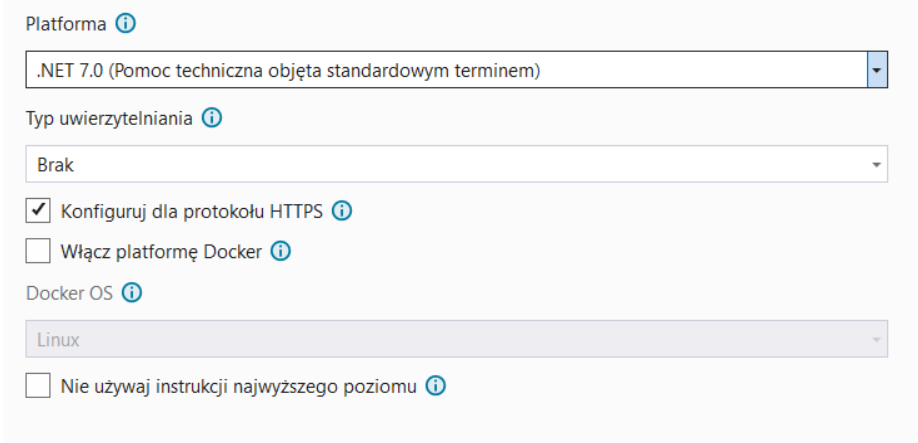
Wprowadzenie:
W ramach cyklu zajęciowego chcielibyśmy zająć się jednym spójnym projektem, w ramach którego moglibyśmy przećwiczyć możliwości aplikacji webowych utworzonych w technologii ASP.NET. Do tego celu wybierzemy problem przydzielania zadań programistycznych do deweloperów. W ramach narzędzia powinniśmy udostępnić możliwość dodania, edycji i usunięcia zadania jak również przydzielenia zadania określonemu deweloperowi. Pojedyncze zadanie powinno cechować się nazwą, opisem oraz pracochłonnością. Do każdego zadania powinna również zostać przydzielona etykieta statusu z ograniczonej puli. Chcielibyśmy zapamiętać przebieg zmiany etykiet statusu w przeciągu jego aktywności.

CZĘŚĆ I Stworzenie projektu oraz odtworzenie rzeczywistości w postaci modeli

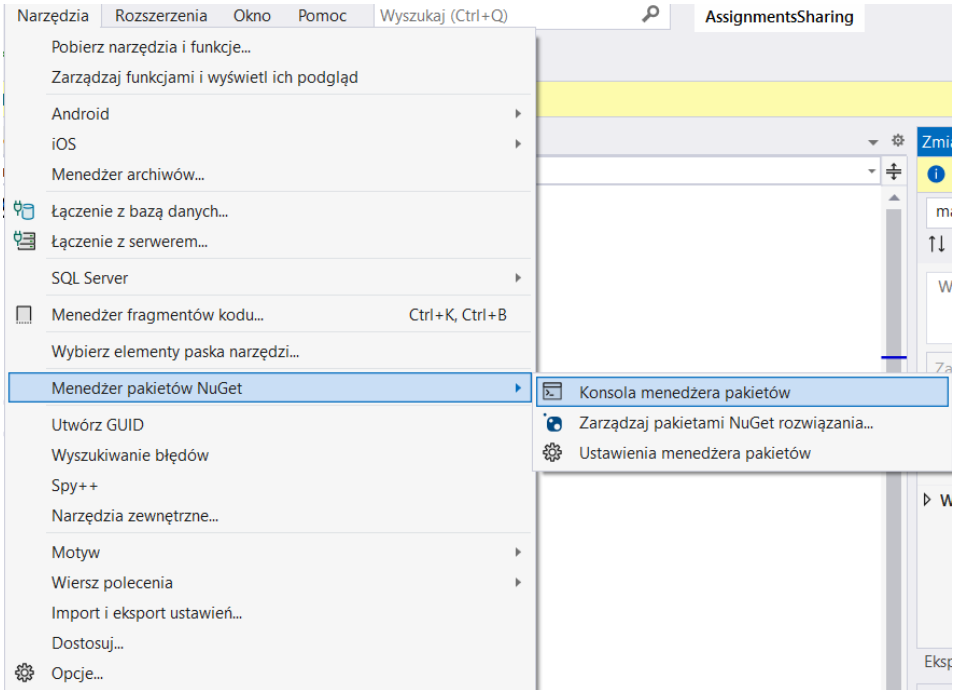
1. Utwórz projekt typu Aplikacja internetowa ASP.NET Core (Model-View-Controller).



Nadaj mu nazwę AssignmentsSharing i w opcjach wybierz następujące ustawienia:

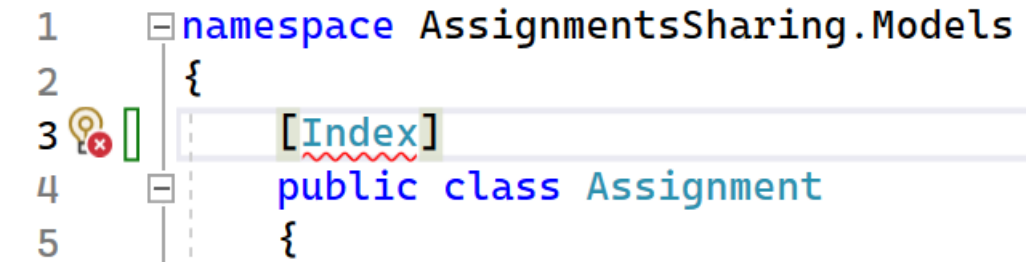


2. Do folderu Models dodaj klasę Assignment reprezentującą zadanie charakteryzującą się właściwościami: Label, Description, TimeCost (liczone w szacowanych godzinach pracy nad zadaniem) oraz Priority (typu int). Zadania o niższym priorytecie powinny mieć niższą wartość Priority.
3. Dodaj kolejną klasę modelu – Status charakteryzującą się właściwościami: StatusType (typ wyliczeniowy z dostępnymi rodzajami statusu) oraz OccuranceTime.
4. Połącz asocjacją [0-1..0-N] modele Assignment oraz Status, tak by każdy status mógł referować do zmieniających się jego statusów (patrz dokument „Asocjacje EF.pdf” dostępny w ramach kursu).
5. Dodaj klasę modelu – Developer cechującą się właściwościami FirstName, LastName oraz Pseudonym.
6. Zadbaj by do każdego zadania mógł być podpięty co najwyżej jeden deweloper wykonujący zadanie. Zastosuj asocjację [0-1..0-N] (patrz dokument „Asocjacje EF.pdf” dostępny w ramach kursu).
7. Dodaj klasę modelu Issue. Klasa ta będzie sprzęgała ze sobą zadania w pule zadań dla określonych użytkowników. Przy użyciu tej klasy łączymy zatem klasy Developer z klasą Assignment przy czym powinno to być połączenie [0-1..0-*] pomiędzy klasami Issue a Assignment oraz [0-N..0..M] pomiędzy Issue i deweloperem. Klasa Issue powinna posiadać właściwości pozwalające na określenie etykiety oraz opisu (patrz dokument „Asocjacje EF.pdf” dostępny w ramach kursu).
8. Zadbaj by wszystkie utworzone przez Ciebie klasy modeli miały pola Id jednoznacznie identyfikujące obiekty tych klasy typu Guid (nie string!).
9. Przy użyciu konsolowego managera pakietów doinstaluj niezbędne pakiety wynikające z użycia Entity Frameworka (jeśli korzystasz z innego środowiska niż VS2022 patrz slajdy wykładu):
    - Otwórz konsolę menedżera pakietów
      (Menu Narzędzia->Menedżer pakietów NuGet->Konsola menedżera pakietów)

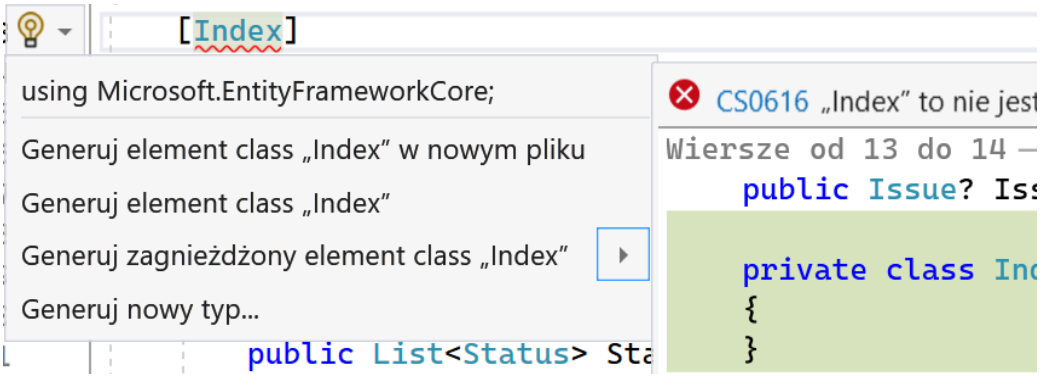


    - W konsoli wpisz następujące komendy:
      Install-Package Microsoft.EntityFrameworkCore
      Install-Package Microsoft.EntityFrameworkCore.Tools
      Install-Package Microsoft.EntityFrameworkCore.Sqlite

10. Przy użyciu atrybutu Index (na klasach modelu) dodaj ograniczenie na unikalność elementów typu Label oraz Pseudonym poszczególnych klas. Pamiętaj, że w celu użycia atrybutu Index (a właściwie klasy atrybutu IndexAttribute w ramach kodu pliku źródłowego) należy użyć przestrzeni nazw Microsoft.EntityFrameworkCore.
np. przed deklaracją klasy Assignment wpisz [Index]:



Środowisko podkreśli atrybut i w ramach hintu (żarówka z przekreśleniem) zaproponuje rozwiązanie problemu. W celu automatyzacji zaaplikowania hintu możesz skorzystać ze skrótu klawiszowego Alt+Enter i wybrać jedną z zaproponowanych opcji:



następnie określ, która spośród właściwości ma być indeksowana tak by ostatecznie kod podpięcia atrybutu indeksowania właściwości w ramach deklaracji klasy Assignment wyglądał następująco:

```
[Index(nameof(Label), IsUnique = true)]
```

UWAGA makro nameof zamienia etykietę elementu składowego, dostępnego w danym kontekście (tutaj właściwości), na ciąg znaków – można by tutaj było zatem zastąpić wywołanie makra ciągiem znaków "Label" i EntityFramework poradziłby sobie ze zindeksowaniem kolumny odpowiadającej właściwości, jednak skorzystanie z makra nameof jest o tyle lepsze, że w przypadku ewentualnej zmiany nazwy właściwości zostanie zmieniony również wstrzyknięty tutaj ciąg znaków.

11. Dodamy teraz i skonfigurujemy klasę kontekstu bazy danych mapującą elementy modeli na tabele bazy:
    - Do głównego folderu projektu dodaj klasę DataContext,
    - Klasę wydziedzicz z klasy DbContext (skorzystaj z tricku automatycznie dodającego przestrzeń nazw do pliku, dla przypomnienia skrót klawiszowy – Alt + Enter)
    - W ramach klasy utwórz konstruktor z parametrem typu IConfiguration i zapisz parametr w prywatnym polu np. _configuration (nie przejmuj się tym że nie wiesz do końca jaki obiekt pod ten parametr konstruktura podstawić i tak tworzenie obiektu kontekstu bazy danych zostanie podzlecone odpowiedniemu serwisowi dzięki określeniu tego parametru w konstruktorze mówimy serwisowi explicite że musi nam go dostarczyć tj. potrzebna nam jest konfiguracja z której będziemy odczytywać określone wpisy, takie podejście nazywami mechanizmem Dependency Injection),

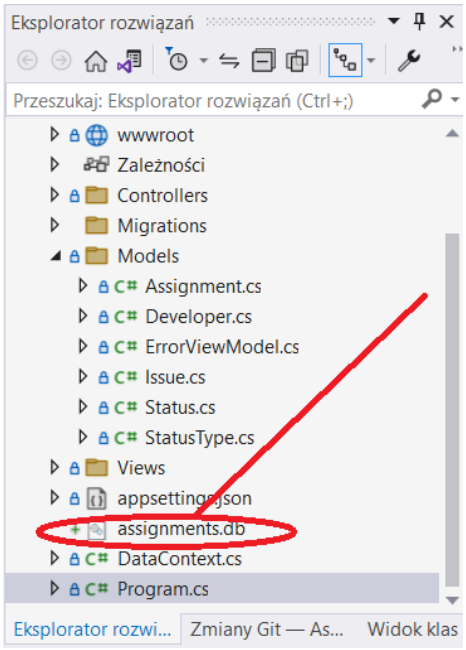
- Konfiguracja, która zostanie przekazana do naszego konstruktora znajduje się w pliku appsettings.json. Dodaj do tego pliku wpis:
```
"ConnectionStrings": {
   "Sqlite": "Data Source=assignments.db"
}
```
- Wróćmy do pliku DataContext.cs, w którym chcemy teraz przeciążyć metodę konfigurującą połączenie z bazą danych. Do tego celu wpiszmy w ramach klasy override. Po naciśnięciu znaku spacji po wpisaniu słówka kluczowego override powinny pojawić się propozycje metod z nadklasy, które można przeciążyć (dla przypomnienia z przedmiotu programowanie obiektowe – żeby przeciążyć metodę w języku C# w nadklasie musimy mieć zadeklarowaną taką samą metodę określoną słówkiem kluczowym virtual). Wybierzemy tutaj metodę OnConfiguring (możemy tutaj przechodzić pomiędzy metodami przy użyciu strzałek góra i dół, wybierać metodę możemy natomiast przy użyciu klawisza Enter).
- W wyniku poprzedniej instrukcji utworzona zostania standardowa implementacja metody OnConfiguring wywołująca implementację metody z nadklasy. My chcielibyśmy tą domyślą funkcjonalność zmienić na skonfigurowanie połączenia z bazą danych Sqlite. Usuńmy zatem zaproponowane ciało metody.
- Skorzystajmy następnie zamiast tego z parametru optionsBuilder, na którym wywołamy metodę UseSqlite. Metoda ta przyjmuje w parametrze tzw. connection string do bazy, do której chcemy się podłączyć. My taki connection string zadeklarowaliśmy już w ramach naszego pliku z konfiguracją. Do tego celu użyjemy wstrzykniętego przy użyciu Dependency Injection pola _configuration wywołując na nim metodę GetConnectionString z parametrem ciągoznakowym "Sqlite" (bo tak nazwaliśmy connection string w pliku appsettings.json). Ostatecznie zatem implementacja metody powinna wyglądać następująco:
```
protected override void OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
{
    optionsBuilder.UseSqlite(_configuration.GetConnectionString("Sqlite"));
}
```
- W ramach kontekstu danych musimy jeszcze określić, które spośród klas stanowią modele rzeczywistości które chcemy traktować jako tabele bazodanowe dostępne w tym kontekście. Żeby określić jakąś klasę jako model tego kontekstu po prostu dodajemy do niego właściwość typu DbSet<KlasaModelu> przykładowo:
```
public virtual DbSet<Assignment> Assignments { get; set; }
```
- Pamiętaj by podobnie dodać pozostałe modele!

12. W głównym pliku aplikacji (Program.cs) zarejestruj usługę tworzącą obiekty kontekstu bazy danych. Skorzystaj do tego celu z metody generycznej AddDbContext przekazując typu kontekstu w generycznym parametrze:
```
builder.Services.AddDbContext<DataContext>();
```
Dzięki temu kontekst bazy danych będzie mógł m.in. wykorzystywać mechanizm Dependency Injection, a my jawnie nie będziemy musieli nigdy tworzyć obiektu kontekstu wszędzie oczekując jedynie że określamy tworzenie takiego kontekstu jako zależność (stąd dependency).
13. Póki co jedynie określiliśmy co chcemy postrzegać jako modele w bazie danych. Nic co zrobiliśmy nie miało jeszcze odzwierciedlenia na fizyczną bazę danych. Żeby utworzyć plikową bazę danych (dlatego skorzystaliśmy z sqlite) Dodamy tzw. migrację tj. krok odzwierciedlenia aktualnego stanu klas programu na bazę danych. Jako że jeszcze żadnej migracji nie stworzyliśmy obecna migracja będzie inicjalizacyjną. Za jej pomocą utworzymy plik z bazą danych. Do tego celu wejdź jeszcze raz w konsolę menedżera pakietów i wpisz następujące komendy (jeśli korzystasz z innego środowiska niż VS2022

patrz slajdy wykładu):

```
Add-Migration InitialMigration
Update-Database
```

Zweryfikuj czy powstał plik z bazą danych wynikający z ConnectionString, który umieściłaś/eś w pliku konfiguracyjnym appsettings.json.



Gdybyśmy dysponowali narzędziami do pracy z bazą plikową sqlite moglibyśmy podejrzeć jakie tabele w ramach bazy danych zostały utworzone:

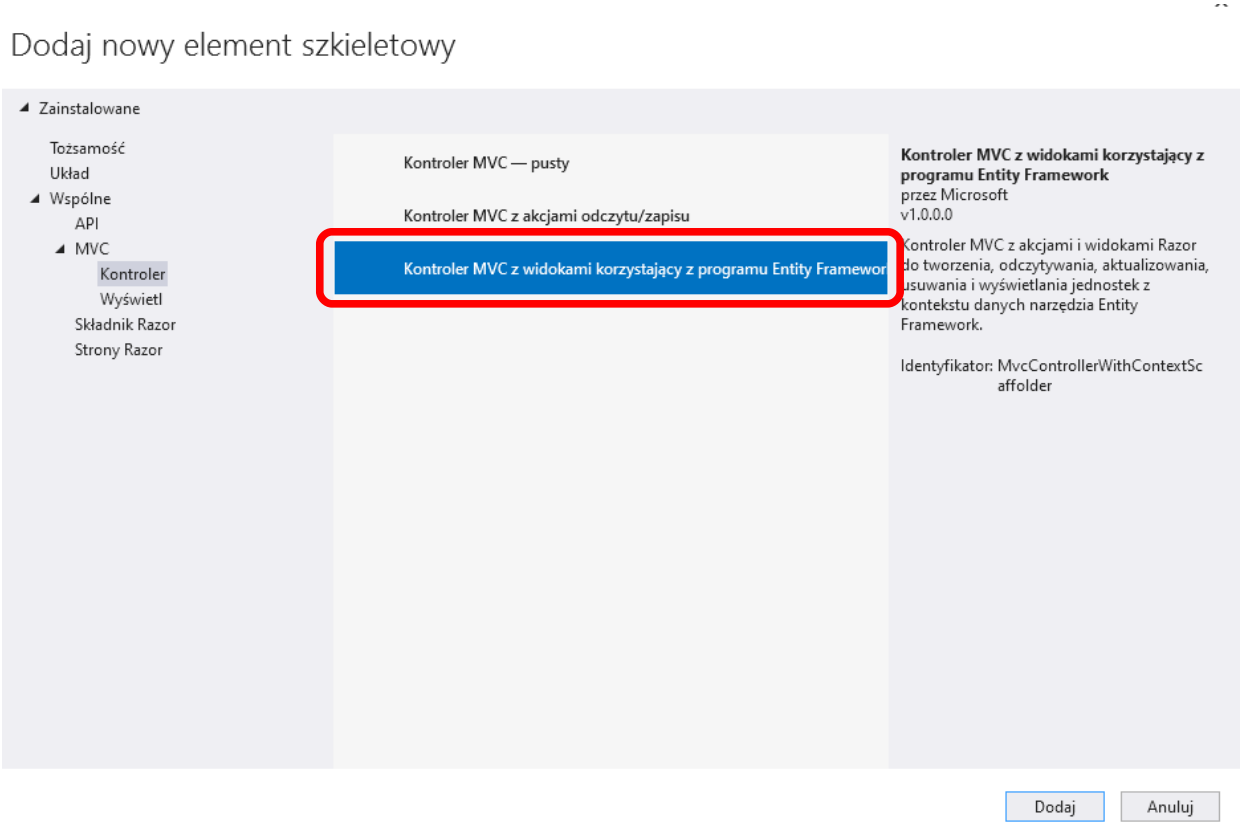
```
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
sqlite> .tables
Assignments            Developers             Statuses
DeveloperIssue         Issues                 __EFMigrationsHistory
sqlite>
```

Dla kompletności pamiętaj by nie zapisywać pliku z bazą danych w repozytorium wersjonowania. W tym celu najlepiej tuż po wywołaniu komendy Update-Database inicjalizującej plik z bazą danych dodać plik z bazą do plików ignorowanych np. w .gitignore w przypadku korzystania z narzędzia git.

CZĘŚĆ II Utworzenie widoków typu CRUD (Create-Read-Update-Delete) dla modeli naszej rzeczywistości

14. Doinstaluj NuGet Microsoft.VisualStudio.Web.CodeGeneration.Design (jeśli przy instalacji będzie problem z wersją .NETa bo np. nie masz najnowszej wersji .NETa 7 zainstalowanej na Twoim komputerze możesz skorzystać ze starszej wersji np. 6.0.10).
15. Wygeneruj kod kontrolera ze standardowymi akcjami oraz widokami dla akcji – dla klasy Assignments, nazwij go AssignmentsController (kliknij prawym przyciskiem myszy na folder Controllers) i wybierz opcję Dodaj->Kontroler... A następnie wybierz opcję Kontroler MVC z widokami korzystający z programu Entity Framework (patrz ilustracja poniżej). Pamiętaj by wybrać nazwę klasę modelu Assignment z kolei klasa kontekstu bazy danych

powinna być ustawiona na utworzony przez nas wcześniej DataContext (jeśli korzystasz z innego środowiska niż VS2022 patrz slajdy wykładu).



16. Przeanalizuj wygenerowany kod i postaraj się zrozumieć implementację poszczególnych akcji (plik Controllers/AssignmentsController.cs). Zastanów się przy tym co by się stało gdybyśmy zamiast nazwy klasy modelu Assignment dla zadania wybrali Task.
17. Uruchom program i przejdź do akcji Index kontrolera Assignments (np. https://localhost:<portusługi>/Assignments/Index).
18. Przy użyciu wygenerowanych widoków spróbuj dodać obiekt typu Assignment do bazy.
19. Zakończ debug działania aplikacji i powtórz krok tworzenia kontrolera/widoków dla pozostałych utworzonych przez Ciebie modeli.

Na kolejnych zajęciach spróbujemy zaimplementować klasy funkcjonalne do naszego projektu.