

ZPR PWr – Zintegrowany Program Rozwoju Politechniki Wrocławskiej

Aplikacje webowe na platformę .NET

W10 – Entity Framework Core

Syllabus

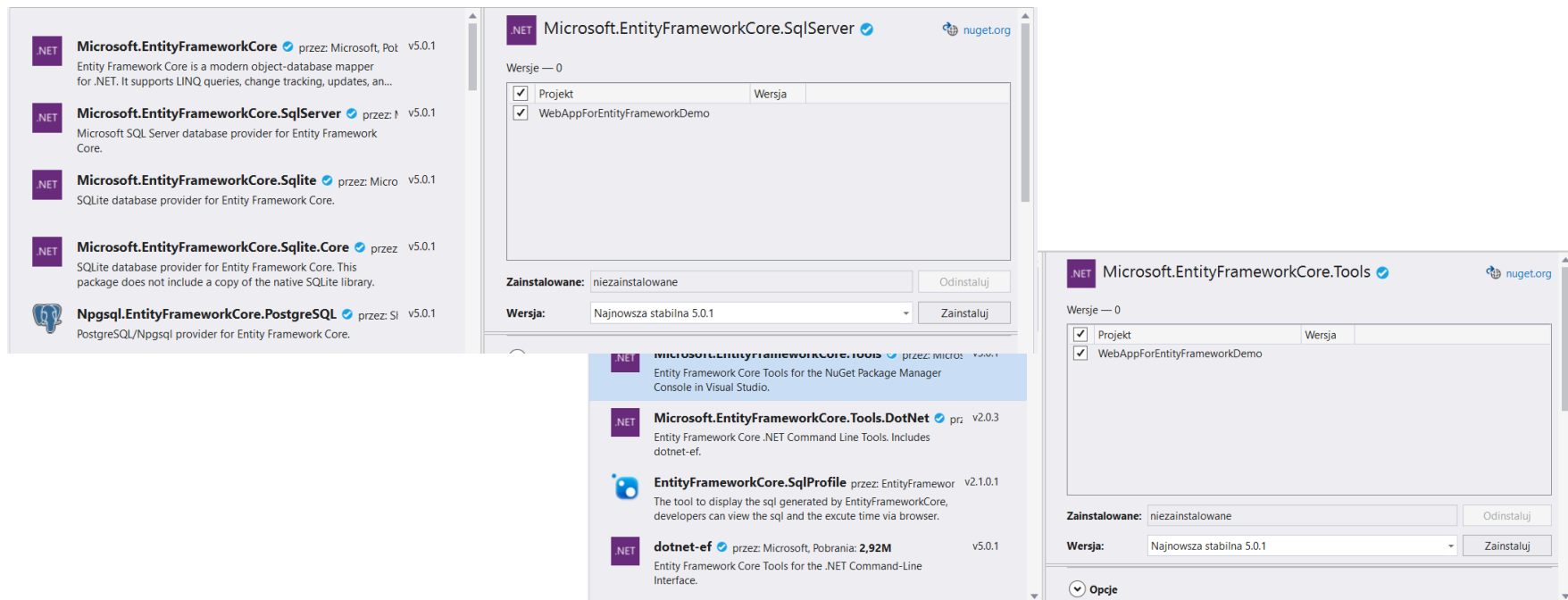
- Instalacja Entity Framework
- EntityFramework – ogólnie
 - **Podejście Code First**
 - Podejście Database First
- Zastosowanie Connection String
- Format JSON
- Klasy `DbContext<>`, `DBSet<>`
- Migracje i aktualizacje bazy danych
- Metoda rozszerzająca `Seed()` – tworzenie danych dla bazy danych
- Generator kodu dla Entity Framework:
 - Kontroler dla klasy z kontekstu wraz z metodami i widokami dla operacji CRUD
- **async, await**, `Task`
- Przykład użycia modelu widokowego przy działaniu dla modelu domenowego
- Adnotacje bazodanowe, dla widoków i walidacji danych
- Relacje między klasami z `DBSet`-ów
 - Jeden do wielu
 - Wiele do wielu
- Podejście **Database First**
 - Struktura bazy w metodach kontekstu
 - Struktura bazy w adnotacjach
- Inne frameworki: Dapper

Entity Framework

Wstep

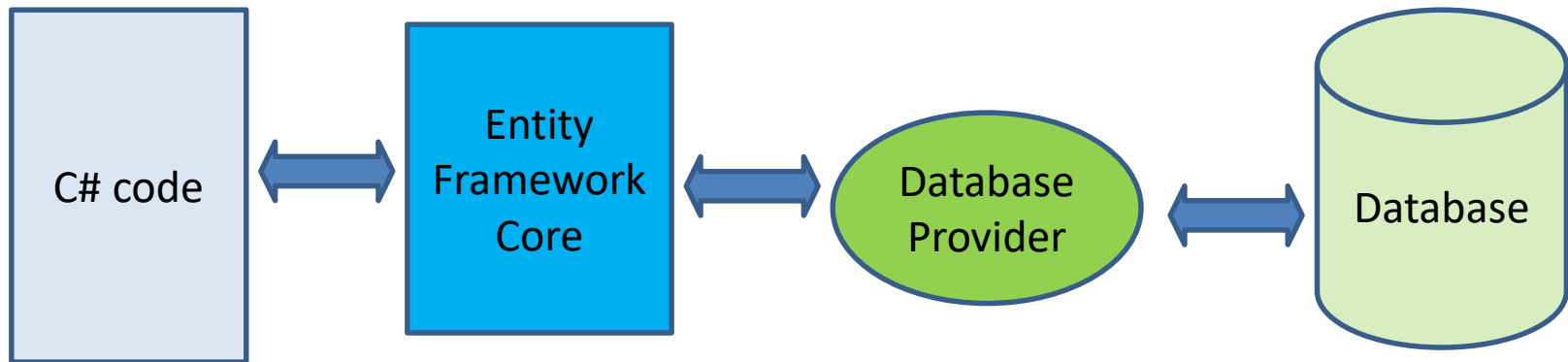
Entity Framework - instalacja

- Należy zainstalować za pomocą NuGet odpowiedni pakiet. Najlepiej:
 - Microsoft.EntityFrameworkCore.SqlServer
 - Wówczas zainstalowane zostaną również pakiety zależne od powyższego Microsoft.EntityFrameworkCore.Relational (dla relacyjnych baz danych) oraz Microsoft.EntityFrameworkCore (dla ogólnych funkcjonalności dla baz danych)
 - Microsoft.EntityFrameworkCore.Tools (dla konsoli pakietów w celu dodawania migracji)
- Istnieją pakiety dla baz danych nie z MS, np. Pomelo.EntityFrameworkCore.MySql
 - Lista dostawców baz danych:
<https://docs.microsoft.com/pl-pl/ef/core/providers/>
- Tworząc projekt może się okazać, że została zainstalowany metapakiet (zestaw pakietów np. Microsoft.AspNetCore.All), w którym już jest EntityFramework.



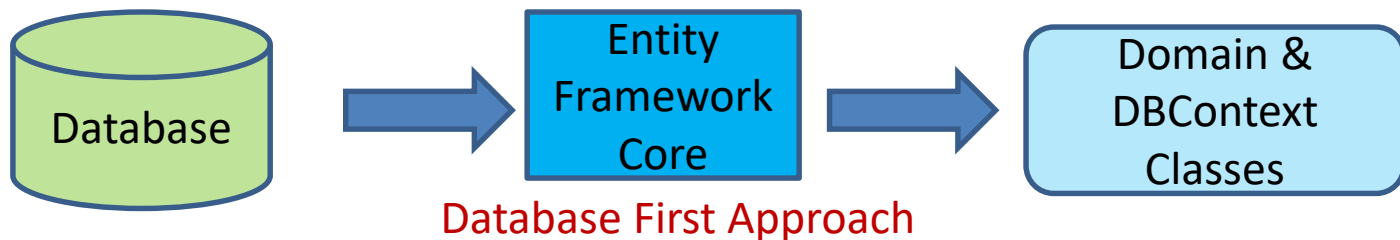
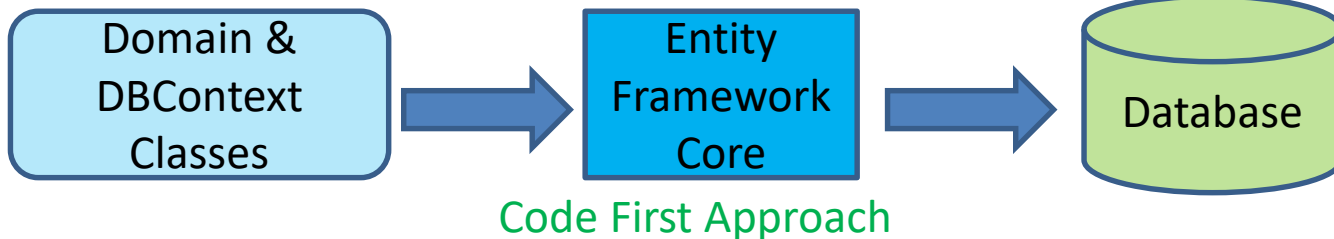
Entity Framework - wstęp

- Entity Framework Core jest narzędziem typu ORM (Object Relational Mapping), pozwalającym odwzorować relacyjną bazę danych za pomocą architektury obiektowej
 - Lekki, rozszerzalny, open source
 - Działa na wielu platformach
 - Oficjalna platforma Microsoftu dostępu do danych.



Entity Framework Core – 2 podejścia

- Dwa podejścia na stworzenie modelu danych:
 - **Code First** – najpierw piszemy kod klas, na podstawie którego tworzony jest model danych oraz struktura bazy danych.
 - *Database First* - podejście to stosujemy gdy mamy już gotową bazę danych. Kod dla klas danych i kontekstu zostanie wytworzony na podstawie analizy bazy danych.
 - Obecnie wsparcie dla tego podejścia jest dość ograniczone
- Podejście nieobecne w EF Core, obecne w EF Framework
 - *Model First* - za pomocą tego podejścia nie musimy pisać żadnego kodu SQL. Wystarczy, że stworzymy model danych w ADO.NET Entity Data Model Designer. Na podstawie stworzonego modelu tworzona jest struktura bazy danych oraz kody klas.
- EF Core pozwala na korzystanie z relacyjnych baz danych ale również z nierelacyjnych baz danych.



Entity Framework

Podejście Code First - ogólnie

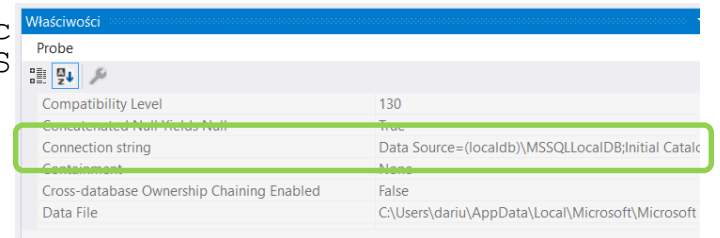
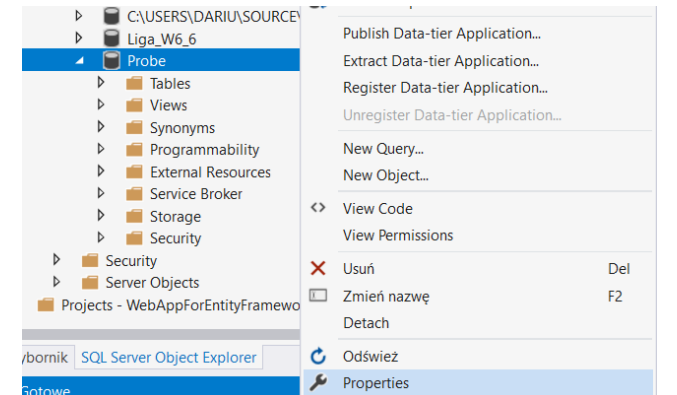
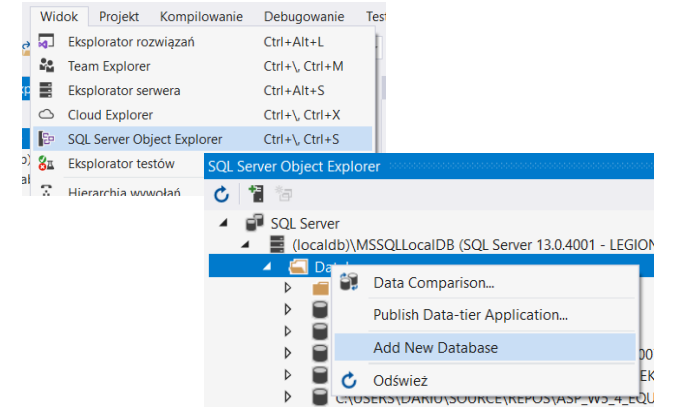
Podejście Code First

Dla luźno powiązanych komponentów:

- Najpierw tworzymy **klasy dla modelu domenowego**
- Następnie tworzymy **klasę kontekstu** (dostępu do bazy danych), wykorzystując zdefiniowany model domenowy
 - Konstruktor z parametrem `options`
- Dodajemy **serwis** związany z **klasą kontekstu** do kontenera serwisów
- W pliku `appsettings.json` dopisujemy **connection string**, łączący z konkretną bazą danych.
- *Ewentualnie* dopisujemy metodę **uzupełniającą bazę danych początkowymi danymi**
 - Najlepiej jako metodę **rozszerzającą** `Seed()`
- W **konsoli menadżera pakietów** wykonujemy **migrację początkową** bazy danych
 - Tworzy się kod EF w C# odpowiedzialny za pierwszą modyfikację bazy danych
 - Tworzy się kod ze stanem (jakie tabele, jakie pola, jakie ograniczenia, powiązania itd.) bazy danych
- **Uaktualniamy bazę danych** (wg stworzonej wcześniej migracji)
 - W konsoli można zobaczyć zapytania SQL, które zostaną uruchomione na serwerze SQL
- **Dopisujemy kod w C#** związany z **kontrolerami, akcjami, widokami** korzystając z klasy kontekstu wstrzykiwanej w konstruktorze
 - W razie potrzeby tworząc klasy modelu widoków, gdy dane z formularza nie mapują się 1-1 do danych domenowych

Tworzenie połączenia do bazy danych – connection string

- Dla celów demonstracyjnych użyta zostanie jednoplikowa baza danych typu MDF
 - Niezależna od połączenia internetowego
 - Nie wymaga tworzenia użytkownika-administratora bazy danych
 - Nie wymaga uwierzytelniania za pomocą konta-hasła (co wymaga wpisania tych danych do connection string)
- Do korzystania z bazy danych potrzebne jest **connection string**, czyli ciąg znaków, w którym są pary klucz-wartość połączone znakiem '=', pary rozdziela średnik. Zawierają one **parametry połączenia** do bazy danych.
- Zamiast szukać jak taki string stworzyć na podstawie dokumentacji, można skorzystać w Visual Studio 2019 z widoku SQL Server Object Explorer. Powinien być widoczny serwer o nazwie „(localdb)\MSSQLLocalDB”. Po stworzeniu nowej bazy w ramach tego serwera (np. o nazwie **Probe**) i rozwinięciu jej, PPM i wybranie „Properties” pozwala znaleźć właściwość „Connection string”.
- Standardowo będzie wyglądać jak poniżej:
 - `Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Probe;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False`
- Najczęściej wystarczy początek jak poniżej:
 - `Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=Probe;Integrated Security=True`
- Para „Integrated Security=True” oznacza, że dostęp jest dozwolony dla zalogowanego użytkownika Windowsa.



Model domenowy i klasa kontekstu bazy danych

```
namespace WebAppForEntityFrameworkDemo.Models
{
    public enum Gender { Female, Male }
    public class Student
    {
        public int Id { get; set; }
        [Required]
        [RegularExpression(@"^[0-9]{1,6}$")]
        public int Index { get; set; }
        [Required]
        [MinLength(2, ErrorMessage="Too short name")]
        [Display(Name="Last Name")]
        [MaxLength(20, ErrorMessage="Too long name, do not exceed {1}")]
        public string Name { get; set; }
        public Gender Gender { get; set; }
        public bool Active { get; set; }
        public int DepartmentId { get; set; }
        [DataType(DataType.DateTime)]
        [Required]
        public DateTime BirthDate { get; set; }
    }
    // constructors
    ...
}
```

```
namespace WebAppForEntityFrameworkDemo.Data
{
    public class MyDbContext:DbContext
    {
        public MyDbContext(DbContextOptions<MyDbContext> options) : base(options)
        {
        }
        public DbSet<Student> Student { get; set; }
    }
}
```

Klasy Entity Framework

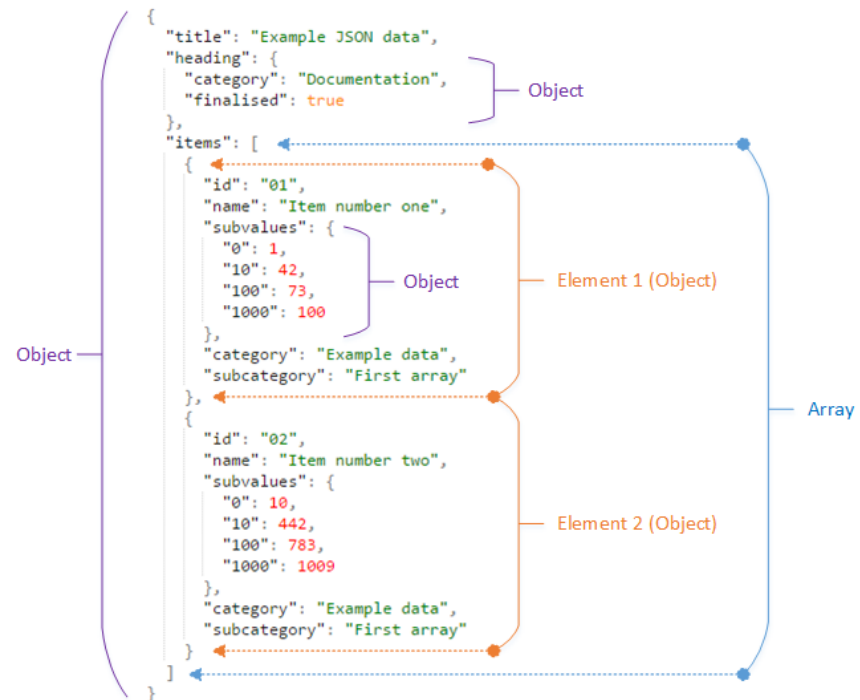
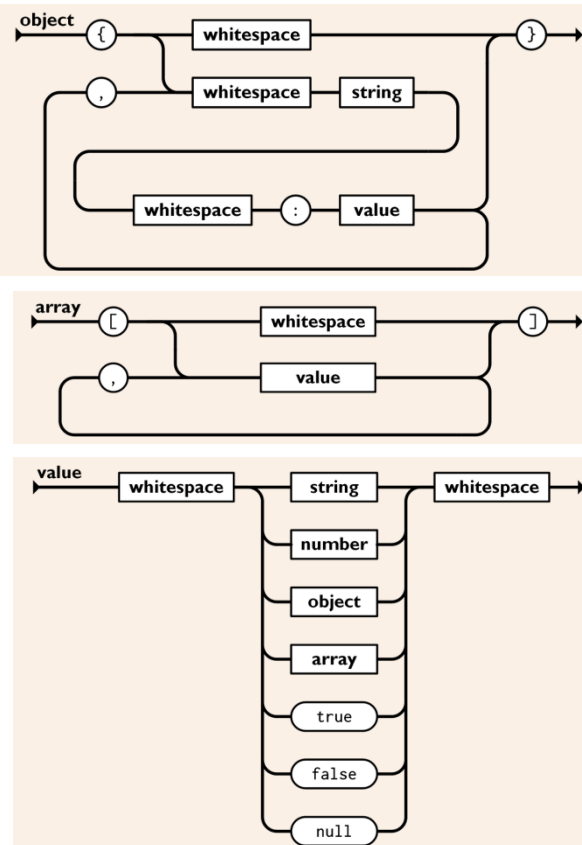
- Klasa `DbContext` to główna klasa, po której należy **dziedziczyć**, aby stworzyć **własny kontekst dostępu** do bazy danych (w przykładzie: `MyDbContext`).
 - Albo po klasach pochodnych po `DbContext`.
- W kontekście należy stworzyć konstruktor z parametrem `DbContextOptions<MyDbContext>` aby przy dodawaniu do kontenera serwisów ustalić connection string.
- Klasa kontekstu jako publiczne właściwości posiada kolekcje generyczne typu `DbSet<>` dla każdej klasy modelu domenowego.
 - Poprzez właściwości klasy `DbSet<>` następuje dostęp do tabel w bazie danych.
- Entity Framework dla każdej `DbSet<X>` i `DbSet<Y>` próbuje określić jakie **relacje** zachodzą **między klasami** `X` i `Y` oraz stara się na tej podstawie stworzyć kod mapujący tabele bazy danych zawierające kolekcje obiektów klasy `X` i `Y`.
 - Np. klasa `X` zawiera **listę obiektów** klasy `Y`: zatem w tabeli `Y` powinien być **klucz obcy** będący **kluczem głównym** klasy `X`
- Jeśli propozycje EF są niepoprawne lub niewystarczające zawsze można stworzyć właściwe w **odpowiedniej** metodzie klasy kontekstu.

Entity Framework

Format JSON, connection string

Format JSON

- JSON, JavaScript Object Notation – tekstowy lekki format wymiany danych komputerowych
 - <https://www.json.org/json-en.html>
- Całość to jeden obiekt w języku Javascript.
- Obiekty zapisane są w nawiasach klamrowych, tablice w nawiasach kwadratowych.
- Obiekt to słownik par klucz-wartość rozdzielonych dwukropkiem, kolejna para po przecinku.
- Dopuszczalne są tylko bardzo podstawowe typy (jak poniżej dla value).
- Przykład z <https://docs.exivty.com/data-pipelines/extract/parslets>



Connection string i wstrzykiwanie

- W `appsettings.json` ustawiamy connection string do bazy `DbForEFdemo` (która jeszcze nie istnieje)
 - W zasadzie całą sekcję (klucz w JSON) `ConnectionStrings` dla takich stringów
 - Każdy connection string ma swoją nazwę (np. klucz „`MyDb`”)



```
1 {
2   "Logging": {
3     "LogLevel": {
4       "Default": "Information",
5       "Microsoft": "Warning",
6       "Microsoft.Hosting.Lifetime": "Information"
7     }
8   },
9   "ConnectionStrings": {
10    "MyDb": "Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=DbForEFdemo;Integrated Security=True"
11  },
12  "AllowedHosts": "*"
13 }
```

- Dodajemy kontekst bazy danych do kontenera serwisów (inaczej niż zwykłe serwisy).
 - Mniej efektywnie przez `AddDbContext<>`
 - Bardziej efektywnie przez `AddDbContextPool<>`
- W parametrze `options` ustawiamy właściwy serwer bazodanowy oraz connection string.

```
public IConfiguration Configuration { get; }

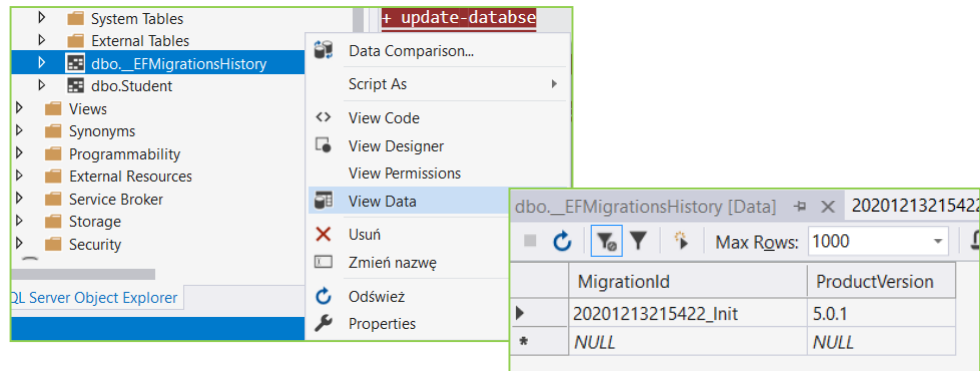
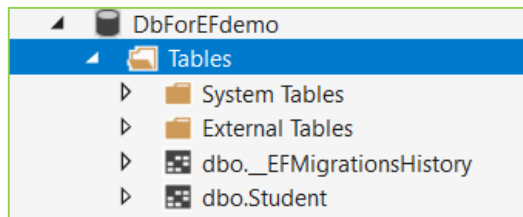
// This method gets called by the runtime. Use this method to add services to
// the container.
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddDbContextPool<MyDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("MyDb")));
}
```

Entity Framework

Migracje, zasiewanie danych

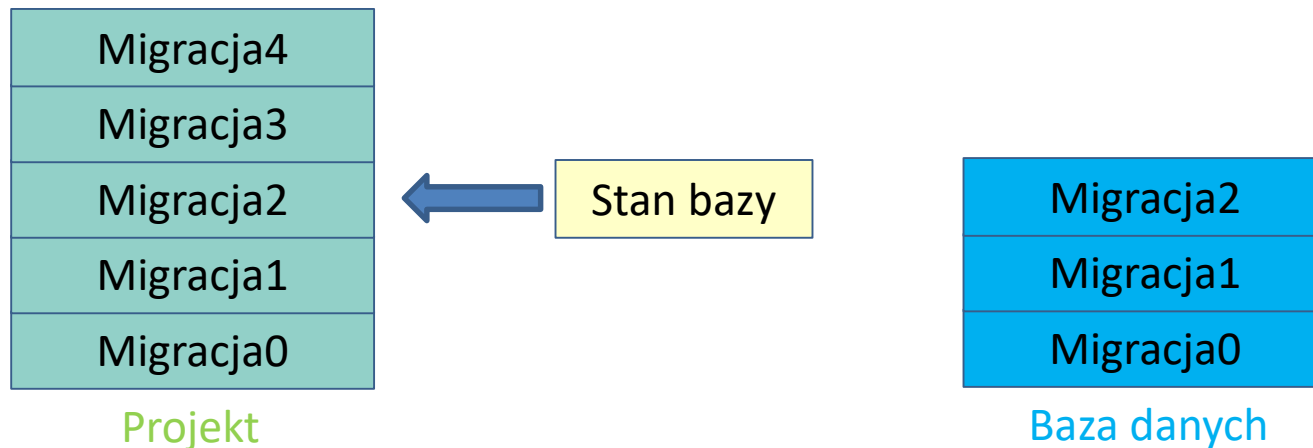
Migracje

- Konsola package managera
- Komenda dodająca migrację o wybranej nazwie:
 - `add-migration Init`
- Powstaje folder Migrations, a w nim plik `.cs` (klasa dziedzicząca po `Migration`) o podanej nazwie poprzedzony stemplem czasu.
 - Dla każdej kolejnej migracji kolejny plik (klasa)
- Oprócz tego plik (klasa) dziedzicząca po `ModelSnapshot` z bieżącym stanem bazy danych
 - Co do struktury i ewentualnie zawartości
- W klasie migracji istnieją dwie metody:
 - `protected override void Up(MigrationBuilder migrationBuilder)`
 - `protected override void Down(MigrationBuilder migrationBuilder)`
- Pierwsza metoda będzie wykonywana, gdy chcemy zaktualizować bazę „do przodu”
- Druga, gdy chcemy wycofać zmiany tej migracji.
- Bieżący stan bazy danych (Snapshot) służy do dostępu do bazy danych i opisuje tabele, kolumny w tabelach, indeksy itd.
- Wykonanie migracji bazy danych w konsoli menadżera pakietów następuje po wpisaniu komendy:
 - `update-database`
 - Do ostatniej migracji
 - `update-database <nazwaMigracji>`
- Po odświeżeniu widoku bazy danych w SQL Server Object Explorer powstały tabele z danymi z modelu oraz dla obsługi migracji.
- Tabela migracji zawiera nazwę migracji ze stemplem czasu
- **Scenariusz użycia:** analiza kodu migracji

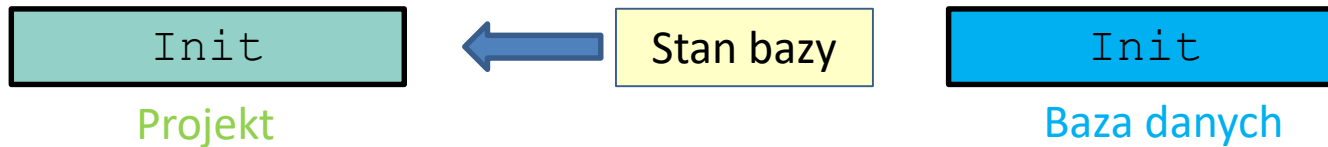


Idea migracji

- Migracje stanowią stos.
- Kolejna migracja jako stan początkowy ma stan z poprzedniej migracji.
- W razie potrzeby należy zmodyfikować metody `Up()` lub `Down()` w przypadku bazy danych zawierającej już informacje.
 - Np. dodanie nowej **wymaganej** (not null) kolumny spowoduje potrzebę określenia jej wartości
- Baza pamięta, z której migracji powstała.
- Dopóki nie wykonamy migrację bazy danych możemy dodawać kolejne migracje lub je usuwać (pojedynczo jak ze stosu), jednak nie niżej niż do migracji zapamiętanej w bazie danych.
 - `Remove-migration`
- Każda zmiana w modelu domenowym powinna powodować stworzenie migracji
 - Entity Framework wykrywa jaka zmiana(zmiany) została dokonana
- Można cofnąć stan bazy do konkretnej migracji poprzez aktualizacje do migracji niżej na stosie migracji:
 - `update-database <nazwaMigracji>`
 - `np.update-database Migracja1`
- Ogólne wykonanie `update-database` sprawdza, czy należy wykonać migrację w górę stosu i uruchomić metody `Up()`, czy też w dół stosu i wykonać metody `Down()` z kolejnych migracji aż do wybranego punktu na stosie.



Scenariusz migracji i aktualizacji



Założenie: Stan początkowy po wykonaniu migracji Init i aktualizacja bazy danych

- 1) Dodanie nowego pola Comment
 - Add-migration AddFieldComment
- 2) Zmiana maksymalnej długości pola Name
 - Add-migration ChangeMaxLenName
- 3) Dodanie pola What
 - Add-migration AddFieldWhat
- 4) Aktualizacja bazy do migracji MaxLenName
 - update-database ChangeMaxLenName
 - Wykonane (operacje Up ()) dwie migracje: AddFieldComment, potem ChangeMaxLenName
- 5) Wycofanie ostatniej migracji AddFieldWhat
 - Remove-migration
- 6) Wycofanie migracji ChangeMaxLenName
 - Remove-migration
 - Niepowodzenie, stan bazy danych nie pozwala
- 7) Aktualizacja bazy do Init (wykonanie metod Down () z dwóch migracji)
 - Update-database Init
- 8) Wycofanie z sukcesem migracji ChangeMaxLenName
 - Remove-migration
- 9) Aktualizacja bazy do AddFieldComment
 - Update-database
 - Już nie ma więcej migracji wyższych migracji
- 10) Wycofanie zmian w bazie danych, czyli powrót do Init
 - Update-database do Init
- 11) Wycofanie migracji AddFieldComment:
 - Remove-migration AddFieldComment

Krok 1

- Dodanie nowego pola Comment
 - Add-migration AddFieldComment

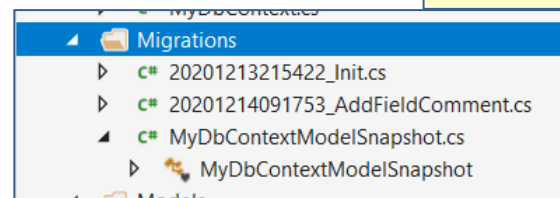
Student.cs

```
public DateTime BirthDate { get; set; }

[MaxLength(100)]
Odwołania: 0
public string Comment { get; set; }

Odwołania: 0
public Student()
```

Migracje



...Snapshot.cs

```
b.Property<DateTime>("BirthDate")
    .HasColumnType("datetime2");

b.Property<string>("Comment")
    .HasMaxLength(100)
    .HasColumnType("nvarchar(100)");

b.Property<int>("DepartmentId")
    .HasColumnType("int");
```

AddFieldComment.cs

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.AddColumn<string>(
        name: "Comment",
        table: "Student",
        type: "nvarchar(100)",
        maxLength: 100,
        nullable: true);
}

1 odwołanie
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropColumn(
        name: "Comment",
        table: "Student");
}
```

AddFieldComment

Init

Projekt



Stan bazy

Init

Baza danych

Krok 2

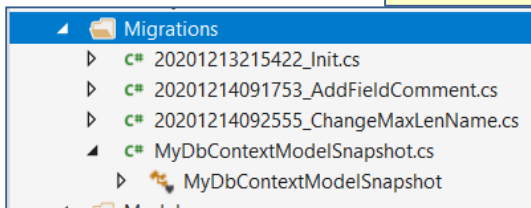
- Zmiana maksymalnej długości pola Name
 - Add-migration ChangeMaxLenName

```
Build succeeded.  
An operation was scaffolded that may result in the loss of data. Please review the migration for accuracy.  
To undo this action, use Remove-Migration.  
PM>
```

Student.cs

```
public int Index { get; set; }  
[Required]  
[MinLength(2, ErrorMessage = "To short name")]  
[Display(Name = "Last Name")]  
[MaxLength(30, ErrorMessage = "To long name, do not exceed {0}")]  
Odwolania: 13  
public string Name { get; set; }
```

Migracje

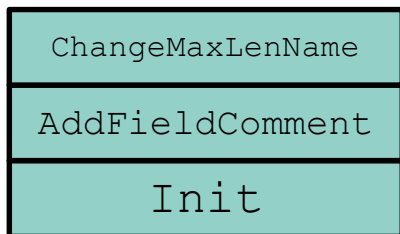


...Snapshot.cs

```
b.Property<int>("Index")  
    .HasColumnType("int");  
  
b.Property<string>("Name")  
    .IsRequired()  
    .HasMaxLength(30)  
    .HasColumnType("nvarchar(30)");  
  
b.HasKey("Id");
```

ChangeMaxLenName.cs

```
protected override void Up(MigrationBuilder migrationBuilder)  
{  
    migrationBuilder.AlterColumn<string>(  
        name: "Name",  
        table: "Student",  
        type: "nvarchar(30)",  
        maxLength: 30,  
        nullable: false,  
        oldClrType: typeof(string),  
        oldType: "nvarchar(20)",  
        oldMaxLength: 20);  
}  
  
Odwolania: 2  
protected override void Down(MigrationBuilder migrationBuilder)  
{  
    migrationBuilder.AlterColumn<string>(  
        name: "Name",  
        table: "Student",  
        type: "nvarchar(20)",  
        maxLength: 20,  
        nullable: false,  
        oldClrType: typeof(string),  
        oldType: "nvarchar(30)",  
        oldMaxLength: 30);  
}
```



Projekt



Stan bazy



Baza danych

Krok 3

- Dodanie pola What
 - Add-migration AddFieldWhat

Student.cs

```
Odwołania: 0
public string Comment { get; set; }

[MaxLength(20)]
Odwołania: 0
public string What { get; set; }

Odwołania: 0
public Student()
```

...Snapshot.cs

```
b.Property<string>("Name")
    .IsRequired()
    .HasMaxLength(30)
    .HasColumnType("nvarchar(30)");

b.Property<string>("What")
    .HasMaxLength(20)
    .HasColumnType("nvarchar(20)");

b.HasKey("Id");
```

AddFieldWhat

ChangeMaxLenName

AddFieldComment

Init

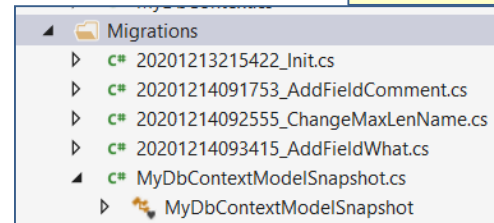
Projekt

Stan bazy

Init

Baza danych

Migracje



AddFieldWhat.cs

```
public partial class AddFieldWhat : Migration
{
    Odwołania: 3
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.AddColumn<string>(
            name: "What",
            table: "Student",
            type: "nvarchar(20)",
            maxLength: 20,
            nullable: true);
    }

    Odwołania: 3
    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropColumn(
            name: "What",
            table: "Student");
    }
}
```

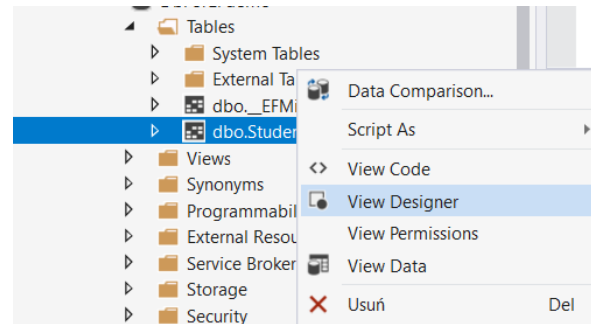
Krok 4

- Aktualizacja bazy do migracji MaxLenName
 - update-database ChangeMaxLenName
 - Wykonane (operacje Up ()) dwie migracje: AddFieldComment, potem ChangeMaxLenName

dbo.Student [Design] 20201214093415_AddFieldWhat.cs 202012140925

Update Script File: dbo.Student.sql

	Name	Data Type	Allow Nulls	Default
Id	Id	int	<input type="checkbox"/>	
Index	Index	int	<input type="checkbox"/>	
Name	Name	nvarchar(30)	<input type="checkbox"/>	
Gender	Gender	int	<input type="checkbox"/>	
Active	Active	bit	<input type="checkbox"/>	
DepartmentId	DepartmentId	int	<input type="checkbox"/>	
BirthDate	BirthDate	datetime2(7)	<input type="checkbox"/>	
Comment	Comment	nvarchar(100)	<input checked="" type="checkbox"/>	



dbo._EFMigrationsHistory [Data] 20201214092555_...ngeMaxL

Max Rows: 1000

MigrationId	ProductVersion
20201213215422_Init	5.0.1
20201214091753_AddFieldComment	5.0.1
20201214092555_ChangeMaxLenName	5.0.1
NULL	NULL

AddFieldWhat
ChangeMaxLenName
AddFieldComment
Init

Projekt



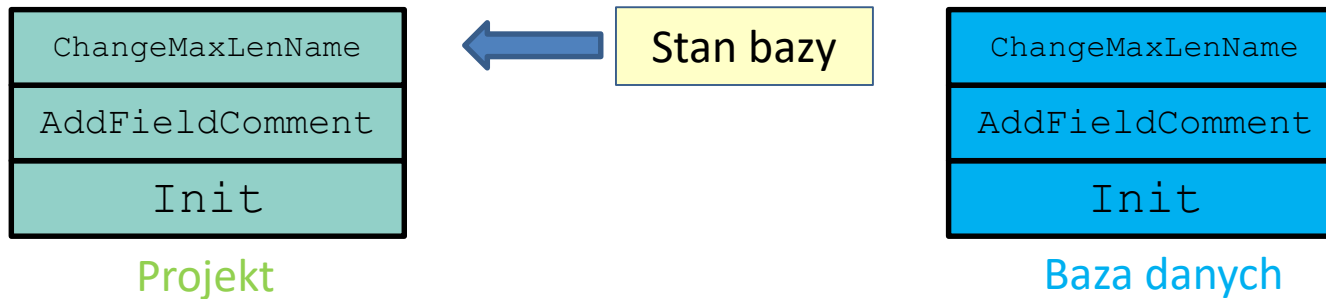
Stan bazy

ChangeMaxLenName
AddFieldComment
Init

Baza danych

Krok 5

- Wycofanie ostatniej migracji AddFieldWhat
 - Remove-migration
 - Powrót do stanu projektu jak przed krokiem 3
 - Należy ręcznie usunąć właściwość What z klasy Student
 - Próba stworzenie nowej migracji powinna stworzyć migrację z pustymi metodami Up () i Down ()



Krok 6

- Wycofanie migracji ChangeMaxLenName
 - Remove-migration
 - Niepowodzenie, stan bazy danych nie pozwala

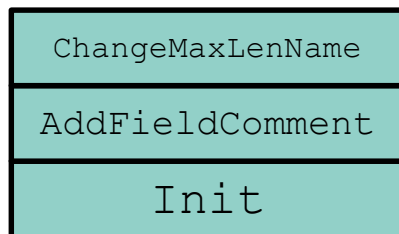
```
PM> remove-migration
Build started...
Build succeeded.
The migration '20201214092555 ChangeMaxLenName' has already been applied to the database. Revert it
and try again. If the migration has been applied to other databases, consider reverting its
changes using a new migration instead.
PM>
```

Krok 7

- Aktualizacja bazy do Init (wykonanie metod Down () z dwóch migracji)
 - Update-database Init
 - Stan jak po kroku 2

dbo.Student [Design] 20201214092555_...ngeMaxLenName.cs Student.c					
Update Script File: dbo.Student.sql					
	Name	Data Type	Allow Nulls	Default	
	Id	int	<input type="checkbox"/>		
	Index	int	<input type="checkbox"/>		
	Name	nvarchar(20)	<input type="checkbox"/>		
	Gender	int	<input type="checkbox"/>		
	Active	bit	<input type="checkbox"/>		
	DepartmentId	int	<input type="checkbox"/>		
	BirthDate	datetime2(7)	<input type="checkbox"/>		
			<input type="checkbox"/>		

dbo._EFMigrationsHistory [Data] 20201214092555_...ngeMaxLen		
Max Rows: 1000		
	MigrationId	ProductVersion
	20201213215422_Init	5.0.1
▶*	NULL	NULL



Projekt



Stan bazy



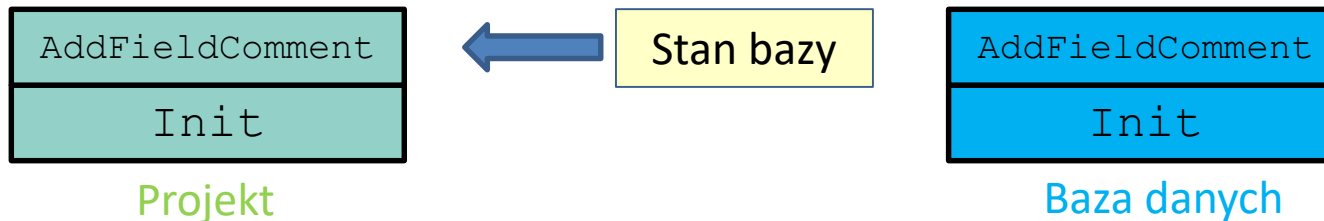
Baza danych

Krok 8

- Wycofanie z sukcesem migracji `ChangeMaxLenName`
 - `Remove-migration ChangeMaxLenName`
 - Stan jako po kroku 1

Krok 9

- 9) Aktualizacja bazy do `AddFieldComment`
 - `Update-database`
 - Już nie ma więcej migracji wyższych migracji



Migracje – informacje inne

- EF analizuje model w kontekście i tworzy kod migracji na podstawie zmiany.
- Jeśli zmian będzie za dużo może zaproponować niewłaściwe operacje na bazie danych
 - Własnoręczne poprawianie metod `Up()` i `Down()`
 - Wykonywanie migracji po każdej zmianie modelu i, gdy już wszystkie pojedyncze zmiany zostały zapamiętane jako oddzielne migracje, wykonać aktualizację bazy danych do ostatniej migracji.
- Drobne zmiany migracyjne pozwalają też łatwiej cofnąć się do pośredniej postaci
- Wada drobnych zmian – mogą być mniej wydajne niż jedna duża zmiana
- Jeśli nastąpiła zmiana w modelu domenowym, w którejś migracji ta zmiana musi być obecna.
 - Decyduje programista kiedy, gdzie i czy w ogóle.
- Usuwanie bazy:
 - `Drop-database`
- Może być wiele kontekstów baz danych, wiele projektów, wiele startup-ów: opcje do wybrania odpowiednich elementów

Dane początkowe Seed()

- Korzystając z metody w kontekście:
 - **protected override void** OnModelCreating(ModelBuilder modelBuilder)
- Można np. dodać kod zapewniający, że pewne dane pojawią się w pustej tabeli
 - Można tak stworzyć pierwszego użytkownika (admina).
 - Najlepiej wykonać to jako metodę rozszerzając

```
namespace WebAppForEntityFrameworkDemo.Data
{
    public class MyDbContext:DbContext
    {
        // ... rest of code
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Seed();
        }
    }
}
```

```
public static class ModelBuilderExtension {
    public static void Seed(this ModelBuilder modelBuilder) {
        modelBuilder.Entity<Student>().HasData(
            new Student() {
                Id = 1,
                Index = 123456,
                Name = "Newman",
                Gender = Gender.Male,
                BirthDate = new DateTime(1999, 10, 10),
                DepartmentId = 1,
                Active = true,
            }, new Student() {
                Id = 2,
                Index = 222222,
                Name = "Yasmin",
                Gender = Gender.Female,
                BirthDate = new DateTime(2000, 2, 2),
                DepartmentId = 2,
                Active = false,
            }
        );
    }
}
```

Najlepiej stworzyć migrację np.
AddExampleStudents
i zaktualizować bazę danych

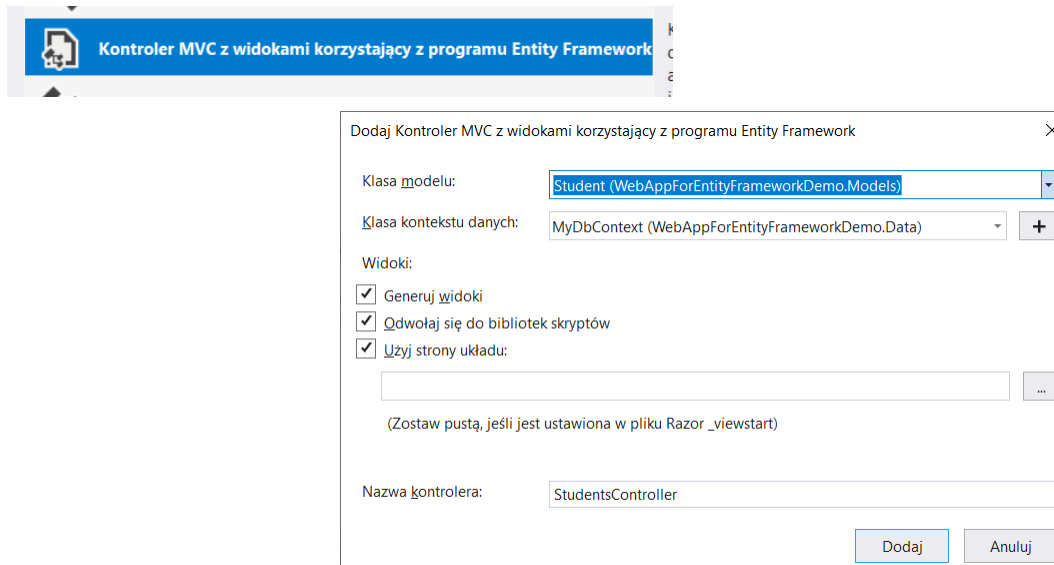
dbo.Student [Data] 20201216142228_A...xampleStudents.cs ModelBuilde							
Max Rows: 1000							
	Id	Index	Name	Gender	Active	DepartmentId	BirthDate
	1	123456	Newman	1	True	1	1999-10-10 00:...
	2	222222	Yasmin	0	False	2	2000-02-02 00:...
►*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Entity Framework

Kontrolery i widoki dla tabel.

Kontroler dla EF

- Stworzenie szkieletu kontrolera dla Entity Framework poprzez wybranie opcji „Kontroler interfejsu API z akcjami używającymi narzędzia Entity Framework”
- Jako klasę modelu wybieramy `Student`, jako klasę kontekstu danych wybieramy `MyDbContext`.
- Domyślnie tworzony jest kontroler o nazwie klasy z dodaną literą `-s` (liczba mnoga).
 - Można zmienić
- Ponieważ wybrana baza danych to `sqlite`, może zostać doinstalowany pakiet dla tej implementacji bazy SQL.
- **W jednym kroku** wytworzony zostanie kontroler wraz z akcjami CRUD oraz z widokami dla wszystkich akcji.
- Uwaga: Zrzuty z ekranu byłyby podobne jak dla poprzedniego wykładu (działającego na kolekcji w pamięci komputera)



Przetwarzanie asynchroniczne C#

Klasa Task, słowa kluczowe async/await

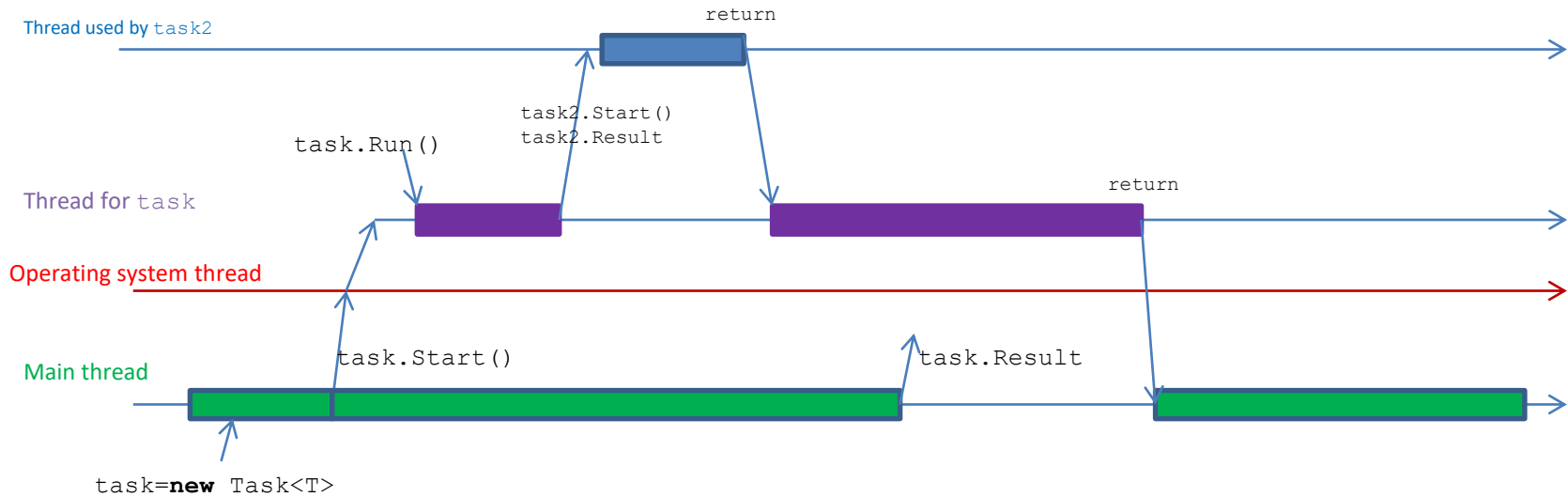
async, await, Task itp.

W dużym uproszczeniu

- Klasa `Task<>` służy do przygotowania zadania do uruchomienia współbieżnego
 - W obiekcie `task` klasy `Task<T>` będzie uruchamiana **nadpisana** metoda `Run()` lub bezparametryczny delegat podany np. w konstruktorze zadania.
 - Uruchamiana jest metoda `Start()`, która przygotowuje nowy wątek, w którym zostanie uruchomiona metoda `Run()`. Jednak już po przygotowaniu do uruchomienia wątku metoda `Start()` kończy swoje działanie
 - Zatem nie mamy wpływu na to, kiedy wykona się metoda `Run()`
 - Na zakończenie działania metody typu `void` można poczekać poprzez `task.Wait()` lub jeśli metoda zwraca typ `T` poprzez odebranie wyniku `task.Result`.
 - Jeśli zadanie się już skończyło, wynik będzie uzyskany natychmiast
 - Jeśli zadanie się nie skończyło, wątek wywołujący `task.Result` jest zawieszony do czasu zakończenia przetwarzania zadania `task`.
 - Diagram poglądowy na następnym slajdzie
- Dużo czasochłonnych metod (np. dostęp do bazy danych) ma swoje wersje z końcówką `Async`:
 - `_context.Student.FirstOrDefault(m => m.Id == id)`
 - `_context.Student.FirstOrDefaultAsync(m => m.Id == id)`
- Jeśli wersja **bez** `Async` zwracała wartość typu `T`, to wersja **z** `Async` zwraca `Task<T>`.
- Słowa kluczowe **async, await** (od C#5.0) ułatwiają szybsze zapisywanie powyżej opisanych kroków w bardziej przejrzysty sposób.

Linia czasu wątków

- Linia czasu wątków



Entity Framework

Przetwarzania asynchroniczne zapytań bazodanowych

async i await - przykład

- Załóżmy, że metoda X uruchomiła poniższą metodę `Details()`
- Słowo kluczowe **async** przed metodą oznacza, że będzie to metoda asynchroniczna zwracająca zadanie `Task<>`
 - W przykładzie to `Task<IActionResult>`
 - Może to być wykonane jak opisano na poprzednim slajdzie, albo jak poniżej
- Słowo kluczowe **await**
 - Tworzy nowe zadanie z kodem metody `Run()` do końca metody `Details()`
 - Uruchamia to zadanie i wątek wraca do wykonywania kolejnych instrukcji metody X

Odwwołania: 0

```
public async Task<IActionResult> Details(int? id)
```

```
{
```

```
    if (id == null)
    {
        return NotFound();
    }
```

Kod wykonany w ramach wątku dla metody X

```
    var student = await _context.Student.FirstOrDefaultAsync(m => m.Id == id);
    if (student == null)
    {
        return NotFound();
    }

    return View(student);
}
```

Metoda `Run()` zadania wynikowego wykonywana w nowym wątku

Wersja synchroniczna akcji Details()

- Wersja synchroniczna, jeden wątek przetwarzania (nazwijmy go wątekX)
 - W trakcie działania `_context.Student.FirstOrDefaultAsync` tworzy się nowy wątek (wątekY), ale ponieważ czekamy na wynik poprzez użycie `.Result`, zatem wątekX musi poczekać, aż wątekY zakończy działanie – brak zysku
 - Zamieniając również tutaj na **wykonanie synchroniczne** (bez końcówki `Async`) poprzez:
`var student = _context.Student.FirstOrDefault(m => m.Id == id);`
nie tracimy czasu na stworzenie wątku i jego usunięcie.

```
public IActionResult Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var student = _context.Student.FirstOrDefault(m => m.Id == id).Result;
    if (student == null)
    {
        return NotFound();
    }

    return View(student);
}
```

Entity Framework

Wytworzony kontroler i widoki

Analiza kodu kontrolera

- Kontroler posiada już wstrzyknięty kontekst bazy danych i pole prywatne do jego przechowywania:
 - **private readonly** MyDbContext _context;
 - **public** StudentsController(MyDbContext context)
- Metody/akcje korzystające z kontekstu bazy danych są w większości metodami asynchronicznymi
 - **async** Task<IActionResult>
- Łączenie danych (data binding) jest bardziej szczegółowe (adnotacja [Bind]):
 - Edit(**int** id, [Bind("Id,Index,Name,Gender,Active,DepartmentId,BirthDate")] Student student)
- Dodanie do kontekstu zamiast dodawanie do odpowiedniego DbSet-u (metoda generyczna kontekstu)
 - _context.Add(student);
- Po zmianie danych w kontekście należy wymusić aktualizację danych w bazie danych poprzez **await** _context.SaveChangesAsync();
- Przykłady:

```
[HttpPost]
[ValidateAntiForgeryToken]
Odwolań: 0
public async Task<IActionResult> Create([Bind("Id,Index,Name,Gender,Active,DepartmentId,BirthDate")] Student student)
{
    if (ModelState.IsValid)
    {
        _context.Add(student);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(student);
}
```

```
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
Odwolań: 0
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var student = await _context.Student.FindAsync(id);
    _context.Student.Remove(student);
    await _context.SaveChangesAsync();
    return RedirectToAction(nameof(Index));
}
```

```
1 odwołanie
private bool StudentExists(int id)
{
    return _context.Student.Any(e => e.Id == id);
}
```

Analiza Widoków

- Więcej tag-helperów
- Parametr Id ustawiony na właściwą wartość Id z modelu domenowego
- Np. Index.cshtml:

```
<td>
  <a asp-action="Edit" asp-route-id="@item.Id">Edit</a> |
  <a asp-action="Details" asp-route-id="@item.Id">Details</a> |
  <a asp-action="Delete" asp-route-id="@item.Id">Delete</a>
</td>
..
```

- W widoku Create.cshtml nie ma kontrolki do wpisania Id
- W widoku Edit.cshtml jest pole ukryte dla Id
 - Wysyłane w żądaniu POST wraz z edytowalnymi polami

```
<form asp-action="Edit">
  <div asp-validation-summary="ModelOnly" class="text-danger"></div>
  <input type="hidden" asp-for="Id" />
  <div class="form-group">
    <label asp-for="Index" class="control-label"></label>
    <input asp-for="Index" class="form-control" />
  </div>
</form>
```

- Kontrolki dla Gender nadal należy poprawić samemu

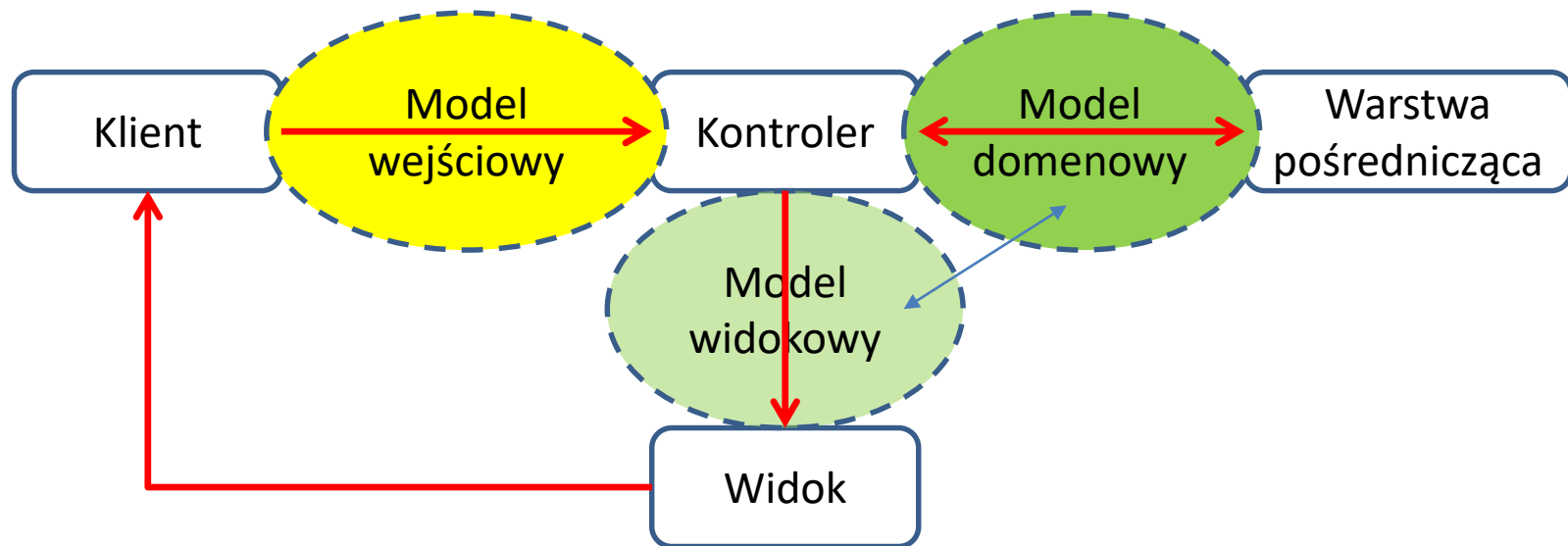
```
<div class="form-group">
  <label asp-for="Gender" class="control-label"></label>
  <select asp-for="Gender" class="custom-select" asp-items="Html.GetEnumSelectList<Gender>()">
    <option value="">Please select</option>
  </select>
  <span asp-validation-for="Gender" class="text-danger"></span>
</div>
```

MVC - modele

Model Wejściowy

Przypomnienie - modele danych w MVC

- Modele danych w MVC dzielimy na:
 - **Model wejściowy**
 - Między klientem a kontrolerem: reprezentuje zestaw danych przesyłanych przez klienta do kontrolera z wykorzystaniem formularza lub URL.
 - **Model domenowy**
 - Między kontrolerem a warstwą pośredniczącą (np. dostęp do bazy danych)
 - **Model widokowy**
 - Między kontrolerem a widokiem: Dane oparte o ten model, odpowiednio przetworzone, trafiają do klienta w postaci kodu HTML.



Przykład użycia modelu widokowego

- Chcemy zamiast zmuszać użytkownika do wstawiania daty w odpowiednim formacie, aby miał 3 osobne kontrolki dla roku, miesiąca i dnia
- Tworzymy klasę modelu widoku `StudentCreateViewModel`

```
namespace WebAppForEntityFrameworkDemo.ViewModels
{
    public class StudentCreateViewModel
    {
        public int Id { get; set; }
        [Required]
        [RegularExpression(@"^[0-9]{1,6}$", ErrorMessage = "Write from 1 to 6 digits")]
        public int Index { get; set; }
        [Required]
        [MinLength(2, ErrorMessage="To short name")]
        [Display(Name="Last Name")]
        [MaxLength(20,ErrorMessage = " To long name, do not exceed {1}")]
        public string Name { get; set; }
        public Gender Gender { get; set; }
        public bool Active { get; set; }
        public int DepartmentId { get; set; }
        [Required]
        [Range(1900,2100)]
        public int Year{ get; set; }
        [Required]
        [Range(1,12)]
        public int Month { get; set; }
        [Required]
        [Range(1, 31)]
        public int Day { get; set; }
    }
}
```

Pozostałe zmiany

- Tworzymy widok Create dla nowej klasy StudentCreateViewModel
 - Tworzenie widoku generatorem kodu dla EF wymaga podania kontekstu, co spowoduje dodanie nadmiarowego DbSet<>
 - Poprawić kontrolkę dla Gender
- Zmienić akcję Create dla POST, aby odebrać dane z ciała żądania jako StudentCreateViewModel.
 - Zmiana w nagłówku metody
 - Dopisać przepakowanie danych do obiektu klasy Student
 - Kod do przepakowania 3 danych do obiektu klasy DateTime

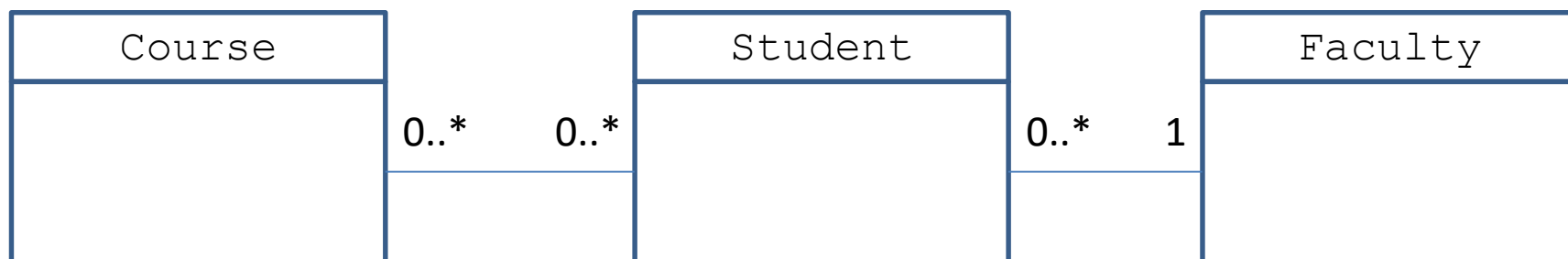
```
[HttpPost]
[ValidateAntiForgeryToken]
Odwołania: 0
public async Task<IActionResult> Create(
    [Bind("Id,Index,Name,Gender,Active,DepartmentId,Year,Month,Day")] StudentCreateViewModel studentView)
{
    if (ModelState.IsValid)
    {
        Student student = new Student()
        {
            Id = studentView.Id,
            Name = studentView.Name,
            Index=studentView.Index,
            Active=studentView.Active,
            DepartmentId=studentView.DepartmentId,
            Gender=studentView.Gender,
            BirthDate=new DateTime(studentView.Year,studentView.Month,studentView.Day)
        };
        _context.Add(student);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    return View(studentView);
}
```

Entity Framework

Relacje między tabelami/DbSet-ami

EF-Code First – powiązania między DbSet-ami

- `Course-Student-Faculty` (w folderze `Models`)
 - Właściwość `Id`, `ID`, `id` (lub właściwość kończąca się na taki ciąg) jest automatycznie oznaczana jako klucz główny.
 - Lub użyć adnotacji `[Key]`
- Przykład zawiera relację „1 do wielu” oraz „wiele do wielu”.



- Zakładamy, że chcemy sami stworzyć widoczne pola połączone z kluczami obcymi w każdej z klas
 - Jeśli tego nie stworzymy EF Core po znalezieniu relacji między klasami w kontekście stworzy pola kluczy obcych w bazie danych, ale z poziomu C# nie będzie do nich dostępu.
- Dla uproszczenia przykładu każda z klas będzie miała tylko nazwę
- Dla łatwiejszej obserwacji będą to np. `FacultyName`, a nie po prostu `Name`

EF-CF-prosty przykład 2/2

- Teoretycznie EF 5.0 wspiera tworzenie relacji „wiele do wielu” po stworzeniu klas jak poniżej
 - Ale może nie zadziałać
 - Tworzy w bazie tabelę niewidoczną (możliwy dostęp, ale złożony) dla programisty

```
public class Course
{
    public int CourseId { get; set; }
    [Required]
    [MaxLength(40)]
    public string CourseName { get; set; }

    ICollection<Student> Student { get; set; } // for many-to-many relaction
}

public class Student
{
    public int StudentId { get; set; }
    [Required]
    [MaxLength(40)]
    public string LastName { get; set; }

    public ICollection<Course> Course { get; set; } // for many-to-many relaction

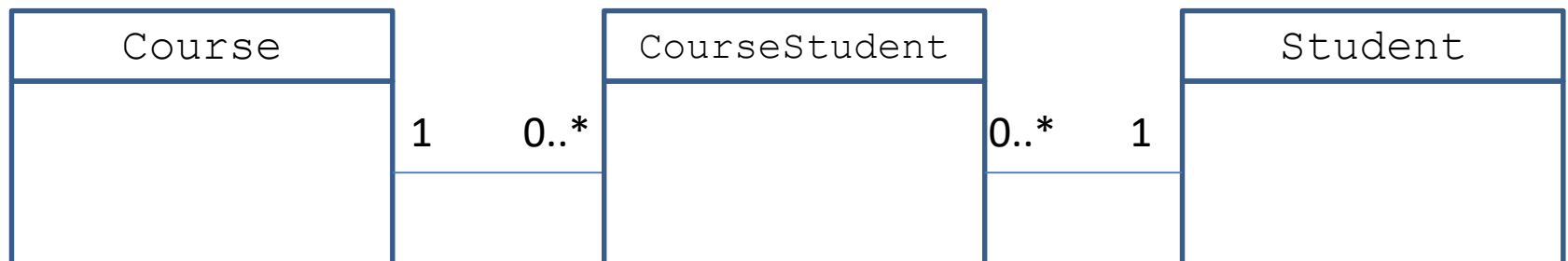
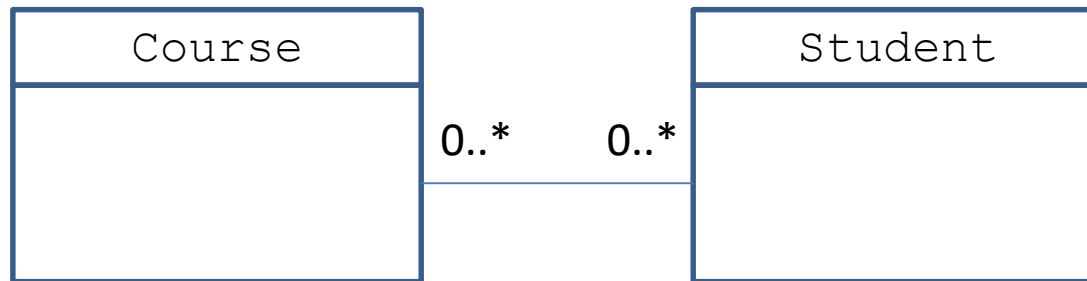
    public int FacultyId { get; set; } // for one-to-many relaction
    public Faculty Faculty { get; set; } // for one-to-many relaction
}

public class Faculty
{
    public int FacultyId { get; set; }
    [Required]
    [MaxLength(40)]
    public string FacultyName { get; set; } // can be Name

    public ICollection<Student> Student { get; set; } // for one-to-many relaction
}
```

Podejście tworzenia własnej relacji „wiele do wielu”

- Założenie: chcemy mieć dostęp do tabeli tworzącej relację „wiele do wielu”
- Relacja „wiele do wielu” to tak naprawdę połączenie dwóch relacji „jeden do wielu”, które pamiętane jest w nowej tabeli
 - Kluczem jest para kluczy z wyjściowych tabel



Własne definiowanie relacji

- Aby własnoręcznie zdefiniować relacje używa się nadpisanej metody w klasie kontekstu:
 - **protected override void** OnModelCreating(ModelBuilder modelBuilder)
- Używa się Fluent API klasy `ModelBuilder`
- Należy stworzyć klasę pośredniczącą dla złożenia dwóch relacji „jeden do wielu”
- I użyć jej w klasach `Student` i `Course`

```
public class CourseStudent{  
    // public int Id { get; set; } // we will use complex key  
    public int CourseId { get; set; }  
    public Course Course { get; set; }  
  
    public int StudentId { get; set; }  
    public Student Student { get; set; }  
}
```

```
public class Course  
{  
    public int CourseId { get; set; }  
    [Required]  
    [MaxLength(40)]  
    public string CourseName { get; set; }  
    public ICollection<CourseStudent> CourseStudents { get; set; }  
}
```

```
public class Student  
{  
    public int StudentId { get; set; }  
    [Required]  
    [MaxLength(40)]  
    public string LastName { get; set; }  
  
    public ICollection<CourseStudent> CourseStudents { get; set; }  
  
    public int FacultyId { get; set; }  
    public Faculty Faculty { get; set; }  
}
```

Modyfikacja kontekstu

- Dodanie nowego DbSet-u.
- Zapisanie relacji za pomocą modelBuilder-a

```
public class UniversityDbContext:DbContext
{
    public UniversityDbContext(DbContextOptions<UniversityDbContext> options) : base(options)
    {
    }
    public DbSet<Student> Students { get; set; }
    public DbSet<Faculty> Faculties { get; set; }
    public DbSet<Course> Courses { get; set; }

    public DbSet<CourseStudent> CourseStudent { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<CourseStudent>()
            .HasKey(t => new { t.CourseId, t.StudentId });

        modelBuilder.Entity<CourseStudent>()
            .HasOne(cs => cs.Course)
            .WithMany(p => p.CourseStudents)
            .HasForeignKey(cs => cs.CourseId);

        modelBuilder.Entity<CourseStudent>()
            .HasOne(cs => cs.Student)
            .WithMany(s => s.CourseStudents)
            .HasForeignKey(cs => cs.StudentId);
    }
}
```


Użycie wiązania

- Przegląd kodu wygenerowanego: relacje „jeden do wielu” jako lista rozwijalna
- Dopisany kod dla zapisów jako przykład relacji „wiele do wielu”
 - Akcja `CreateEnroll` dla GET i POST w kontrolerze `Student-a`
 - Widok dla tej akcji
 - Modyfikacja widoku akcji `Details()` i jej widoku.

```
1 odwołanie
public IActionResult CreateEnroll()
{
    ViewData["StudentId"] = new SelectList(_context.Students, "StudentId", "LastName");
    ViewData["CourseId"] = new SelectList(_context.Courses, "CourseId", "CourseName");
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
1 odwołanie
public async Task<IActionResult> CreateEnroll([Bind("StudentId,CourseId")] CourseStudent courseStudent)
{
    try
    {
        if (ModelState.IsValid)
        {
            _context.CourseStudent.Add(courseStudent);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }
        return View(nameof(Index));
    }
    catch (DbUpdateException)
    {
        TempData["Error"] = "Duplicate enrollment";
        return RedirectToAction(nameof(CreateEnroll));
    }
}
```

Create Enrolment

Choose student and course

Duplicate enrollment

StudentId

Student 1

CourseId

Course 1

Enroll

[Back to List](#)

```
@model WebAppEntityFrameworkRelations.Models.CourseStudent
@{
    ViewData["Title"] = "Create Enrolment";
}
<h1>@ViewData["Title"]</h1>
<h4>Choose student and course</h4>
<hr />
<div class="row">
    <div class="col-md-4">
        <div class="text-danger">@TempData["Error"]</div>
        <form asp-action="CreateEnroll">
            <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            <div class="form-group">
                <label asp-for="StudentId" class="control-label"></label>
                <select asp-for="StudentId" class="form-control" asp-items="ViewBag.StudentId"></select>
            </div>
            <div class="form-group">
                <label asp-for="CourseId" class="control-label"></label>
                <select asp-for="CourseId" class="form-control" asp-items="ViewBag.CourseId"></select>
            </div>
            <div class="form-group">
                <input type="submit" value="Enroll" class="btn btn-primary" />
            </div>
        </form>
    </div>
    <div>
        <a asp-action="Index">Back to List</a>
    </div>
</div>
@section Scripts {
    @await Html.RenderPartialAsync("_ValidationScriptsPartial");
}
```

Details – akcja i widok

```
@model WebAppEntityFrameworkRelations.Models.Student

@{
    ViewData["Title"] = "Details";
}

<h1>Details</h1>

<div>
    <h4>Student</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.LastName)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.LastName)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Faculty)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Faculty.FacultyName)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.CourseStudents)
        </dt>
        <dd class="col-sm-10">
            @foreach (var item in ViewData["Courses"] as IEnumerable<Course>)
            {
                <p>@item.CourseId, @item.CourseName</p>
            }
        </dd>
    </dl>
</div>
<div>
    <a asp-action="Edit" asp-route-id="@Model.StudentId">Edit</a> |
    <a asp-action="Index">Back to List</a>
</div>
```

```
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var student = await _context.Students
        .Include(s => s.Faculty)
        .FirstOrDefaultAsync(m => m.StudentId == id);
    ViewData["Courses"] = _context.CourseStudent
        .Where(c => c.StudentId == id)
        .Join(_context.Courses, cs => cs.CourseId, c => c.CourseId, (cs, c) => c)
        .ToList();

    if (student == null)
    {
        return NotFound();
    }

    return View(student);
}
```

Details

Student

LastName	Student 1
Faculty	Faculty 1
CourseStudents	1, Course 1
	2, Course 2
	4, Course 4

[Edit](#) | [Back to List](#)

DbSet<>

- Klasy `DbContext` i `DbSet<>` ukrywają dostęp do tabel bazy danych w możliwie optymalny sposób:
 - Zacztyją z bazy tylko te dane, które potrzeba w danej chwili
 - Jeśli wcześniejsze zapytanie (w ramach tego samego żądania HTTP, lub w ramach puli kontekstów) już pewne kolekcje pobrała z bazy danych, nie będą ponownie pobierane
 - Zapisują dane do bazy tylko na wyraźną potrzebę
 - Stara się je wykonać w najmniejszej liczbie zapytań do bazy
- Starają się z jednej strony ograniczyć częstotliwość dostępu do bazy, a z drugiej strony - ilość danych przechowywaną w kolekcjach w pamięci komputera.

Entity Framework

Podejście Database First - ogólnie

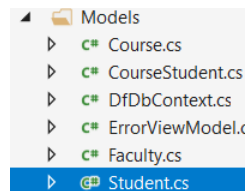
EF Core – Database First

- Podejście to zakłada istnienie bazy danych.
- Polega na wykonaniu w konsoli pakietów NuGet komendy `Scaffold-DbContext`, po której następuje connection string oraz nazwa providera bazy danych.
- Warto też dodać np. opcję `-OutputDir <folderName>`, aby tworzył klasy w wybranym folderze
- Nazwy klas to nazwy tabel bez końcówki `-s`.
- Standardowo wszelkie ograniczenia, relacje itd. zapisuje w postaci Fluent Api:
 - Można zmienić na adnotacje opcją `-DataAnnotations`
- Standardowa **nazwa klasy** kontekstu zawiera **całą ścieżkę** dostępu do bazy, warto zmienić to opcja `-Context <className>`
- Scenariusz użycia:
 - Wykonać `Scaffold-DbContext` bez opcji `-DataAnnotations`
 - Analiza instrukcji Fluent API
 - Wykonać `Scaffold-DbContext` z opcją `-DataAnnotations`
 - Analiza instrukcji Fluent API
 - Analiza adnotacji
- Oczywisty brak migracji
- Brak dodania kontekstu do kontenera
- Nie wytwarza się connection string – ostrzeżenie, że warto go stworzyć
- Relacje „wiele do wielu” nie zawsze dobrze zinterpretowane
 - Zawsze z tabelą pośredniczącą w kontekście

Przykładowe komendy Scaffold-DbContext

- Nowy projekt z zainstalowanym EF Core
- Scaffold-DbContext "Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=UnivDb;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Context DfDbContext

```
PM> Scaffold-DbContext "Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=UnivDb;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Context DfDbContext
Build started...
Build succeeded.
To protect potentially sensitive information in your connection string, you should move it out of source code and use environment variables or the Data Protection library. For more guidance on storing connection strings, see http://go.microsoft.com/fwlink/?LinkId=2131148. For more guidance on storing connection strings, see http://go.microsoft.com/fwlink/?LinkId=723263.
PM>
```



```
public partial class Student
{
    Odwołania: 0
    public Student()
    {
        CourseStudents = new HashSet<CourseStudent>();
    }

    Odwołania: 0
    public int StudentId { get; set; }
    1 odwołanie
    public string LastName { get; set; }
    Odwołania: 2
    public int FacultyId { get; set; }

    1 odwołanie
    public virtual Faculty Faculty { get; set; }
    Odwołania: 2
    public virtual ICollection<CourseStudent> CourseStudents { get; set; }
}
```

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
    if (!optionsBuilder.IsConfigured)
    {
        #warning To protect potentially sensitive information in your connection string, you should move
        optionsBuilder.UseSqlServer("Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=UnivDb;Integrated Security=True");
    }
}

Odwołania: 0
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.HasAnnotation("Relational:Collation", "SQL_Latin1_General_CP1_CI_AS");

    modelBuilder.Entity<Course>(entity =>
    {
        entity.Property(e => e.CourseName)
            .IsRequired()
            .HasMaxLength(40);
    });

    modelBuilder.Entity<CourseStudent>(entity =>
    {
        entity.HasKey(e => new { e.CourseId, e.StudentId });
    });
}
```

Przykładowe komendy Scaffold-DbContext z adnotacjami

- Scaffold-DbContext "Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=UnivDb;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Context DfDbContext **-DataAnnotations**

```
public partial class DfDbContext : DbContext
{
    Odwołania: 0
    public DfDbContext()
    {
    }

    Odwołania: 0
    public DfDbContext(DbContextOptions<DfDbContext> options)
        : base(options)
    {
    }

    Odwołania: 0
    public virtual DbSet<Course> Courses { get; set; }
    Odwołania: 0
    public virtual DbSet<CourseStudent> CourseStudents { get; set; }
    Odwołania: 0
    public virtual DbSet<Faculty> Faculties { get; set; }
    Odwołania: 0
    public virtual DbSet<Student> Students { get; set; }

    Odwołania: 0
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        if (!optionsBuilder.IsConfigured)
        {
            To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid scaffolding the connection string by using the Name= syntax to read it from configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see http://go.microsoft.com/fwlink/?linkid=723263.
            optionsBuilder.UseSqlServer("Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=UnivDb;Integrated Security=True");
        }
    }

    Odwołania: 0
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.HasAnnotation("Relational:Collation", "SQL_Latin1_General_CP1_CI_AS");

        modelBuilder.Entity<CourseStudent>(entity =>
        {
            entity.HasKey(e => new { e.CourseId, e.StudentId });
        });

        OnModelCreatingPartial(modelBuilder);
    }

    1 odwołanie
    partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
}
```

```
PM> Scaffold-DbContext "Data Source=(localdb)\MSSQLLocalDB;Initial Catalog=UnivDb;Integrated Security=True" Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Context DfDbContext -DataAnnotations
Build started...
Build succeeded.
To protect potentially sensitive information in your connection string, you should move it out of source code. You can avoid scaffolding the connection string by using the Name= syntax to read it from configuration - see https://go.microsoft.com/fwlink/?linkid=2131148. For more guidance on storing connection strings, see http://go.microsoft.com/fwlink/?linkid=723263.
PM>
```

Models

- Course.cs
- CourseStudent.cs
- DfDbContext.cs
- ErrorViewModel.cs
- Faculty.cs
- Student.cs

```
[Index(nameof(FacultyId), Name = "IX_Students_FacultyId")]
Odwołania: 6
public partial class Student
{
    Odwołania: 0
    public Student()
    {
        CourseStudents = new HashSet<CourseStudent>();
    }

    [Key]
    Odwołania: 0
    public int StudentId { get; set; }
    [Required]
    [StringLength(40)]
    Odwołania: 0
    public string LastName { get; set; }
    Odwołania: 2
    public int FacultyId { get; set; }

    [ForeignKey(nameof(FacultyId))]
    [InverseProperty("Students")]
    1 odwołanie
    public virtual Faculty Faculty { get; set; }
    [InverseProperty(nameof(CourseStudent.Student))]
    1 odwołanie
    public virtual ICollection<CourseStudent> CourseStudents { get; set; }
}
```

Relacje w podejściu Database First

- Analiza wytworzonego kodu
- Relacje jako Fluent API – wszystko w jednej funkcji
 - Bazodanowo bardziej spójne
 - Silnik Razor nie ma informacji przydatnych przy sprawdzaniu poprawności modelu
- Relacje jako adnotacje – rozrzucone po klasach
 - Bazodanowo – trzeba przeglądać klasy aby znaleźć relacje i ograniczenia
 - Silnik Razor posiada część informacji do sprawdzania poprawności modelu
- Brak wsparcia do relacji many to many
 - Takie relacje są reprezentowane jako tabela pośrednicząca z relacjami one-to-many

Informacje różne

- Pola niewymagane:
 - Typ nullowalny: `object` lub `int?` itp.
- Indeksy w bazie danych
 - Nie mogą być dla pól typu „nvarchar(max)”, który jest domyślny dla typu `string`
 - Operacje na tak długim polu są nieefektywne
- Adnotacje dla schematu baz danych
 - Zespół `System.ComponentModel.DataAnnotations.Schema`
 - Zmiana nazwy tabeli w bazie danych:
`[Table("StudentMaster")] public class Student{ ... }`
 - Zmiana nazwy kolumny w tabeli bazy danych:
`[Column("Name")] public string StudentName { get; set; }`
 - `[NotMapped]`
 - Inny typ niż standardowy: `[Column(TypeName=„varchar(10)“)]`
 - itd.
- Jeśli kolejnej części zapytania LINQ nie można uruchomić na serwerze SQL (np. metody napisanej w C#), to w tym miejscu kończy się zapytanie do serwera bazy danych i dopiero po odebraniu wyniku początkowej części zapytania następuje wykonanie reszty w aplikacji ASP .Net.
 - Będzie wyjątek podczas wykonania
 - Należy wywołać `ToList()` i resztę zapytania
 - To może być bardzo nieefektywne!
 - Zamiast własnej metody lepiej używać wprost wyrażenia lambda

Dapper

- Dapper - inny framework ORM
- Bardziej dla znających SQL
 - Należy wprost używać języka SQL
 - Przez to powstaje kod wydajniejszy w działaniu
 - Kod bezpieczniejszy
 - Lepszy dla dużych baz danych (od początku myśli się o efektywności zapytań)
- EF w rękach niedoświadczonego dewelopera
 - Pozwala szybko wygenerować działający kod
 - Nie wymaga znajomości SQL

ale

 - Tworzy ogólnie niebezpieczny kod
 - Jeśli niemożliwy jest dostęp do serwera SQL spoza aplikacji webowej ASP i zapytania nie są dostępne z zewnątrz, to nie jest problem
 - Tworzy (bardzo) nieefektywny kod
 - Jeśli aplikacja nie działa na dużych zestawach danych, brak efektywności może nie być obserwowalna przez użytkownika
- Aby powyższe negatywne cechy wyeliminować trzeba jednak dogłębnie zrozumieć komunikację z serwerem SQL oraz tworzenie zapytań:
 - Wymagana głębsza wiedza z LINQ
 - Wymagana wiedza z SQL
 - Wymagana głębsza wiedza z EF