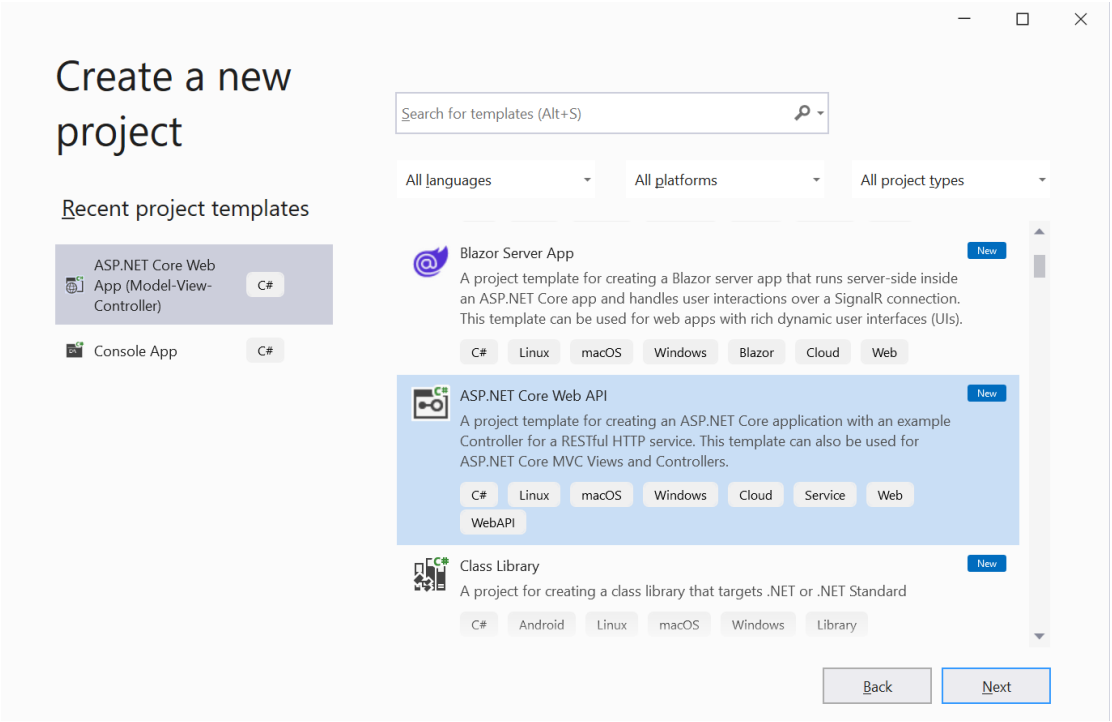
Microsoft .NET:
ASP.NET Web API, Entity Framework (Code First from Database), LINQ, REST

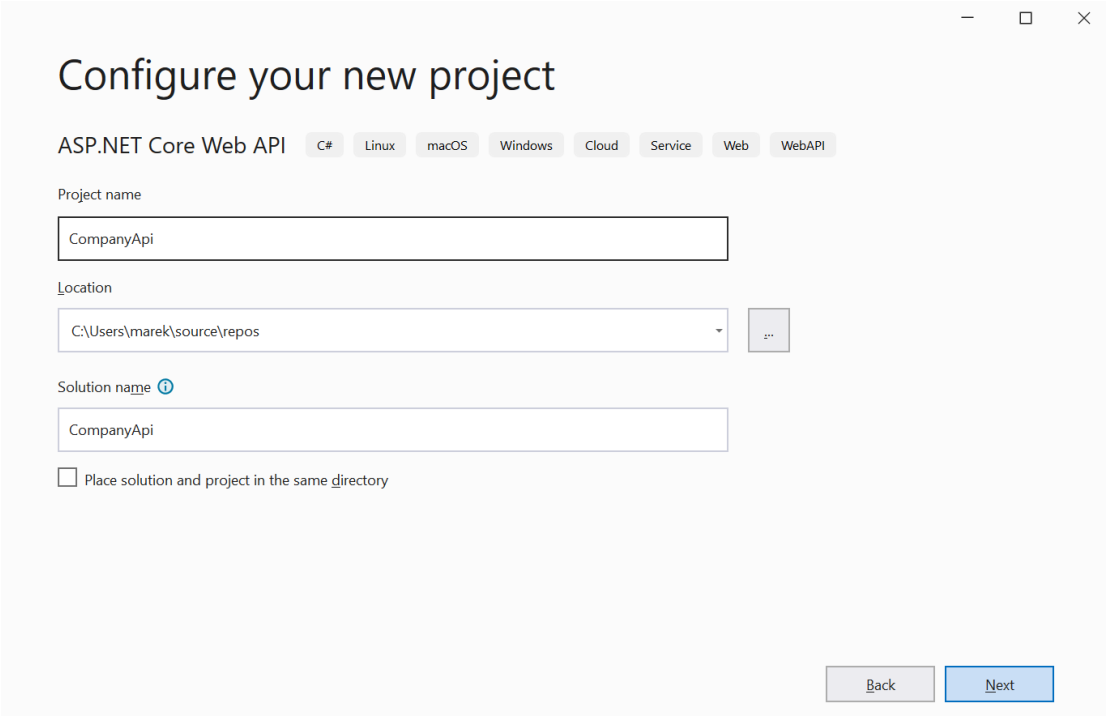
Do realizacji ćwiczeń potrzebne jest zintegrowane środowisko programistyczne Microsoft Visual Studio 2022.

Celem ćwiczeń jest wykorzystanie technologii ASP.NET do implementacji Web API. Warstwa dostępu do danych będzie oparta na Entity Framework z wykorzystaniem strategii Code First from database oraz zapytaniach LINQ.

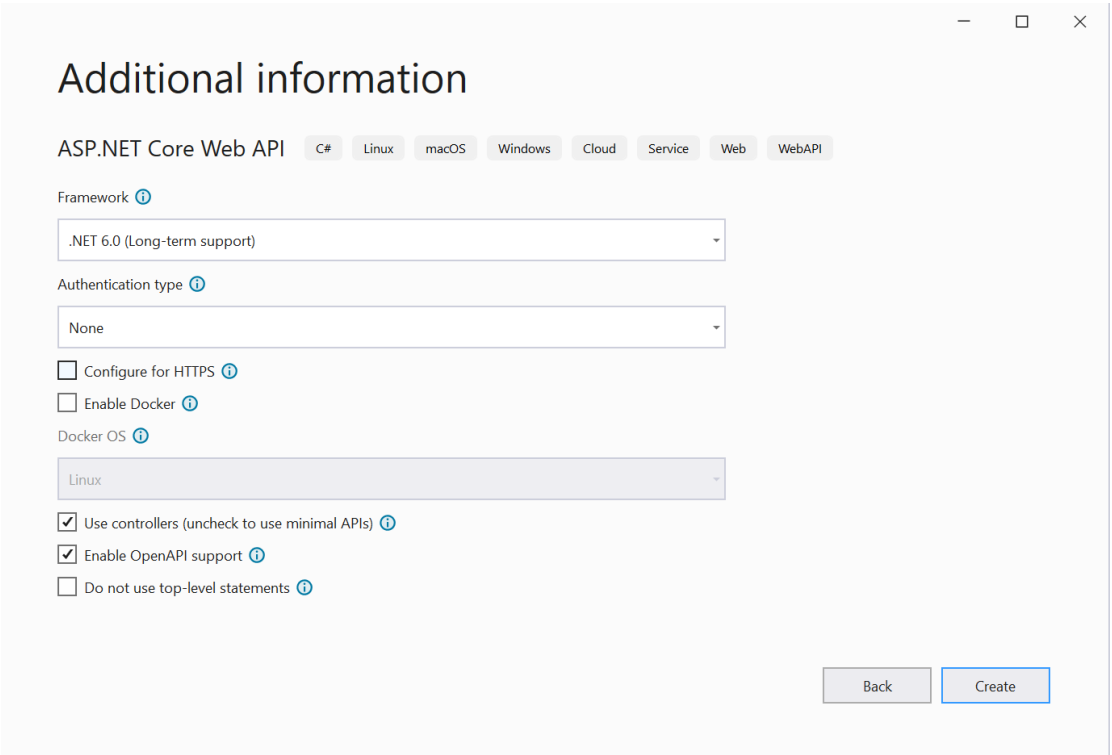
1. Utworzenie nowego projektu (Project).
   a) Uruchom narzędzie Microsoft Visual Studio.
   b) Z menu głównego wybierz File→New→Project (lub Create a new project z ekranu startowego). Wybierz szablon ASP.NET Core Web API. Kliknij przycisk Next.



   c) Zmień proponowaną nazwę projektu na „CompanyApi”. Zaakceptuj zaproponowany katalog lub zmień go na inny gdy nie masz prawa zapisu w proponowanym katalogu. Pozostałe opcje pozostaw domyślne. Kliknij przycisk Next.

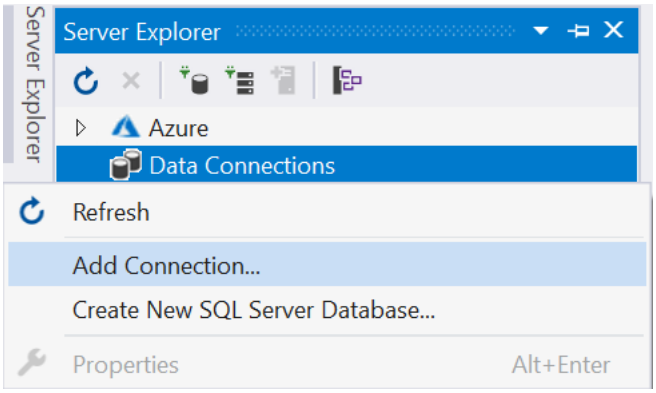


d) W oknie wyboru wersji frameworka .NET wybierz .NET 6.0 (Long-term support). Odznacz pole wyboru Configure for HTTPS. Pozostałe opcje pozostaw domyślne i kliknij przycisk Create.

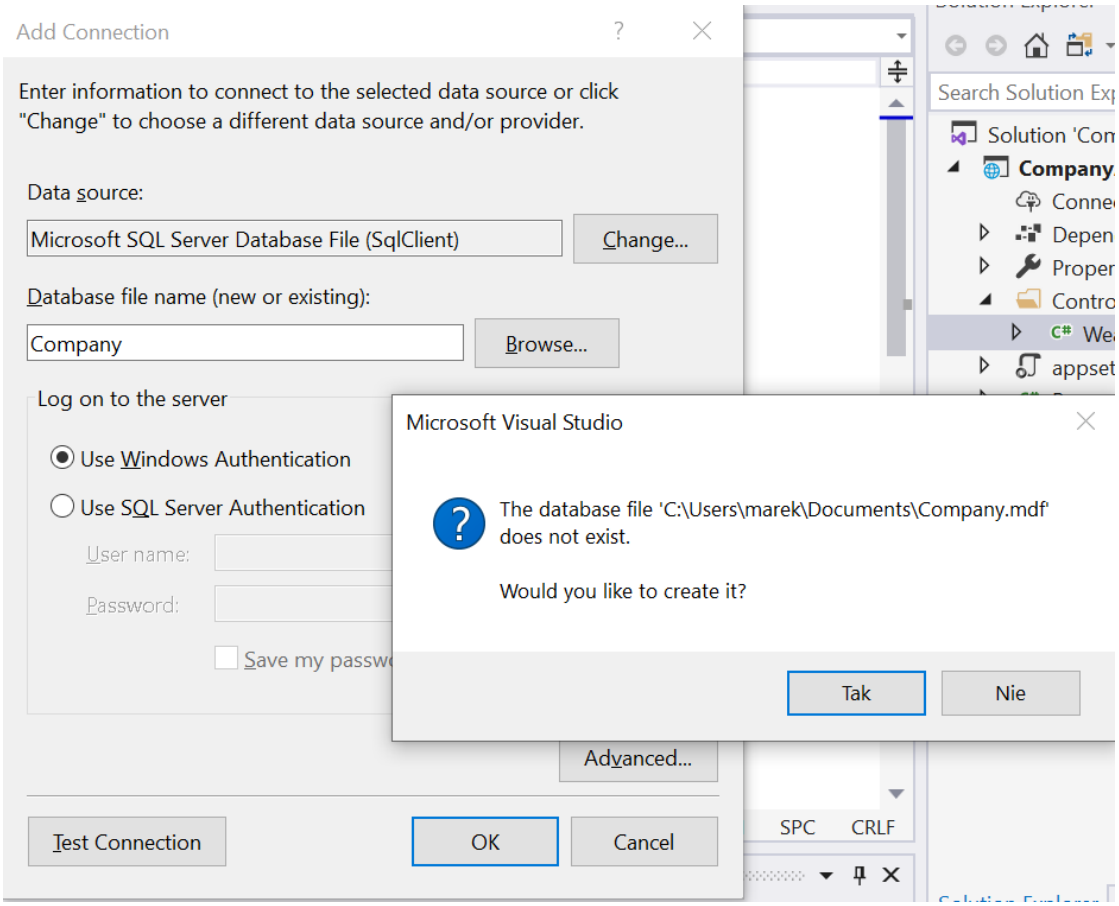


2. Utworzenie bazy danych SQL Server LocalDb

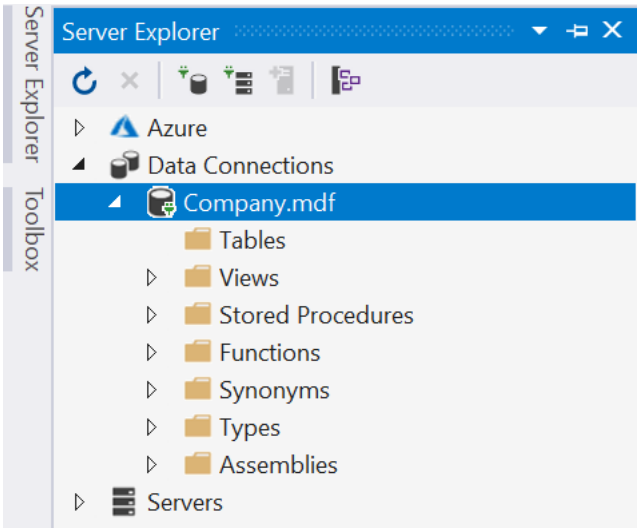
a) Otwórz panel Server Explorer i z poziomu węzła Data Connections wybierz opcję Add Connection.



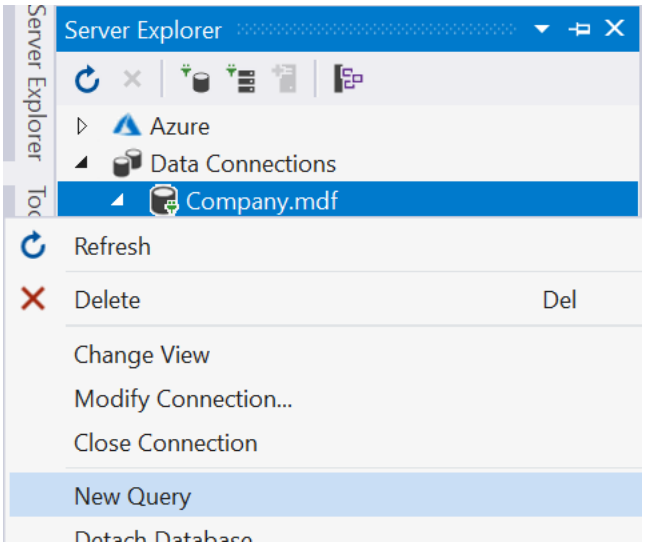
b) W kreatorze nowego połączenia jako Data Source wybierz Microsoft SQL Server Database File (SqlClient), a jako nazwę pliku wprowadź „Company”. Ponieważ taki plik jeszcze istnieje, wraz z definicją połączenia zostanie utworzona nowa baza danych. (Nie klikaj Test Connection gdyż połączenie przed utworzeniem pliku bazy danych nie zadziała.) Kliknij OK. Powinno zostać wyświetlone okienko z pytaniem o utworzenie pliku bazy danych. Odpowiedz twierdząco.



c) Rozwiń węzeł utworzonego połączenia z bazą danych, aby upewnić się, że baza danych w tej chwili nie zawiera żadnych tabel.



d) Kliknij prawym klawiszem myszy na węźle połączenia i wybierz opcję New Query.



e) W oknie poleceń SQL wklej poniższy skrypt tworzący tabele Department i Employee oraz wypełniający je danymi:

```
CREATE TABLE Department
      (DepartmentId INT IDENTITY CONSTRAINT PK_Department PRIMARY KEY,
       Name VARCHAR(20));

CREATE TABLE Employee
   (EmployeeId INT  IDENTITY CONSTRAINT PK_Employee PRIMARY KEY,
      FirstName VARCHAR(15),
      LastName VARCHAR(15),
      ManagerId INT CONSTRAINT FK_Employee_Employee REFERENCES
Employee(EmployeeId),
      Salary DECIMAL(6,2) CONSTRAINT CK_Salary CHECK(Salary>100),
      Bonus DECIMAL(6,2),
      DepartmentId INT CONSTRAINT FK_Employee_Department REFERENCES
Department(DepartmentId));

INSERT INTO Department(Name) VALUES ('Sales');
INSERT INTO Department(Name) VALUES ('Marketing');
```

```
INSERT INTO Department(Name) VALUES ('Research');
INSERT INTO Department(Name) VALUES ('Administration');

INSERT INTO
Employee(FirstName,LastName,ManagerId,Salary,Bonus,DepartmentId)
VALUES ('Daenerys', 'Targaryen', NULL, 8000, 900, 4);
INSERT INTO
Employee(FirstName,LastName,ManagerId,Salary,Bonus,DepartmentId)
VALUES ('Jaime', 'Lannister', 1, 6000, NULL, 4);
INSERT INTO
Employee(FirstName,LastName,ManagerId,Salary,Bonus,DepartmentId)
VALUES ('Jon', 'Snow', 1, 6000, NULL, 4);
INSERT INTO
Employee(FirstName,LastName,ManagerId,Salary,Bonus,DepartmentId)
VALUES ('Jorah', 'Mormont', 2, 3500, 300, 2);
INSERT INTO
Employee(FirstName,LastName,ManagerId,Salary,Bonus,DepartmentId)
VALUES ('Davos', 'Seaworth', 2, 3000, 500, 1);
INSERT INTO
Employee(FirstName,LastName,ManagerId,Salary,Bonus,DepartmentId)
VALUES ('Arya', 'Stark', 3, 4900, NULL, 3);
INSERT INTO
Employee(FirstName,LastName,ManagerId,Salary,Bonus,DepartmentId)
VALUES ('Theon', 'Greyjoy', 3, 5300, NULL, 3);
```

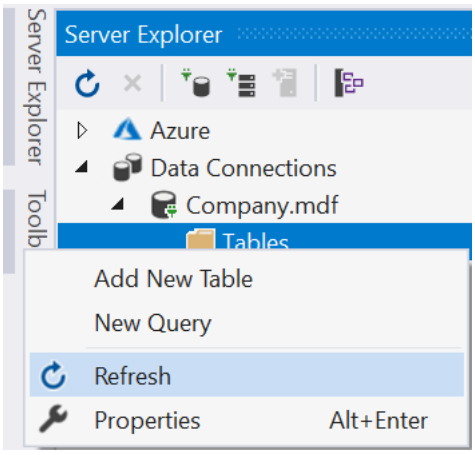
Komentarz: Nazwy tabel w liczbie pojedynczej mogą razić purystów modelu relacyjnego. Jednakże, dzięki przyjętej konwencji, wygenerowane kreatorem klasy encji będą miały poprawne nazwy w liczbie pojedynczej. W przypadku nazw tabel w liczbie mnogiej, w celu uzyskania nazw klas w liczbie pojedynczej należałoby ręcznie poprawić wygenerowany kod, gdyż aktualnie kreator `Scaffold-DbContext` nie oferuje opcji automatycznej konwersji nazw.

f) Przeanalizuj strukturę tworzonych przez skrypt tabel zwracając uwagę na następujące związki reprezentowane przez klucze obce:

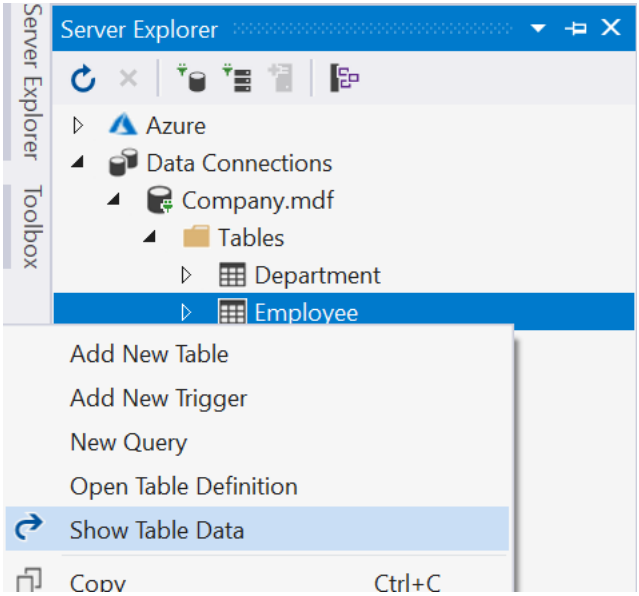
- pracownik może mieć szefa, którym jest inny pracownik
- pracownik może być przypisany do departamentu

g) Uruchom skrypt poprzez kliknięcie ikony Execute. Przejrzyj komunikaty w panelu komunikatów, aby upewnić się, że skrypt wykonał się bez błędów.

h) W panelu Server Explorer odśwież gałąź Tables i upewnij się, że baza danych zawiera teraz tabele Department i Employee.



i) Obejrzyj zawartość utworzonych tabel korzystając z opcji Show Table Data.



3. Dodanie do projektu pakietów Entity Framework i dostawcy usług dla SQL Server.

a) Uruchom konsolę menedżera pakietów opcją Tools -> NuGet Package Manager -> Package Manager Console z głównego menu.

b) Z poziomu konsoli menedżera pakietów uruchom kolejno następujące komendy:

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
Install-Package Microsoft.EntityFrameworkCore.Design
Install-Package Microsoft.EntityFrameworkCore.Tools
```

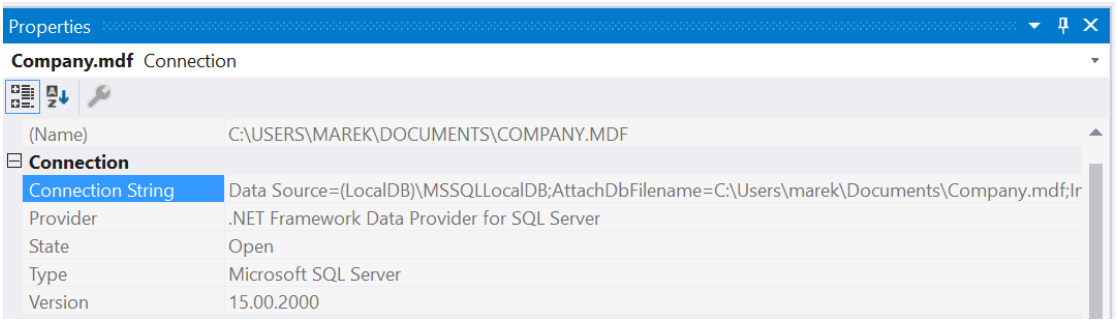
Komentarz: Sam pakiet EF Core nie musi być instalowany jawnie gdyż zostanie zainstalowany wraz z pakietem dostawcy EF Core dla serwera SQL Server jako jego zależność. Pozostałe dwa jawnie instalowane pakiety są wymagane w związku z wybraną w projekcie strategią inżynierii wstecznej w oparciu o istniejącą bazę danych.

4. Utworzenie klasy kontekstu bazy danych i klas encji na podstawie istniejącej bazy danych.

a) Przejdź do edycji pliku konfiguracyjnego aplikacji `appsettings.json`. Dodaj na końcu (przed ostatnim nawiasem klamrowym, oddzielając przecinkiem od poprzedniej opcji) poniższą sekcję z łańcuchem połączenia z bazą danych. (Uwaga: W miejscu trzech kropek w kolejnym kroku zostanie wstawiony faktyczny łańcuch połączenia z bazą danych)

```
  "ConnectionStrings": {
    "CompanyDbContext":
    "..."
  }
```

b) Z poziomu panelu Server Explorer wyświetl właściwości zdefiniowanego połączenia z bazą danych opcją Properties z menu kontekstowego.



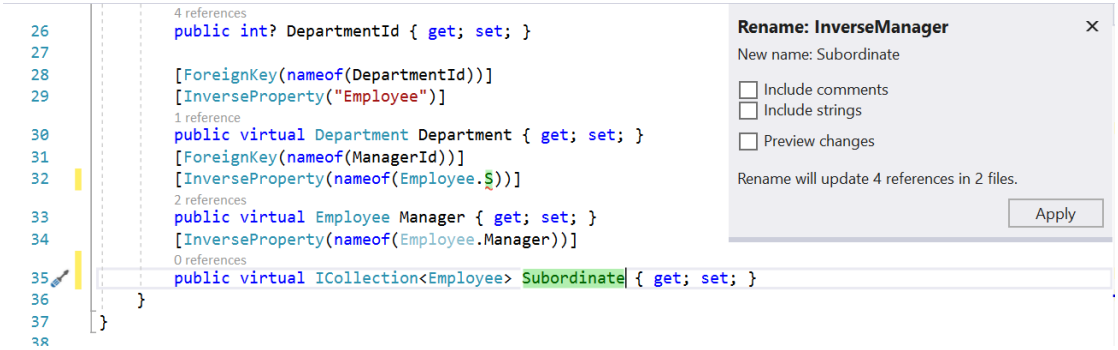
c) Skopiuj do schowka wartość właściwości Connection String (zapewne będzie się ona różnić ścieżką do pliku w porównaniu z przedstawioną na powyższym obrazku). Wklej ją w miejsce trzech kropek w łańcuchu połączenia z bazą danych w pliku konfiguracyjnym `appsettings.json`, zastępując pojedyncze ukośniki podwójnymi (zgodnie z zasadami dla łańcuchów znaków w JSON).

d) Z poziomu konsoli menedżera pakietów uruchom następującą komendę (w jednej linii, ze spacjami zamiast przejść do nowej linii):

```
Scaffold-DbContext -Connection name=CompanyDbContext
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
-Context CompanyDbContext -DataAnnotations
```

e) Obejrzyj wygenerowaną klasę kontekstu bazy danych i klasy encji. W klasie kontekstu odszukaj właściwości reprezentujące kolekcje encji z bazy danych. Czy kreator wygenerował nazwy tych właściwości w liczbie mnogiej? Obecnie (od EF Core 5.0) opcja „pluralize” jest domyślnie włączona, można ją wyłączyć flagą `-NoPluralize`. W klasach encji zwróć uwagę na obecność zarówno właściwości reprezentujących kolumny kluczy obcych jak i właściwości nawigacyjnych pozwalających na dostęp do powiązanych instancji encji (dotyczą one w naszym modelu dwukierunkowych związków: szef-podwładni i departament-pracownicy).

f) Nazwa właściwości nawigacyjnej prowadzącej do podwładnych pracownika wygenerowana przez kreator (InverseManager) jest mało przekonująca. Zmień ją na Subordinate, korzystając w opcji Rename menu kontekstowego w edytorze kodu (wywołaj menu prawym klawiszem myszy z miejsca w kodzie, w którym ma być dokonana zmiana). Gdy pojawi się okno kreatora Rename nadpisz ręcznie w jednym z miejsc w kodzie starą nazwę właściwości nową nazwą i kliknij Apply.



g) Zapisz wszystkie zmiany i przebuduj projekt (Build).

5. Konfiguracja kontekstu bazy danych jako usługi w aplikacji.

a) W pliku Program.cs przed instrukcją tworzącą obiekt aplikacji wstaw poniższy kod:

```
builder.Services.AddDbContext<CompanyDbContext>(options =>
      options.UseSqlServer(builder.Configuration
            .GetConnectionString("CompanyDbContext")));
```

b) Po dodaniu powyższej instrukcji, do listy instrukcji using dodaj instrukcję importującą przestrzeń nazw klasy kontekstu bazy danych oraz poniższą instrukcję importującą przestrzeń nazw EF Core:

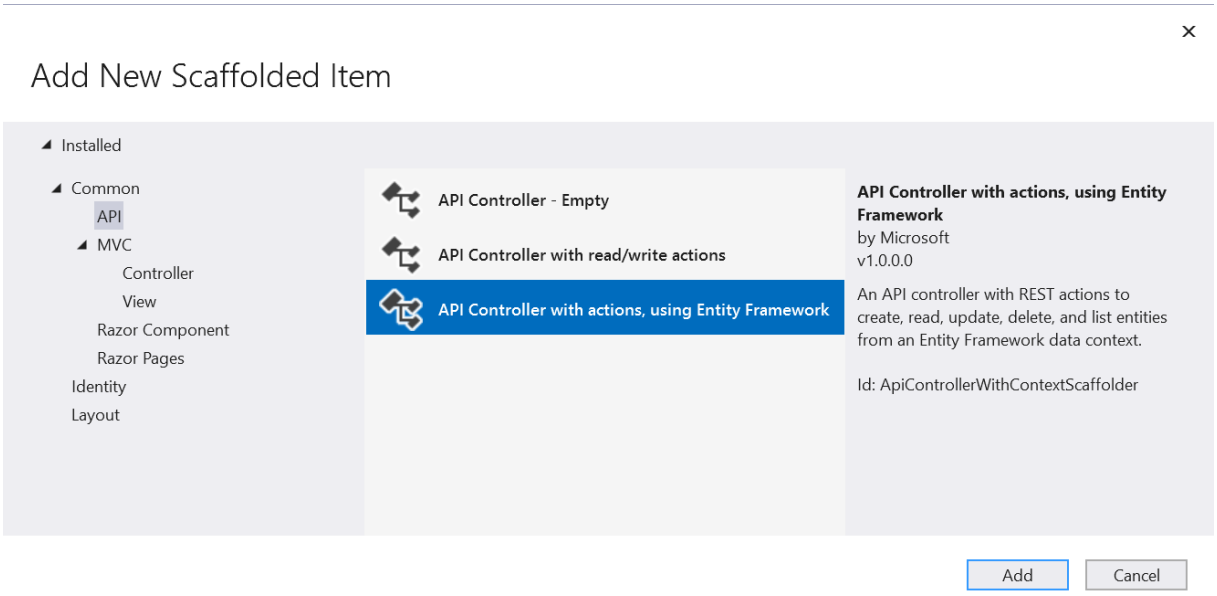
```
using Microsoft.EntityFrameworkCore;
```

c) Przejdź do edycji klasy kontekstu bazy danych. Ponieważ przekazanie nazwy łańcucha połączeniowego bazy danych do kontekstu bazy danych odbywa się teraz w pliku Program, metoda OnConfiguring klasy kontekstu utworzona przez kreator jest już niepotrzebna. Usuń więc ją.

d) Zapisz wszystkie zmiany.

e) Przebuduj projekt, aby upewnić się, że nie zawiera błędów (Build).

6. Z poziomu węzła Controllers projektu wywołaj operację Add→Controller (kreator nowego kontrolera można również uruchomić operacją Add→New Scaffolded Item). Wybierz szablon API Controller with actions, using Entity Framework i kliknij Add.

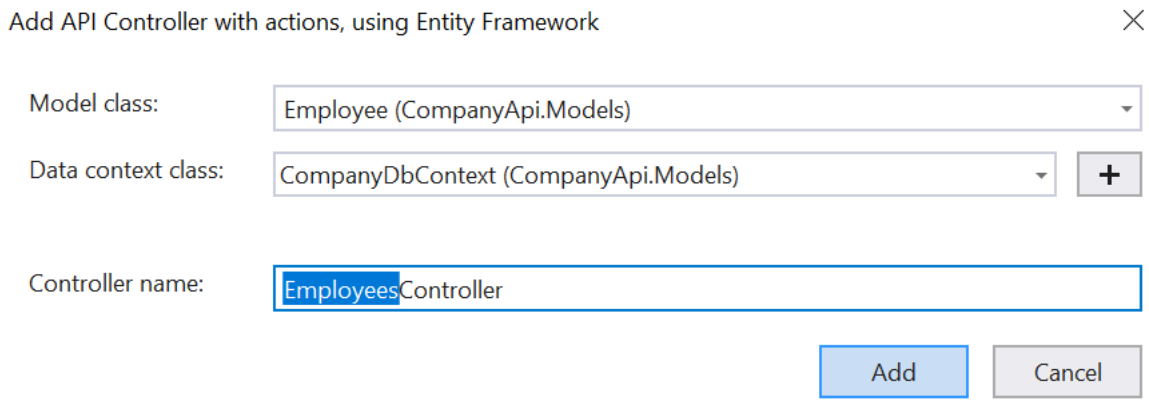


W kreatorze dodawania kontrolera:

a) Jako klasę modelu wskaż klasę encji Employee

b) Jako klasę kontekstu wybierz CompanyDbContext

c) Zaakceptuj proponowaną nazwę kontrolera EmployeesController



7. Analogicznie utwórz kontroler dla encji Department.

8. Przeanalizuj i porównaj kod poszczególnych metod obu kontrolerów, zwracając szczególną uwagę na adnotacje opisujące klasy i metody oraz na konstrukcje związane z asynchronicznym przetwarzaniem akcji.

9. Uruchom aplikację. Ponieważ w kreatorze projektu pozostawiliśmy zaznaczone wsparcie Open API, w przeglądarce zostanie otwarta strona z dokumentacją API wygenerowana przez Swagger, umożliwiająca przetestowanie API. Odszukaj sekcję dotyczącą zasobu reprezentującego prognozę pogody i pobierz ją metodą GET. Następnie w innej karcie przeglądarki wprowadź w pasku adresu adres URI zasobu z prognozą pogody i pobierz ją bezpośrednio (bez pośrednictwa Swaggera).

10. Włącz tymczasowo i przetestuj wsparcie dla XML jako alternatywy dla formatu JSON:

a) W pliku Program.cs zastąp instrukcję konfigurującą kontrolery poniższą, aktywującą obsługę formatu XML i możliwość negocjacji preferowanego formatu wymiany danych przez przeglądarki za pomocą nagłówka Accept protokołu HTTP:

```
builder.Services.AddControllers(options =>
        {
            options.RespectBrowserAcceptHeader = true;
        }).AddXmlSerializerFormatters();
```

b) Zapisz zmiany w projekcie, uruchom aplikację, a następnie odśwież w przeglądarce stronę Swaggera. Pobierz prognozę pogody wybierając z listy dostępnych typów zawartości application/xml. Następnie w innej karcie przeglądarki wprowadź w pasku adresu adres URI zasobu z prognozą pogody i pobierz ją bezpośrednio (bez pośrednictwa Swaggera).

c) Przywróć poprzednią wersję instrukcji konfigurującej kontrolery w pliku Program.cs:

```
builder.Services.AddControllers();
```

Komentarz: Jeśli z punktu widzenia klientów tworzonego API obsługa XML jest niepotrzebna, to lepiej jej po stronie API nie włączać.

d) Zapisz wszystkie zmiany i przebuduj projekt, aby upewnić się, że nie zawiera błędów (Build).

11. Obejrzyj adnotacje opisujące klasę kontrolera obsługującego departamenty i jej metody akcji. Wywnioskuj jak powinien wyglądać adres URI reprezentujący wszystkie departamenty oraz pojedynczy departament. Uruchom aplikację kombinacją klawiszy Ctrl+F5 (Start

Without Debugging). Zajrzyj na stronę Swaggera by potwierdzić postaci adresów URI. Następnie w przeglądarce (bezpośrednio z poziomu paska adresu, bez użycia Swaggera):

a) Wyświetl wszystkie departamenty.
b) Wyświetl departament o identyfikatorze 2.
c) Sprawdź czy adresy URI adresujące zasoby Web API są wrażliwe na wielkość liter.
d) Sprawdź zachowanie API przy podaniu nieistniejącego identyfikatora departamentu.

Komentarz: Domyślna serializacja instancji encji Department skutkuje uwzględnieniem w dokumencie JSON pustej tablicy pracowników. Jest to mało użyteczne. Zajmiemy się tym problemem później.

12. Analogicznie do departamentów, spróbuj wyświetlić w przeglądarce wszystkich pracowników lub pojedynczego pracownika. Zinterpretuj zgłoszony wyjątek i spróbuj wskazać w klasie Employee źródło problemu.

Komentarz: Udostępnianie encji ORM bezpośrednio poprzez API jest często wskazywane jako zła praktyka (organizacja danych może nie być zorientowana na potrzeby klienta, brak kontroli nad zakresem udostępnianych danych, zależność API i klienta od modelu z niższej warstwy aplikacji). Ponadto, w przypadku cyklicznych zależności między encjami, skończy się zaobserwowanym błędem w ASP.NET. Rozwiązaniem wszystkich wspomnianych problemów jest wykorzystanie obiektów DTO (ang. data transfer object) do reprezentacji udostępnianych przez API danych.

13. Utworzenie klasy DTO dla encji Employee.

a) Z poziomu folderu Models wywołaj kreator tworzenia nowej klasy. Nazwij ją EmployeeDTO.
b) Umieść w ciele klasy DTO poniższe właściwości (są to wszystkie właściwości encji Employee z pominięciem właściwości nawigacyjnych i bez atrybutów Data Annotations):

```
public int EmployeeId { get; set; }
public string FirstName { get; set; }
public string LastName { get; set; }
public int? ManagerId { get; set; }
public decimal? Salary { get; set; }
public decimal? Bonus { get; set; }
public int? DepartmentId { get; set; }
```

14. Utworzenie metod do konwersji (w obie strony) między instancjami klasy encji Employee i EmployeeDTO w klasie kontrolera obsługującego pracowników:

```
private static EmployeeDTO EmployeeToDTO(Employee employee) =>
    new EmployeeDTO
    {
        EmployeeId = employee.EmployeeId,
        FirstName = employee.FirstName,
        LastName = employee.LastName,
        ManagerId = employee.ManagerId,
        Salary = employee.Salary,
        Bonus = employee.Bonus,
```

```
                DepartmentId = employee.DepartmentId
            };

        private static Employee DTOToEmployee(EmployeeDTO employeeDTO) =>
            new Employee
            {
                EmployeeId = employeeDTO.EmployeeId,
                FirstName = employeeDTO.FirstName,
                LastName = employeeDTO.LastName,
                ManagerId = employeeDTO.ManagerId,
                Salary = employeeDTO.Salary,
                Bonus = employeeDTO.Bonus,
                DepartmentId = employeeDTO.DepartmentId
            };
```

15. Modyfikacja metod akcji kontrolera obsługującego pracowników tak, aby akceptowały od klientów i zwracały klientom obiekty DTO w miejsce instancji encji.

Uwaga: Możesz podmienić cały kod metod na podane poniżej implementacje lub dokonać zmian krok po kroku. Miejsca, w których zostały dokonane zmiany, zostały w kodzie wytłuszczone.

a) Metodę zwracającą wszystkich pracowników zastąp poniższą implementacją:

```
[HttpGet]
public async Task<ActionResult<IEnumerable<EmployeeDTO>>> GetEmployee()
{
            return await _context.Employees
                .Select(e => EmployeeToDTO(e))
                .ToListAsync();
}
```

Komentarz: Zamiast konwersji po odczytaniu listy instancji encji zapytaniem LINQ, konwersja została zrealizowana w ciele zapytania w klauzuli Select. Integracja zapytań z językiem programowania to fundament LINQ.

b) Metodę zwracającą pojedynczego pracownika zastąp poniższą implementacją:

```
        [HttpGet("{id}")]
        public async Task<ActionResult<EmployeeDTO>> GetEmployee(int id)
        {
            var employee = await _context.Employees.FindAsync(id);

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

            return EmployeeToDTO(employee);
        }
```

c) Metodę podmieniającą pracownika zastąp poniższą implementacją:

```
    [HttpPut("{id}")]
    public async Task<IActionResult> PutEmployee(int id,
                                                 EmployeeDTO employeeDTO)
    {
            if (id != employeeDTO.EmployeeId)
            {
                return BadRequest();
            }
```

```
            _context.Entry(DTOToEmployee(employeeDTO)).State =
                      EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!EmployeeExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }

            return NoContent();
    }
```

d) Metodę dodającą nowego pracownika zastąp poniższą implementacją:

```
[HttpPost]
public async Task<ActionResult<EmployeeDTO>> PostEmployee(EmployeeDTO
                                                          employeeDTO)
{
     _context.Employees.Add(DTOToEmployee(employeeDTO));
     await _context.SaveChangesAsync();

     return CreatedAtAction("GetEmployee",
            new { id = employeeDTO.EmployeeId }, employeeDTO);
}
```

e) Metodę usuwającą pracownika zastąp poniższą implementacją:

```
    [HttpDelete("{id}")]
    public async Task<ActionResult<EmployeeDTO>> DeleteEmployee(int id)
    {
            var employee = await _context.Employees.FindAsync(id);
            if (employee == null)
            {
                return NotFound();
            }

            _context.Employees.Remove(employee);
            await _context.SaveChangesAsync();

            return EmployeeToDTO(employee);
    }
```

16. Zapisz wszystkie zmiany, przebuduj i uruchom aplikację.

17. Przetestuj z poziomu przeglądarki, czy pobieranie wszystkich pracowników i wybranego pracownika działa teraz poprawnie.

Zadanie do samodzielnego wykonania

1. Zmień obsługę departamentów przez API tak, aby była realizowana poprzez obiekty DTO, analogicznie do obsługi pracowników.

a) Utwórz klasę DepartmentDTO zawierającą dwie właściwości: identyfikator i nazwę departamentu.

b) Utwórz w klasie kontrolera obsługującego departamenty parę prywatnych statycznych metod do konwersji między instancjami encji Department a obiektami DTO.

c) Zmodyfikuj metody akcji kontrolera obsługującego departamenty tak aby akceptowały od klientów i zwracały klientom obiekty DTO w miejsce instancji encji.

2. Przetestuj z poziomu przeglądarki, że pobieranie wszystkich departamentów i wybranego departamentu nadal działa po dokonanych zmianach w aplikacji.

3. Korzystając ze Swaggera przetestuj wszystkie akcje kontrolera obsługującego departamenty:

a) Pobierz wszystkie departamenty.

b) Pobierz departament o identyfikatorze 2.

c) Dodaj nowy departament. (Wskazówka: Prześlij dokument JSON zawierający tylko nazwę departamentu, bez identyfikatora.) Zwróć uwagę na status odpowiedzi HTTP.

d) Pobierz wszystkie departamenty z poziomu przeglądarki aby upewnić się, że departament został dodany.

e) Zmodyfikuj nazwę dodanego przez siebie departamentu. (Wskazówka: Zwróć uwagę na zgodność wartości identyfikatora zawartego w przesyłanym dokumencie JSON z identyfikatorem zawartym w adresie URI.) Sprawdź status odpowiedzi HTTP.

f) Pobierz wszystkie departamenty z poziomu przeglądarki aby upewnić się, że departament został zmodyfikowany.

g) Usuń dodany przez siebie departament. Sprawdź status odpowiedzi HTTP.

h) Pobierz wszystkie departamenty z poziomu przeglądarki aby upewnić się, że departament został usunięty.