

Dawid Majchrowski

.Net, Entity Framework

Sprawozdanie – 10.11.2019

Sprawozdanie kontynuujemy od miejsca zakończenia ćwiczeń na następującym stanie (IV j.):

- CategoryForm

```
ConsoleApplication1.CategoryForm
CategoryForm.Loading

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.Entity;

namespace ConsoleApplication1
{
    public partial class CategoryForm : Form
    {
        public CategoryForm()
        {
            InitializeComponent();
            Load += new EventHandler(CategoryForm_Load);
        }

        private void categoryDataGridView_CellContentClick(object sender, DataGridViewCellEventArgs e)
        {
        }

        private void CategoryForm_Load(object sender, System.EventArgs e)
        {
            ProdContext db = new ProdContext();
            db.Categories.Load();
            this.categoryBindingSource.DataSource = db.Categories.Local.ToBindingList();
        }
    }
}
```

- Aplikacja konsolowa

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            using (ProdContext db = new ProdContext()){
                Console.WriteLine("Podaj nazwe kategorii: ");
                var category = Console.ReadLine();
                db.Categories.Add(new Category { Name = category });
                db.SaveChanges();

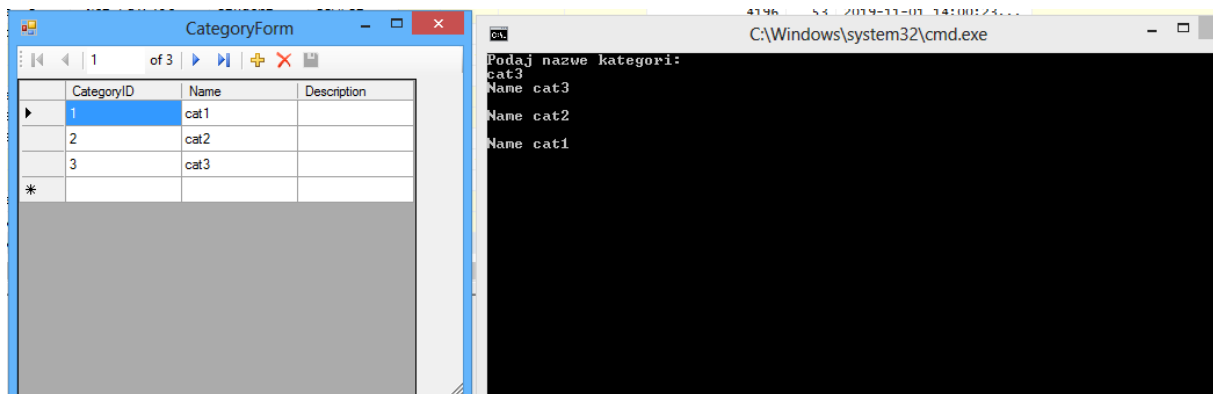
                var query = from cat in db.Categories
                            orderby cat.Name descending
                            select cat.Name;

                foreach (var item in query){
                    Console.WriteLine("Name {0}\n", item);
                }

                CategoryForm form = new CategoryForm();
                form.ShowDialog();
            }
        }
    }

    class Customer
    {
        [Key]
        public string CompanyName { get; set; }
        public string Description { get; set; }
    }
}
```

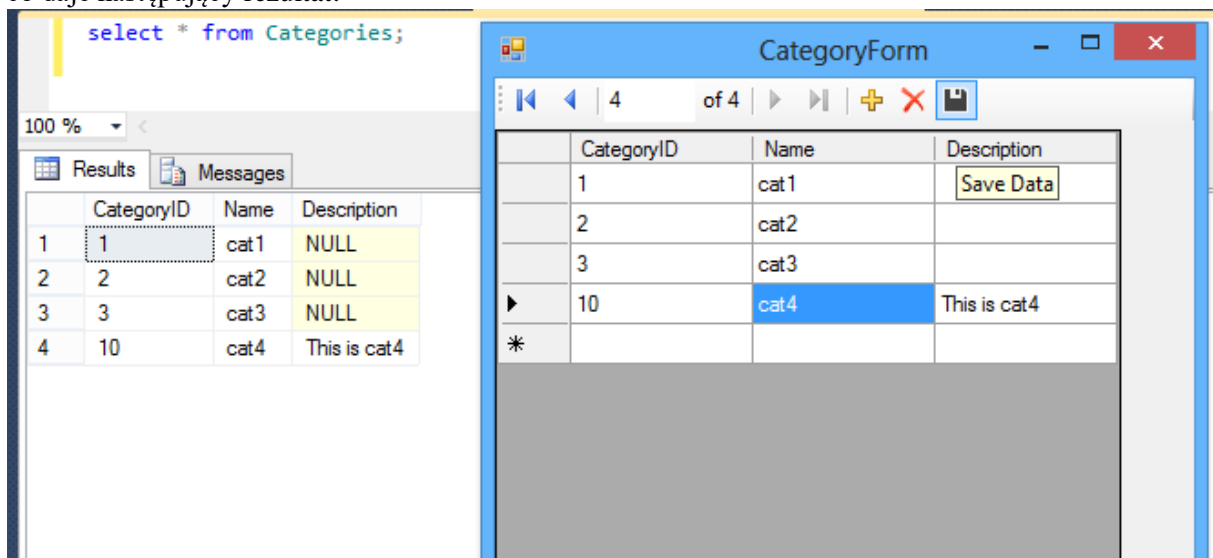
- Aktualny stan po odpaleniu aplikacji



Kolejnym zadaniem jest dodanie obsługi zmian oraz zapisu danych w formularzu. Robimy to w następujący sposób:

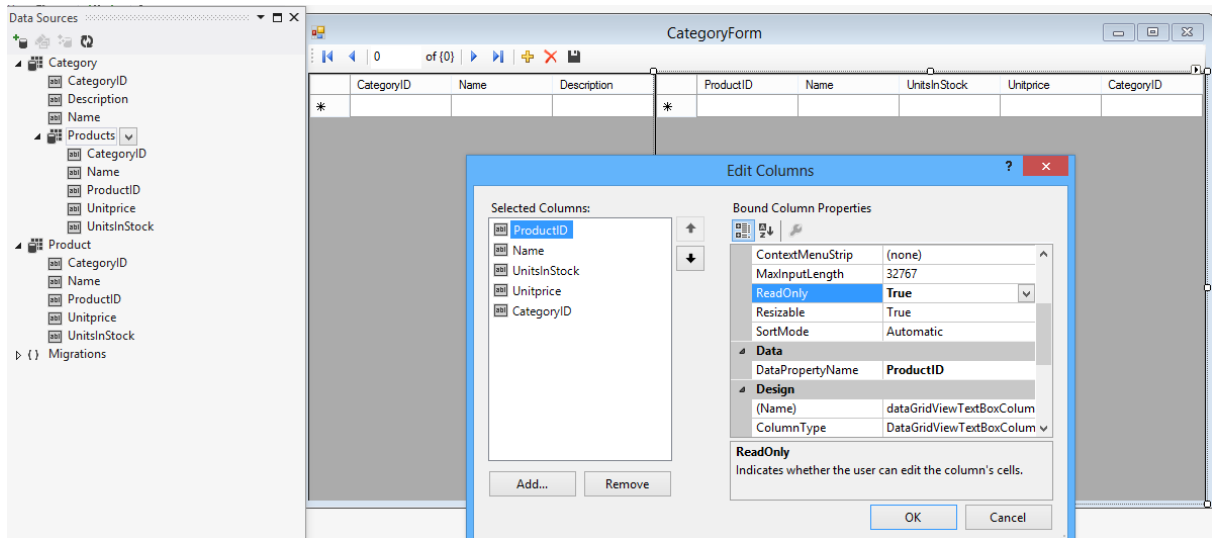
```
ProdContext db;  
  
public CategoryForm()  
{  
    InitializeComponent();  
    Load += new EventHandler(CategoryForm_Load);  
}  
  
private void categoryDataGridView_CellContentClick(object sender, DataGridViewCellEventArgs e)  
{  
  
}  
  
private void CategoryForm_Load(object sender, System.EventArgs e)  
{  
    db = new ProdContext();  
    db.Categories.Load();  
    this.categoryBindingSource.DataSource = db.Categories.Local.ToBindingList();  
}  
  
private void categoryBindingNavigatorSaveItem_Click(object sender, EventArgs e)  
{  
    db.SaveChanges();  
    this.categoryDataGridView.Refresh();  
}
```

co daje następujący rezultat:



Po dodaniu nowej kategorii, sprawdzamy, czy działa usuwanie usuwając nowo dodaną kategorię oraz ponownie zapisując dane do bazy, obserwując zachowanie zgodnie z założeniem.

Kolejnym krokiem jest dodanie produktów do naszej aplikacji konsolowej, robimy to w ten sam sposób co dodanie kategorii (przeciągamy źródło danych z „data source”), ustawiamy odpowiednie pola na do odczytu, oraz tworzymy obsługę eventu następującego po naciśnięciu odpowiedniej komórki z kategorią.



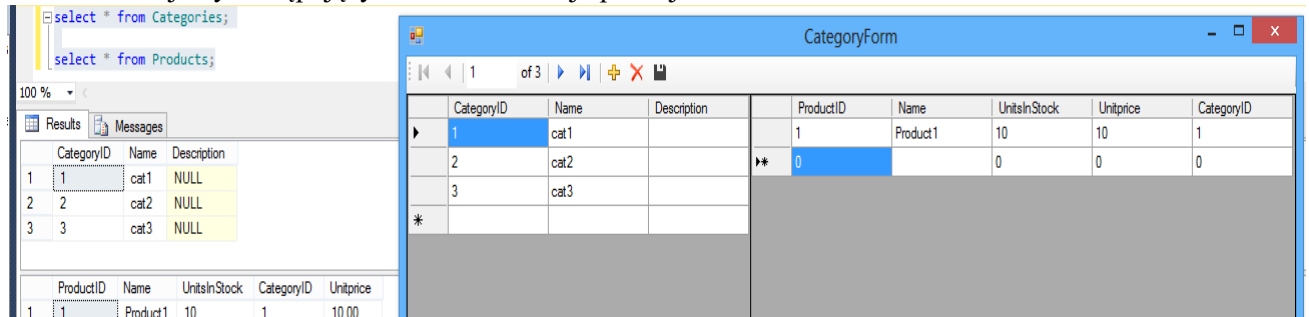
Zapytania zgodnie z poleceniem obsługę formularza tworzymy w obu notacjach.

```
private void categoryDataGridView_CellClick(object sender, DataGridViewCellEventArgs e)
{
    DataGridView dgv = sender as DataGridView;

    if (dgv == null)
        return;
    if (dgv.CurrentRow.Selected)
    {
        int CategoryId = (int)dgv.CurrentRow.Cells[0].Value;
        //var query = from prod in db.Products
        //              where prod.CategoryID == CategoryId
        //              select prod;
        var query = db.Products.Where(prod => prod.CategoryID == CategoryId);

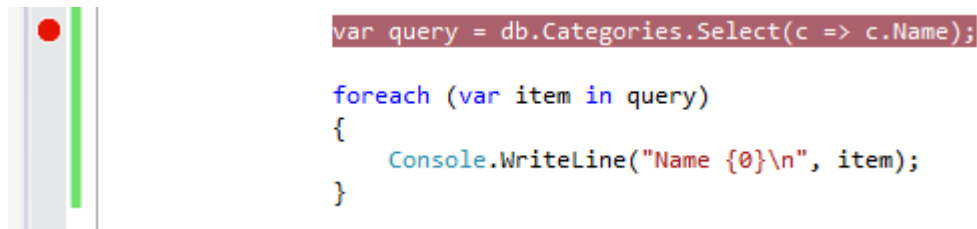
        this.productsBindingSource.DataSource = new BindingList<Product>(query.ToList());
    }
}
```

Oraz obserwujemy następujący rezultat w naszej aplikacji:



Wracamy do części konsolowej i dodajemy odpowiednie metody, które:

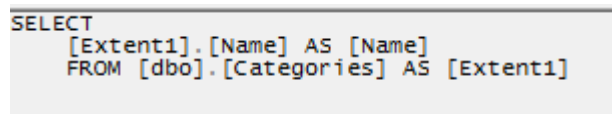
- i) Nazwy kategorii – method based syntax



```
var query = db.Categories.Select(c => c.Name);

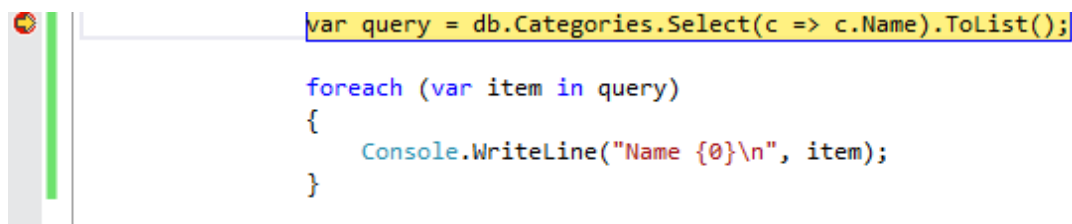
foreach (var item in query)
{
    Console.WriteLine("Name {0}\n", item);
}
```

Po krokowym przejściu, obserwujemy na profilerze egzekucję następującego zapytania, dopiero w momencie wywołania pętli foreach, czyli zgodnie z przewidywaniem, gdyż nie wymusiliśmy natychmiastowej egzekucji zapytania.



```
SELECT
    [Extent1].[Name] AS [Name]
FROM [dbo].[Categories] AS [Extent1]
```

Dokonujemy drobnej zmiany, wymuszając natychmiastową egzekucję i obserwujemy w profilerze to samo zapytanie, ale wykonane w momencie deklaracji zapytania, a nie przy egzekucji pętli foreach.

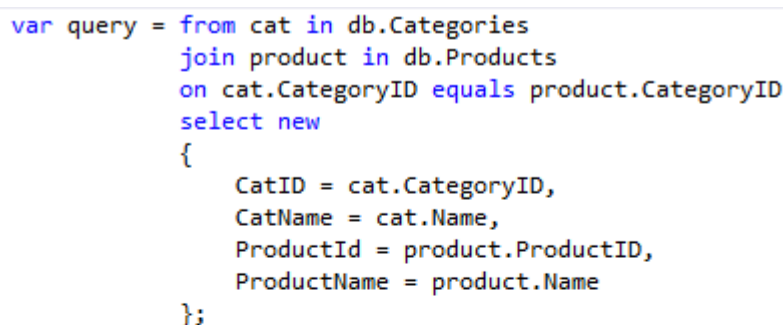


```
var query = db.Categories.Select(c => c.Name).ToList();

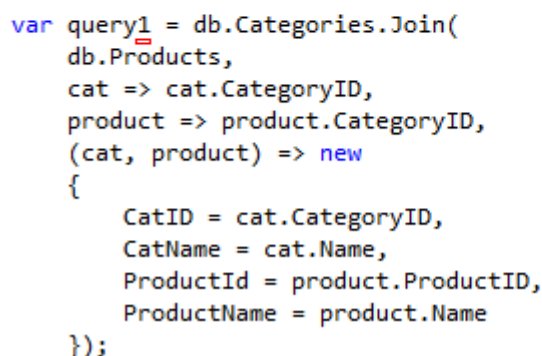
foreach (var item in query)
{
    Console.WriteLine("Name {0}\n", item);
}
```

- ii) Wszystkie kategorie i produkty (w obu notacjach)

- Joigny (Obie notacje)



```
var query = from cat in db.Categories
            join product in db.Products
            on cat.CategoryID equals product.CategoryID
            select new
            {
                CatID = cat.CategoryID,
                CatName = cat.Name,
                ProductId = product.ProductID,
                ProductName = product.Name
            };
};
```



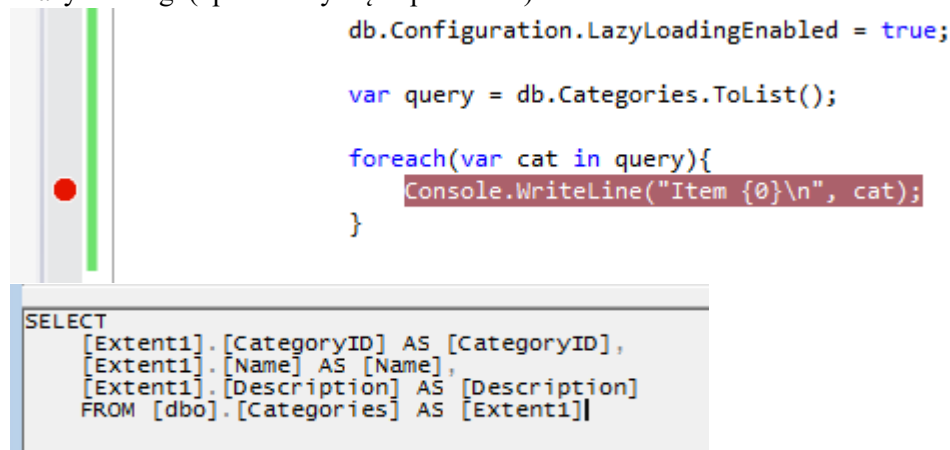
```
var query1 = db.Categories.Join(
    db.Products,
    cat => cat.CategoryID,
    product => product.CategoryID,
    (cat, product) => new
    {
        CatID = cat.CategoryID,
        CatName = cat.Name,
        ProductId = product.ProductID,
        ProductName = product.Name
    });
};
```

- Navigation property

```
var query = from cat in db.Categories
             select new
             {
                 catId = cat.CategoryID,
                 catName = cat.Name,
                 prods = cat.Products
             };

foreach (var cat in query)
{
    foreach (var prod in cat.prods)
    {
        Console.WriteLine("Item {0} {1} {2}\n", prod.ProductID, prod.Name, cat.catName);
    }
}
```

-Lazy loading (upewniamy się w profilerze)



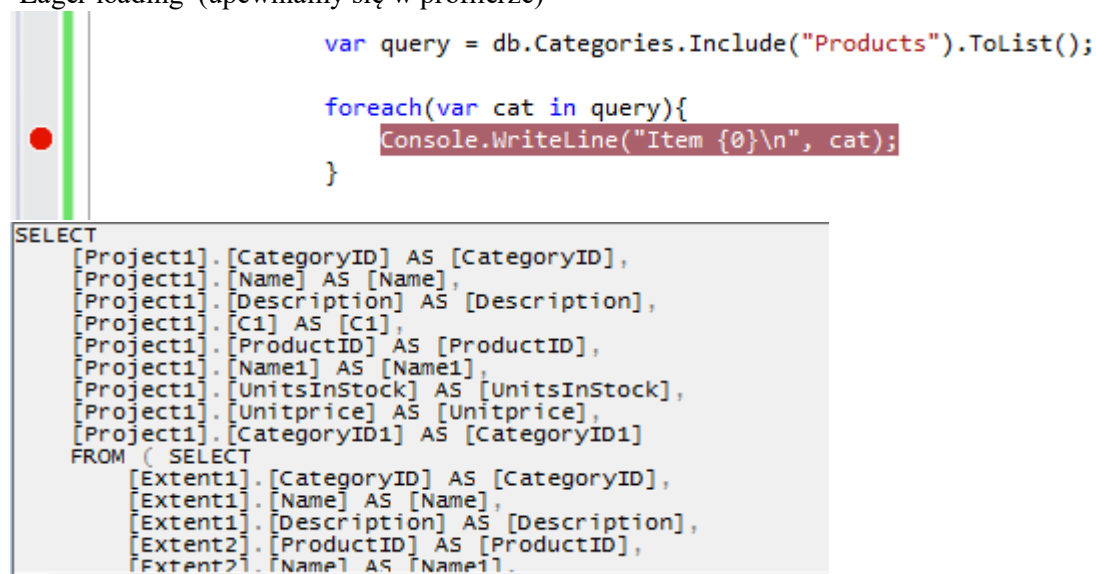
```
db.Configuration.LazyLoadingEnabled = true;

var query = db.Categories.ToList();

foreach(var cat in query){
    Console.WriteLine("Item {0}\n", cat);
}
```

```
SELECT
    [Extent1].[CategoryID] AS [CategoryID],
    [Extent1].[Name] AS [Name],
    [Extent1].[Description] AS [Description]
FROM [dbo].[Categories] AS [Extent1]
```

-Eager loading (upewniamy się w profilerze)



```
var query = db.Categories.Include("Products").ToList();

foreach(var cat in query){
    Console.WriteLine("Item {0}\n", cat);
}
```

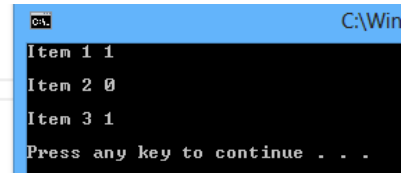
```
SELECT
    [Project1].[CategoryID] AS [CategoryID],
    [Project1].[Name] AS [Name],
    [Project1].[Description] AS [Description],
    [Project1].[C1] AS [C1],
    [Project1].[ProductID] AS [ProductID],
    [Project1].[Name1] AS [Name1],
    [Project1].[UnitsInStock] AS [UnitsInStock],
    [Project1].[Unitprice] AS [Unitprice],
    [Project1].[CategoryID1] AS [CategoryID1]
FROM ( SELECT
        [Extent1].[CategoryID] AS [CategoryID],
        [Extent1].[Name] AS [Name],
        [Extent1].[Description] AS [Description],
        [Extent2].[ProductID] AS [ProductID],
        [Extent2].[Name] AS [Name1],
        [Extent1].[UnitsInStock] AS [UnitsInStock],
        [Extent1].[Unitprice] AS [Unitprice],
        [Extent1].[CategoryID] AS [CategoryID1]
    FROM [dbo].[Categories] AS [Extent1]
    INNER JOIN [dbo].[Products] AS [Extent2] ON [Extent1].[CategoryID] = [Extent2].[CategoryID] ) AS [Project1]
```

Wszystkie powyższe zapytania dają oczekiwane rezultaty.

iii) Dla każdej kategorii pokaż liczbę produktów (jeśli dla kategorii brak produktu - wyświetl 0)

- Method syntax

```
var query = db.Categories.Include("Products");
foreach (var cat in query)
{
    Console.WriteLine("Item {0} {1}\n", cat.CategoryID, cat.Products.Count());
}
```



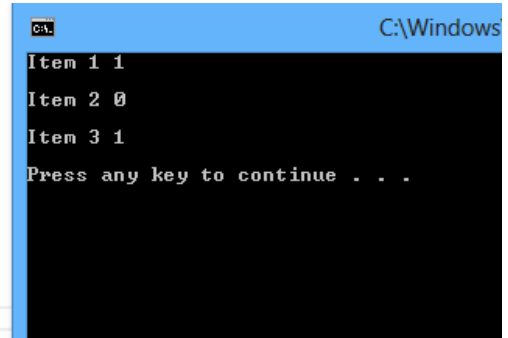
C:\Win

```
Item 1 1
Item 2 0
Item 3 1
Press any key to continue . . .
```

- Query syntax

```
var query = from cat in db.Categories.Include("Products")
            select new
            {
                catId = cat.CategoryID,
                count = cat.Products.Count()
            };

foreach (var cat in query)
{
    Console.WriteLine("Item {0} {1}\n", cat.catId, cat.count);
}
```



C:\Windows

```
Item 1 1
Item 2 0
Item 3 1
Press any key to continue . . .
```


Tym samym kończymy zadania laboratoryjne, jako zadanie domowe tworzymy aplikację webową z użyciem Asp.Net Core Web API oraz frameworka Angular 9.0. Tym razem nie będziemy już korzystać z maszyny wirtualnej, gdyż Asp.Net Core jest dostępny dla VS w wersji 2015+. Założenie aplikacji, to możliwość dodawania kategorii oraz produktów do bazy danych, korzystając z aplikacji webowej.


Tworzymy aplikację ASP.NET Core :


Utwórz nową aplikację internetową platformy ASP.NET Core


.NET Core


ASP.NET Core 3.0


**Pusty**
Pusty szablon projektu służący do tworzenia aplikacji platformy ASP.NET Core. Ten szablon nie ma żadnej zawartości.

**API**
Szablon projektu służący do tworzenia aplikacji platformy ASP.NET Core z przykładowym kontrolerem obsługującym usługę HTTP RESTful. Tego szablonu można także użyć dla widoków i kontrolerów platformy ASP.NET Core MVC.

**Aplikacja internetowa**
Szablon projektu służący do tworzenia aplikacji ASP.NET Core z przykładowymi stronami ASP.NET Core Razor.

**Aplikacja internetowa (Model-View-Controller)**
Szablon projektu służący do tworzenia aplikacji platformy ASP.NET Core z przykładowymi widokami i kontrolerami platformy ASP.NET Core MVC. Tego szablonu można także użyć dla usług HTTP RESTful.

**Angular**
Szablon projektu umożliwiający tworzenie aplikacji ASP.NET Core z użyciem platformy Angular.

**React.js**

Uwierzytelnianie
Bez uwierzytelniania
[Zmień](#)

Zaawansowane
☐ Konfiguruj dla protokołu HTTPS
☐ Włącz obsługę platformy Docker
(Wymaga elementu [Docker Desktop](#))

Linux

Autor: Microsoft
Źródło: .NET Core 3.0.0


Wstecz


Utwórz


[Pobierz dodatkowe szablony projektu](#)



Instalujemy potrzebne pakiety:

- Cors (Nie będziemy korzystać ze zbudowanej strony statycznej, requesty na frontendzie będą wysyłane na odpowiedni port oraz ścieżkę)

**Microsoft.AspNetCore.Cors** przez: Microsoft, Pobrania: **38,2M** v2.2.0
CORS middleware and policy for ASP.NET Core to enable cross-origin resource sharing. Commonly used types:

**Microsoft.AspNetCore.Mvc.Cors** przez: Microsoft, Pobrania: **43,4M** v2.2.0
ASP.NET Core MVC cross-origin resource sharing (CORS) features.

**Microsoft.AspNetCore.Cors** przez: Microsoft, Pobrania: **21,6M** v5.2.7
This package contains the core components to enable Cross-Origin Resource Sharing (CORS) in ASP.NET.

**Microsoft.AspNetCore.Cors** 

Wersja:

Najnowsza stabilna 2.2.0

Zainstaluj

Opcje

Opis
CORS middleware and policy for ASP.NET Core to enable cross-origin resource sharing. Commonly used types: [Microsoft.AspNetCore.Cors.Defaults.CorsOptions](#)

- Entity, to samo co w laboratorium


Przeglądaj


Zainstalowane


Aktualizacje


Entity

Uwzględnij wersję wstępną

**EntityFramework** przez: Microsoft, Pobrania: **75,2M** v6.3.0
Entity Framework is Microsoft's recommended data access technology for new applications.

**Microsoft.EntityFrameworkCore** przez: Microsoft, Pobrania: **59,9M** v3.0.0
Entity Framework Core is a lightweight and extensible version of the popular Entity Framework data access technology.

**Microsoft.EntityFrameworkCore.Relational** przez: Microsoft, Pobrania: **60,3M** v3.0.0
Shared Entity Framework Core components for relational database providers.

**Microsoft.EntityFrameworkCore.Design** przez: Microsoft, Pobrania: **38,5M** v3.0.0
Shared design-time components for Entity Framework Core tools.



Licencja na każdy pakiet jest udzielana przez jego właściciela. Firma NuGet nie ponosi odpowiedzialności za pakiety innych firm ani nie udziela na nie żadnych licencji.

☐ Nie pokazuj ponownie

Menedżer pakietów NuGet: Web

Źródło pakietów:

nuget.org

**Microsoft.EntityFrameworkCore** 

Wersja:

Najnowsza stabilna 3.0.0

Zainstaluj

Opcje

Opis
Entity Framework Core is a lightweight and extensible version of the popular Entity Framework data access technology.

Commonly Used Types:
[Microsoft.EntityFrameworkCore.DbContext](#)
[Microsoft.EntityFrameworkCore.DbSet](#)









Wersja: 3.0.0
Autorzy: Microsoft
Licencja: [Apache-2.0](#)
Data publikacji: poniedziałek, 23 września 2019 (23.09.2019)
Adres URL projektu: <https://docs.microsoft.com/ef/core/>

-SqlServer (Konfiguracja bazy danych z ConnectionStringa)

Przeglądaj Zainstalowane Aktualizacje Konsoliduj



Entity ☐ Uwzględnij wersję wstępną

Źródło pakietów:

	Microsoft.EntityFrameworkCore.Tools  przez: Microsoft, Pobrania: 32,4M v3.0.0 Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.
	Microsoft.EntityFrameworkCore.SqlServer  przez: Microsoft, Pobrania: 39,9M v3.0.0 Microsoft SQL Server database provider for Entity Framework Core.
	Microsoft.EntityFrameworkCore.InMemory  przez: Microsoft, Pobrania: 28,3M v3.0.0 In-memory database provider for Entity Framework Core (to be used for testing purposes).
	Microsoft.EntityFrameworkCore.Sqlite  przez: Microsoft, Pobrania: 17,8M v3.0.0 SQLite database provider for Entity Framework Core.

Licencja na każdy pakiet jest udzielana przez jego właściciela. Firma NuGet nie ponosi odpowiedzialności za pakiety innych firm ani nie udziela na nie żadnych licencji.

☐ Nie pokazuj ponownie

Microsoft.EntityFrameworkCore  









Wersje — 0

<input checked="" type="checkbox"/>	Projekt	Wersja
<input checked="" type="checkbox"/>	Web	

Zainstalowane:



Wersja:

-Tools (Zarządzanie Migracjami i bazą danych z menadżera pakietu)

	Microsoft.EntityFrameworkCore.Tools  przez: Microsoft, Pobrania: 32,4M v3.0.0 Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.
	Microsoft.EntityFrameworkCore.Design  przez: Microsoft, Pobrania: 38,6M v3.0.0 Shared design-time components for Entity Framework Core tools.
	Microsoft.EntityFrameworkCore.Tools.DotNet  przez: Microsoft, Pobrania: 4,14M v2.0.3 Entity Framework Core .NET Command Line Tools. Includes dotnet-ef.
	Microsoft.EntityFrameworkCore  przez: Microsoft, Pobrania: 60,1M v3.0.0 Entity Framework Core is a lightweight and extensible version of the popular Entity Framework data access technology.

Licencja na każdy pakiet jest udzielana przez jego właściciela. Firma NuGet nie ponosi odpowiedzialności za pakiety innych firm ani nie udziela na nie żadnych licencji.

☐ Nie pokazuj ponownie

Microsoft.EntityFrameworkCore  

Wersja:

Opis

Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.

Enables these commonly used commands:

- Add-Migration
- Drop-Database
- Get-DbContext
- Scaffold-DbContext
- Script-Migrations
- Update-Database

Wersja: 3.0.0

Autorzy: Microsoft

Tworzymy model (ten sam co na laboratorium, bez Customerów)

- Dodajemy opcje w konstruktorze DbContextu, żeby dodawać bazę danych z appsettings.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

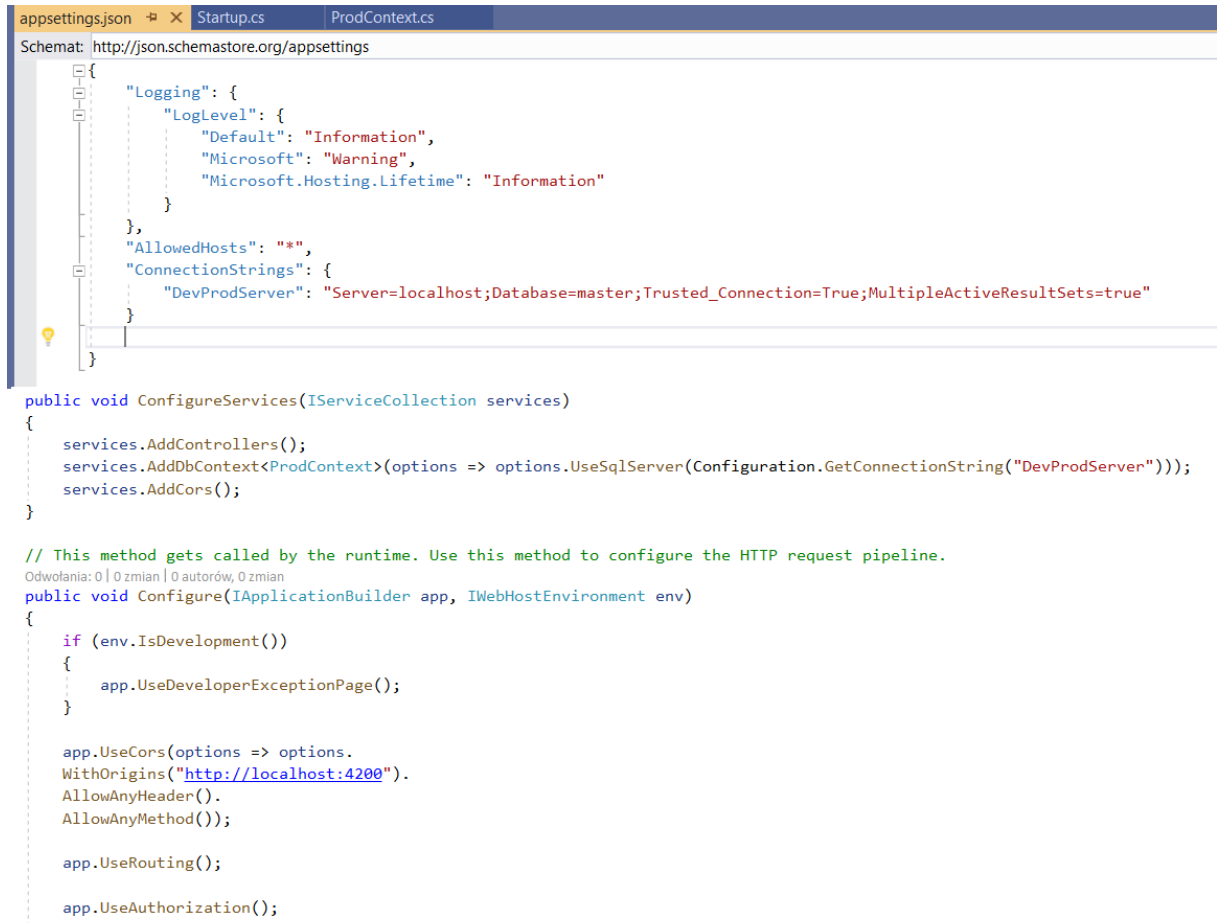
namespace Web.Models
{
    Odwołania: 2 | 0 zmian | 0 autorów, 0 zmian
    public class ProdContext : DbContext
    {
        Odwołania: 0 | 0 zmian | 0 autorów, 0 zmian
        public ProdContext(DbContextOptions<ProdContext> options):base(options)
        {
        }
        Odwołania: 0 | 0 zmian | 0 autorów, 0 zmian
        public DbSet<Category> Categories { get; set; }
        Odwołania: 0 | 0 zmian | 0 autorów, 0 zmian
        public DbSet<Product> Products { get; set; }
        Odwołania: 0 | 0 zmian | 0 autorów, 0 zmian
        public DbSet<Customer> Customers { get; set; }
    }
}
```

Rozwiązanie „Web” (liczba projektów: 1 z 1)

- Web
 - Connected Services
 - Properties
 - Zależności
 - Controllers
 - Models
 - + C# Category.cs
 - + C# ProdContext.cs
 - + C# Product.cs
 - + appsettings.json
 - + Program.cs
 - + Startup.cs

Podpinamy bazę danych

- Dodajemy odpowiedni ConnectionString dla naszego serwera. (w moim przypadku lokalna baza danych, można ustawić wedle uznania).
- Przy okazji aktywujemy CORS, aby przyjmował zapytania od strony (frontend na porcie 4200).



The screenshot shows the Visual Studio IDE with two files open: `appsettings.json` and `Startup.cs`. The `appsettings.json` file is selected, showing its schema and content. The `Startup.cs` file is also visible, showing the `ConfigureServices` and `Configure` methods.

```
appsettings.json  X Startup.cs  ProdContext.cs
Schemat: http://json.schemastore.org/appsettings
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DevProdServer": "Server=localhost;Database=master;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}

public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();
    services.AddDbContext<ProdContext>(options => options.UseSqlServer(Configuration.GetConnectionString("DevProdServer")));
    services.AddCors();
}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
Odwolania: 0 | 0 zmian | 0 autorów, 0 zmian
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseCors(options => options.
        WithOrigins("http://localhost:4200").
        AllowAnyHeader().
        AllowAnyMethod());

    app.UseRouting();

    app.UseAuthorization();
}
```

Sama aplikacja backendowa natomiast będzie chodziła na porcie 5000 (nie ustawiamy statycznie zbudowanej strony, więc wymagane będzie włącznie frontendu i backendu jednocześnie)



The screenshot shows the Visual Studio IDE with the `launchSettings.json` file open. The schema is shown as '<Nie wybrano schematu>'. The content of the file is as follows:


```
launchSettings.json  X
Schemat: <Nie wybrano schematu>
{
  "$schema": "http://json.schemastore.org/launchsettings.json",
  "profiles": {
    "Web": {
      "commandName": "Project",
      "launchBrowser": false,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

Skrypt uruchomieniowy dla windowsa (batch), uruchamia w tle proces backendu (port 5000) oraz frontend (port 4200) w przeglądarce (będzie opisany później).

```
start.bat — Notatnik
Plik Edycja Format Widok Pomoc
cd web/
start /b dotnet run
cd ../frontend
npm install & ng serve --open
```

Dodajemy potrzebne 2 kontrolery (dla produktów i kategorii)

Kontroler... Dodaj

 **Kontroler interfejsu API z akcjami używający narzędzia Entity Framework**


Dodaj Kontroler interfejsu API z akcjami używający narzędzia Entity Framework ×


Klasa modelu: Product (Web.Models) ▾

Klasa kontekstu danych: ProdContext (Web.Models) ▾ +

Nazwa kontrolera: ProductsController

Dodaj Anuluj

 Controllers

 ProductsController.cs

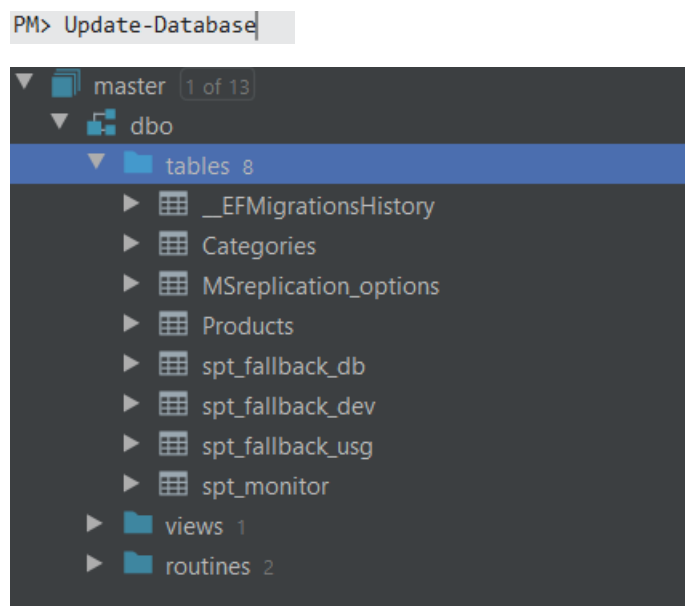
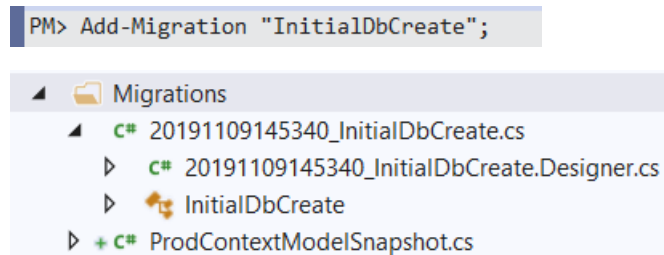
```
namespace Web.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    1 odwołanie | 0 zmian | 0 autorów, 0 zmian
    public class ProductsController : ControllerBase
    {
        private readonly ProdContext _context;

        Odwołania: 0 | 0 zmian | 0 autorów, 0 zmian
        public ProductsController(ProdContext context)
        {
            _context = context;
        }

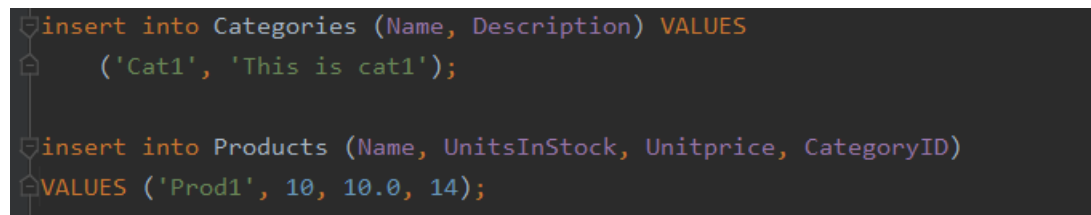
        // GET: api/Products
        [HttpGet]
        Odwołania: 0 | 0 zmian | 0 autorów, 0 zmian
        public async Task<ActionResult<IEnumerable<Product>>> GetProducts()
        {
            return await _context.Products.ToListAsync();
        }
    }
}
```

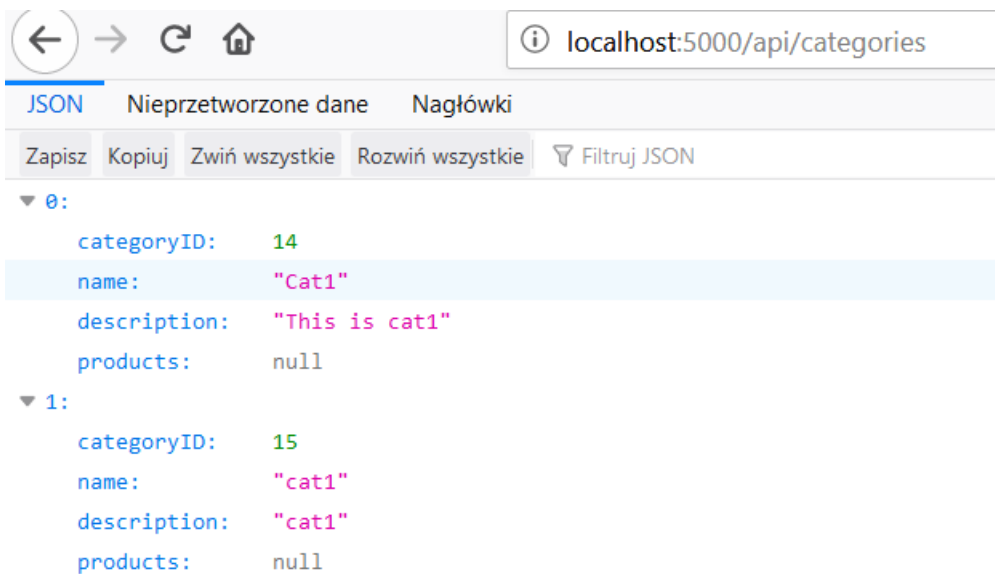
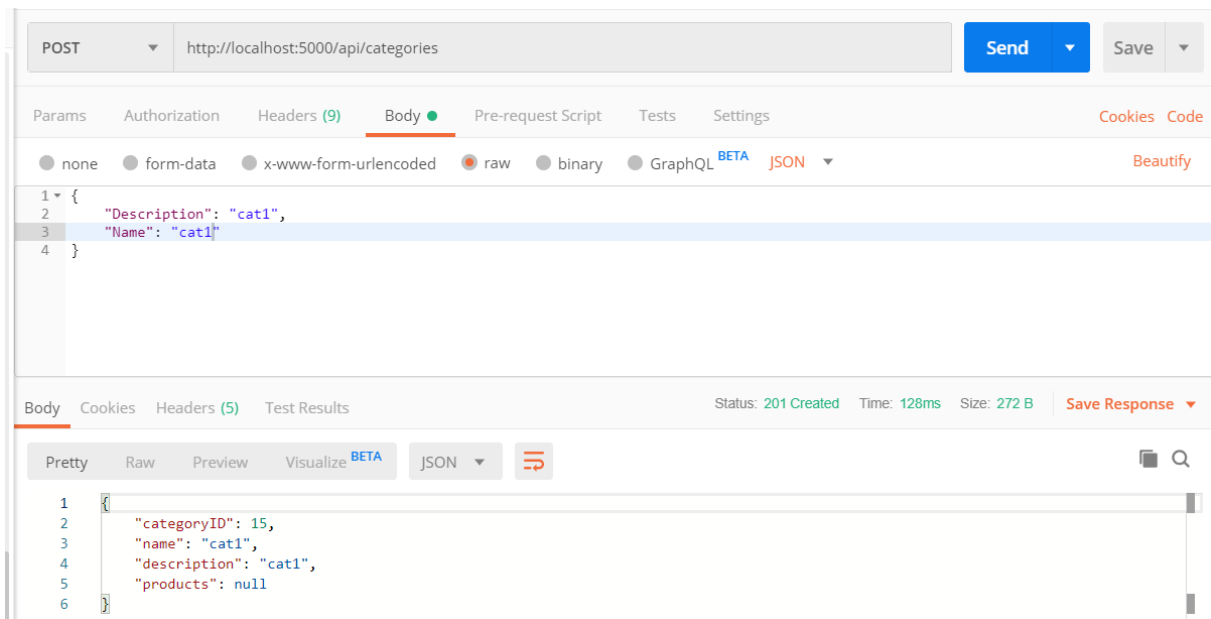
Analogicznie dla Kategorii.

Korzystając z managera tworzymy inicjalizujące migracje oraz aktualizujemy bazę.



Jak widać wszystko przebiega pomyślnie, zatem możemy sprawdzić działanie naszych dodanych kontrolerów.





Wszystko działa zgodnie z założeniem, wprowadzamy ostatnią zmianę w kontrolerze, z której będziemy korzystać w przyszłości.

Możliwość wyszukiwania produktów na podstawie Id kategorii, za pomocą następującej metody.

```
// GET: api/Products/byCategory1
[HttpGet("byCategory{id}")]
Odwolania: 0 | 0 zmian | 0 autorów, 0 zmian
public async Task<ActionResult<IEnumerable<Product>>> GetProductsByCategory(int id)
{
    return await _context.Products.Where(x => x.CategoryID == id).ToListAsync();
}
```

Tworzymy frontend (zależności):

```
$ ng --version

Angular CLI: 8.3.18
Node: 13.1.0
OS: win32 x64
Angular:
...

Package                           Version
-----
@angular-devkit/architect         0.803.18
@angular-devkit/core              8.3.18
@angular-devkit/schematics        8.3.18
@schematics/angular              8.3.18
@schematics/update                0.803.18
rxjs                              6.4.0

$ ng new frontend
CREATE frontend/angular.json (3609 bytes)
CREATE frontend/package.json (1294 bytes)
CREATE frontend/README.md (1026 bytes)
CREATE frontend/tsconfig.json (543 bytes)
CREATE frontend/tslint.json (1953 bytes)
CREATE frontend/.editorconfig (246 bytes)
CREATE frontend/.gitignore (631 bytes)
```

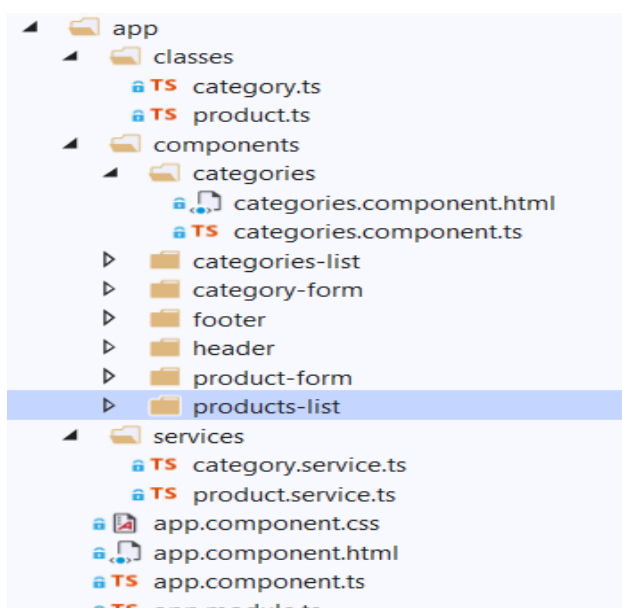
Głównie będziemy korzystać z angular material, dlatego dodajemy do projektu

```
$ ng add @angular/material
```

Dodajemy również bootstrapa

```
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw&
```

Struktura aplikacji



Dwie klasy (odpowiadające klasą kontekstowym bez kluczy obcych)

```
export class Category {  
    categoryID: number;  
    name: string;  
    description: string;  
}  
  
export class Product {  
    productID: number;  
    name: string;  
    unitsInStock: number;  
    unitprice: number;  
}
```

Dwa główne serwisy wysyłające restowe zapytania.

```
refreshList(categoryID: number) {  
    this.http.get<Product[]>(`${this.rootURL}/byCategory${categoryID}`)  
        .toPromise().then(res => {this.productList = res});  
    this.resetForm();  
}  
  
public postProduct(data: Product, categoryID: number): Observable<Product> {  
    let obj = {  
        Name: data.name,  
        UnitsInStock: data.unitsInStock,  
        Unitprice: data.unitprice,  
        CategoryID: categoryID  
    }  
    return this.http.post<Product>(this.rootURL, obj);  
}  
  
public putProduct(data: Product, categoryID: number): Observable<any> {  
    let obj = {  
        ProductID: data.productID,  
        Name: data.name,  
        UnitsInStock: data.unitsInStock,  
        Unitprice: data.unitprice,  
        CategoryID: categoryID  
    }  
    return this.http.put<any>(`${this.rootURL}/${data.productID}`, obj);  
}  
  
public deleteProduct(id: number): Observable<Product> {  
    return this.http.delete<Product>(`${this.rootURL}/${id}`);  
}
```

Siedem Komponentów

- Header
- Footer
- Categories główny komponent, na którym wyświetlane są :
 - Kategorie (Formularz + Aktualny Stan wszystkich kategorii)
 - Produkty (Formularz + Aktualny Stan wszystkich produktów, dla wybranej kategorii)

Z komponentów zostały wyrzucone pliki spec.ts (testy) oraz .css (wszystkie arkusze stylów są w globalnym pliku styles.css)

Html komponentu „Categories”, odpowiedzialny za wyświetlanie albo produktów albo kategorii.

```
<div>
  <app-header [title]="title" (homeClick)="onHomeClick($event)"></app-header>
  <div class="jumbotron">
    <div class="row">
      <div class="col-md-5">
        <app-category-form *ngIf = "!categoryID"></app-category-form>
        <app-product-form [categoryID]="categoryID" *ngIf = "categoryID"></app-product-form>
      </div>
      <div class="col-md-7">
        <app-categories-list *ngIf="!categoryID" (categoryEmitter)="assignID($event)"></app-categories-list>
        <app-products-list [categoryID]="categoryID" *ngIf="categoryID"></app-products-list>
      </div>
    </div>
  </div>
</div>
```

Wybrany kod do wyświetlania produktów:

```
export class ProductFormComponent implements OnInit {

  @Input() categoryID: number;

  constructor(private productService: ProductService) { }

  ngOnInit() {
    this.productService.resetForm();
  }

  onSubmit(form: NgForm) {
    if (form.value.productID == null)
      this.insertRecord(form);
    else
      this.updateRecord(form);
  }

  insertRecord(form: NgForm) {
    this.productService.postProduct(form.value, this.categoryID).subscribe(res => {
      this.productService.refreshList(this.categoryID);
    });
  }

  updateRecord(form: NgForm) {
    this.productService.putProduct(form.value, this.categoryID).subscribe(res => {
      this.productService.refreshList(this.categoryID);
    });
  }
}
```

```
<form #form="ngForm" (submit)="onSubmit(form)" autocomplete="off">
  <input type="hidden" name="productID" #productID="ngModel" [(ngModel)]= "productService.formData.productID">
  <div class="form-group">
    <label>Product Name</label>
    <input name="name" #name="ngModel" [(ngModel)]= "productService.formData.name" class="form-control" required>
    <div class="validation-error" *ngIf="name.invalid && name.touched">This field is required.</div>
  </div>
  <div class="form-group">
    <label>Units In Stock</label>
    <input name="unitsInStock" #unitsInStock="ngModel" [(ngModel)]= "productService.formData.unitsInStock" class="form-control" required>
    <div class="validation-error" *ngIf="unitsInStock.invalid && unitsInStock.touched">This field is required.</div>
  </div>
  <div class="form-group">
    <label>Unit Price</label>
    <input name="unitprice" #unitprice="ngModel" [(ngModel)]= "productService.formData.unitprice" class="form-control" required>
    <div class="validation-error" *ngIf="unitprice.invalid && unitprice.touched">This field is required.</div>
  </div>
  <div class="form-group">
    <button type="submit" [disabled]="form.invalid" class="btn btn-lg btn-block">SUBMIT</button>
  </div>
</form>
```

```

<table class="table table-hover">
  <tr>
    <td>Product Name</td>
    <td>Units In Stock</td>
    <td>Unit Price</td>
  </tr>
  <tr *ngFor="let product of productService.productList">
    <td (click)="populateForm(product)">{{product.name}}</td>
    <td (click)="populateForm(product)">{{product.unitsInStock}}</td>
    <td (click)="populateForm(product)">{{product.unitprice}}</td>
    <td><button (click)="onDelete(product.productId)" class="btn btn-sm btn-outline-danger">X</button></td>
  </tr>
</table>

```

Działanie aplikacji (po uruchomieniu skryptu z działającym backendem):

Category Registration

Category Name

Description

SUBMIT

Dawid Majchrowski 2019

Dodajemy kilka kategorii za pomocą formularza:

Category Registration

Category Name

This field is required.

Description

SUBMIT

Category Name	Description		
cat1	desc1	M	X
cat2	desc2	M	X
cat3	desc3	M	X

Dawid Majchrowski 2019

```
select * from Categories;
```

CategoryID	Name	Description
16	cat1	desc1
17	cat2	desc2
18	cat3	desc3

Naciskamy na tabele po prawej stronie, na wysokości cat2

Category Registration

Category Name

cat2

Description

desc2

SUBMIT

Category Name	Description		
cat1	desc1	M	X
cat2	desc2	M	X
cat3	desc3	M	X

Dawid Majchrowski 2019

Zmieniamy opis na „desc4” i klikamy submit

Category Registration

Category Name

Description

SUBMIT

Category Name	Description		
cat1	desc1	M	X
cat2	desc4	M	X
cat3	desc3	M	X

Dawid Majchrowski 2019

```
select * from Categories;
```

	CategoryID	Name	Description
1	16	cat1	desc1
2	17	cat2	desc4
3	18	cat3	desc3

Update danych działa poprawnie. Klikamy teraz na pole M po prawej stronie tabeli (cat 2). Pojawia nam się pole do rejestracji produktów, zatem dodajemy ponownie ich kilka. Zauważmy, że w tym miejscu potrzebujemy produkt należące tylko dla danej kategorii, dlatego skorzystamy z dodanego przez nas getByCategory w kontrolerze produktu.

```
refreshList(categoryID: number) {
  this.http.get<Product[]>(`${this.rootURL}/byCategory${categoryID}`)
    .toPromise().then(res => {this.productList = res});
  this.resetForm();
}
```

Product Registration

Product Name

Units In Stock

0

Unit Price

0

SUBMIT

Product Name	Units In Stock	Unit Price

Dawid Majchrowski 2019

Dodajemy kilka produktów. Dla kilku różnych kategorii.

Product Name

Units In Stock

0

Unit Price

0

SUBMIT

Product Name	Units In Stock	Unit Price	
Prod1ForCat2	1	2	X
Prod2ForCat2	31	33	X

Dawid Majchrowski 2019

```
7 select * from Categories;
8
9 select * from Products;
```

ProductID	Name	UnitsInStock	Unitprice	CategoryID
40	Prod1ForCat2	1	2.0000	17
41	Prod2ForCat2	31	33.0000	17
42	Prod1ForCat3	1	0.0000	18

Usuwamy Prod1ForCat2 oraz Cała kategorie 3 i sprawdzamy co się stanie.

```
9 select * from Products;
```

ProductID	Name	UnitsInStock	Unitprice	CategoryID
41	Prod2ForCat2	31	33.0000	17

Jak widać usunięcie kategorii usuwa również produkty do nie należące.

Analogicznie sprawdzamy resztę funkcjonalności.

Kod projektu:

<https://github.com/majchrow/Databases/tree/master/assignment2/WebApi>