

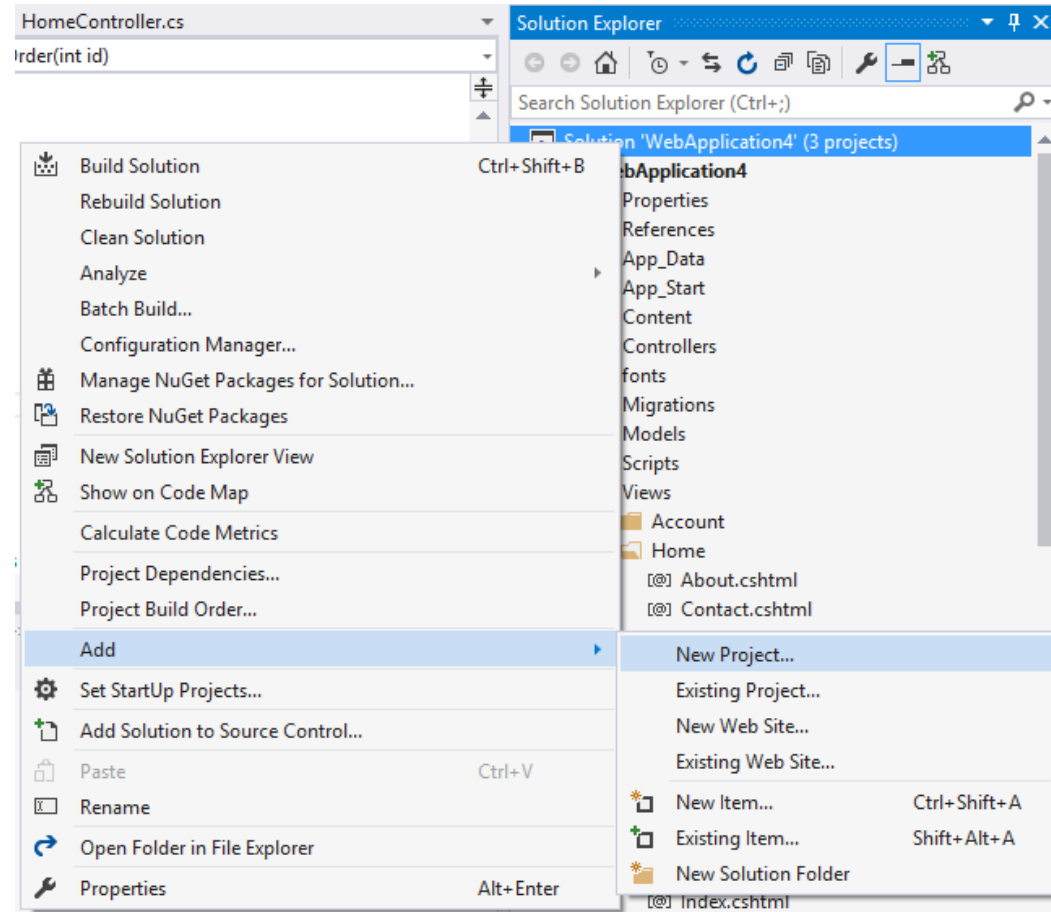
Kurs ASP.NET MVC

CRUD i Entity Framework

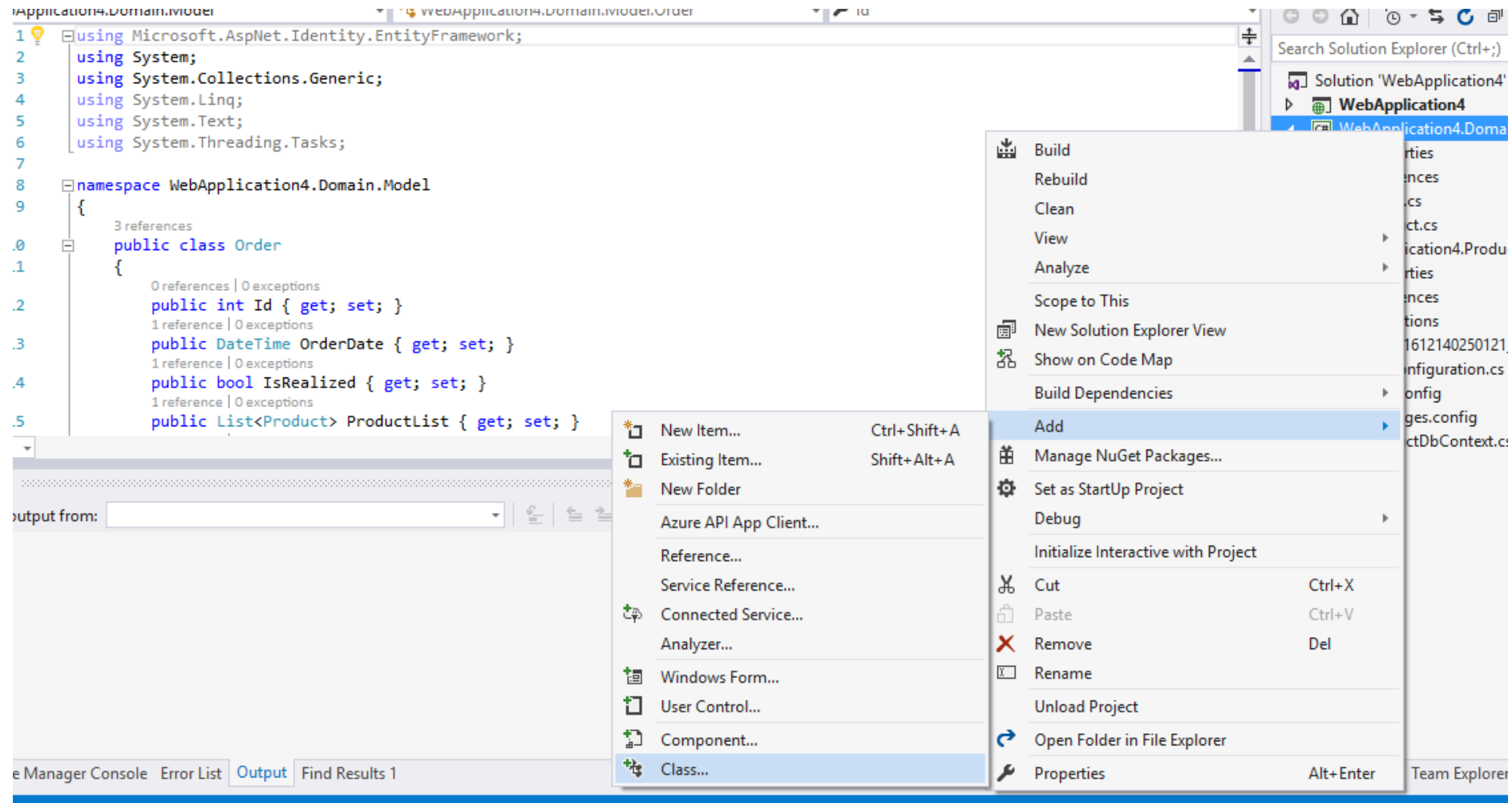
Plan:

- Stworzyć model danych
- Stworzyć i uruchomić bazę danych
- Stworzyć podstawowe widoki i kontrolery

Dodajemy nowy projekt typu "Class Library" i nazywamy go Domain.Model



Utwórzmy nowe klasy opisujące obiekty przechowywane w naszej bazie danych

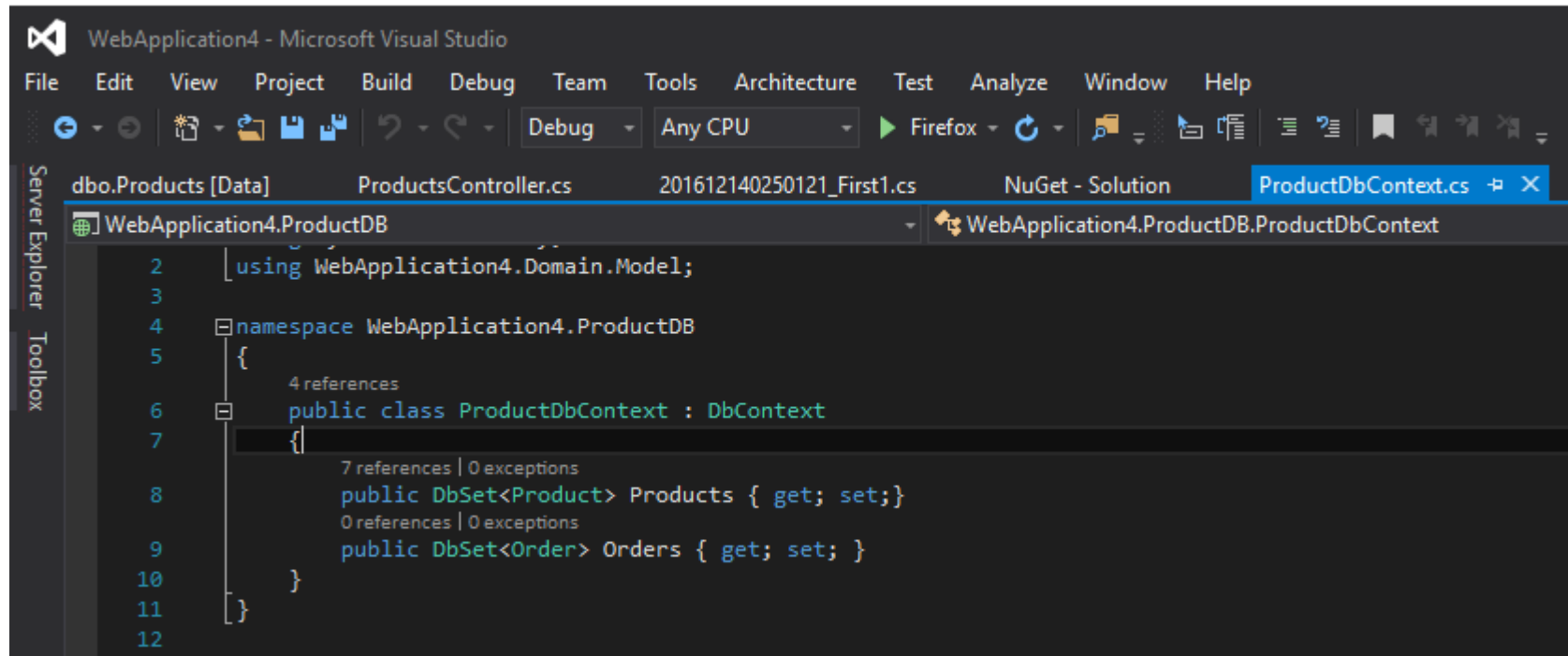


Utwórzmy tak jak wcześniej nowy projekt "class library" i nazwijmy go database.

Dodajmy w nim klasę ProductDbContext dziedziczącą po DbContext.

Klasa ta zawiera deklaracje tabel w bazie
(publiczne właściwości typu DbSet<nazwa_model>)

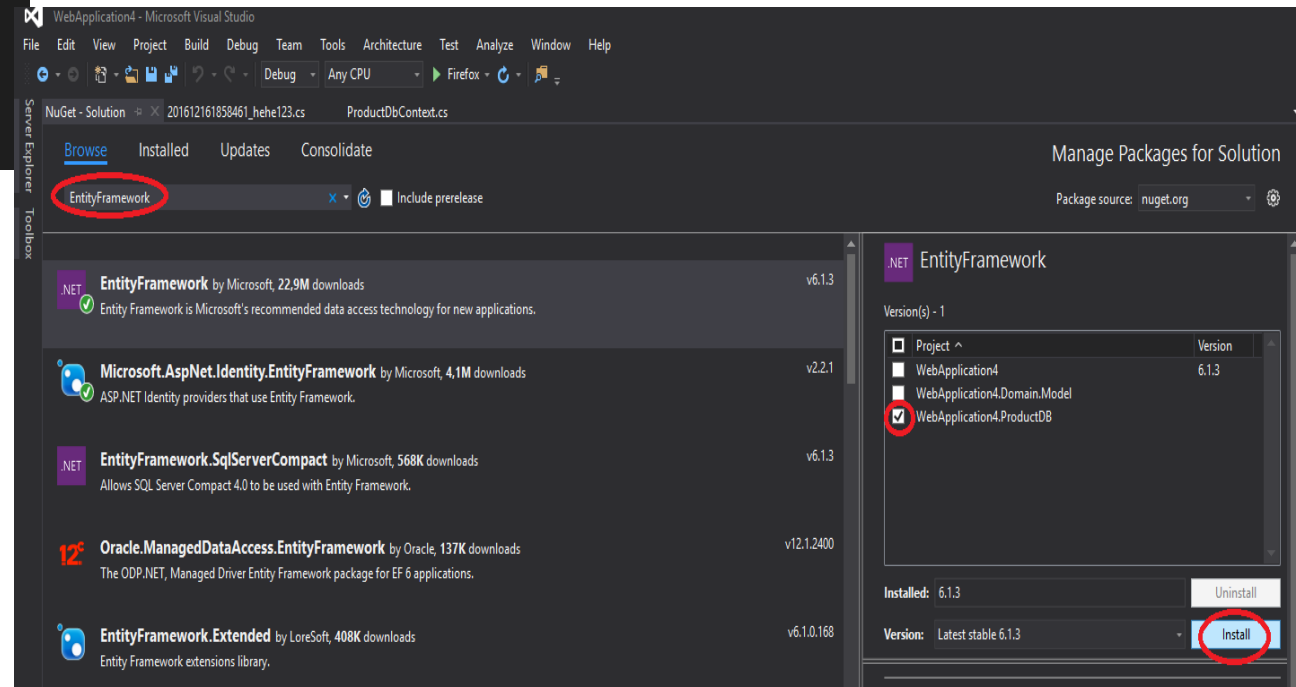
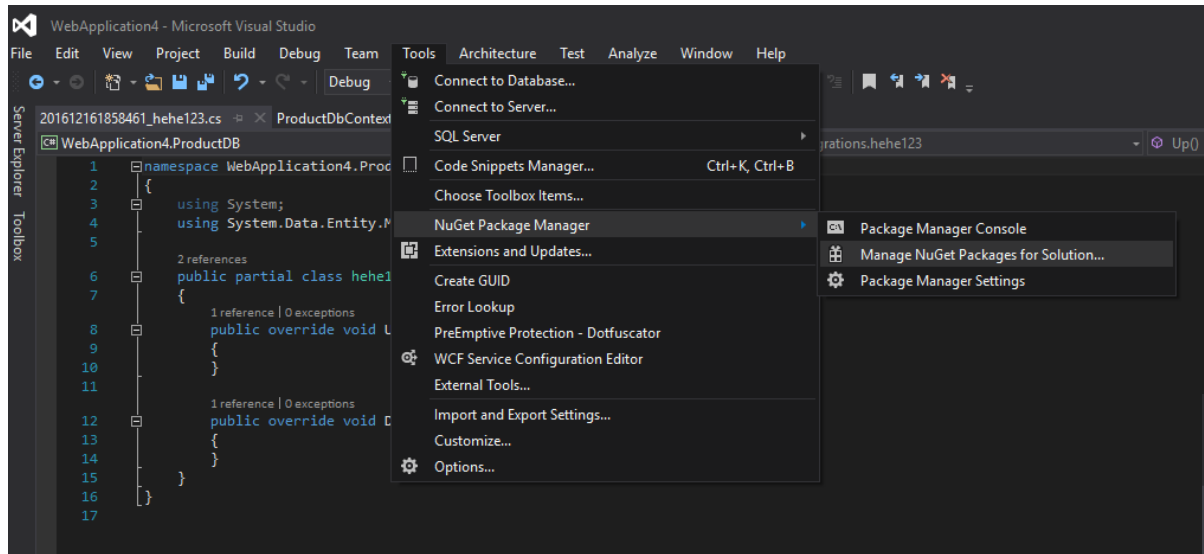
Tworzymy ją aby zdefiniować tabele naszej bazy i móc się z nią komunikować.



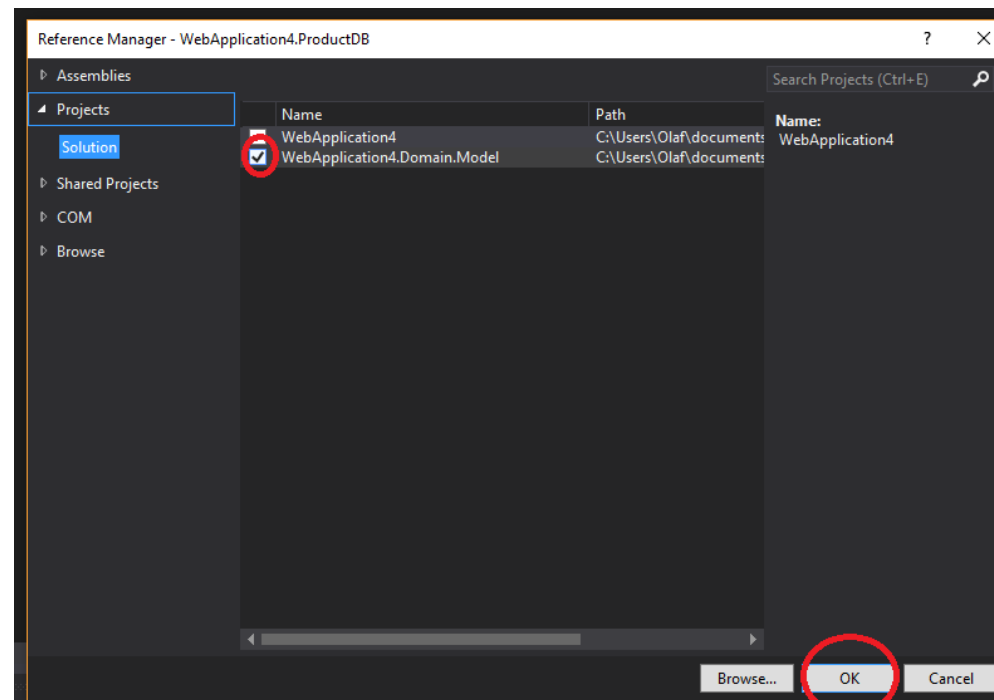
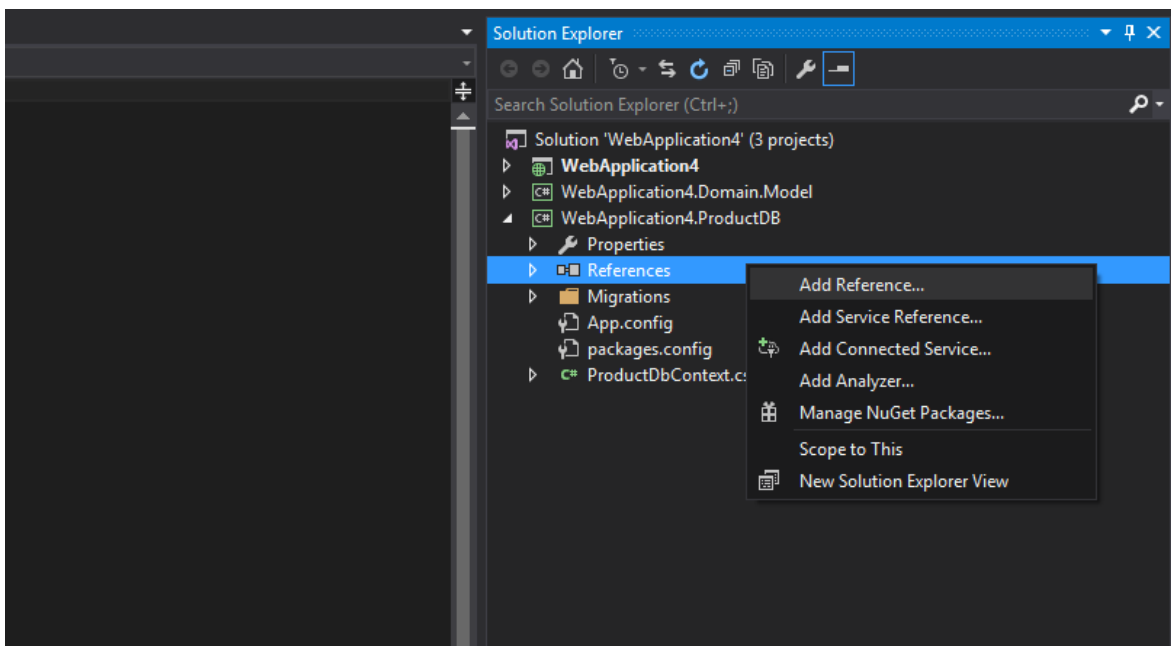
The screenshot shows the Visual Studio IDE with the 'ProductDbContext.cs' file open. The file is part of a class library project named 'WebApplication4.ProductDB'. The code defines a 'ProductDbContext' class that inherits from 'DbContext'. It includes two DbSet properties: 'Products' of type 'DbSet<Product>' and 'Orders' of type 'DbSet<Order>'. The code is as follows:

```
2 using WebApplication4.Domain.Model;
3
4 namespace WebApplication4.ProductDB
5 {
6     4 references
7     public class ProductDbContext : DbContext
8     {
9         7 references | 0 exceptions
10        public DbSet<Product> Products { get; set; }
11        0 references | 0 exceptions
12        public DbSet<Order> Orders { get; set; }
13    }
14 }
```

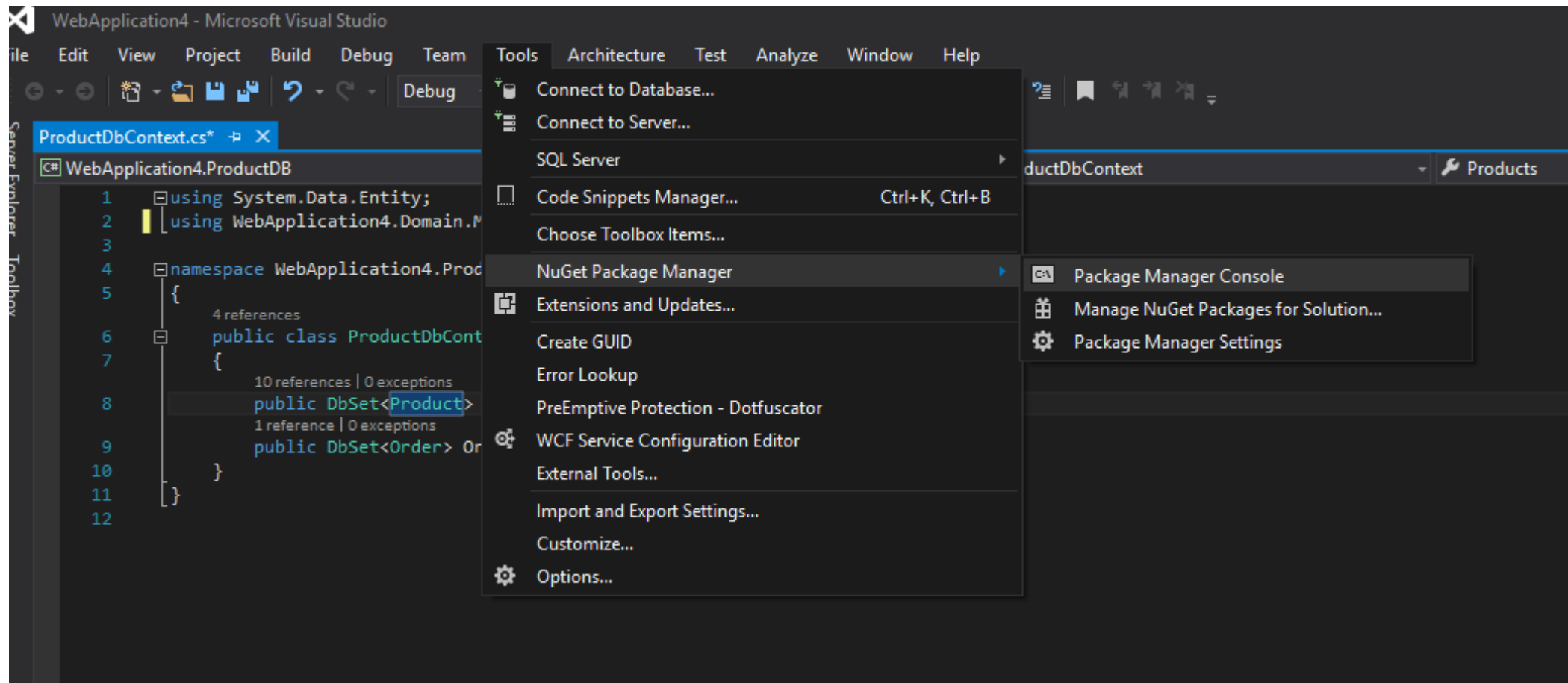
Dodajemy do projektu z bazą danych biblioteki Entity Framework – potrzebne, by uruchomić „silnik” naszej bazy danych, bez tego nie utworzymy bazy danych!



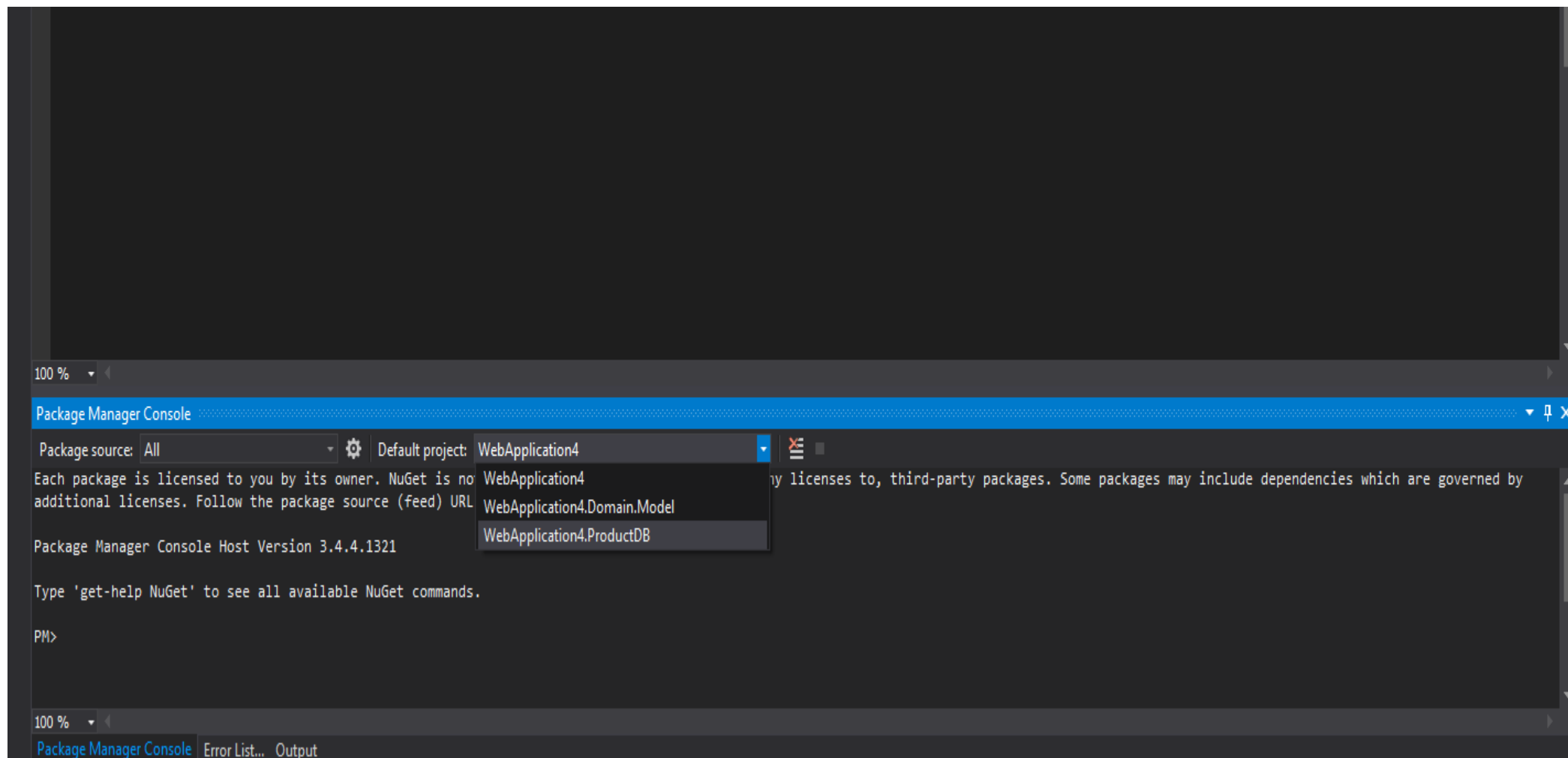
Dodajmy referencje do projektu Database – dajmy mu dostęp do projektu model. Po dodaniu możemy używać w projekcie z ProductDB klas z projektu Model! (jeśli wciąż podkreśla nazwy klas stawiamy kursor w miejscu podkreślenia, wciskamy Ctrl + kropka i wybieramy „using...”).



Odpalamy konsolkę, czas utworzyć naszą lokalną bazę :)



Wybieramy projekt z bazą z rozwijalnej listy, i kolejno wbijamy do konsoli: enable-migrations, add-migration „InitialMigration” (lub jakakolwiek inna nazwa pierwszej migracji), update-database



Nasza nowo powstała baza danych w żaden sposób nie komunikuje się z głównym projektem, w którym zawiera się nasza strona, dlatego czas ją podłączyć, robimy to w następujący sposób:

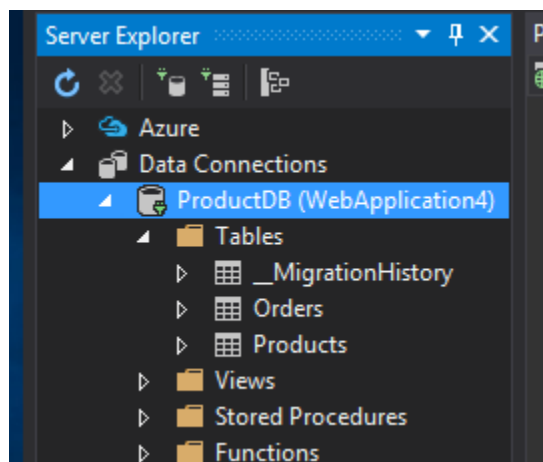
W głównym projekcie znajduje się plik Web.config, otwieramy go, dublujemy linijkę pomiędzy tagami <connectionStrings> </connectionStrings> i podmieniamy w niej dane:

name – nazwa bazy danych,

AttachDbFilename – w folderze z projektem (domyslnie Dokumenty/Visual Studio

2015/Projects/nazwa_projektu) wchodzimy do katalogu App_Data i podstawiamy nazwę odpowiedniego pliku (rozpoznanie po nazwie) który tam znajdziemy do tego parametru

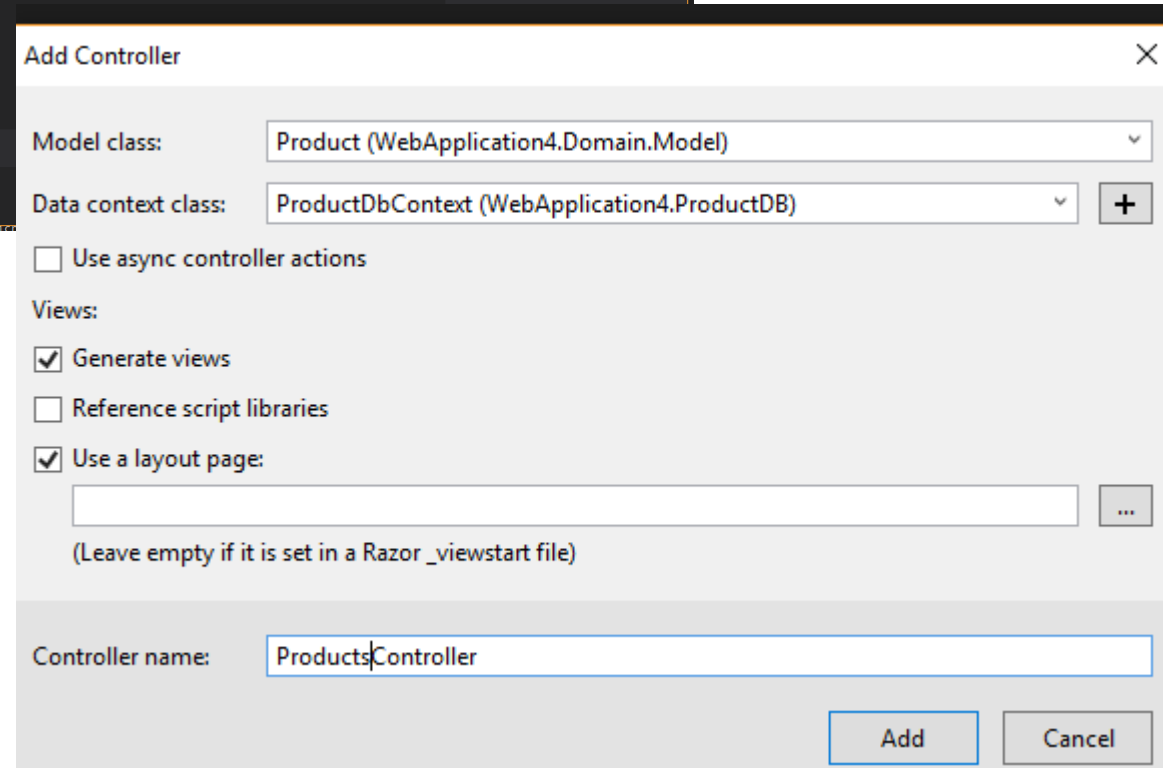
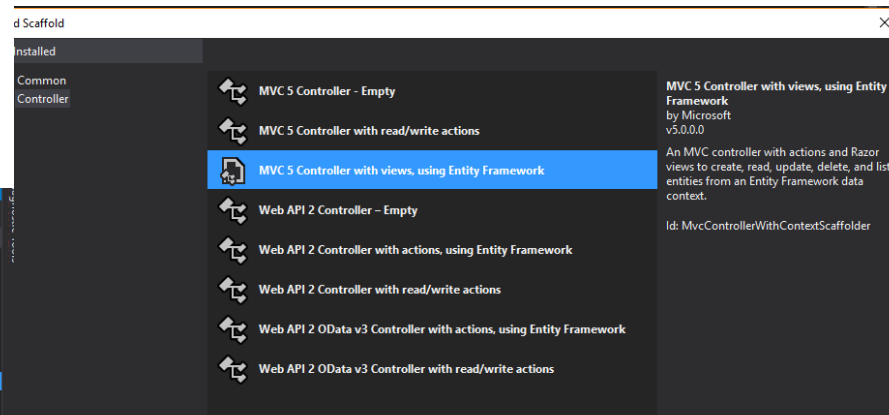
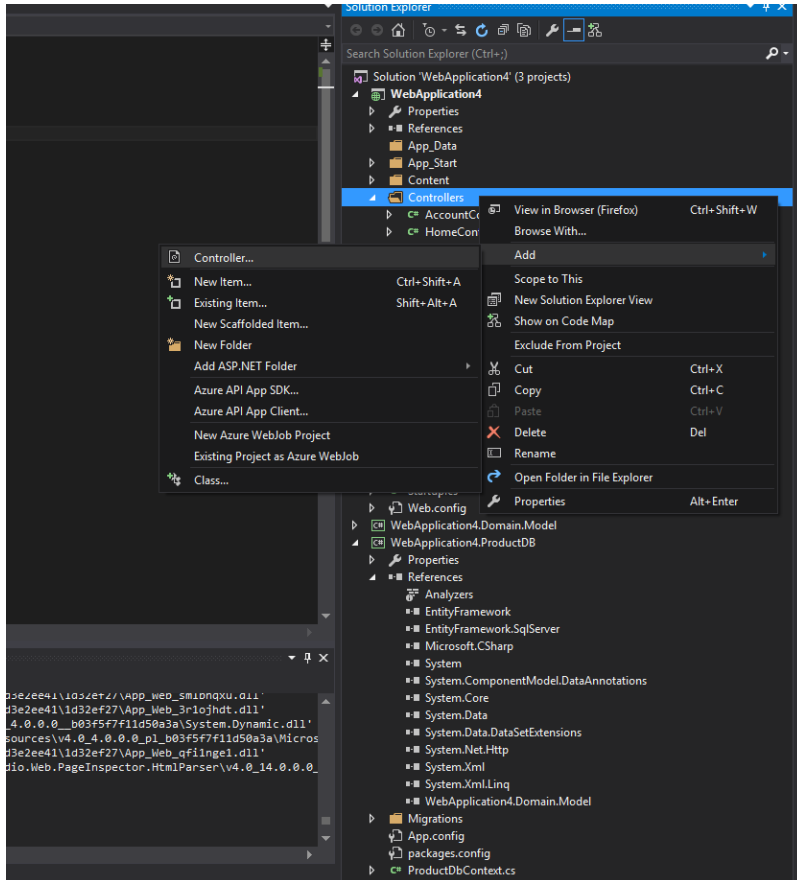
Teraz nasza baza już jest podpięta do aplikacji. Po uruchomieniu, w menu View>Server Explorer można zobaczyć połączenie do niej, automatycznie utworzone podczas migracji tabele i ich kolumny. Są takie same jak właściwości w naszym modelu!



Tworzymy Controller!

Jak wiemy z poprzednich zajęć, jest to pośrednik między Modelem (który u nas obsługuje) a widokiem (czyli tym co widzi użytkownik).

Przyjmuje on dane, obrabia je i zwraca do bazy lub użytkownika w formie widoku. Tym razem nie będziemy pisać wszystkiego ręcznie, wykorzystamy wbudowany w Visual Studio generator.



Wybieramy „MVC 5 Controller with Views, using Entity Framework”
Wybieramy model na którym chcemy pracować(jeżeli go nie ma,
dodajemy do głównego projektu referencję do projektu z modelem)),
W polu Data context class wybieramy połączenie bazy danych czyli w naszym przypadku
Klasę ProduktDbContext. Klikamy ok.

```
ProductsController.cs 201612140250121_First1.cs NuGet - Solution ProductDbContext.cs
WebApplication4
using WebApplication4.ProductDB;
namespace WebApplication4.Controllers
{
    References
    public class ProductsController : Controller
    {
        private ProductDbContext db = new ProductDbContext();

        // GET: Products
        References | 1 request | 0 exceptions
        public ActionResult Index()
        {
            return View(db.Products.ToList());
        }

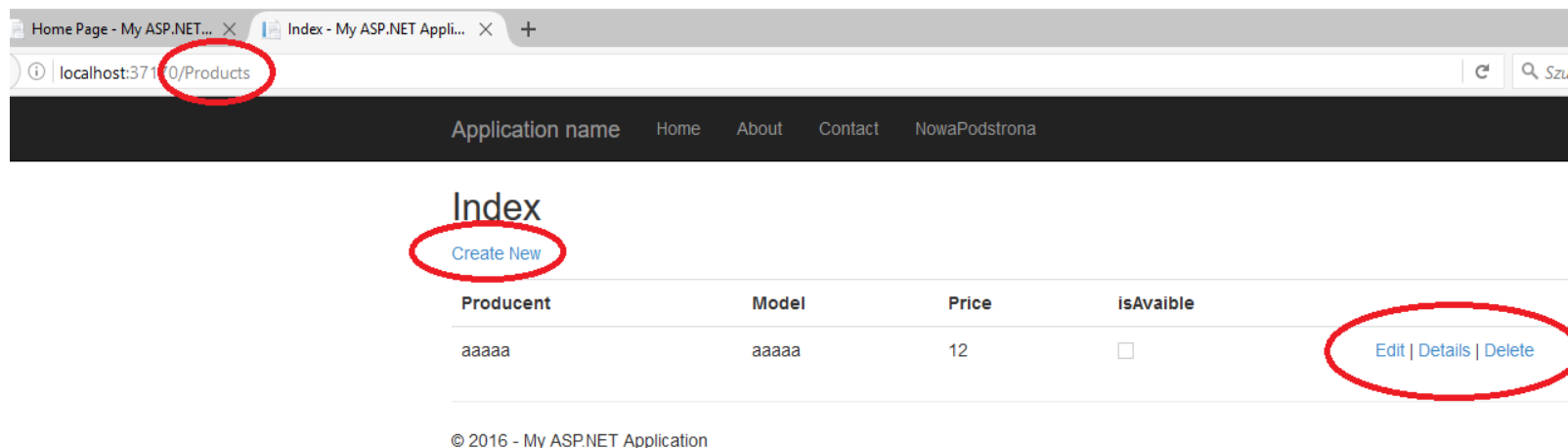
        // GET: Products/Details/5
        References | 0 requests | 0 exceptions
        public ActionResult Details(int? id)
        {
            if (id == null)
            {
                return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
            }
            Product product = db.Products.Find(id);
            if (product == null)
            {
                return HttpNotFound();
            }
            return View(product);
        }

        // GET: Products/Create
        References | 0 requests | 0 exceptions
        public ActionResult Create()
        {
            return View();
        }

        // POST: Products/Create
        // To protect from overposting attacks, please enable the specific properties you want to bind to, for
    }
}
```

Utworzył się gotowy controller z akcjami CRUD (Create Read Update Delete) oraz widoki do niego w

Po uruchomieniu aplikacji i odwołaniu się do akcji index naszego nowego Controllera mamy możliwość dodania nowego produktu do tabeli w naszej bazie danych :)



The screenshot shows a web browser with two tabs: 'Home Page - My ASP.NET...' and 'Index - My ASP.NET Appli...'. The address bar shows 'localhost:3710/Products'. The page has a dark navigation bar with links: 'Application name', 'Home', 'About', 'Contact', and 'NowaPodstrona'. Below the navigation bar, the word 'Index' is displayed. Under 'Index', there is a blue link 'Create New'. Below this is a table with the following columns: 'Producent', 'Model', 'Price', 'isAvaible', and an action column. The table contains one row with the values 'aaaaa', 'aaaaa', '12', and an unchecked checkbox. The action column for this row contains the links 'Edit | Details | Delete'. At the bottom of the page, there is a copyright notice: '© 2016 - My ASP.NET Application'.

Producent	Model	Price	isAvaible	
aaaaa	aaaaa	12	<input type="checkbox"/>	Edit Details Delete

Słowniczek pojęć:

Entity Framework – silnik Bazy Danych platformy .NET

Referencja – in. odwołanie, dodając referencję do projektu lub biblioteki, umożliwia to korzystanie z niego.

Zadanie domowe:

Utwórz projekt o nazwie application i przenieś do niego wszystkie odwołania do bazy danych.

W projekcie Application powinny być zaimplementowane metody dodaj,usun itp.

Pamiętaj o referencji do projektu z bazą!

W Controllerze mają być tylko wywołania metod (np.. dodajDoBazy) a nie bezpośrednie działanie na bazie!

Tak, żeby główny projekt nie musiał korzystać z ProductDbContext, ale korzystał z projektu Application,

Który dopiero korzysta z bazy danych i ProductDbContext.

W ten sposób oddzielisz i uniezależnisz logikę połączenia

Z bazą danych, co sprawi że nie będziesz powtarzać kodu oraz ułatwi testowanie.

Budując aplikację w ten sposób stosujemy wzorzec Onion Architecture.