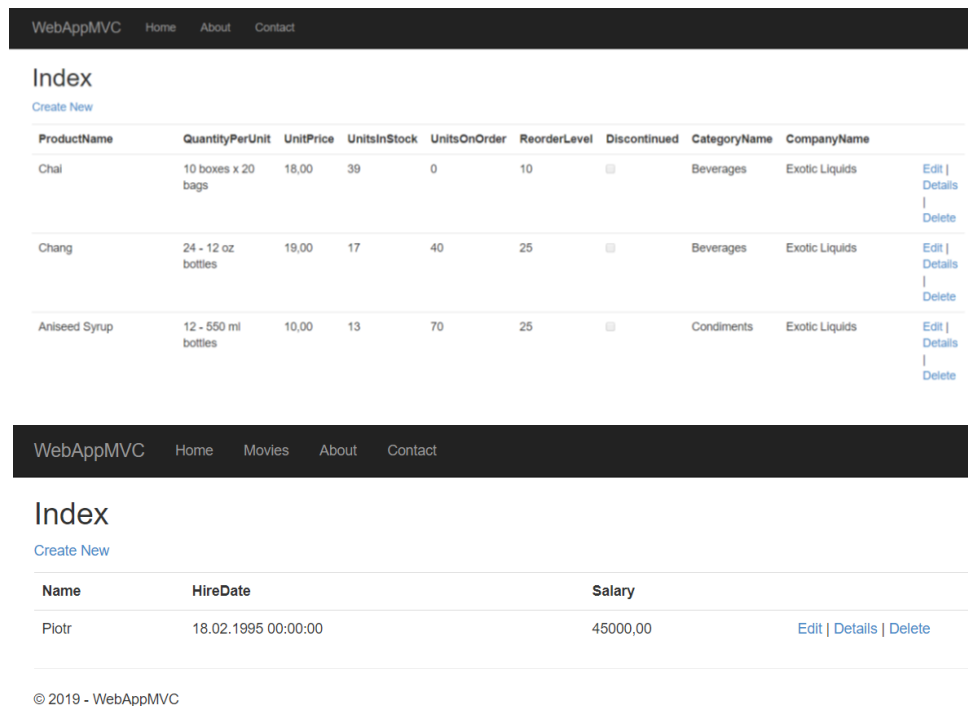
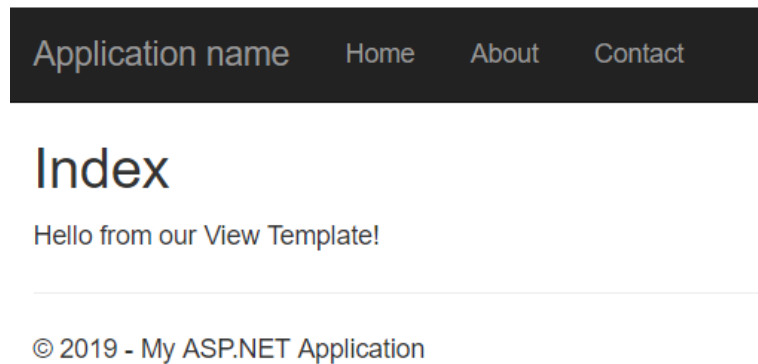


Laboratorium 8

Wprowadzenie do ASP.NET MVC

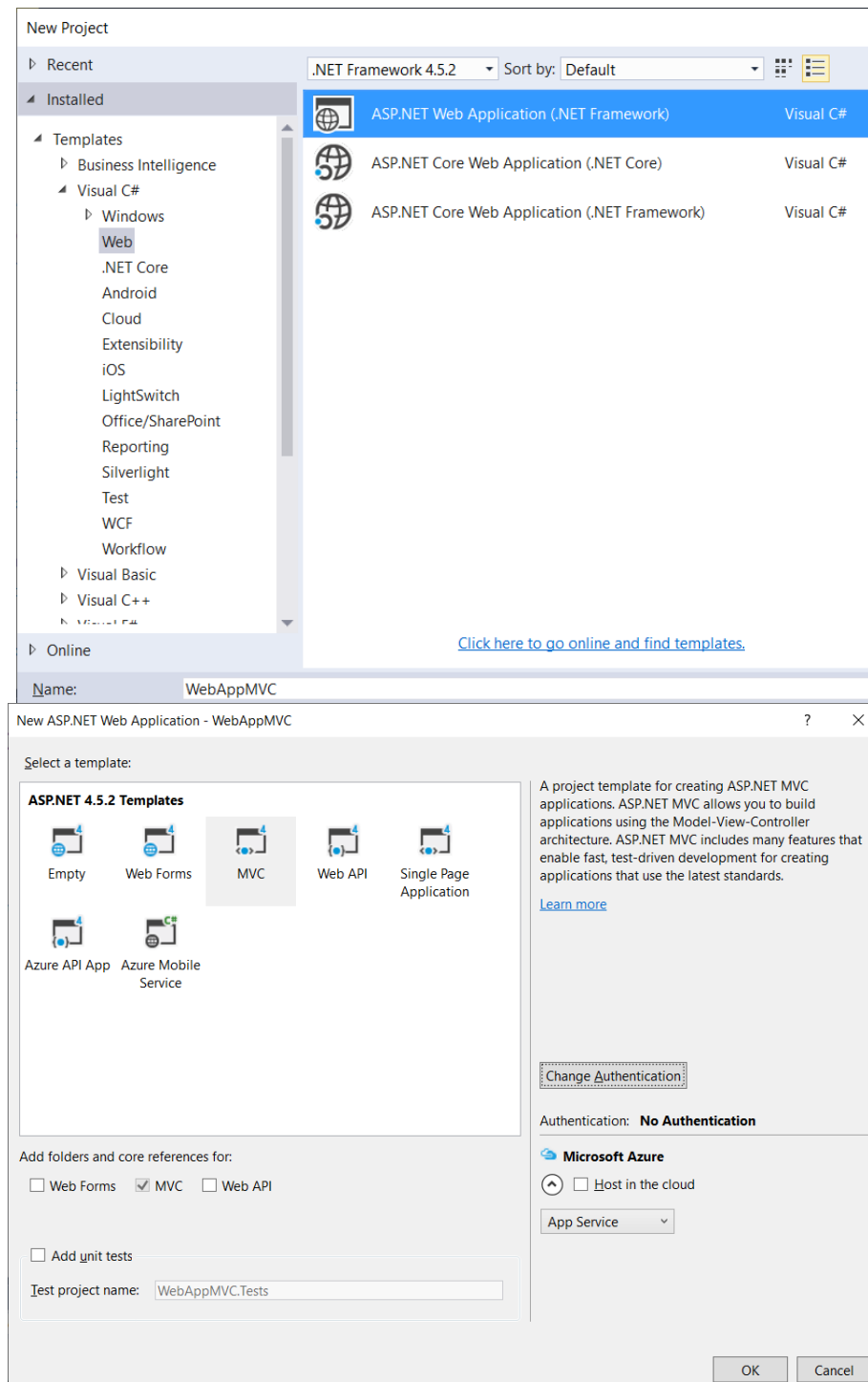
Celem ćwiczenia jest przygotowanie prostej aplikacji ASP.NET MVC, w której warstwa prezentacji przedstawia dane w sposób pokazany na rysunku (w wersji początkowej) a docelowo operacje CRUD na bazie danych.

ASP.NET MVC wykorzystuje kontrolery i widoki. Strony Razor są alternatywą ASP.NET, oparty na modelu, który sprawia, że tworzenie interfejsu użytkownika sieci web jest łatwiejsze i bardziej wydajne.



1. Tworzymy aplikację WebAppMVC

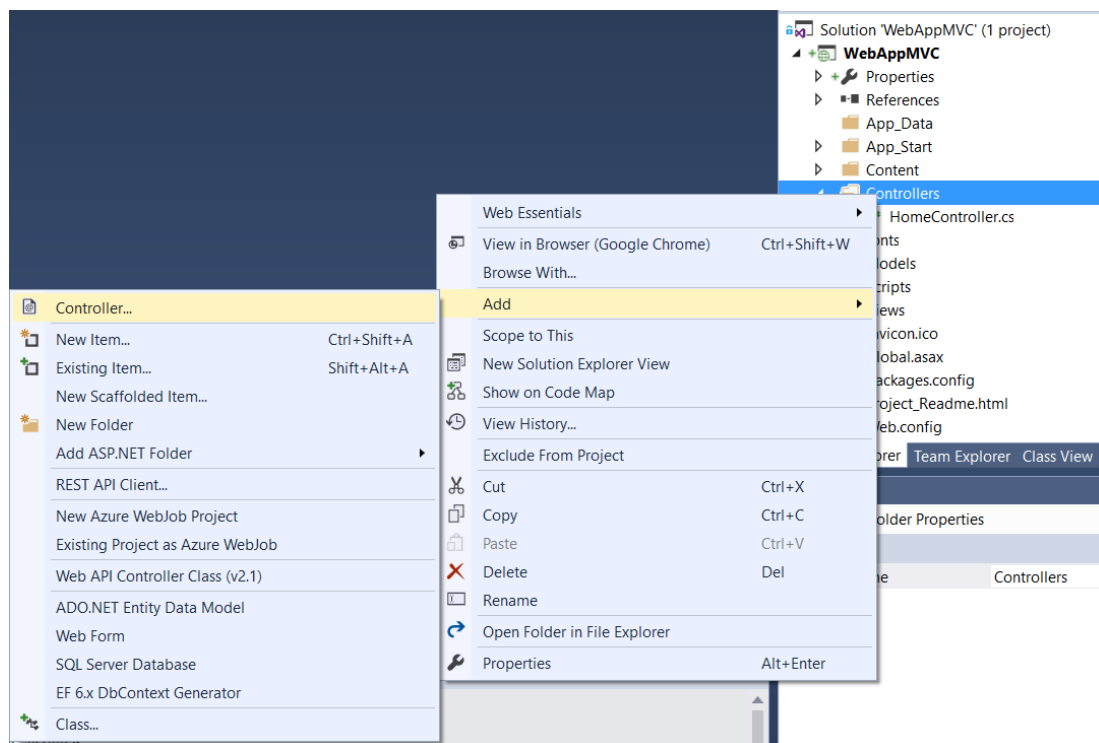
Otwieramy nowy project - ASP.NET Web Application (.NET Framework)



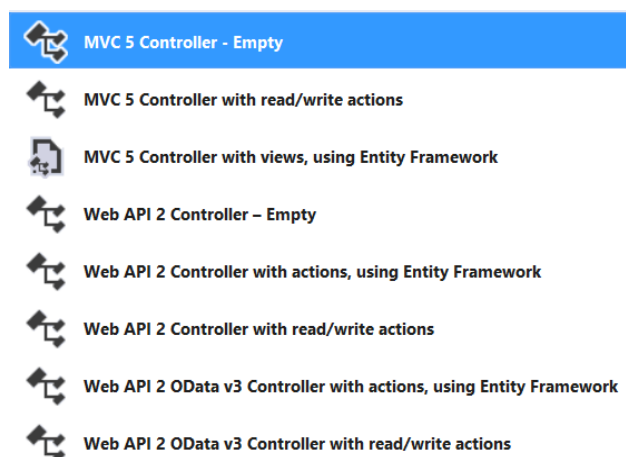
Uruchamiamy aplikację i przeglądamy jej elementy w przeglądarce po uruchomieniu serwera aplikacji.

2. Dodawanie kontrolera

Tworzymy klasę kontrolera o nazwie HelloWorldController. W Eksploratorze rozwiązań, kliknij prawym przyciskiem myszy folder Controllers, a następnie kliknij przycisk Dodaj, następnie kontroler.

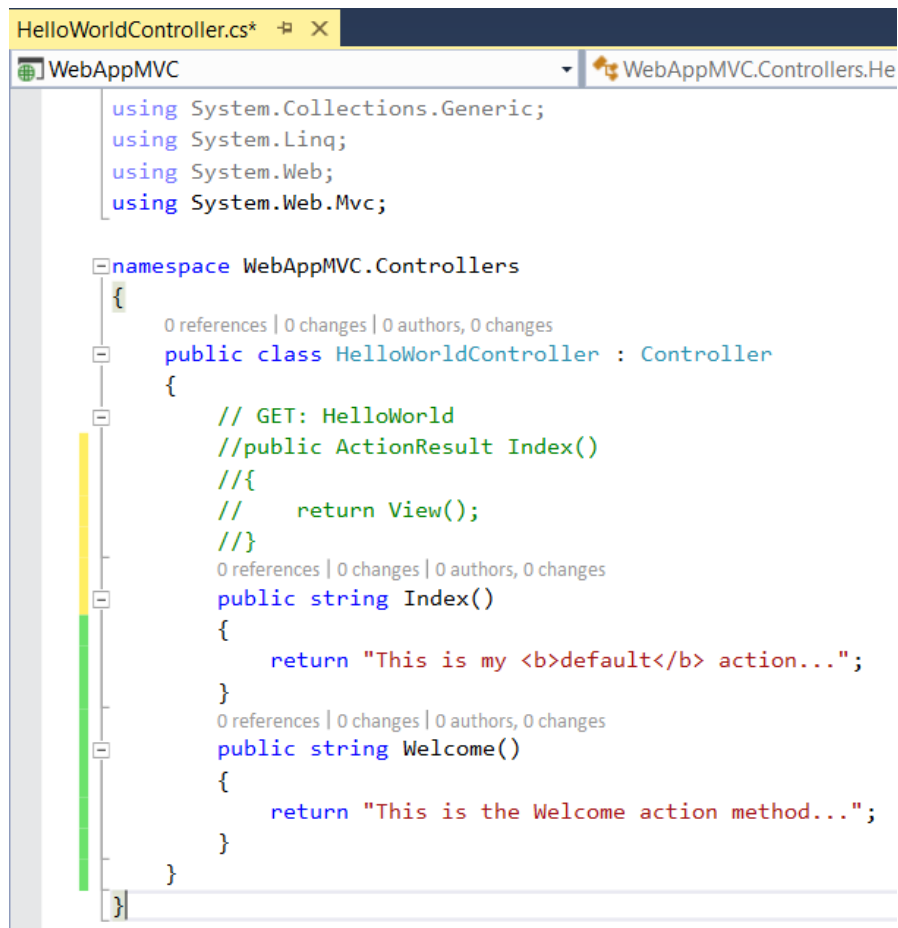


W oknie dialogowym szkieletu (Scaffold), wybierz przycisk MVC 5 Controller - Empty, a następnie kliknij przycisk Dodaj.



Został utworzony plik o nazwie *HelloWorldController.cs* i nowy pusty w środku folder *Views\HelloWorld*.

Zastąp zawartość pliku *HelloWorldController.cs* następującym kodem.



```
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

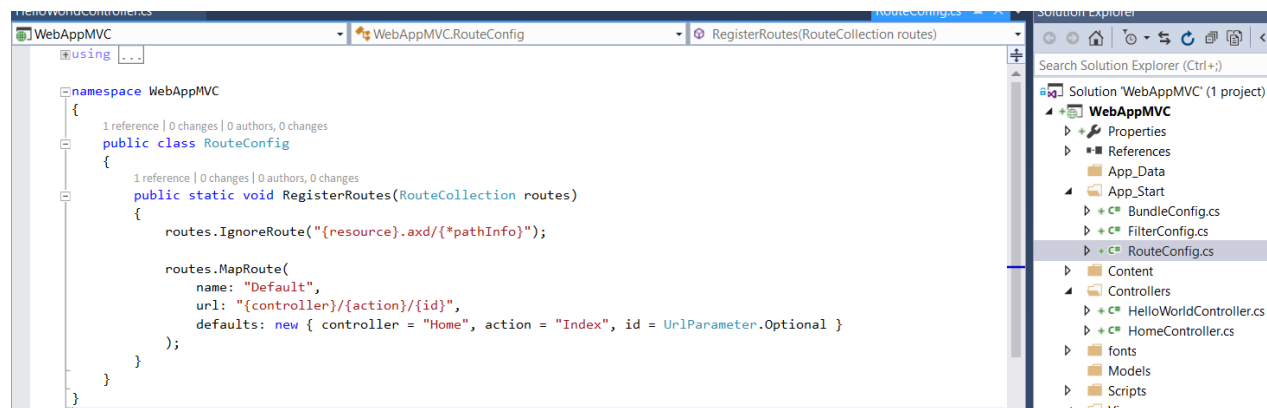
namespace WebAppMVC.Controllers
{
    0 references | 0 changes | 0 authors, 0 changes
    public class HelloWorldController : Controller
    {
        // GET: HelloWorld
        //public ActionResult Index()
        //{
        //    return View();
        //}
        0 references | 0 changes | 0 authors, 0 changes
        public string Index()
        {
            return "This is my <b>default</b> action...";
        }
        0 references | 0 changes | 0 authors, 0 changes
        public string Welcome()
        {
            return "This is the Welcome action method...";
        }
    }
}
```

Sprawdzamy wywołanie stron:

- **http://localhost:1234/HelloWorld**
lub
- **http://localhost:1234/HelloWorld/Index**
oraz
- **http://localhost:1234/HelloWorld/Welcome**

ASP.NET MVC wywołuje kontrolera klasy (i różne metody akcji w nich), w zależności od przychodzącego adresu URL. Logika routingu domyślnego adresu URL używany przez program ASP.NET MVC używa formatu następującego:
/[Controller]/[ActionName]/[Parameters]

Plik routingu znajduje się w projekcie w pliku **App_Start/RouteConfig.cs**.



W **HelloWorldController.cs** dokładamy nową akcję **Welcome1** z parametrami

```
public string Welcome()
{
    return "This is the Welcome action method...";
}

public string Welcome1(string name, int numTimes = 1)
{
    return HttpUtility.HtmlEncode("Hello " + name + ", NumTimes is: " + numTimes);
}
```

Uruchamianie adresu z parametrami wygląda następująco:

<http://localhost:1234/HelloWorld/Welcome1?name=Nowak&numtimes=3>

Opis metody `HttpUtility.HtmlEncode` –

https://docs.microsoft.com/pl-pl/dotnet/api/system.web.httputility.htmlencode?redirectedfrom=MSDN&view=netframework-4.8#System_Web_HttpUtility_HtmlEncode_System_Object

Dokładamy nowy element routingu:

```
public class RouteConfig
{
    1 reference | 0 changes | 0 authors, 0 changes
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
        );

        routes.MapRoute(
            name: "Hello",
            url: "{controller}/{action}/{name}/{numTimes}"
        );
    }
}
```

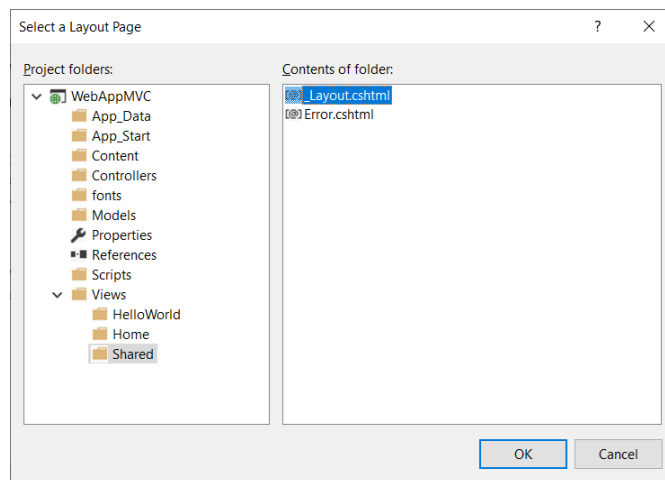
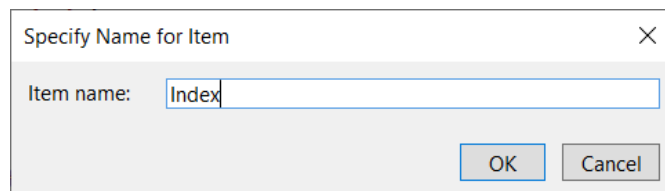
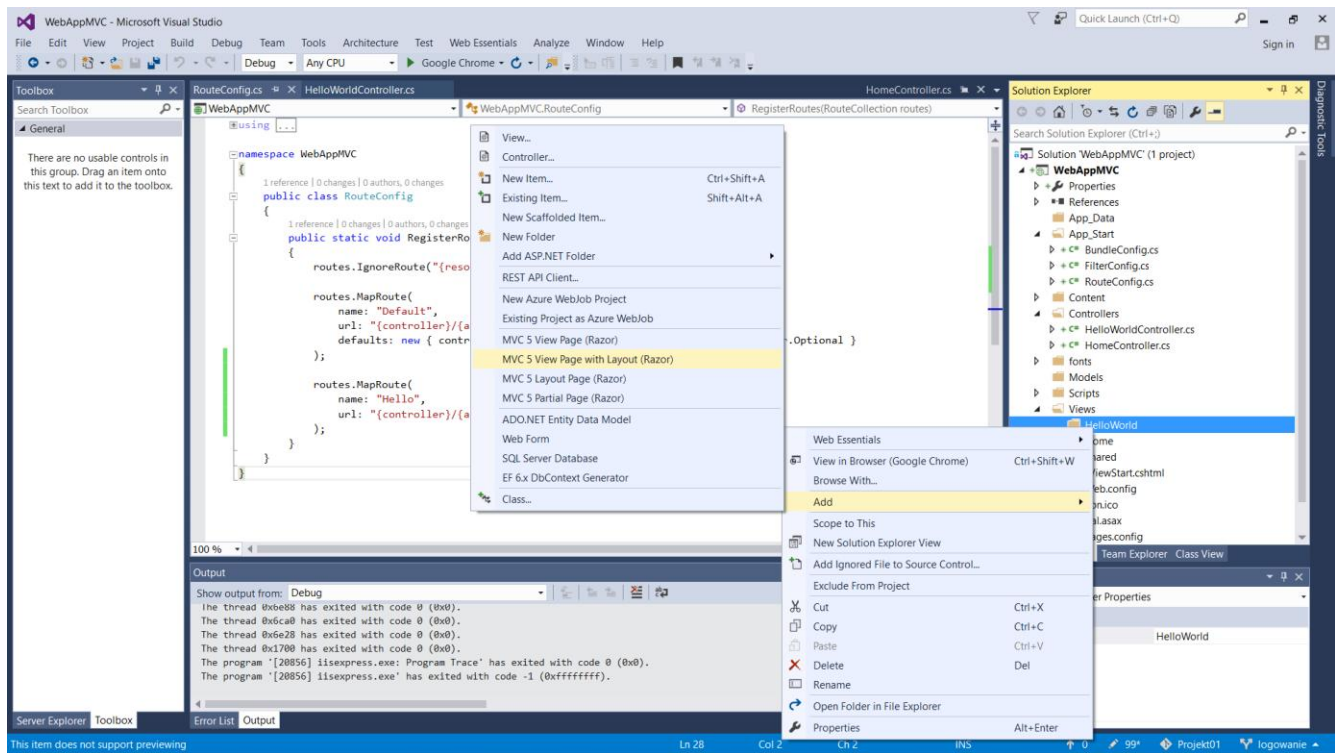
Uruchamianie adresu z parametrami w następujący sposób:

<http://localhost:1234/HelloWorld/Welcome1/Nowak/3>

W naszym przypadku kontroler zwraca HTML bezpośrednio jako string co normalnie nie jest tak realizowane tylko za pomocą metody View() – w dalszej części ćwiczenia.

3. Dodawanie widoku do akcji danego kontrolera

Modyfikujemy HelloWorldController w celu użycia widoku pliku szablonu



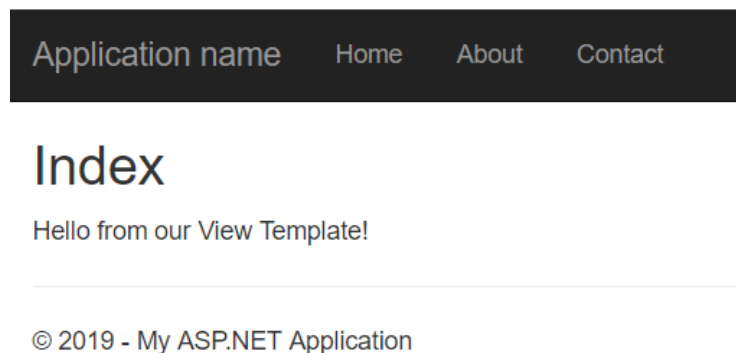
Zostanie utworzony plik: **WebAppMVC\Views\HelloWorld\Index.cshtml**.

Zamieniamy dla akcji Index typ na ActionResult i return View();

```
0 references | 0 changes | 0 authors, 0 changes
} public class HelloWorldController : Controller
{
    // GET: HelloWorld
    //public ActionResult Index()
    //{
    //    return View();
    //}
    0 references | 0 changes | 0 authors, 0 changes
    public ActionResult Index()
    {
        return View();
    }
    0 references | 0 changes | 0 authors, 0 changes
    public string Welcome()
    {
        return "This is the Welcome action method...";
    }
    0 references | 0 changes | 0 authors, 0 changes
    public string Welcome1(string name, int numTimes = 1)
    {
        return HttpUtility.HtmlEncode("Hello " + name + ", NumTimes is: " + numTimes);
    }
}
}
```

```
Index.cshtml  RouteConfig.cs  HelloWorldController.cs
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>Hello from our View Template!</p>
```

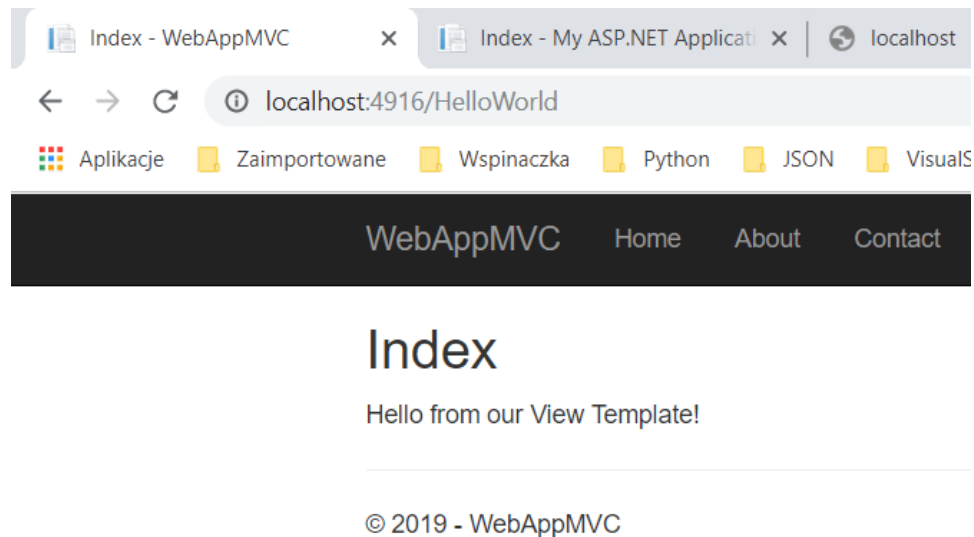
I przy uruchomieniu <http://localhost:1234/HelloWorld/Index> otrzymamy



4. Zmienianie widoków i układu strony

Najpierw należy zmienić "Nazwę aplikacji" widoczną u góry strony. Ten tekst jest wspólny dla każdej strony. Faktycznie wartość ta jest stosowana w jednym miejscu w projekcie, nawet jeśli jest wyświetlana na każdej stronie w aplikacji. Przejdź do /Views/Shared, a następnie otwórz plik _Layout.cshtml.

- Zmiana nazwy aplikacji na WebAppMVC w pliku Views\Shared_Layout.cshtml (w title, menu i footer)



5. Przekazywanie danych z kontrolera do widoku

Dla kontrolera HelloWorldController zmieniamy Welcome1 z dwoma parametrami i definiujemy widok o tej samej nazwie.

```
namespace WebAppMVC.Controllers
{
    0 references | 0 changes | 0 authors, 0 changes
    public class HelloWorldController : Controller
    {
        0 references | 0 changes | 0 authors, 0 changes
        public ActionResult Index()
        {
            return View();
        }
        0 references | 0 changes | 0 authors, 0 changes
        public string Welcome()
        {
            return "This is the Welcome action method...";
        }
        0 references | 0 changes | 0 authors, 0 changes
        public ActionResult Welcome1(string name, int numTimes = 1)
        {
            ViewBag.Message = "Hello " + name;
            ViewBag.NumTimes = numTimes;

            return View();
        }
    }
}

HelloWorldController.cs* | Welcome1.cshtml | _ViewStart.cshtml

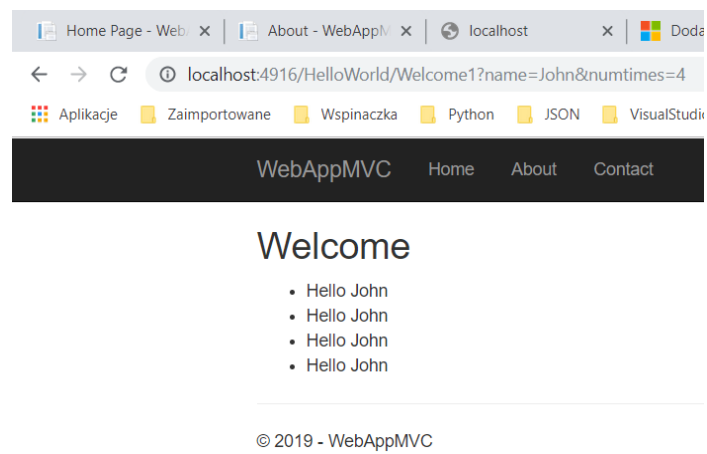
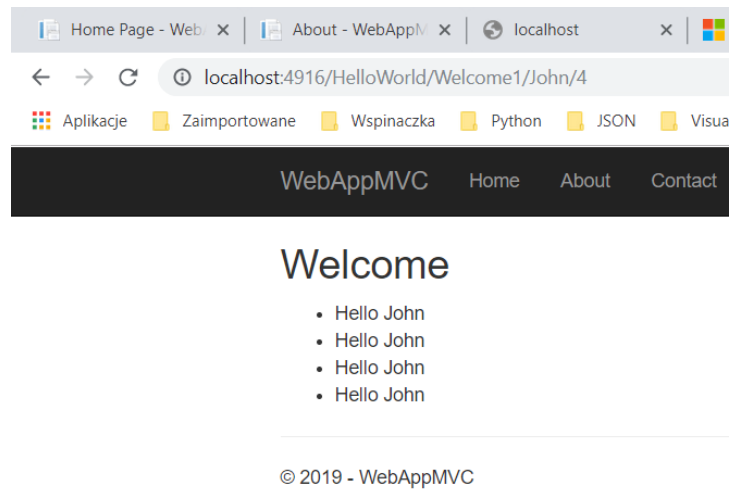
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

@{
    ViewBag.Title = "Welcome";
}

<h2>Welcome</h2>

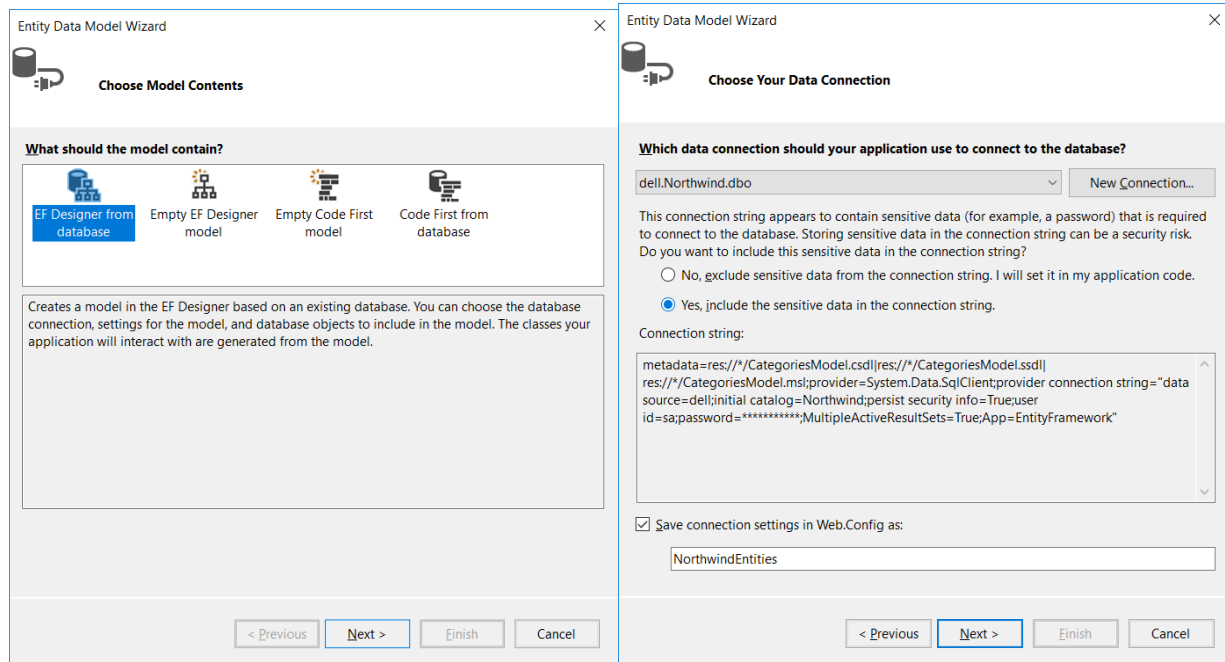
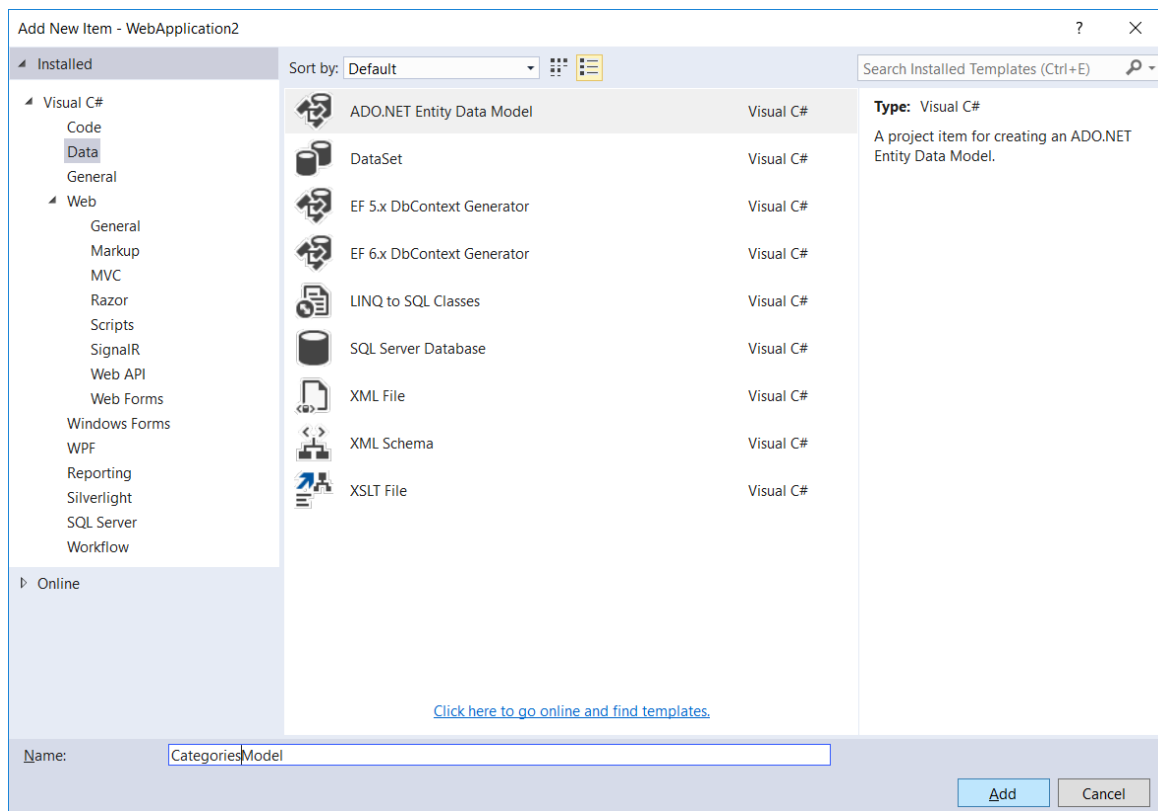
<ul>
    @for (int i = 0; i < ViewBag.NumTimes; i++)
    {
        <li>@ViewBag.Message</li>
    }
</ul>
```

I testujemy wywołanie:



6. Definiowanie Modelu z wykorzystaniem istniejącej struktury bazy danych (Database First)

(do modelu EF dokładamy Suppliers, Categories i Products)



Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

- ☒ dbo
 - ☒ Categories
 - ☐ CustomerCustomerDemo
 - ☐ CustomerDemographics
 - ☐ Customers
 - ☐ Employees
 - ☐ Employees1
 - ☐ EmployeeTerritories
 - ☐ MyTest
 - ☐ Numbers
 - ☐ Order Details
 - ☐ Orders
 - ☒ Products
 - ☐ Region

☒ Pluralize or singularize generated object names

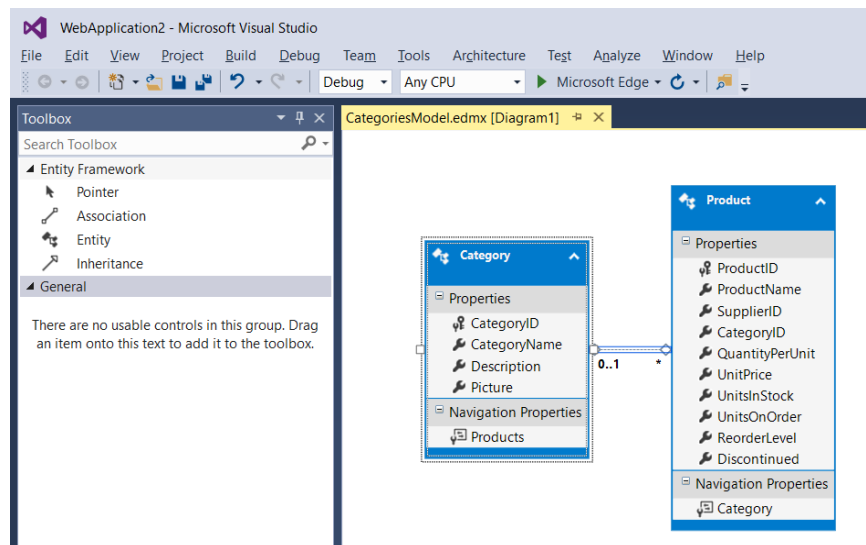
☒ Include foreign key columns in the model

☒ Import selected stored procedures and functions into the entity model

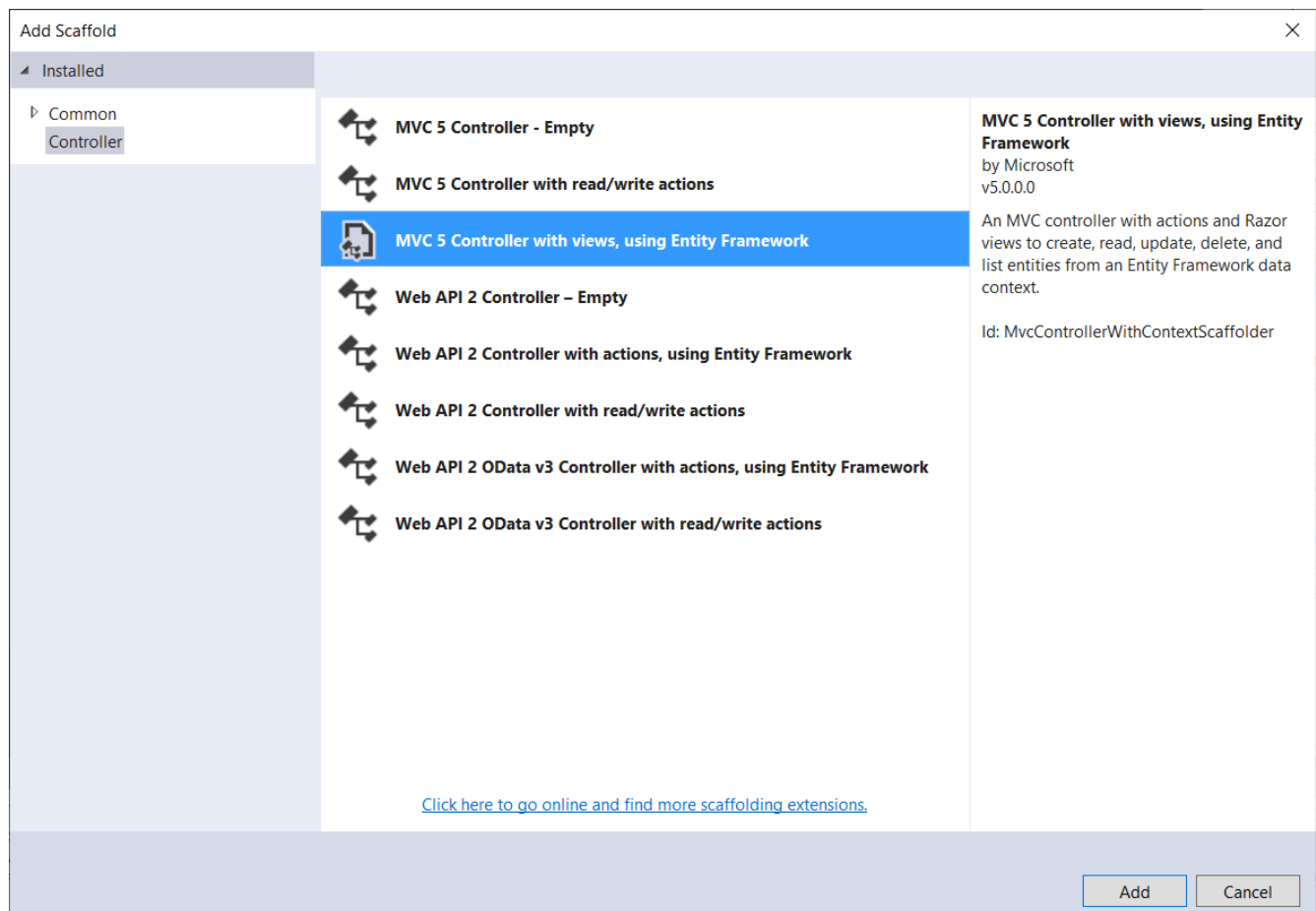
Model Namespace:

NorthwindModel

< Previous Next > Finish Cancel



7. Dodawanie Modelu (w MVC)



Sprawdzić ten sam krok ale wybierając opcję wcześniejszą np. dla Modelu Category.

Add Controller

Model class:
Product (WebAppMVC)

Data context class:
NorthwindEntities (WebAppMVC)
+

☐ Use async controller actions

Views:

☒ Generate views
☒ Reference script libraries
☒ Use a layout page:
...

(Leave empty if it is set in a Razor `_viewstart` file)

Controller name:
ProductsController

Add
Cancel

Wynik:

WebAppMVC
Home
About
Contact

Index
Create New

ProductName	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued	CategoryName	CompanyName	
Chai	10 boxes x 20 bags	18,00	39	0	10	<input type="checkbox"/>	Beverages	Exotic Liquids	Edit Details Delete
Chang	24 - 12 oz bottles	19,00	17	40	25	<input type="checkbox"/>	Beverages	Exotic Liquids	Edit Details Delete
Aniseed Syrup	12 - 550 ml bottles	10,00	13	70	25	<input type="checkbox"/>	Condiments	Exotic Liquids	Edit Details Delete

Dodać w menu odpowiednio elementy: Categories, Suppliers, Products i dla każdego z nich powinno pojawić się odpowiednia obsługa danych. Zmodyfikować zawartość każdego z elementów menu pod daną aplikację. Dodać wyszukiwanie danych np. Produktów. Dodać pokazywanie elementów z bazy danych po 10 elementów. Zamiast elementu Edit, Delete podmienić dowolne ikonki. Co zrobić z polem Unitprice przy wstawianiu danych jeśli podamy 10.23 lub 10,23 (przy wartości 10 jest OK).

Uwaga: You may not be able to enter decimal points or commas in the Price field. To support jQuery validation for non-English locales that use a comma (",") for a decimal point, and non US-English date formats, you must include *globalize.js* and your specific *cultures/globalize.cultures.js* file(from <https://github.com/jquery/globalize>) and JavaScript to use `Globalize.parseFloat`. I'll show how to do this in the next tutorial. For now, just enter whole numbers like 10.

8. Tworzenie parametrów połączenia i praca z bazą danych SQL Server LocalDB

Dodaj poniższe parametry połączenia do <connectionStrings> pliku *Web.config*.

```
<add name="EmployeeDBContext"
      connectionString="Data Source=(LocalDb)\MSSQLLocalDB;Initial
Catalog=MVC_Emp;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|\Employees.mdf"
      providerName="System.Data.SqlClient"
/>
```

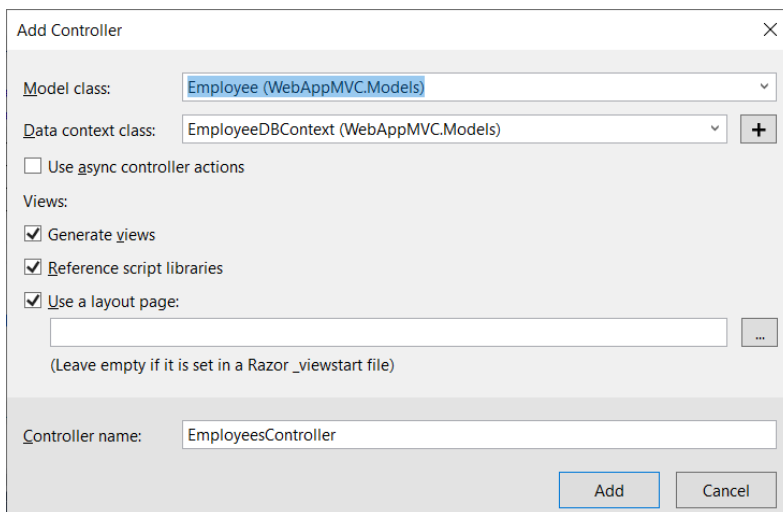
Do katalogu Models wstawiamy klasę o nazwie np. Employee

```
using System;
using System.Data.Entity;

namespace MvcMovie.Models
{
    public class Employee
    {
        public int ID { get; set; }
        public string Name { get; set; }
        public DateTime HireDate { get; set; }
        public decimal Salary { get; set; }
    }

    public class EmployeeDBContext : DbContext
    {
        public DbSet<Employee> Employees { get; set; }
    }
}
```


Dodajemy kontroler o nazwie EmployeesController



Add Controller

Model class: Employee (WebAppMVC.Models)

Data context class: EmployeeDbContext (WebAppMVC.Models)

☐ Use async controller actions

Views:

☒ Generate views

☒ Reference script libraries

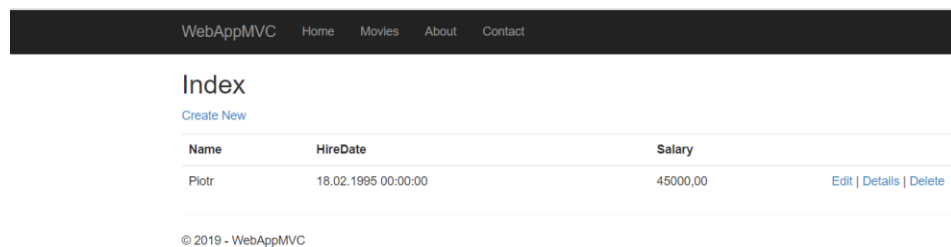
☒ Use a layout page:

(Leave empty if it is set in a Razor _viewstart file)

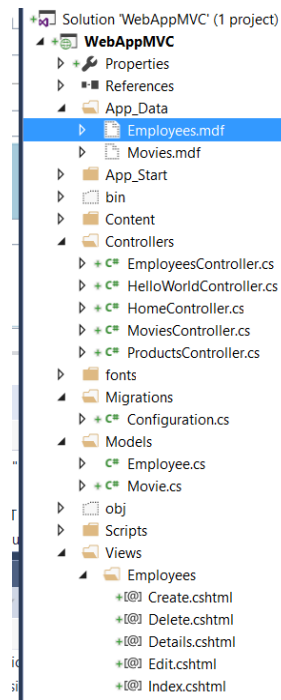
Controller name: EmployeesController

Add Cancel

Po uruchomieniu aplikacji i dodaniu nowego rekordu baza danych o nazwie Employees będzie znajdowała się w katalogu App_Data (włączamy tylko oglądanie wszystkich plików, aby zobaczyć pliki. Następnie przejrzyj wygenerowany kod.



Name	HireDate	Salary	
Piotr	18.02.1995 00:00:00	45000,00	Edit Details Delete



9. Dodawanie metody wyszukiwania i widoku wyszukiwania

Dla kontrolera ProductsController dla akcji Index dokładamy zmienną searchString, którą uwzględniamy w danych przekazywanych do widoku.

```
public ActionResult Index(string searchString)
{
    var products = db.Products.Include(p => p.Category).Include(p => p.Supplier);
    if (!String.IsNullOrEmpty(searchString))
    {
        products = products.Where(s => s.ProductName.Contains(searchString));
    }
    return View(products);
    //var products = from m in db.Products
    //                select m;
}
```

Otwórz plik Views\Products\Index.cshtml, po kodzie @Html.ActionLink("Create New", "Create"), dodaj formularz, które przedstawiono poniżej:

```
<p>
    @Html.ActionLink("Create New", "Create")

    @using (Html.BeginForm()) {
        <p> Title: @Html.TextBox("SearchString") <br />
        <input type="submit" value="Filter" /></p>
    }
</p>
```

W efekcie otrzymamy:

The screenshot shows the 'Index' page of a web application. At the top is a navigation bar with links: WebAppMVC, Home, Movies, About, and Contact. Below the navigation bar is the 'Index' title and a 'Create New' link. A search form is present with a text input labeled 'Title: cha' and a 'Filter' button. Below the form is a table with 9 columns: ProductName, QuantityPerUnit, UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued, CategoryName, and CompanyName. The table contains three rows of product data. Each row has links for 'Edit', 'Details', and 'Delete' at the end. At the bottom of the page, there is a copyright notice: '© 2019 - WebAppMVC'.

ProductName	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued	CategoryName	CompanyName
Chai	10 boxes x 20 bags	18,00	39	0	10	<input type="checkbox"/>	Beverages	Exotic Liquids
Chang	24 - 12 oz bottles	19,00	17	40	25	<input type="checkbox"/>	Beverages	Exotic Liquids
Chartreuse verte	750 cc per bottle	18,00	69	0	5	<input type="checkbox"/>	Beverages	Aux joyeux ecclésiastiques

Gdyby była potrzeba wyświetlenia w pasku adresu parametrów metodą GET to wstawiamy: `Html.BeginForm("Index", "Products", FormMethod.Get)` zamiast `Html.BeginForm()`

10. Dodawanie wyszukiwania według nazwy kategorii

Zadanie:

Wybieramy nazwę dostępnych kategorii z pola listy i na tej podstawie wybieramy wszystkie produkty z danej kategorii. Dodatkowo zostawiamy wyszukiwanie wg. nazwy produktu.

WebAppMVC Home Movies About Contact

Index

[Create New](#)

CategoryName: Confections Title: Sir

Filter

ProductName	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued	CategoryName	CompanyName
Sir Rodney's Marmalade	30 gift boxes	81,00	40	0	0	<input type="checkbox"/>	Confections	Specialty Biscuits, Ltd.
Sir Rodney's Scones	24 pkgs. x 4 pieces	10,00	3	40	5	<input type="checkbox"/>	Confections	Specialty Biscuits, Ltd.

© 2019 - WebAppMVC

```
public ActionResult Index(string searchString, string searchCategory)
{
    var products = db.Products.Include(p => p.Category).Include(p => p.Supplier);
    var categoryName = from c in db.Categories
                       orderby c.CategoryName
                       select c.CategoryName ;
    ViewBag.searchCategory = new SelectList(categoryName.Distinct());

    if (!string.IsNullOrEmpty(searchString))
    {
        products = products.Where(s => s.ProductName.Contains(searchString));
    }
    if (!string.IsNullOrEmpty(searchCategory))
    {
        products = products.Where(x => x.Category.CategoryName == searchCategory);
    }
    return View(products);
}
```

Otwórz plik `Views\Products\Index.cshtml`, po kodzie `@Html.ActionLink("Create New", "Create")`, dodaj formularz, które przedstawiono poniżej:

```
@using (Html.BeginForm("Index", "Products", FormMethod.Get))
{
    <p>
        CategoryName: @Html.DropDownList("SearchCategory", "All")
        Title: @Html.TextBox("SearchString") <br />
        <input type="submit" value="Filter" />
    </p>
}
```