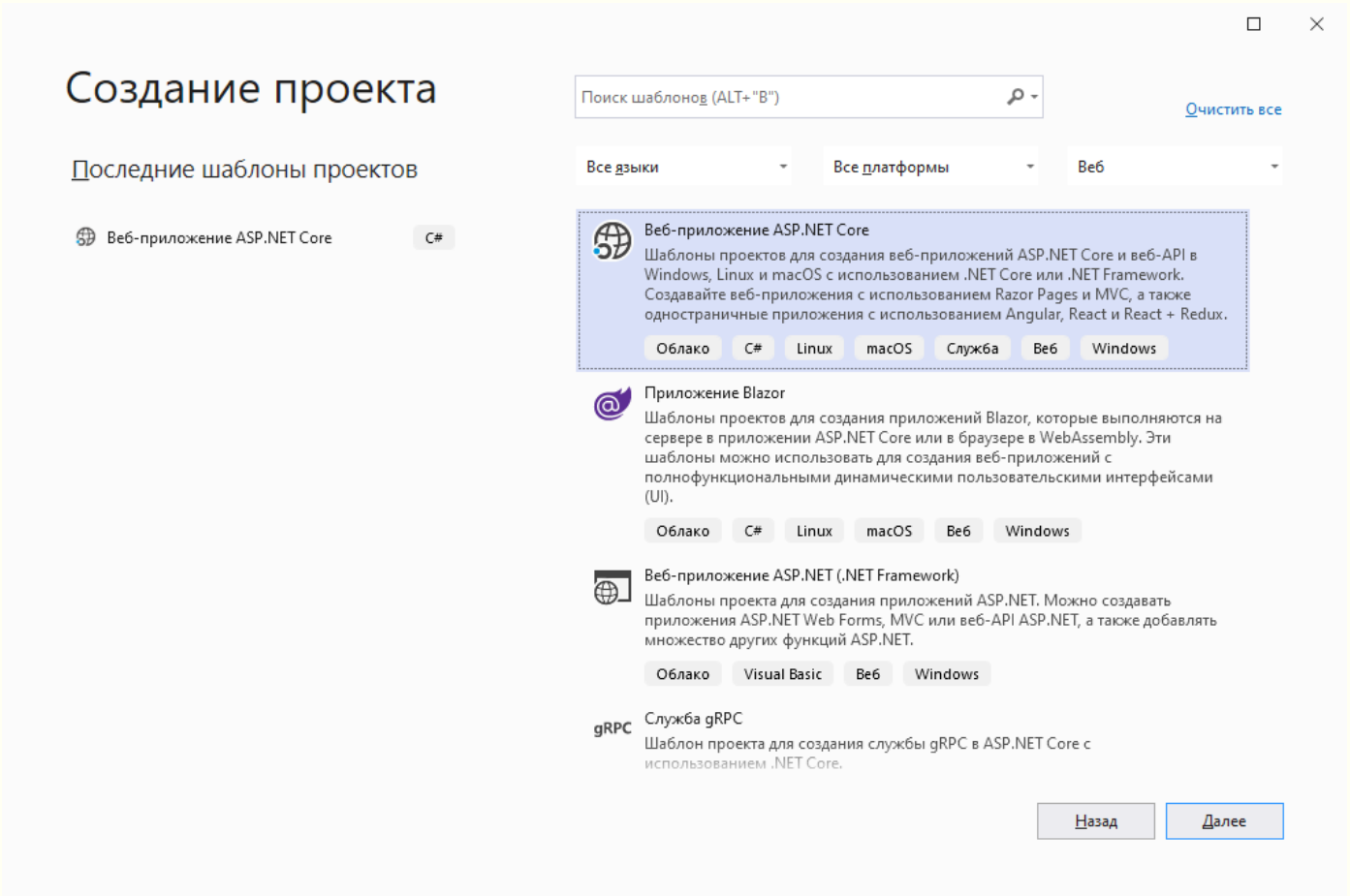
Building Web Applications
with ASP.NET Core MVC

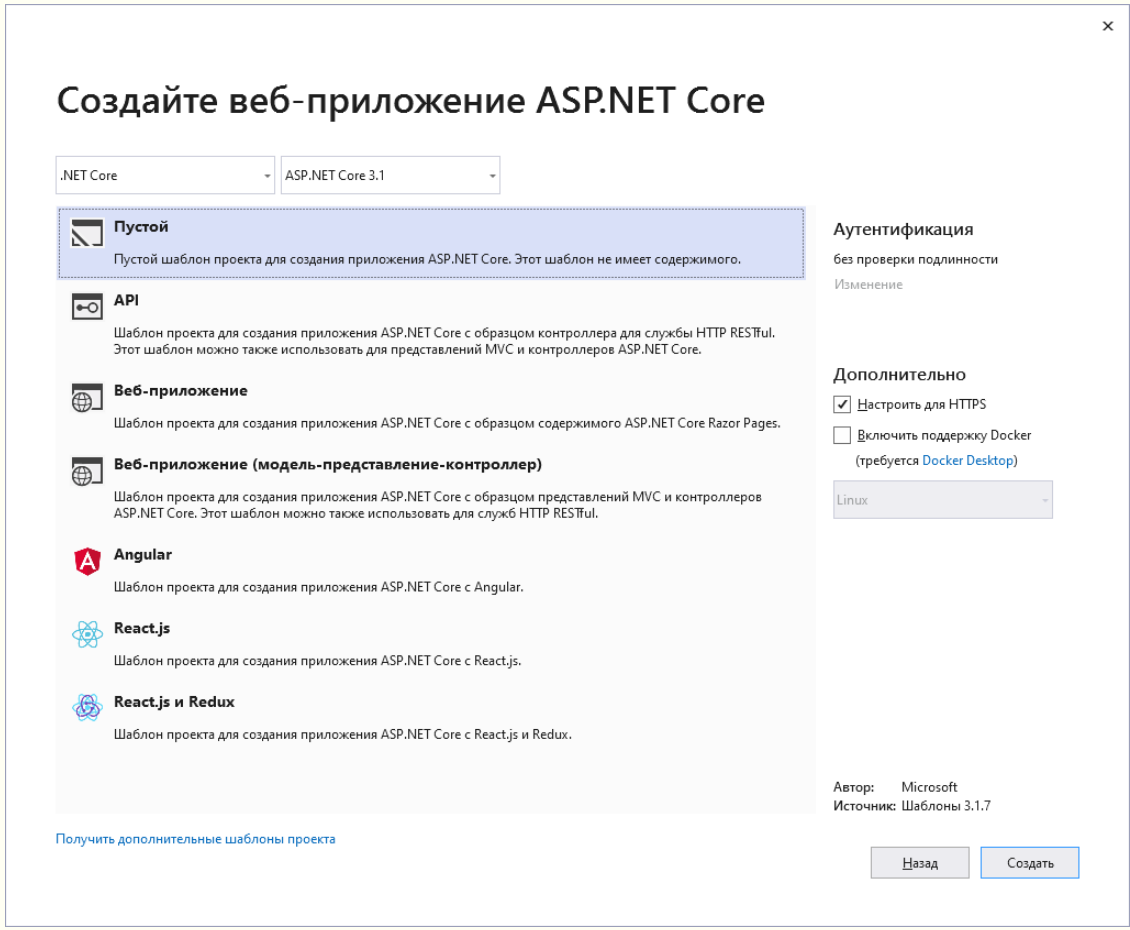
Gill Cleeren
CTO XPIRIT BELGIUM
@gillcleeren www.snowball.be

ОСНОВИ РОЗРОБКИ ВЕБ-САЙТІВ НА БАЗІ ТЕХНОЛОГІЇ ASP.NET CORE MVC

Питання 5.2 (відео 1-39)

Створення проекту в середовищі Visual Studio



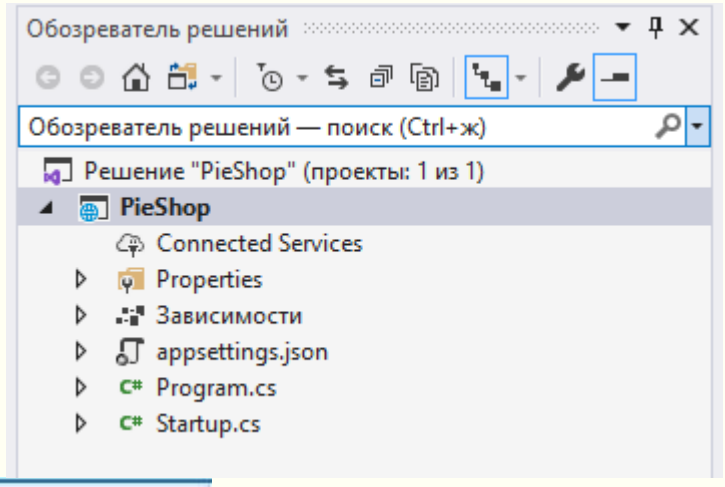


- Створимо заготовку для онлайн-магазину пирогів

У структурі проекту немає NuGet-пакетів

PieShop.csproj

```
1  <Project Sdk="Microsoft.NET.Sdk.Web">
2
3    <PropertyGroup>
4      <TargetFramework>netcoreapp3.1</TargetFramework>
5    </PropertyGroup>
6
7  </Project>
8
```



- Старіша версія для ASP.NET Core 2.1

```
1  <Project Sdk="Microsoft.NET.Sdk.Web">
2
3    <PropertyGroup>
4      <TargetFramework>netcoreapp2.1</TargetFramework>
5    </PropertyGroup>
6
7    <ItemGroup>
8      <Folder Include="wwwroot\" />
9    </ItemGroup>
10
11   <ItemGroup>
12     <PackageReference Include="Microsoft.AspNetCore.App" />
13     <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.1.2" PrivateAssets="
14   </ItemGroup>
15
16 </Project>
```

Конфігурація сайту (файл Program.cs)

Program.cs | PieShop.csproj

PieShop | PieShop.Program

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Hosting;
6  using Microsoft.Extensions.Configuration;
7  using Microsoft.Extensions.Hosting;
8  using Microsoft.Extensions.Logging;
9
10 namespace PieShop
11 {
12     public class Program
13     {
14         public static void Main(string[] args)
15         {
16             CreateHostBuilder(args).Build().Run();
17         }
18
19         public static IHostBuilder CreateHostBuilder(string[] args) =>
20             Host.CreateDefaultBuilder(args)
21                 .ConfigureWebHostDefaults(webBuilder =>
22                 {
23                     webBuilder.UseStartup<Startup>();
24                 });
25     }
26 }
```

- У методі Main налаштовується хост (конфігурується сервер та конвеєр обробки запитів – request processing pipeline).
  - Метод CreateHostBuilder() встановить значення за умовчанням для додатку – викличе статичний метод CreateDefaultBuilder() класу Host (v2.1 – WebHost, підтримується й далі).
- За умовчанням веб-додаток розташовується на власному веб-сервері Kestrel.
  - Зазвичай доступ через IIS залишається, відповідне посилання заховане в дефолтних налаштуваннях.
- Клас Startup є конфігуратором, який описує здійснення налаштування сайту перед запуском.

Конфігурація сайту (файл Startup.cs)

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        ...
    }

    public void Configure(IApplicationBuilder app,
        IHostingEnvironment env)
    {
        ...
    }

}
```

- Передбачає наступні дії:
  - Визначення конвеєра обробки запитів (метод ConfigureServices – реєструє служби за допомогою ін'єкції залежностей).
  - Конфігурація всіх служб, потрібних у додатку (метод Configure).
- Методи автоматично викликаються ASP.NET Core.
  - Працює вбудована система ін'єкції залежностей на базі інтерфейсу IServiceProvider.

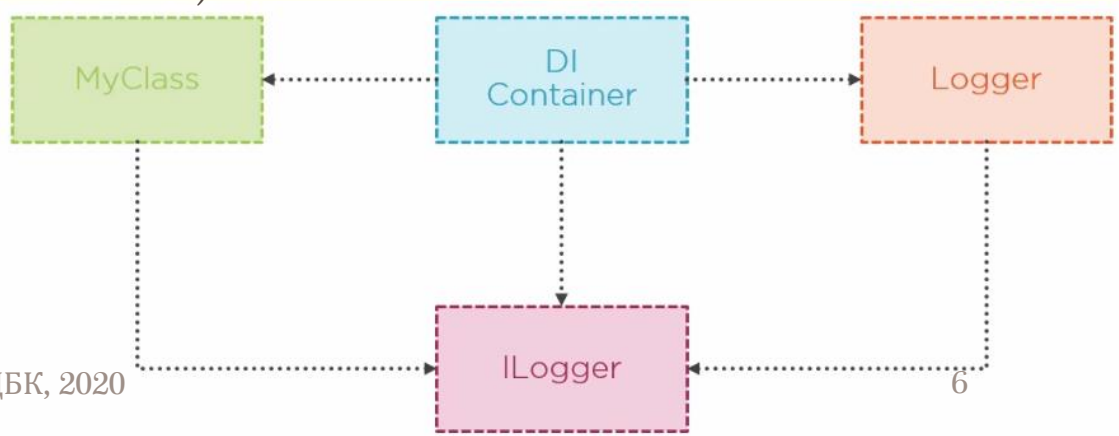
Коротко про ін'єкції залежностей (Dependency Injection, DI)

Сильне зв'язування (Tight Coupling)

- Потрібний інший спосіб передачі екземпляру класу Logger у клас MyClass.
- Можливий вихід – створення спеціального конструктора, проте екземпляри потрібно явно, вручну конструювати за кожної потреби.

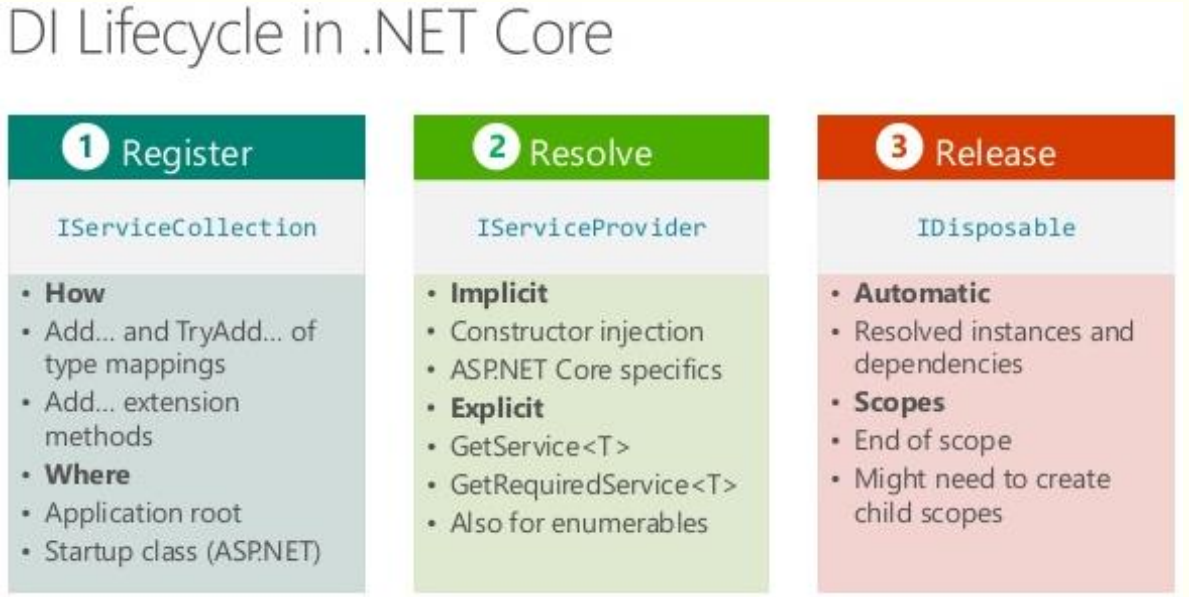
Послаблення зв'язків за допомогою ін'єкції залежностей

- Впроваджуємо інтерфейс ILogger та контейнер ін'єкції залежностей (DI-контейнер). У контейнері:
  - Реєструються всі залежності: ILogger і Logger (коли реалізується ILogger, надати екземпляр класу Logger)
  - Вирішуються потреби одних компонентів в екземплярах інших (усі частини додатку знають про DI-контейнер, він видає одним блокам екземпляри інших).



Метод ConfigureService()

```
public void ConfigureServices(IServiceCollection services)
{
    //register framework services
    services.AddControllersWithViews();

    //register our own services (more later)

}
```

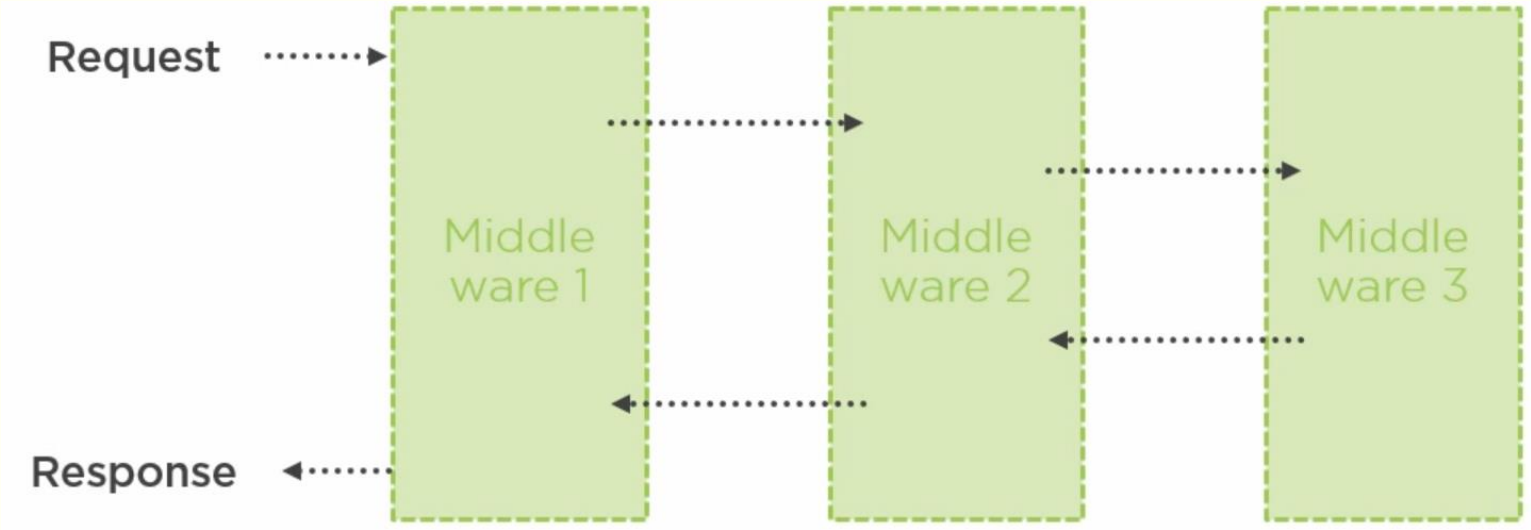


- Вбудований DI-контейнер доступний інтерфейсу IServiceProvider, який управляє колекцією сервісів IServiceCollection.
  - Під ***сервісами*** тут розуміємо об'єкти з певною функціональністю, які надаються іншим частинам додатку.
  - У методі ConfigureServices() можемо додавати нові сервіси
- ASP.NET Core містить багато вбудованих сервісів.
  - У прикладі в коді – підтримка шаблону MVC (у попередніх версіях – метод addMvc()).

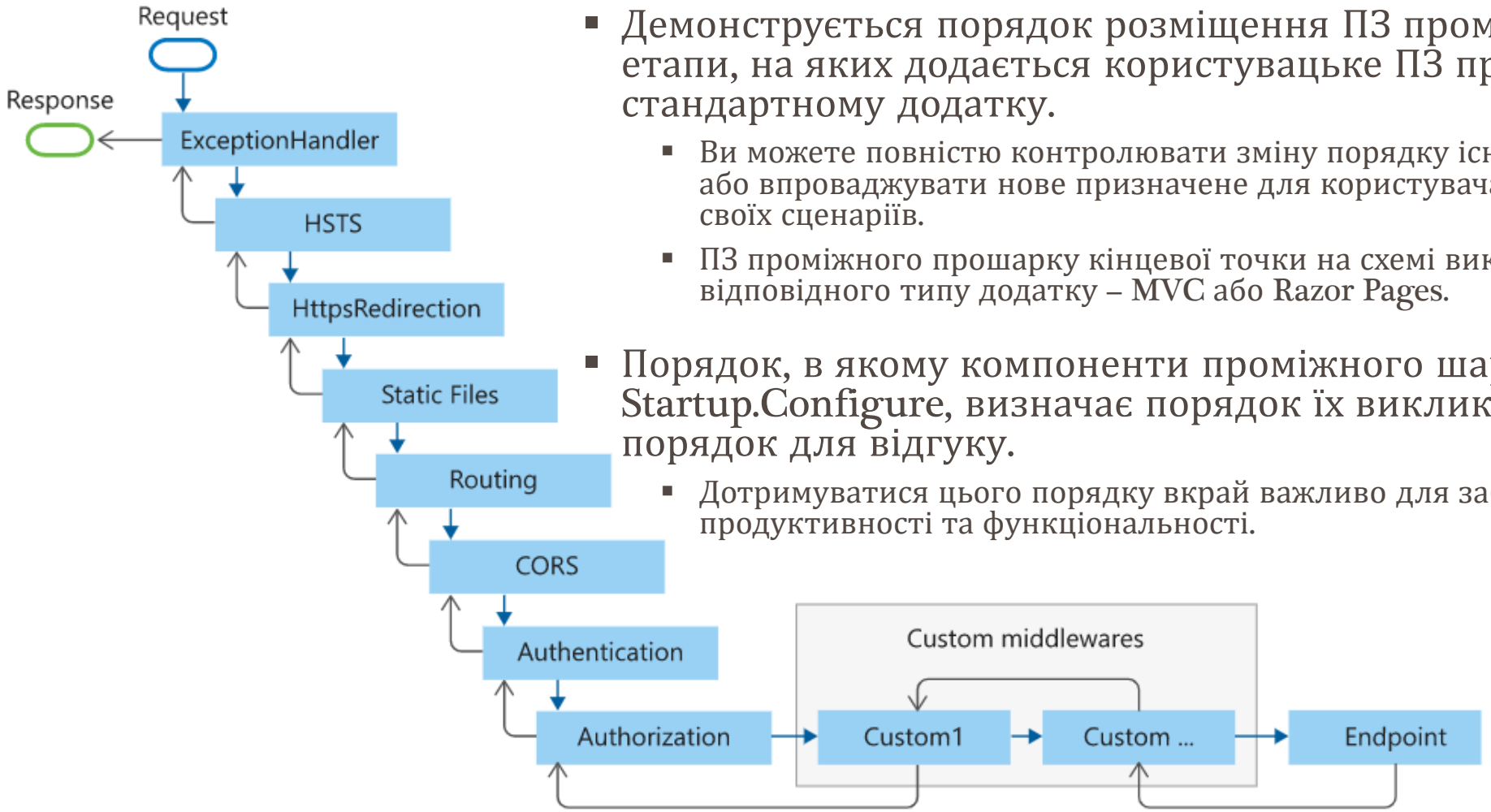
Метод Configure()

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env)
{

    //add middleware components here

}
```

- Налаштовується конвеєр запитів, який складається з послідовності зв'язаних компонентів (middleware-компонентів, компоненти проміжного прошарку).
  - Перехоплюють та обробляють HTTP-запити (requests), які надходять;
  - Продукують HTTP-відгук (response).
- Для построения конвейера запросов используются делегаты запроса.
  - Они обрабатывают каждый HTTP-запрос.
  - Для их настройки служат методы расширения Run, Map и Use.



Повний конвеєр обробки запитів ASP.NET Core MVC та Razor Pages

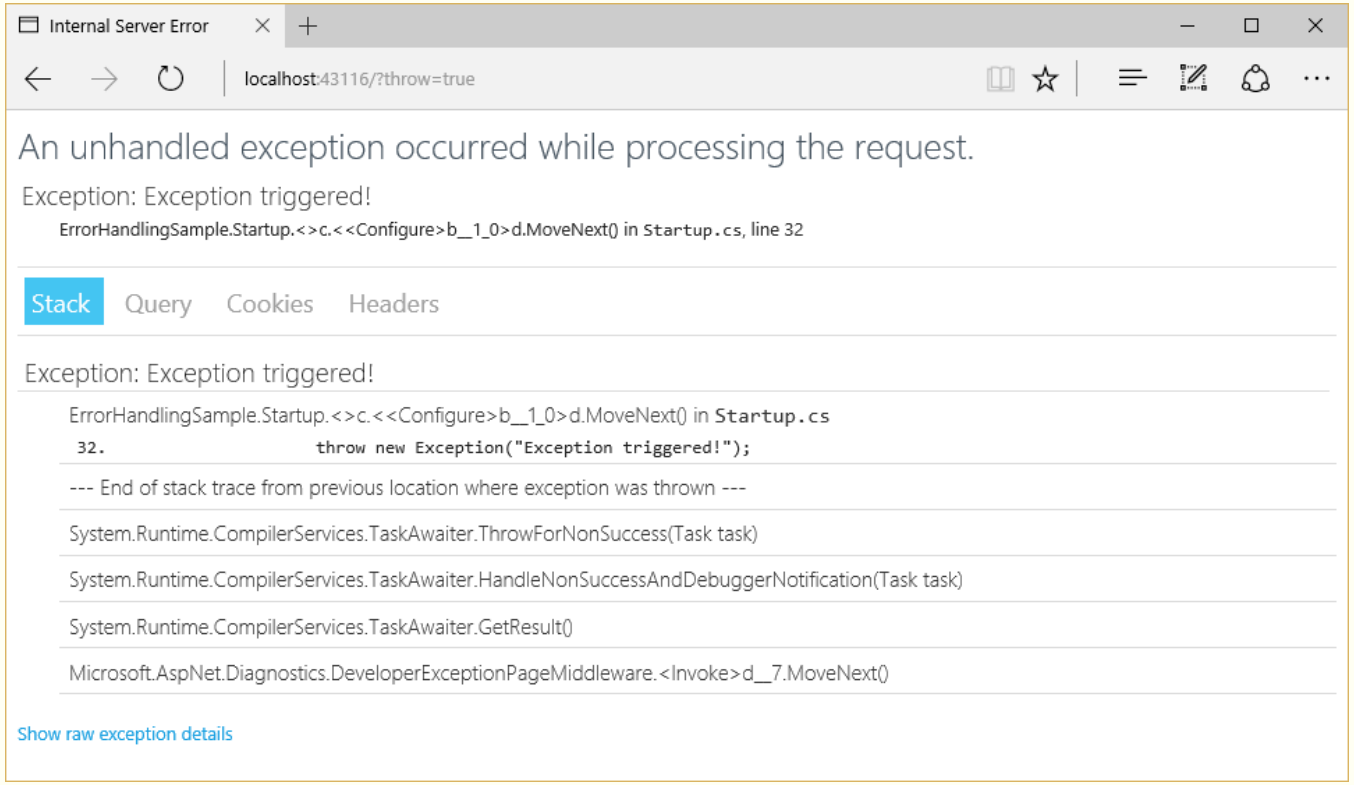


- Демонструється порядок розміщення ПЗ проміжного прошарку та етапи, на яких додається користувацьке ПЗ проміжного прошарку, в стандартному додатку.
  - Ви можете повністю контролювати зміну порядку існуючого ПЗ проміжного прошарку або впроваджувати нове призначене для користувача ПЗ проміжного прошарку для своїх сценаріїв.
  - ПЗ проміжного прошарку кінцевої точки на схемі виконує конвеєр фільтра для відповідного типу додатку – MVC або Razor Pages.
- Порядок, в якому компоненти проміжного шару додаються в метод Startup.Configure, визначає порядок їх виклику при запитах і зворотний порядок для відгуку.
  - Дотримуватися цього порядку вкрай важливо для забезпечення безпеки, продуктивності та функціональності.

Приклад методу Configure()

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env)
{
    app.UseDeveloperExceptionPage();
    app.UseStatusCodePages();
    app.UseStaticFiles();
    app.UseEndpoints(endpoints =>
    { ... });
}
```

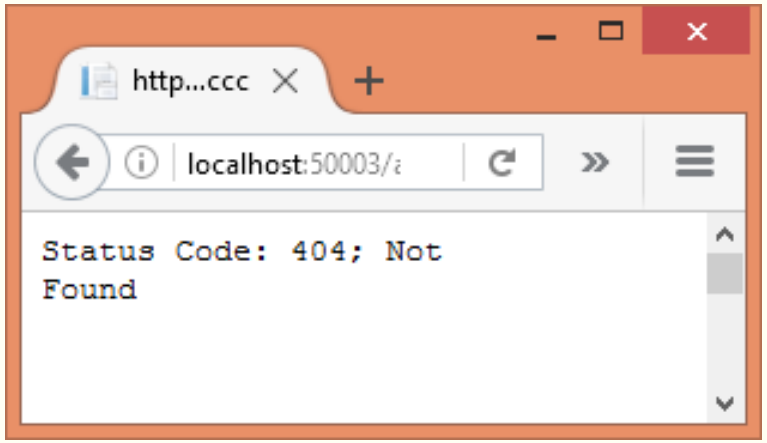
- UseDeveloperExceptionPage() – дозволяє використання сторінки винятків для розробника (Developer Exception Page):



Приклад методу Configure()

```
public void Configure(IApplicationBuilder app,
    IHostingEnvironment env)
{
    app.UseDeveloperExceptionPage();
    app.UseStatusCodePages();
    app.UseStaticFiles();
    app.UseEndpoints(endpoints =>
    { ... });
}
```

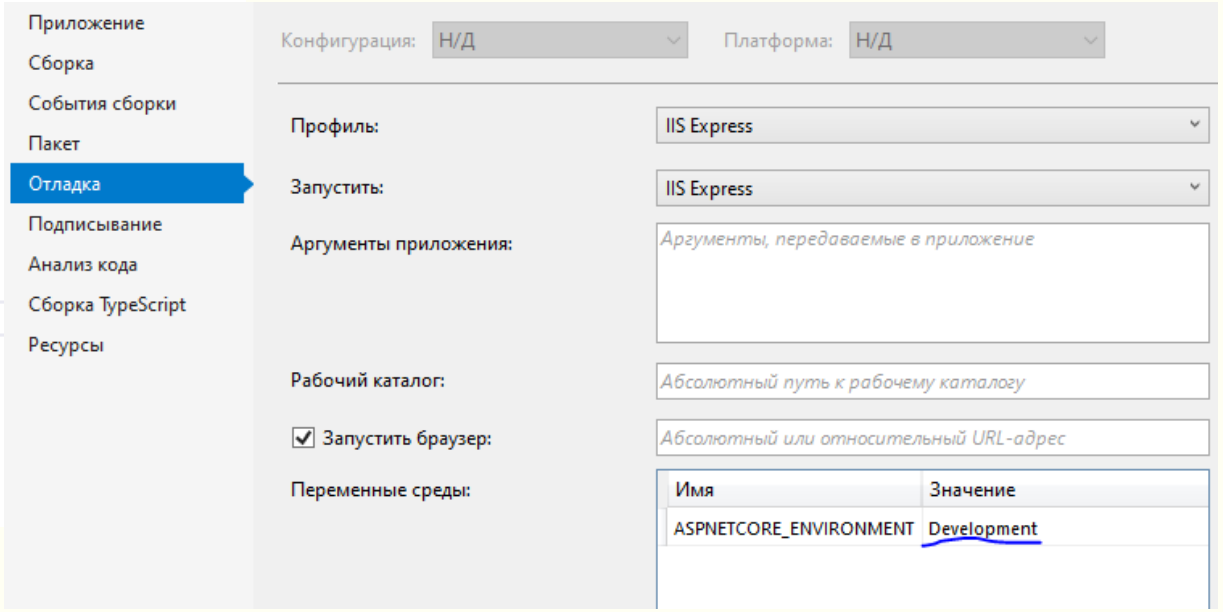
- UseStatusCodePages() – додає підтримку текстових кодів стану HTTP:



- UseStaticFiles() – підтримка роботи зі статичними файлами в запитах.
- UseEndpoints() – додає компонент проміжного прошарку, який направлятиме запити до коректної кінцевої точки

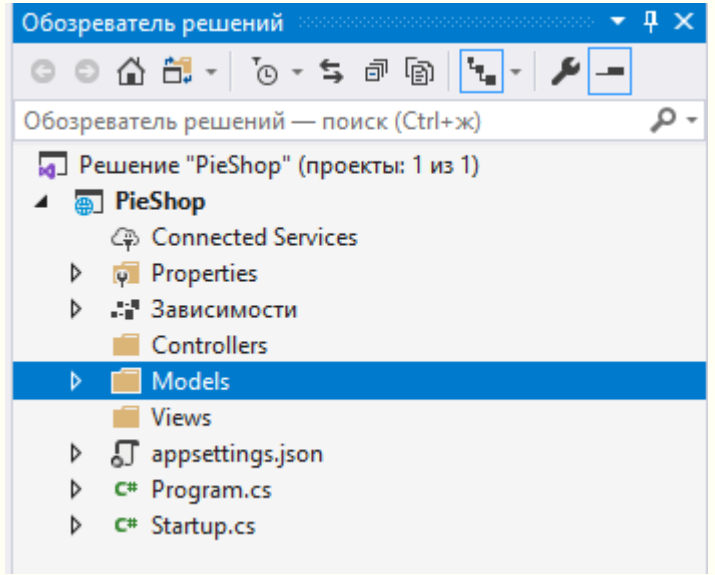
Приклад конфігурації в Startup.cs

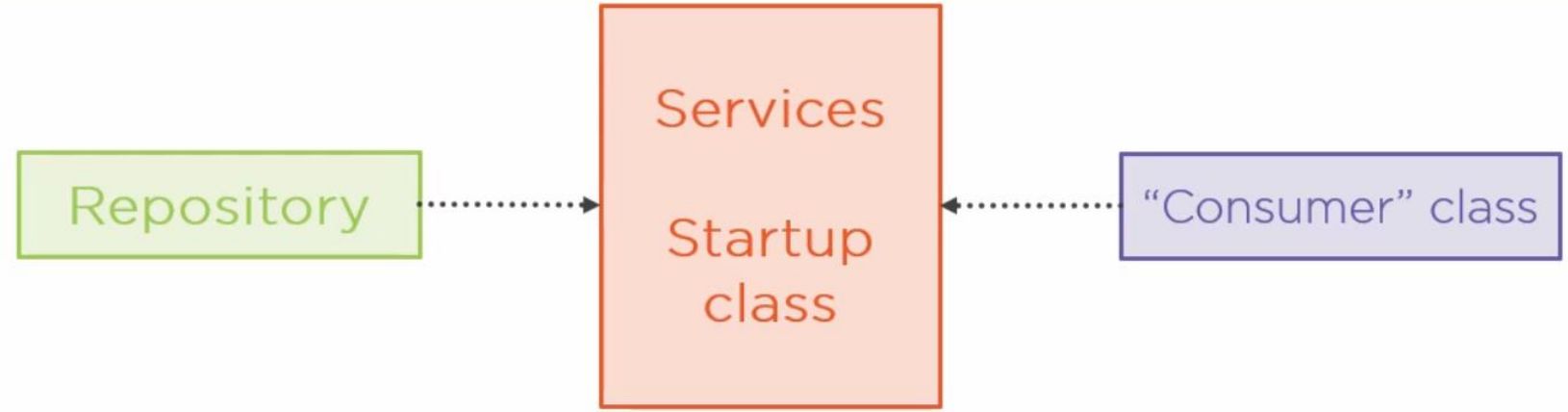
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews(); // до версії 3.0 - services.AddMvc();
}

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

    app.UseHttpsRedirection();  // конвертація HTTP-запитів у HTTPS
    app.UseStaticFiles();       // додавання підтримки статичних файлів
                                // (зображення, js, css тощо)
                                // стандартний каталог для цих файлів - wwwroot
    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        // заміняємо стандартний "Hello, world!"
        // відображаємо запит, який надійшов, на дію контролера
        // перенаправлення стане потрібним надалі в проекті
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
```



Створення веб-сторінки

- Процес:
  - Створити модель та репозиторій
  - Створити контролер
  - Додати представлення
  - Стилізувати представлення
- Репозиторій підключатиметься як сервіс через ін’єкції залежностей





Створення рівня моделі

```
namespace PieShop.Models
{
    public class Pie
    {
        public int PieId { get; set; }
        public string Name { get; set; }
        public string ShortDescription { get; set; }
        public string LongDescription { get; set; }
        public string AllergyInformation { get; set; }
        public decimal Price { get; set; }
        public string ImageUrl { get; set; }
        public string ImageThumbnailUrl { get; set; }
        public bool IsPieOfTheWeek { get; set; }
        public bool InStock { get; set; }
        public int CategoryId { get; set; }
        public Category Category { get; set; }
    }
}
```

```
namespace PieShop.Models
{
    // інтерфейс репозиторія
    public interface IPieRepository
    {
        IEnumerable<Pie> AllPies { get; }
        IEnumerable<Pie> PiesOfTheWeek { get; }
        Pie GetPieById(int pieId);
    }
}
```

- Рівень моделі містить опис предметної області (domain data, група класів) та логіку управління даними
  - Предметна область – пироги в продажу

```
namespace PieShop.Models
{
    public class Category
    {
        public int      CategoryId { get; set; }
        public string   CategoryName { get; set; }
        public string   Description { get; set; }
        public List<Pie> Pies { get; set; }
    }
}
```

```
namespace PieShop.Models
{
    public interface ICategoryRepository
    {
        IEnumerable<Category> AllCategories { get;  }
    }
}
```

Опис моку репозиторія пирогів

```
namespace PieShop.Models
{
    public class MockPieRepository : IPieRepository
    {
        private readonly ICategoryRepository _categoryRepository = new MockCategoryRepository();

        public IEnumerable<Pie> AllPies =>
            new List<Pie>
            {
                new Pie {PieId = 1, Name="Полуничний пиріг", Price=155.50M, ShortDescription="Lorem Ipsum",
                    LongDescription="Icing carrot cake jelly-o cheesecake. Sweet roll marzipan marshmallow toffee brownie brownie candy tootsie roll. Chocolate cake gingerbread t
                    Category = _categoryRepository.AllCategories.ToList()[0], ImageUrl="https://gillcleerenpluralsight.blob.core.windows.net/files/strawberrypie.jpg",
                    InStock=true, IsPieOfTheWeek=false, ImageThumbnailUrl="https://gillcleerenpluralsight.blob.core.windows.net/files/strawberrypiesmall.jpg"},
                new Pie {PieId = 2, Name="Чізкейк", Price=218.95M, ShortDescription="Lorem Ipsum",
                    LongDescription="Icing carrot cake jelly-o cheesecake. Sweet roll marzipan marshmallow toffee brownie brownie candy tootsie roll. Chocolate cake gingerbread t
                    Category = _categoryRepository.AllCategories.ToList()[1], ImageUrl="https://gillcleerenpluralsight.blob.core.windows.net/files/cheesecake.jpg",
                    InStock=true, IsPieOfTheWeek=false, ImageThumbnailUrl="https://gillcleerenpluralsight.blob.core.windows.net/files/cheesecakesmall.jpg"},
                new Pie {PieId = 3, Name="Ревеневий пиріг", Price=215.00M, ShortDescription="Lorem Ipsum",
                    LongDescription="Icing carrot cake jelly-o cheesecake. Sweet roll marzipan marshmallow toffee brownie brownie candy tootsie roll. Chocolate cake gingerbread t
                    Category = _categoryRepository.AllCategories.ToList()[0],ImageUrl="https://gillcleerenpluralsight.blob.core.windows.net/files/rhubarbpie.jpg",
                    InStock=true, IsPieOfTheWeek=true, ImageThumbnailUrl="https://gillcleerenpluralsight.blob.core.windows.net/files/rhubarbpiesmall.jpg"},
                new Pie {PieId = 4, Name="Гарбузовий пиріг", Price=132.50M, ShortDescription="Lorem Ipsum",
                    LongDescription="Icing carrot cake jelly-o cheesecake. Sweet roll marzipan marshmallow toffee brownie brownie candy tootsie roll. Chocolate cake gingerbread t
                    Category = _categoryRepository.AllCategories.ToList()[2], ImageUrl="https://gillcleerenpluralsight.blob.core.windows.net/files/pumpkinpie.jpg",
                    InStock=true, IsPieOfTheWeek=true, ImageThumbnailUrl="https://gillcleerenpluralsight.blob.core.windows.net/files/pumpkinpiesmall.jpg"}
            };

        public IEnumerable<Pie> PiesOfTheWeek { get; }

        public Pie GetPieById(int pieId)
        {
            return AllPies.FirstOrDefault(p => p.PieId == pieId);
        }
    }
}
```

Опис моку репозиторія категорій та підключення репозиторіїв

```
namespace PieShop.Models
{
    public class MockCategoryRepository : ICategoryRepository
    {
        public IEnumerable<Category> AllCategories =>
            new List<Category>
            {
                new Category {CategoryId=1, CategoryName="Фруктові пироги", Description="Вітамінні бомби"},
                new Category {CategoryId=2, CategoryName="Чізкейки", Description="Творогова насолода"},
                new Category {CategoryId=3, CategoryName="Сезонні пироги", Description="Для Вашого настрою"}
            };
    }
}
```

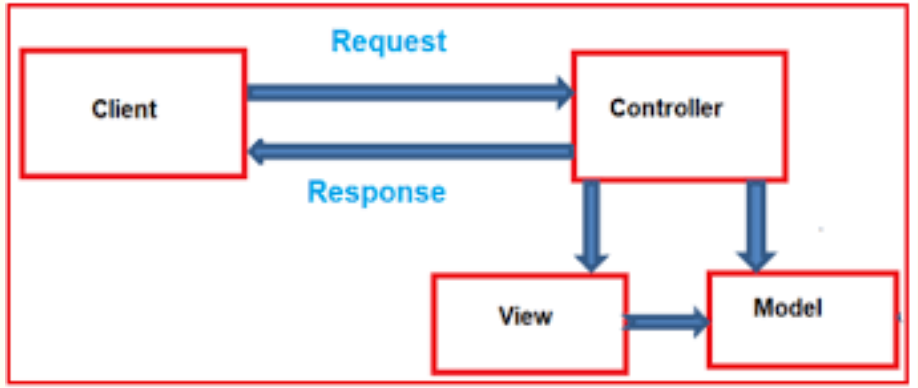
- У файлі Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
    // реєстрація репозиторія в DI-контейнері
    // тепер можемо звертатись до IPieRepository будь-де в проекті
    services.AddScoped<IPieRepository, MockPieRepository>();

    services.AddScoped<ICategoryRepository, MockCategoryRepository>();

    services.AddControllersWithViews(); // до версії 3.0 - services.AddMvc();
}
```

- Опції реєстрації сервісу в контейнері:
  - ***AddTransient*** – при кожному зверненні до сервісу отримується новий, «чистий» екземпляр
  - ***AddSingleton*** – при кожному зверненні до сервісу отримується один і той же екземпляр
  - ***AddScoped*** – при кожному HTTP-запиті створюється новий екземпляр, проте у зверненнях в межах одного запиту – один і той же екземпляр.

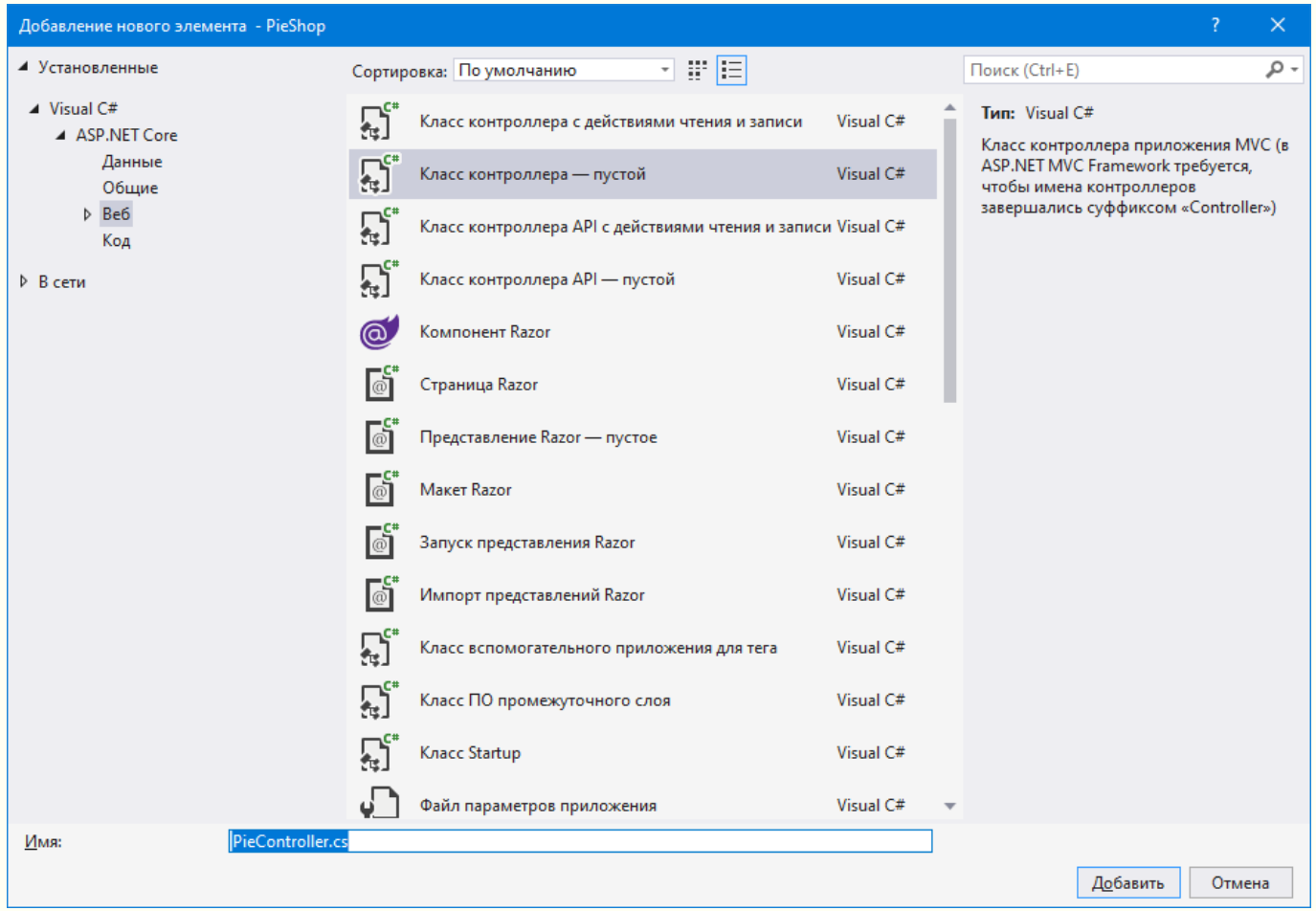
Рівень контролера

- На цьому рівні відбувається:
  - Взаємодія з користувачем
  - Оновлення даних з рівня моделі у відповідь на дії користувача без розуміння того, в якій формі ці дані зберігаються
- Контролер – клас, породжений від базового класу Controller.
  - Назва контролеру за угодою включає суфікс Controller
  - Реакція контролера на запити реалізується у вигляді action-методу з класу, який викликатиметься при їх надходженні




```
public class PieController : Controller
{
    public ViewResult Index()  <····· Action
    {
        return View();         <····· View to show
    }
}
```

Реальний контролер




```
namespace PieShop.Controllers
{
    public class PieController : Controller
    {
        // інтерфейси для управління репозиторіями
        private readonly IPieRepository _pieRepository;
        private readonly ICategoryRepository _categoryRepository;

        // ініціалізуємо інтерфейси в конструкторі
        // вони вже були зареєстровані в Startup.cs (DI-контейнері)
        // і будуть впроваджуватись (inject) автоматично
        public PieController(IPieRepository pieRepository,
                             ICategoryRepository categoryRepository)
        {
            // доступ до рівня моделі
            _pieRepository = pieRepository;
            _categoryRepository = categoryRepository;
        }

        // обробка взаємодії з користувачем
        // відображаємо всі пироги
        public ViewResult List()
        {
            return View(_pieRepository.AllPies);
        }

    }
}
```

Рівень представлення

- HTML-шаблон з розміткою веб-сторінки для візуалізації даних з рівня моделі
  - У файлах *.cshtml з Razor-синтаксисом (дозволяє додавати C#-код до HTML-розмітки)
- Типи представлень в ASP.NET Core:
  - plain
  - strongly-typed (використовується в більшості випадків)
- Для кожного контролера потрібна відповідна директорія з представленням у папці Views

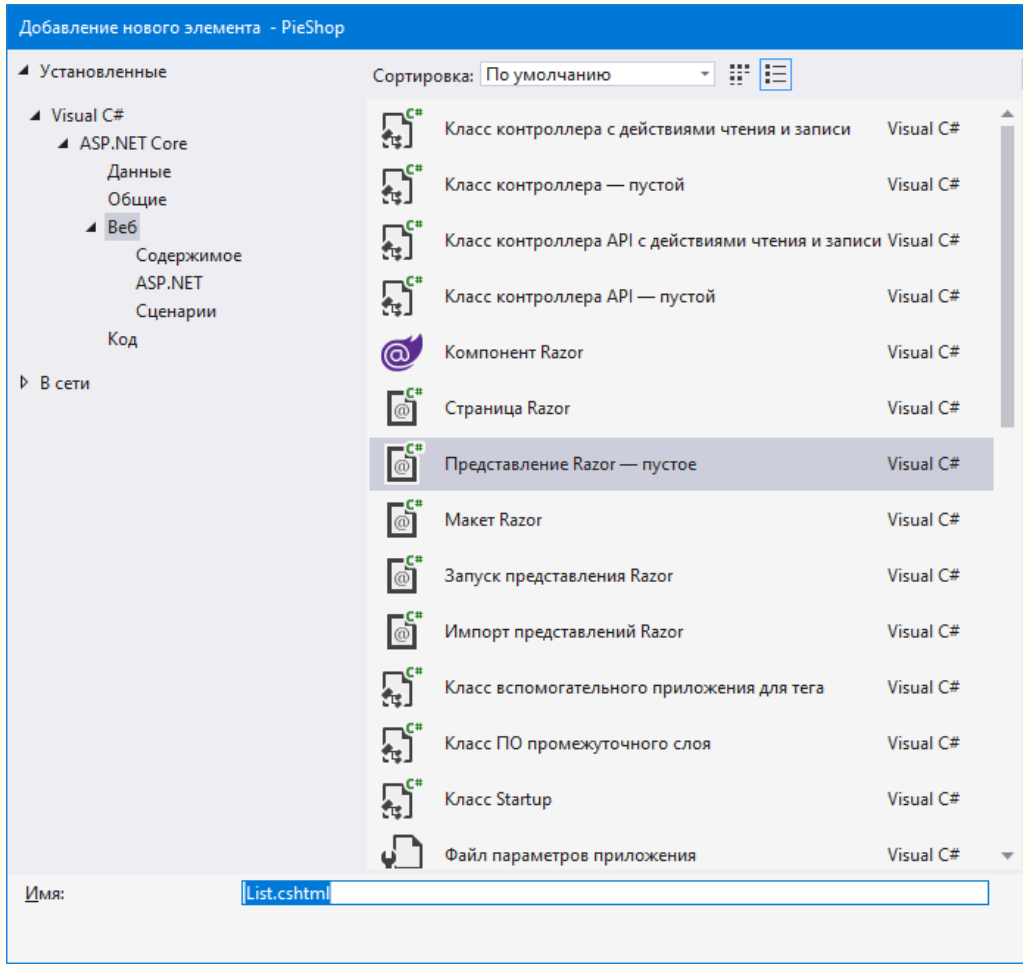
Строго типізоване представлення

```
@model IEnumerable<Pie>
<html>
  <body>
    <div>
      @foreach (var pie in Model.Pies)
      {
        <div>
          <h2>@pie.Name</h2>
          <h3>@pie.Price.ToString("c")</h3>
          <h4>@pie.Category.CategoryName</h4>
        </div>
      }
    </div>
  </body>
</html>
```

- Спочатку включаємо тип об’єкту, який буде передаватись у динамічну форму (Razor-синтаксис у першому рядку).
- Використовуємо Razor-синтаксис з C#-циклом foreach() для динамічного створення HTML-елементів.
  - За допомогою Razor-виразів отримуємо доступ до властивостей елементів списку (Name, Price, Category)
- Контролер повинен повертати список пирогів для відображення

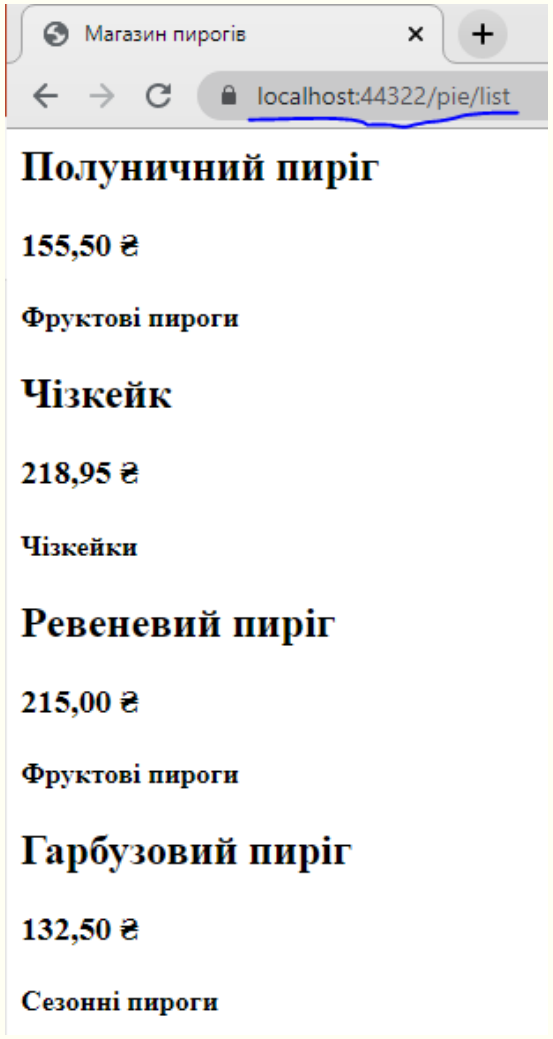
```
public ViewResult List()
{
    return View(_pieRepository.AllPies);
}
```

Реалізуємо представлення




```
1  @model IEnumerable<PieShop.Models.Pie>
2
3  <html>
4  <head>
5      <meta charset="utf-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport"
8            content="width=device-width, initial-scale=1">
9      <title>Магазин пирогів</title>
10 </head>
11
12 <body>
13     @foreach (var pie in Model)
14     {
15         <div>
16             <h2>@pie.Name</h2>
17             <h3>@pie.Price.ToString("c")</h3>
18             <h4>@pie.Category.CategoryName</h4>
19         </div>
20     }
21 </body>
22 </html>
```

Запуск представлення та передача додаткових даних

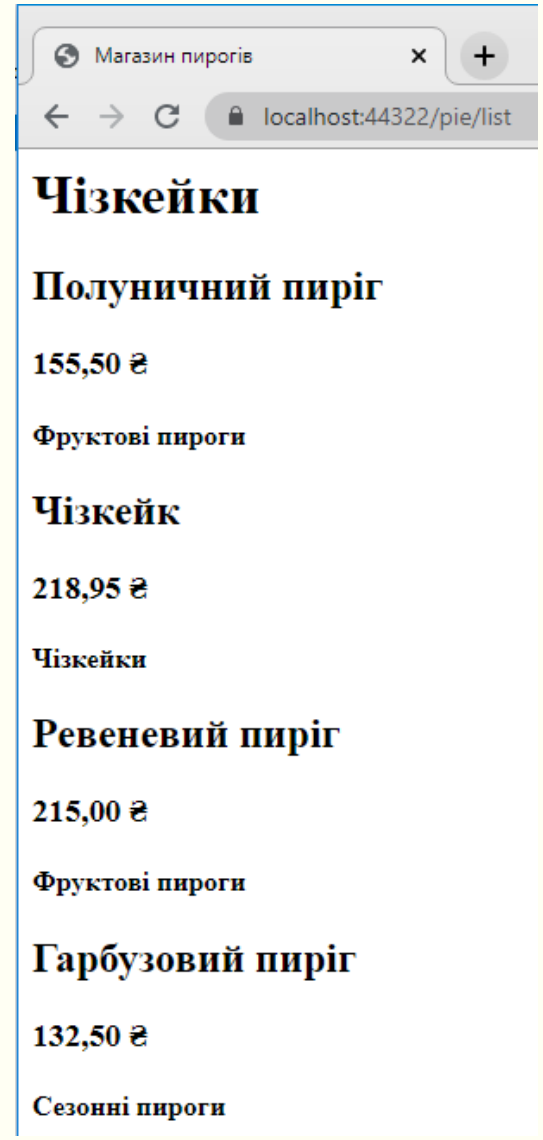


- Основні способи передачі додаткових даних:
  - Динамічна змінна ViewBag
  - Формування моделі для представлення (рівень ViewModel)
- Додамо заголовок за допомогою змінної ViewBag:
  - У контролері:

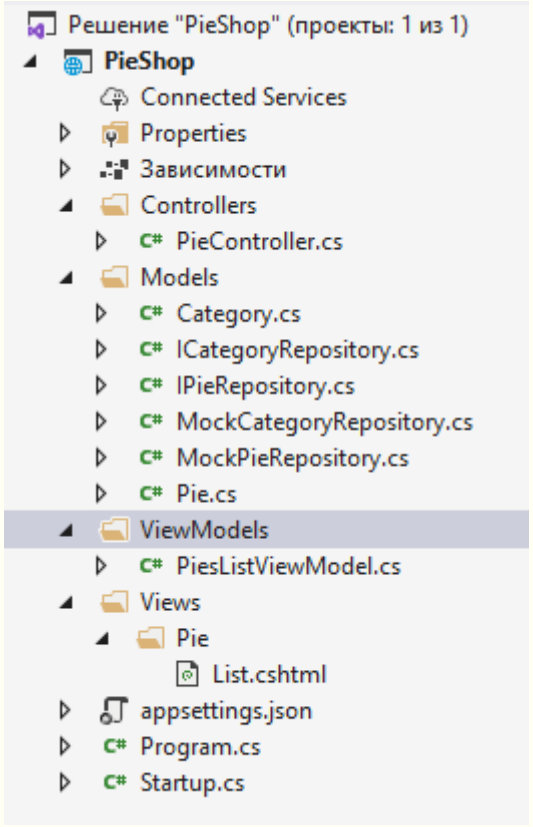
```
// обробка взаємодії з користувачем
// відображаємо всі пироги
public ViewResult List()
{
    ViewBag.CurrentCategory = "Чізкейки";
    return View(_pieRepository.AllPies);
}
```

  - У представленні:

```
<body>
    <h1>@ViewBag.CurrentCategory</h1>
    @foreach (var pie in Model)
    {
        <div>
```



Кращий підхід – ViewModel-клас: збиратиме потрібні для відображення дані з різних джерел



- Відобразимо не лише дані щодо пирога, але й його категорію:

```
namespace PieShop.ViewModels
{
    public class PiesListViewModel
    {
        public IEnumerable<Pie> Pies { get; set; }
        public string CurrentCategory { get; set; }
    }
}
```

- У контролері

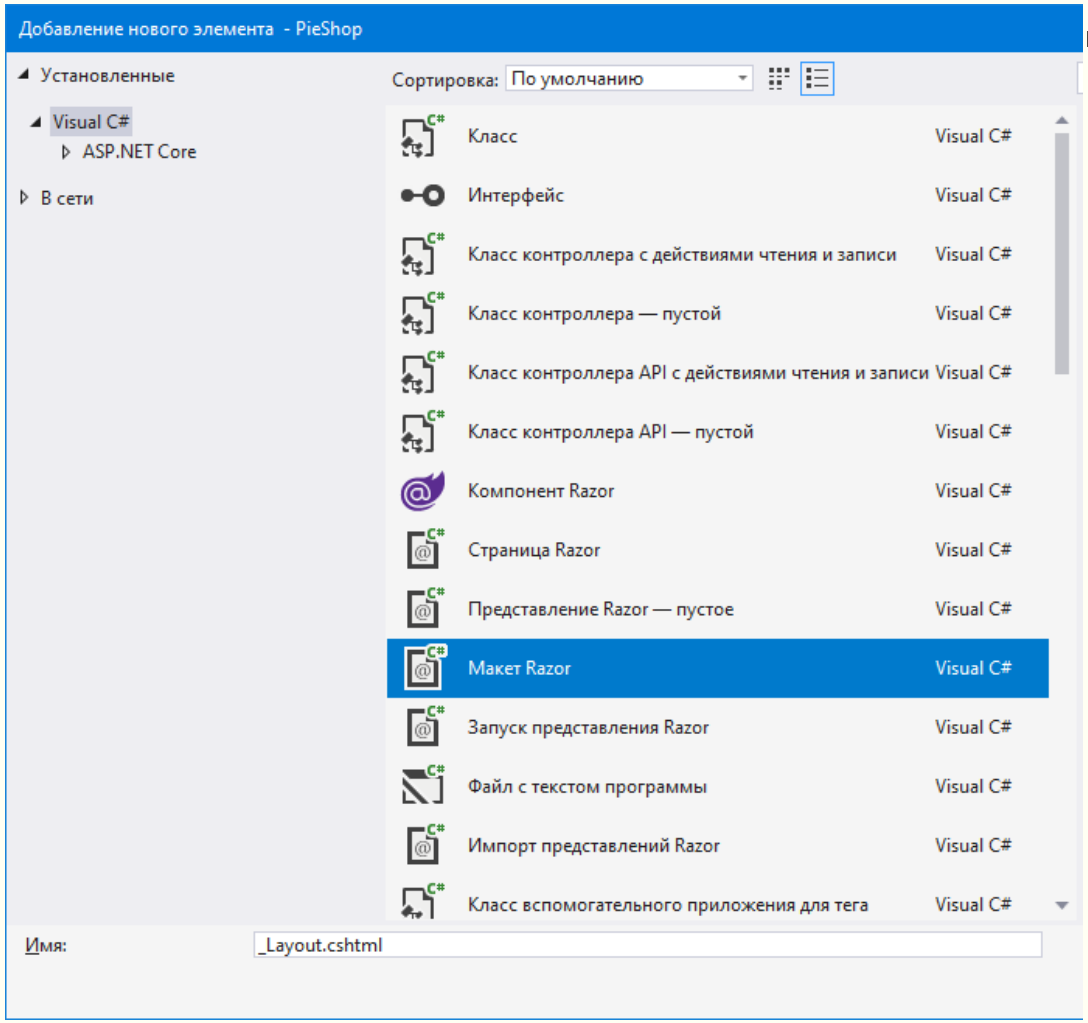
```
// обробка взаємодії з користувачем
// відображаємо всі пироги
public ViewResult List()
{
    //ViewBag.CurrentCategory = "Чізкейки";
    PiesListViewModel piesListViewModel = new PiesListViewModel();
    piesListViewModel.Pies = _pieRepository.AllPies;
    piesListViewModel.CurrentCategory = "Чізкейки";

    return View(piesListViewModel);
}
```

Оновлюємо представлення

```
1  @model PieShop.ViewModels.PiesListViewModel
2
3  <html>
4  <head>
5      <meta charset="utf-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport"
8            content="width=device-width, initial-scale=1">
9      <title>Магазин пирогів</title>
10 </head>
11
12 <body>
13     <h1>@Model.CurrentCategory</h1>
14     @foreach (var pie in Model.Pies)
15     {
16         <div>
17             <h2>@pie.Name</h2>
18             <h3>@pie.Price.ToString("c")</h3>
19             <h4>@pie.Category.CategoryName</h4>
20         </div>
21     }
22 </body>
23 </html>
```

- (1) Змінився шлях до рівня моделі
- (13) ViewBag замінився на Model
- (14) Model замінився на Model.Pies

Макетування веб-сторінок



- Створюється спеціальна папка Shared для спільного доступу різних представлень.
  - Спеціальний файл-макет _Layout.cshtml вже містить базову HTML-розмітку, в яку можна додавати «чисті» представлення.
  - Замість RenderBody підключається файл з представленням

_Layout.cshtml | PiesListViewModel.cs | List.cshtml* | PieController.cs* | MockCategoryRepository.cs | ICategory

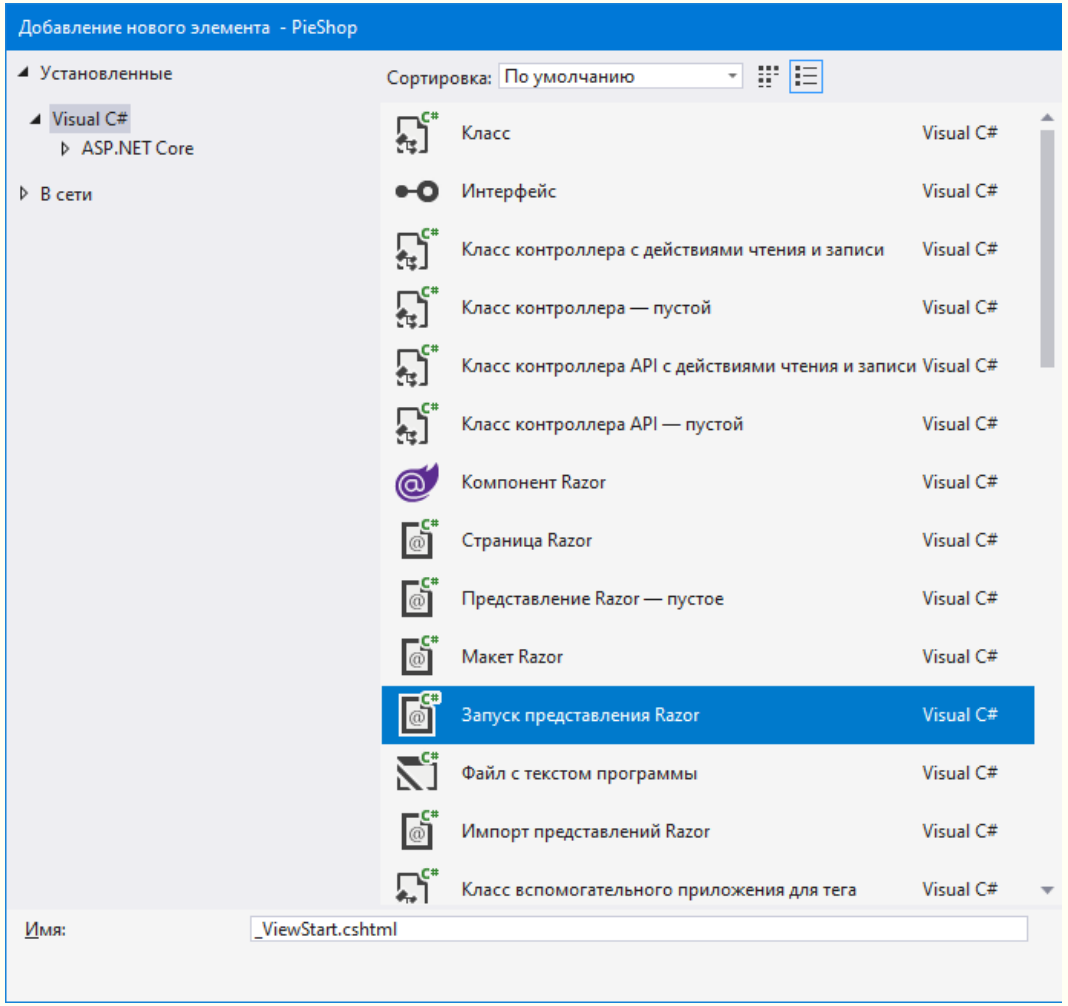
```
1  <!DOCTYPE html>
2
3  <html>
4  <head>
5      <meta name="viewport" content="width=device-width" />
6      <title>@ViewBag.Title</title>
7  </head>
8  <body>
9      <div>
10         @RenderBody()
11     </div>
12 </body>
13 </html>
```

Оновимо представлення для подальшого підключення до макету

_Layout.cshtml | PiesListViewModel.cs | List.cshtml* | PieController.cs* | MockCategory

```
1  @model PieShop.ViewModels.PiesListViewModel
2
3  <h1>@Model.CurrentCategory</h1>
4  @foreach (var pie in Model.Pies)
5  {
6      <div>
7          <h2>@pie.Name</h2>
8          <h3>@pie.Price.ToString("c")</h3>
9          <h4>@pie.Category.CategoryName</h4>
10     </div>
11 }
```

- Потрібно, щоб макет запускав дану розмітку, проте посилання на неї ще немає.
  - Передбачається спеціальний ViewStart-файл, який буде виконуватись при кожному запуску представлення.
  - Можна розташувати або в папці представлення, або на рівень вище.
  - Назва файлу повинна залишитись як є.



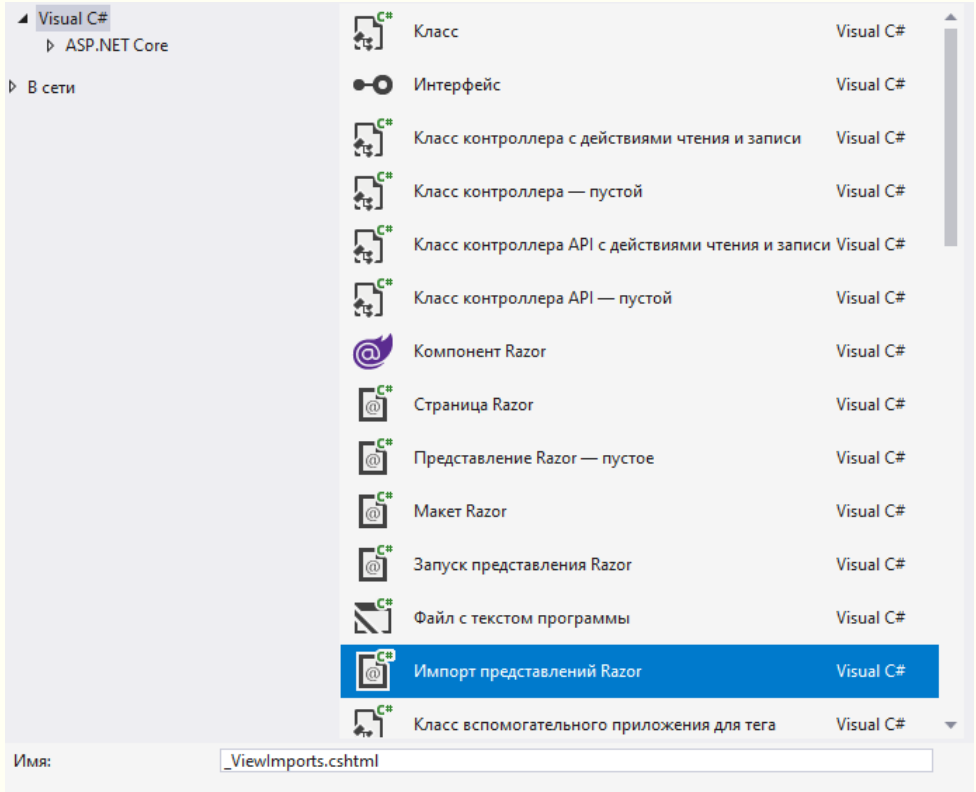
Файли _ViewStart.cshtml та _ViewImports.cshtml

- Файл _ViewStart.cshtml вказує на макет для запуску (без cshtml)

_ViewStart.cshtml | _Layout.cshtml | PiesListViewModel.cs

```
@{
    Layout = "_Layout";
}
```

- Файл _ViewImports.cshtml дозволяє зручніше організувати імпорт:



_ViewImports.cshtml* | _ViewStart.cshtml | _Layout.cshtml

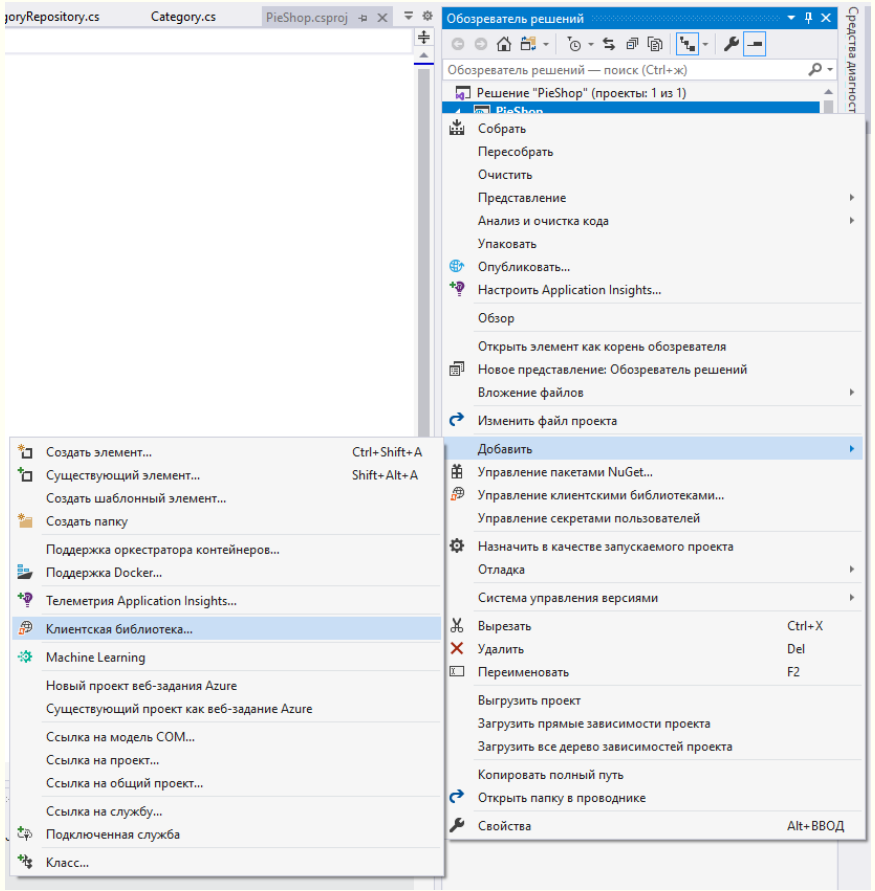
```
@using PieShop.ViewModels
```

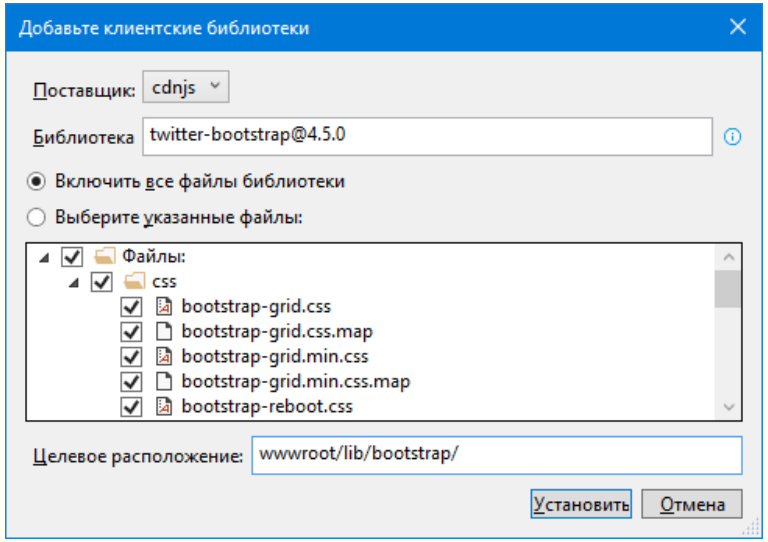
_ViewImports.cshtml* | List.cshtml* | _ViewStart.cshtml | _Layout.cshtml | PiesListViewM

```
@model PiesListViewModel

<h1>@Model.CurrentCategory</h1>
@foreach (var pie in Model.Pies)
{
    <div>
        <h2>@pie.Name</h2>
        <h3>@pie.Price.ToString("c")</h3>
        <h4>@pie.Category.CategoryName</h4>
    </div>
```

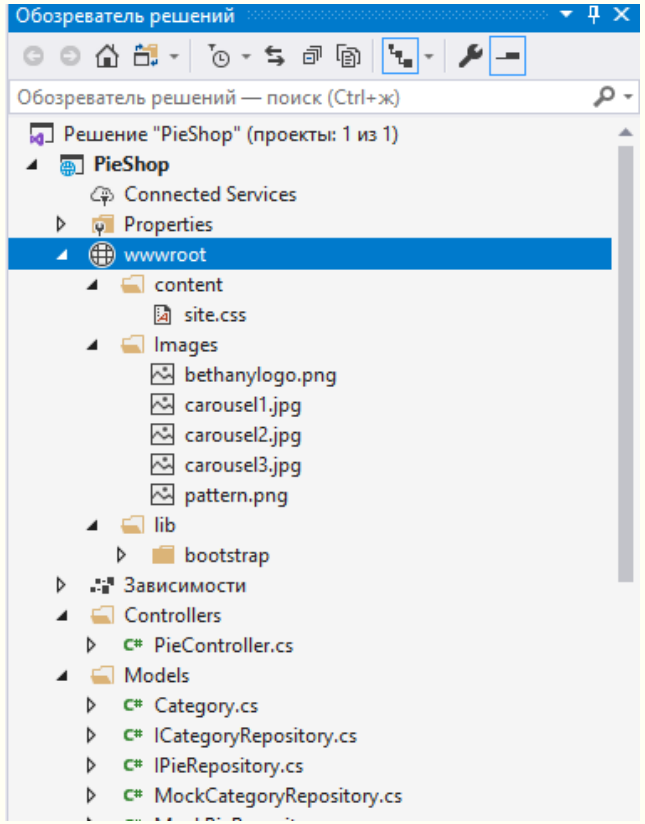
Стилізація представлення

- Підключимо в проект Bootstrap

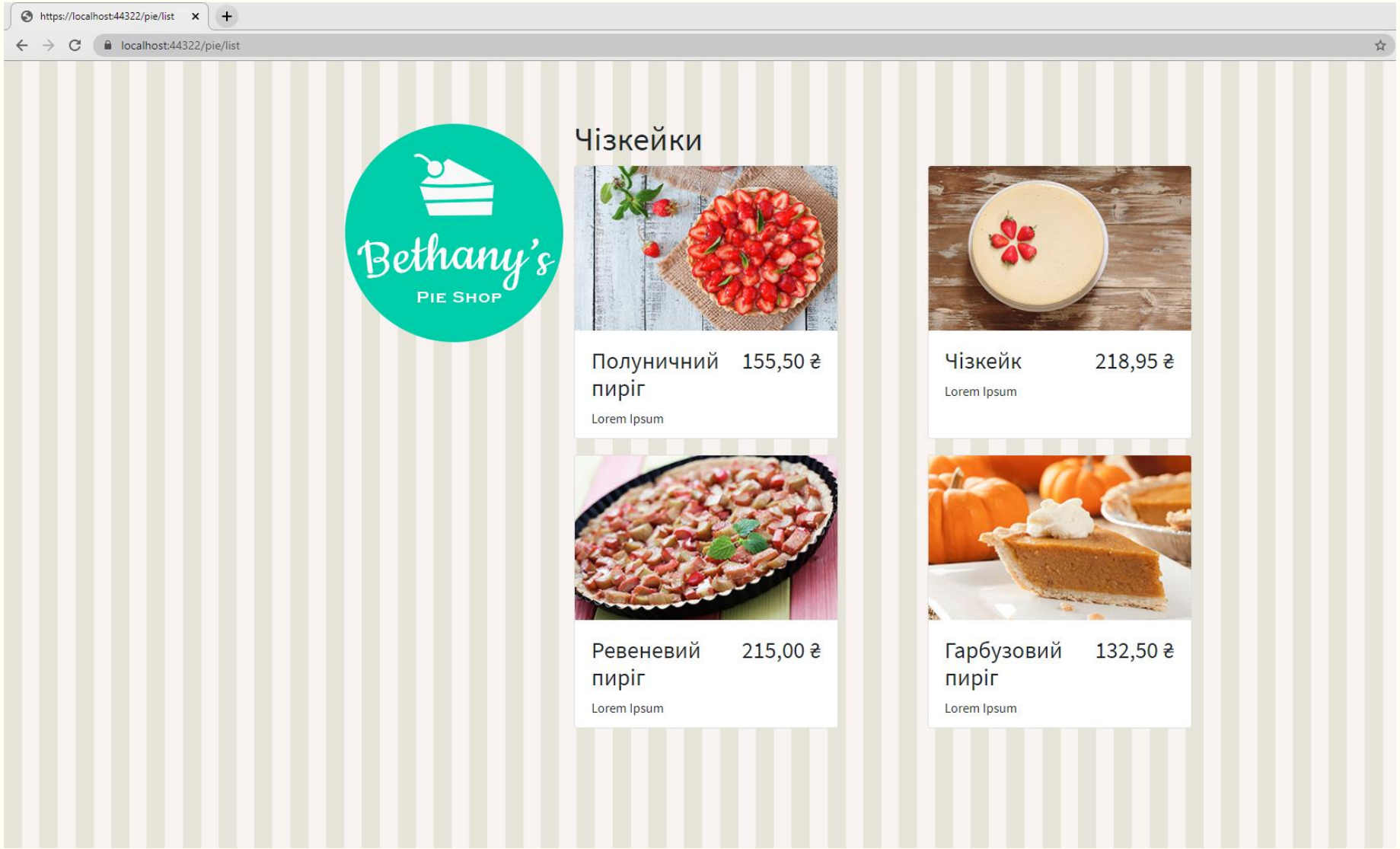




- Оскільки файли бібліотеки статичні, збережемо їх у папку wwwroot.
  - Додамо зображення, які будуть використовуватись на сайті, у папку wwwroot/images
  - Додамо свій css-файл у папку wwwroot/content



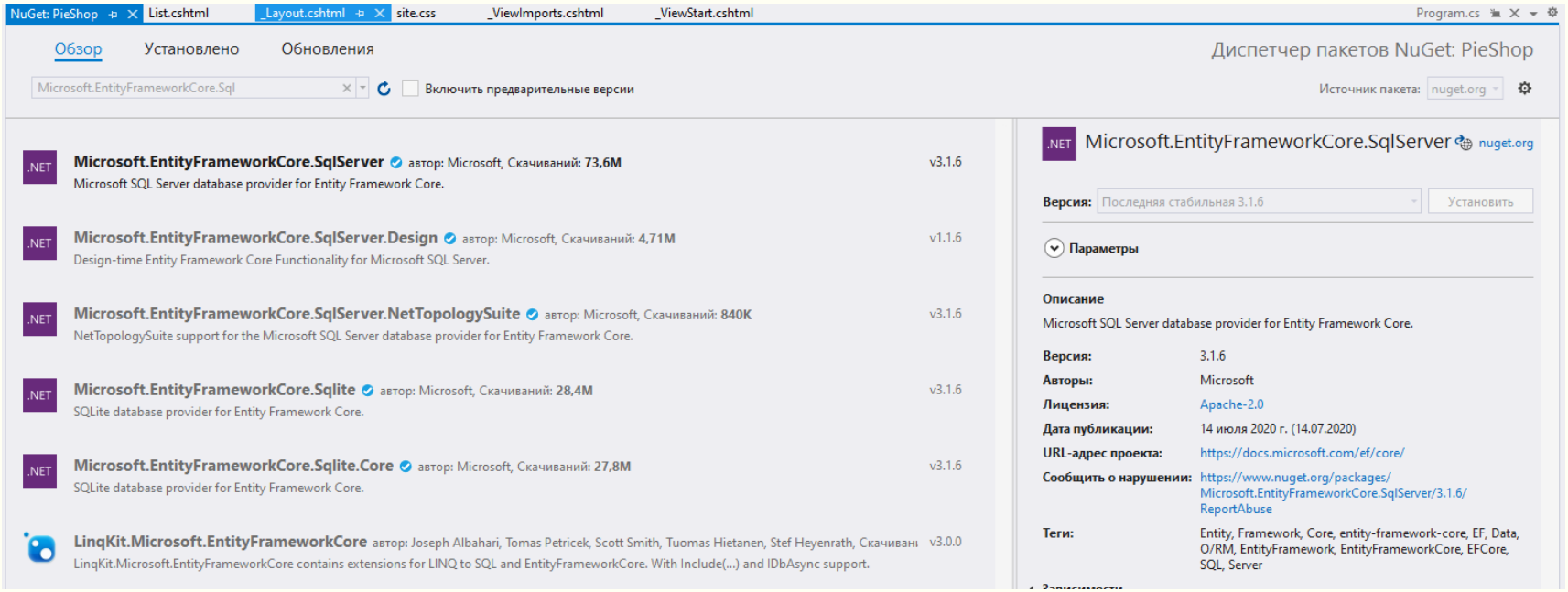
Оновимо представлення та макет

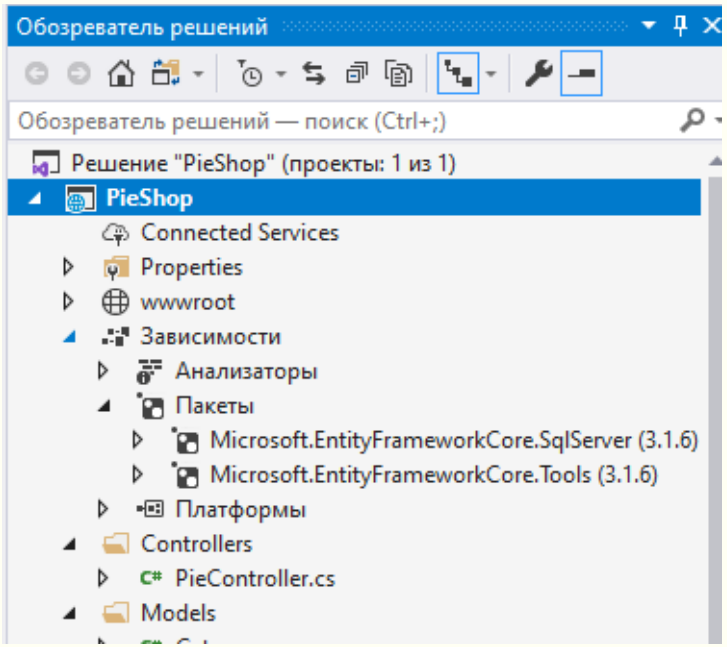


Підключимо базу даних

1) Додаємо необхідні пакети (потрібно для v3.0, для раніших версій – пропускається)

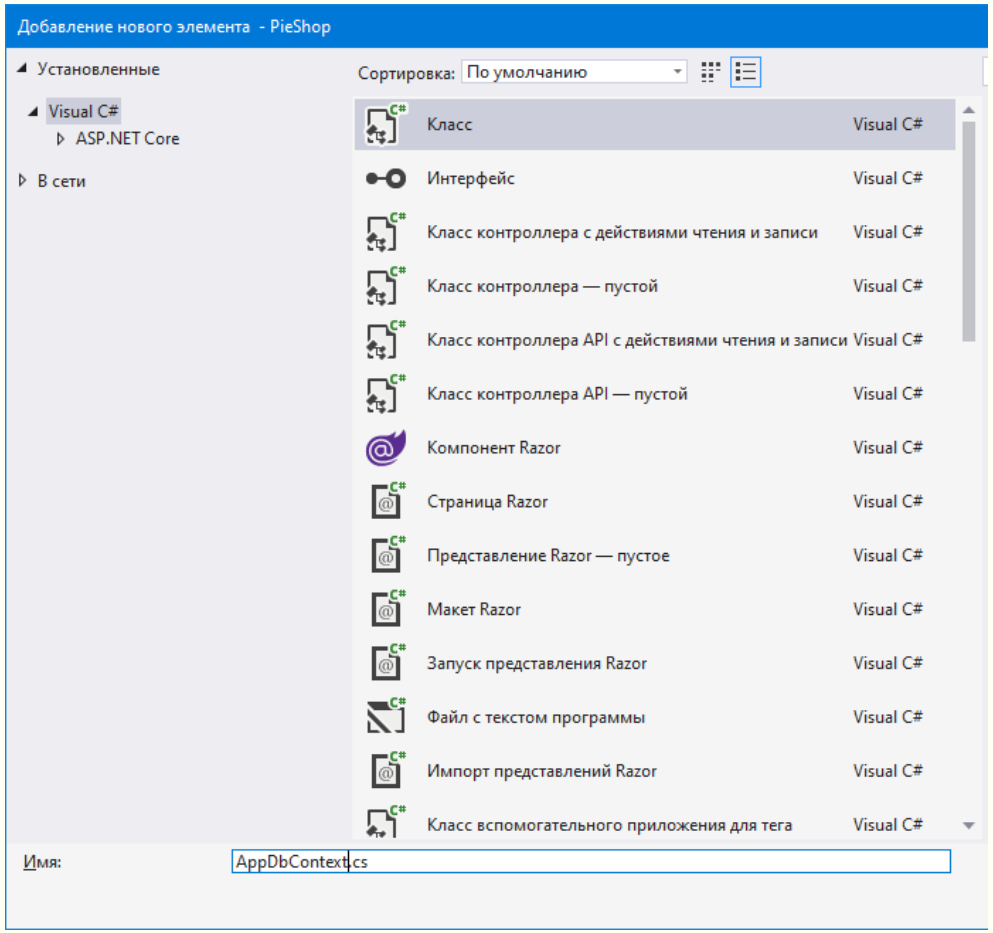
- *Microsoft.EntityFrameworkCore.SqlServer, Microsoft.EntityFrameworkCore.Tools*





Підключимо базу даних

- 2) Створимо контекст DbContext
  - Додаємо до моделі клас AppDbContext, породжений від DbContext




```
namespace PieShop.Models
{
    public class AppDbContext : DbContext
    {
        // для виконання обов'язково повинен бути екземпляр DbContextOptions
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
        {

        }

        // сутності, якими будуть керувати (таблиці БД)
        public DbSet<Pie> Pies { get; set; }
        public DbSet<Category> Categories { get; set; }

    }
}
```

Конфігуруємо веб-додаток для взаємодії з майбутньою БД

- Додамо рядок підключення в appsettings.json

appsettings.json | AppDbContext.cs | NuGet: PieShop | List.cshtml | _Layout.cshtml | site.css | _ViewImports.cshtml | _ViewStart.cshtml

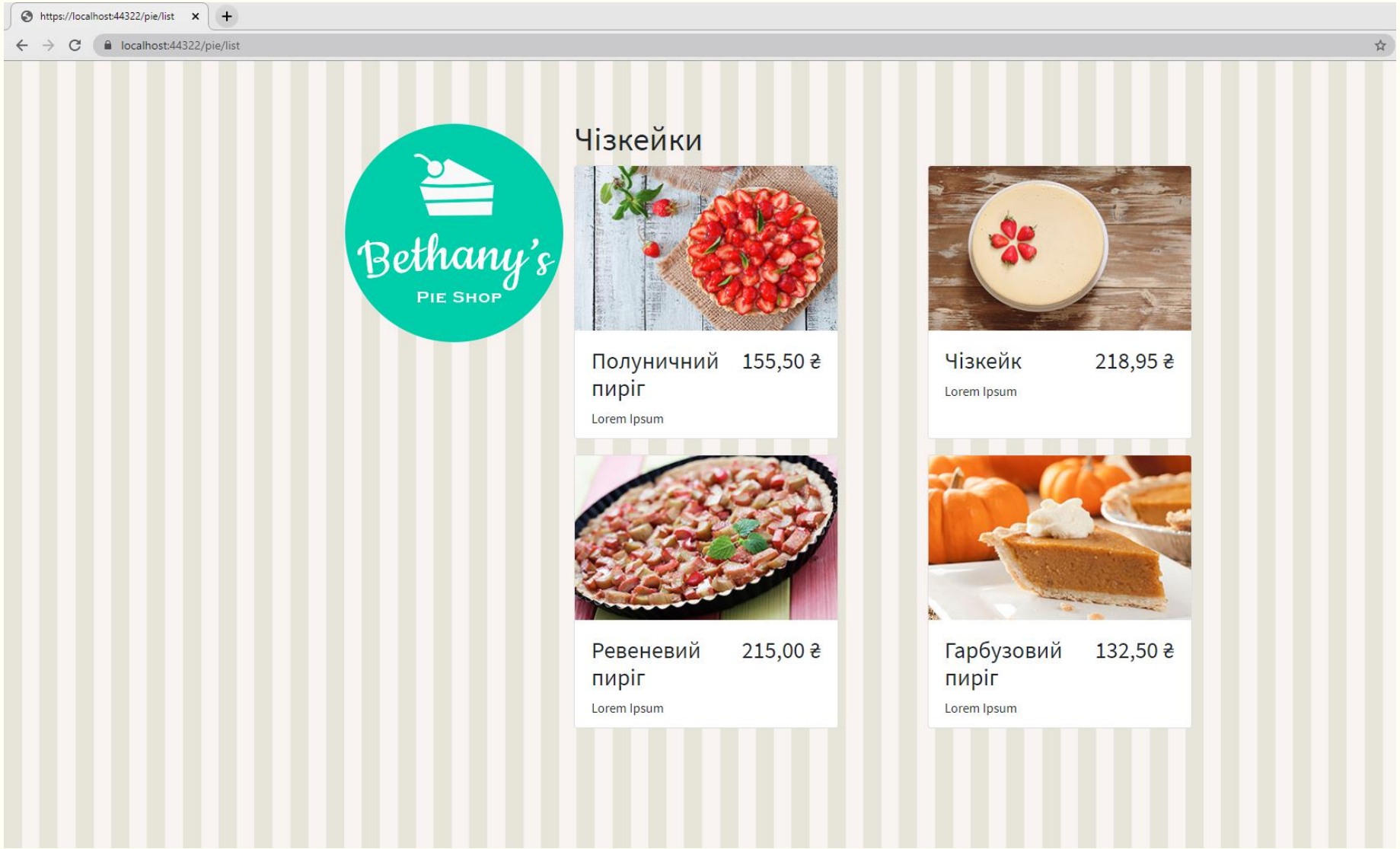
https://json.schemastore.org/appsettings

```
{
  "ConnectionStrings": {
      "BloggingDatabase": "Server=(localdb)\\mssqllocaldb;Database=PieShop;Trusted_Connection=True;"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

- Додамо сервіс, щоб розкрити базу даних для додатку в Startup.cs:

```
public class Startup
{
    // appsetting.json автоматично зчитується в екземпляр IConfiguration
    public IConfiguration Configuration { get; set; }

    // ін'єкція IConfiguration-об'єкта в конструктор класу
    public Startup(IConfiguration configuration)
    {
        Configuration = Configuration;
    }

    public void ConfigureServices(IServiceCollection services)
    {
        // викликаємо метод-розширення AddDbContext з опцією використання SQL Server
        // UseSqlServer приймає connection string
        services.AddDbContext<AppDbContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("DefaultConfiguration")));

        // реєстрація репозиторія в DI-контейнері
        // тепер можемо звертатись до IPieRepository будь-де в проекті
        services.AddScoped<IPieRepository, MockPieRepository>();
```

Оновимо представлення та макет



Робота з реальними даними за допомогою Entity Framework Core

- Підтримує тільки Code-First підхід

Class

```
public class Pie
{
    public int PieId { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
}
```

Table

PieId	Int (PK)
Name	String
Description	string

Додаємо EF Core в додаток

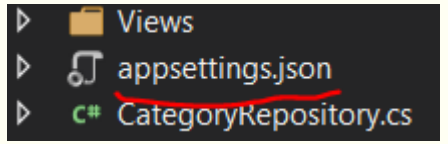
- 1) Маємо модель предметної області (доменну модель) як набір класів, з яких EF Core створить базу даних.
- 2) Потрібно створити клас контексту БД – місток між кодом та реальною БД.
  - Він управляє об'єктами-сутностями протягом виконання додатку, буде заповнювати об'єкти даними з БД, а також відстежувати зміни та зберігати оновлені дані в БД.
  - Назвемо його AppDbContext (обов'язкове наслідування від базового класу DbContext) та сформуємо каркас:

```
public class AppDbContext : DbContext
{

   public AppDbContext
      (DbContextOptions<AppDbContext> options): base(options)
   {
   }

    public DbSet<Pie> Pies { get; set; }

}
```

  - Зазвичай у ньому розкриваються DbSet-властивості для кожної сутності, що розкривається чи управляється.
  - За допомогою DbSet<Pie>-властивості організується доступ до даних з таблиці Pie.

Додаємо EF Core в додаток

- 3) Конфігурування додатку з використанням connection string.
  - До .NET Core ці налаштування містились у файлі WebConfig.
  - Нині вони розташовуються в файлі appsettings.json.
  - Структурно вигляд наступний:




```
{
   "ConnectionStrings": {
      "DefaultConnection":
         "Server=(localdb)\\mssqllocaldb;
         Database=BethanysPieShop;
         Trusted_Connection=True;
         MultipleActiveResultSets=true“
   }
}
```

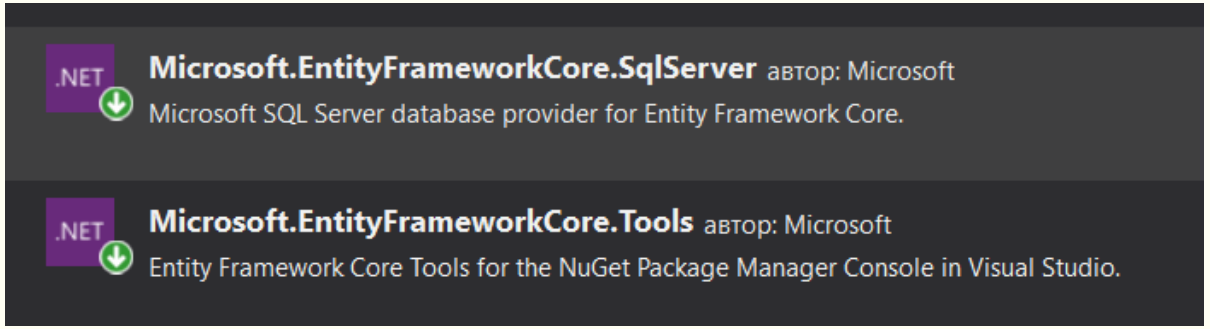
  - Далі вказуємо, що будемо використовувати EF Core – реєструємо сервіси в методі ConfigureServices():

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<AppDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddScoped<ICategoryRepository, CategoryRepository>();
    services.AddScoped<IPieRepository, PieRepository>();

    services.AddControllersWithViews();//services.AddMvc(); would also work still
}
```

- 4) Додати до проєкту пакети (для .NET Core 3.0+)

Демонстрація

- Для пункту 4 додаємо NuGet-пакети:



- У папку Model додамо клас AppDbContext.
  - При успадкуванні від DbContext повинен бути присутнім об'єкт DbContextOptions, який передається як параметр спеціального конструктора.

```
public class AppDbContext: DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
    {

    }

    public DbSet<Category> Categories { get; set; }
    public DbSet<Pie> Pies { get; set; }
}
```

Демонстрація

- Додаємо контекст БД в конфігурацію запуску (через ін'єкцію в конструктор), підключаючи налаштування DefaultConnection:

```
public class Startup
{
    public IConfiguration Configuration { get; }

    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    // This method gets called by the runtime. Use this method to add services to the container.
    // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddDbContext<AppDbContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

        services.AddScoped<ICategoryRepository, MockCategoryRepository>();
        services.AddScoped<IPieRepository, MockPieRepository>();

        services.AddControllersWithViews();//services.AddMvc(); would also work still
    }
```

Подальший розвиток

- Для доступу до даних таблиці Pie застосовуватимемо LINQ-оператори до властивості Pies.
  - Тут отримуватимемо не лише торти, але й категорії, до яких вони належать (оператор Include()), причому відбиратимемо лише торти тижня:

```
_appDbContext.Pies.
  Include(c => c.Category).Where(p => p.IsPieOfTheWeek);
```

- Також контекст БД застосовується для оновлення даних у БД.
  - На даний момент додаток візуалізує тільки read-only список тортів.
  - Пізніше будемо додавати код для вставки та оновлення даних у БД, на зразок корзини (shoppingCart).
  - Після завершення заповнення корзини зміни фіксуються за допомогою методу SaveChanges() з метою синхронізаіції з БД.
  - Каркас подібного коду для вставки даних:

```
foreach (var shoppingCartItem in shoppingCartItems)
{
    var orderDetail = new OrderDetail
    {
        Amount = shoppingCartItem.Amount,
        PieId = shoppingCartItem.Pie.PieId,
        Price = shoppingCartItem.Pie.Price
    };
    order.OrderDetails.Add(orderDetail);
}

_appDbContext.Orders.Add(order);

_appDbContext.SaveChanges();
```

Демонстрація створення реального репозиторію даних

```
public class PieRepository: IPieRepository
{
    private readonly AppDbContext _appDbContext;

    public PieRepository(AppDbContext appDbContext)
    {
        _appDbContext = appDbContext;
    }

    public IEnumerable<Pie> AllPies
    {
        get
        {
            return _appDbContext.Pies.Include(c => c.Category);
        }
    }

    public IEnumerable<Pie> PiesOfTheWeek
    {
        get
        {
            return _appDbContext.Pies.Include(c => c.Category).Where(p => p.IsPieOfTheWeek);
        }
    }

    public Pie GetPieById(int pieId)
    {
        return _appDbContext.Pies.FirstOrDefault(p => p.PieId == pieId);
    }
}
```

- Створимо в папці Model клас PieRepository, який реалізуватиме інтерфейс IPieRepository.
  - Реалізація методів з інтерфейсу IPieRepository буде за кулісами використовувати контекст БД для роботи з нею, зокрема через властивість Pie.
  - Оскільки клас Pie містить властивість типу Category, метод AllPies() буде повертати для тортів і їх категорію з бази даних.

Демонстрація створення реального репозиторію даних

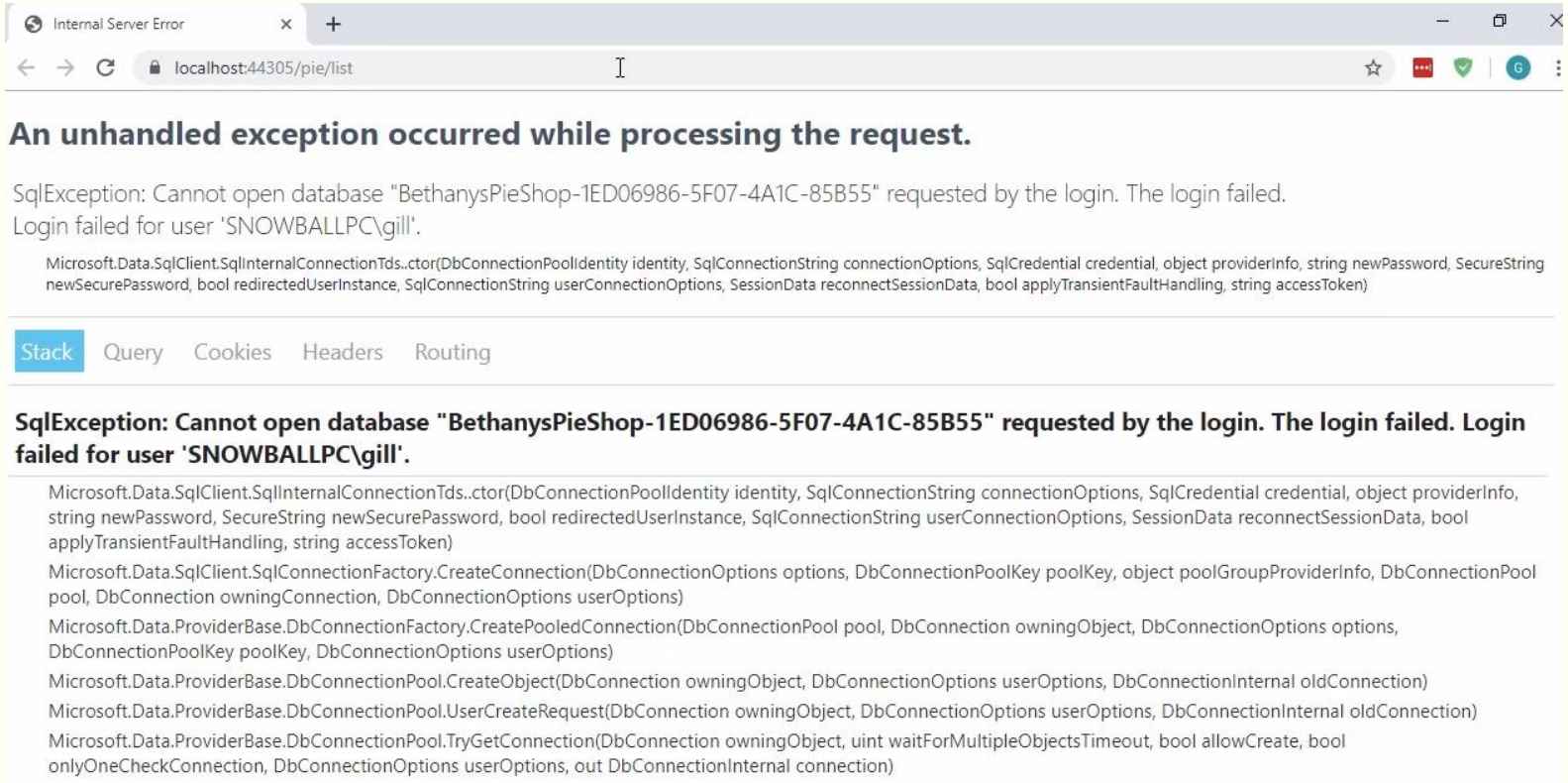
- Створимо клас CategoryRepository, який реалізуватиме інтерфейс ICategoryRepository.

```
public class CategoryRepository: ICategoryRepository
{
    private readonly AppDbContext _appDbContext;

    public CategoryRepository(AppDbContext appDbContext)
    {
        _appDbContext = appDbContext;
    }

    public IEnumerable<Category> AllCategories => _appDbContext.Categories;
}
```

- Замінимо використання мок-репозиторіїв на реальні в файлі Startup.cs:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<AppDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddScoped<ICategoryRepository, CategoryRepository>();
    services.AddScoped<IPieRepository, PieRepository>();

    services.AddControllersWithViews();//services.AddMvc(); would also work still
}
```

Запуск сайту на даному етапі



- Ми ще не створили реальної бази даних.
  - Перехід на цю сторінку зумовлений налаштуваннями в методі Configure():

```
public void Configure(IApplicationBuilder app,
                      IWebHostEnvironment env)
{
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();
        …
```

Створення бази даних

- Створення БД з програмного коду відбувається за допомогою міграцій EF Core.

  - Створення міграцій відбувається в Package Manager Console у Visual Studio (Вид – Другие окна – Консоль диспетчера пакетов).
  - Основні команди:

```
>add-migration <MigrationName>
>update-database
```

  - Створення міграції не виконує автоматичну синхронізацію даних, тому потрібно запускати команду update-database.

Ініціалізація бази даних

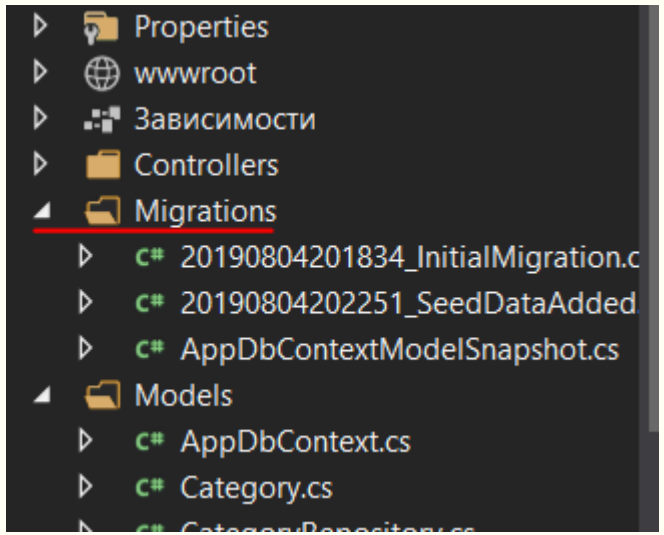
- У програмному коді можна перевіряти, чи є деякі дані в БД.
  - Якщо їх немає, виконуємо вставку.

Демонстрація створення БД

- Назвемо першу міграцію InitialMigration

```
PM> add-migration InitialMigration
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Model.Validation[30000]
      No type was specified for the decimal property 'Price' on entity type 'Pie'. This will cause values to be silently truncated if they do not fit in the default precision
and scale. Explicitly specify the SQL server column type that can accommodate all the values in 'OnModelCreating' using 'HasColumnType()', specify precision and scale using
'HasPrecision()' or configure a value converter using 'HasConversion()'.
```

- Додається нова директорія, в яку буде поміщатись міграційний код:
- Буде створено клас InitialMigration з методами Up() і Down().
- Up() займається синхронізацією даних у додатку та БД.
- Down() відповідає за скасування (revert) міграції.

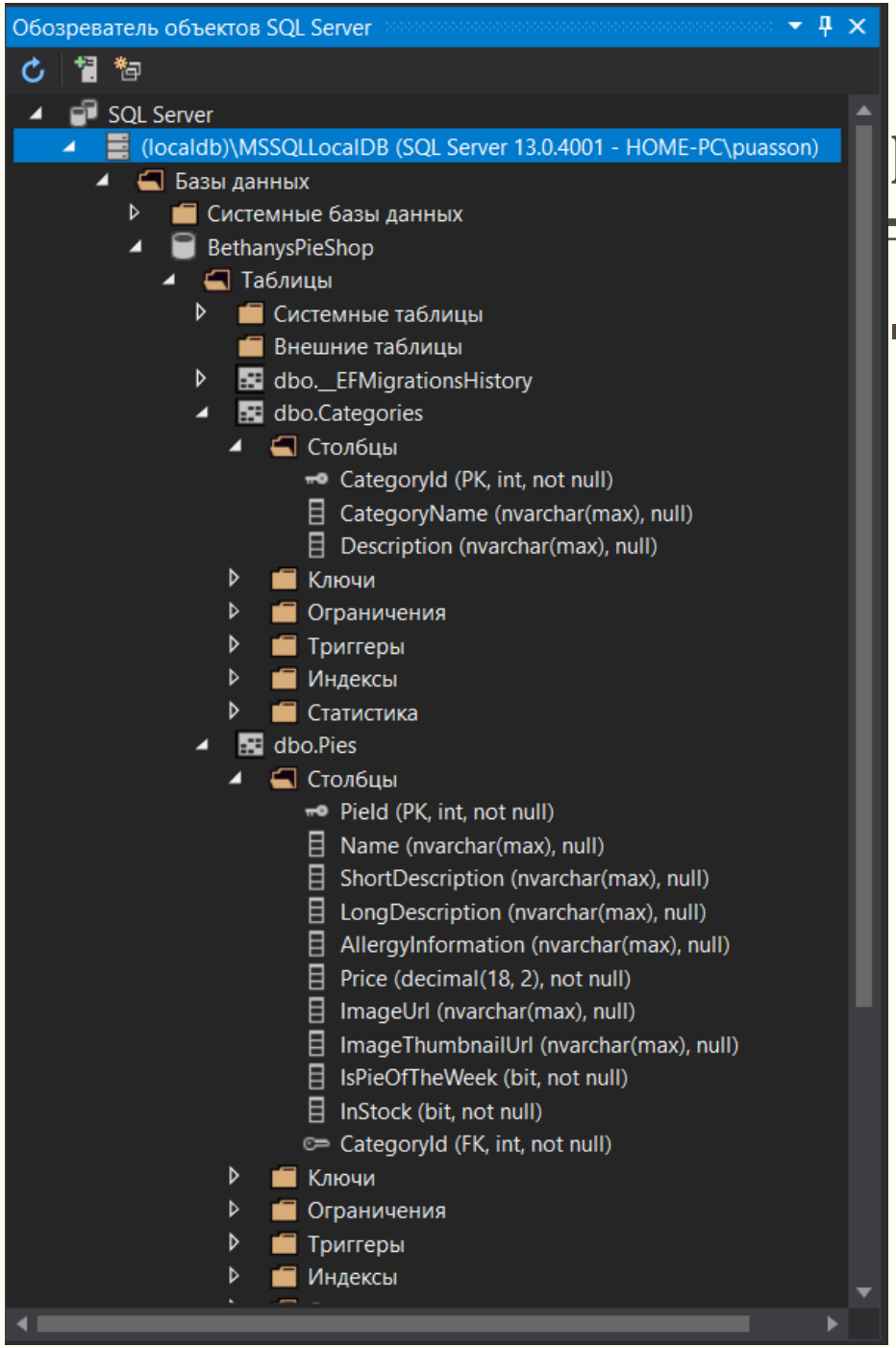


Демонстрація створення БД: метод Up()

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "Categories",
        columns: table => new
        {
            CategoryId = table.Column<int>(nullable: false)
                .Annotation("SqlServer:ValueGenerationStrategy",
                   SqlServerValueGenerationStrategy.IdentityColumn),
            CategoryName = table.Column<string>(nullable: true),
            Description = table.Column<string>(nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Categories", x => x.CategoryId);
        });
```

- Код створює 2 таблиці: Categories та Pies.
  - Для створення бази даних виконайте команду update-database у консолі

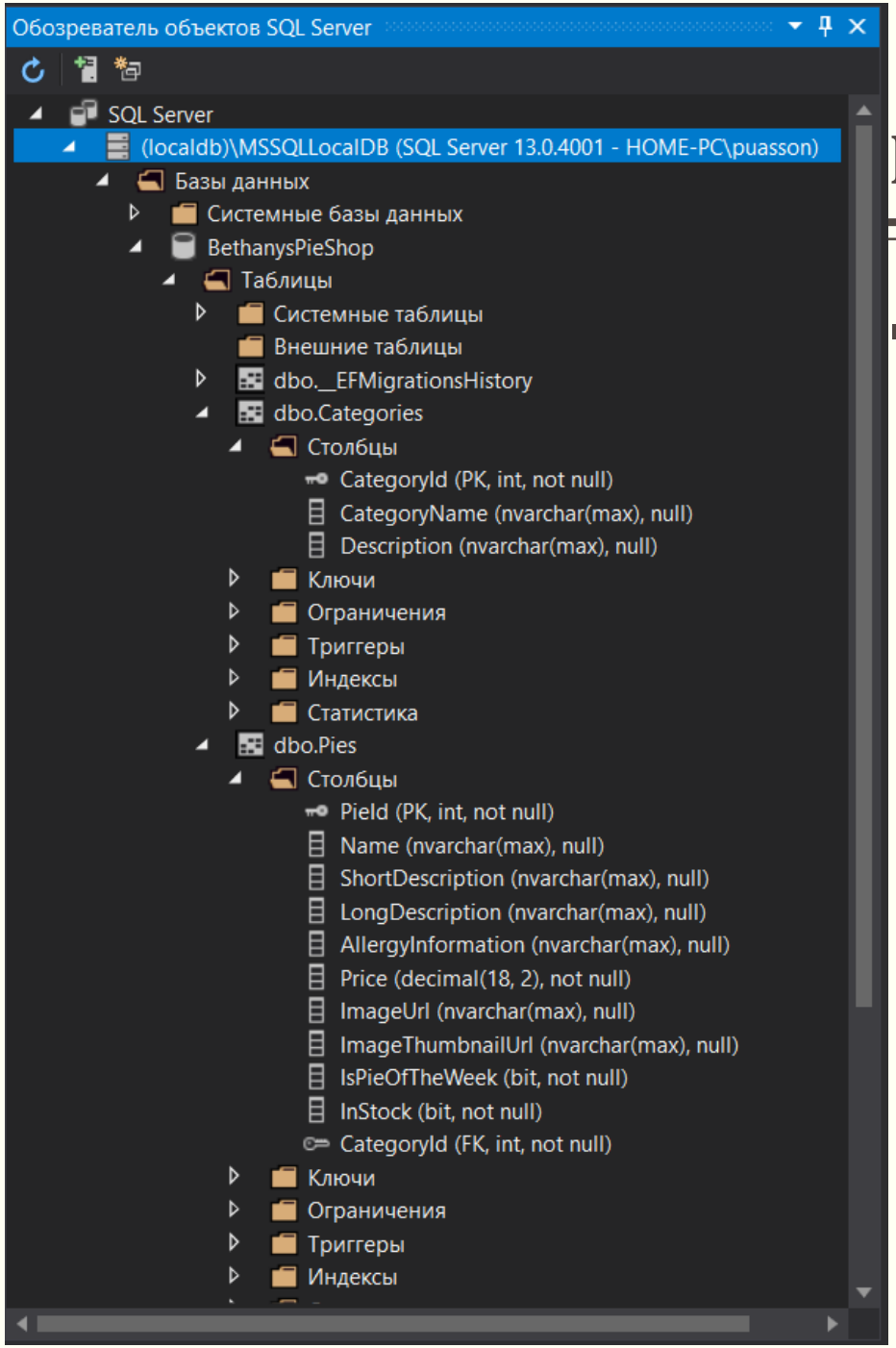

```
migrationBuilder.CreateTable(
    name: "Pies",
    columns: table => new
    {
        PieId = table.Column<int>(nullable: false)
            .Annotation("SqlServer:ValueGenerationStrategy",
               SqlServerValueGenerationStrategy.IdentityColumn),
        Name = table.Column<string>(nullable: true),
        ShortDescription = table.Column<string>(nullable: true),
        LongDescription = table.Column<string>(nullable: true),
        AllergyInformation = table.Column<string>(nullable: true),
        Price = table.Column<decimal>(nullable: false),
        ImageUrl = table.Column<string>(nullable: true),
        ImageThumbnailUrl = table.Column<string>(nullable: true),
        IsPieOfTheWeek = table.Column<bool>(nullable: false),
        InStock = table.Column<bool>(nullable: false),
        CategoryId = table.Column<int>(nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Pies", x => x.PieId);
        table.ForeignKey(
            name: "FK_Pies_Categories_CategoryId",
            column: x => x.CategoryId,
            principalTable: "Categories",
            principalColumn: "CategoryId",
            onDelete: ReferentialAction.Cascade);
    });

migrationBuilder.CreateIndex(
    name: "IX_Pies_CategoryId",
    table: "Pies",
    column: "CategoryId");
}
```



Ініціалізація бази даних

- Конфігурацію процесу створення бази даних рекомендується покладати на метод OnModelCreating(), який розмістимо в класі контексту БД:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    //seed categories
    modelBuilder.Entity<Category>().HasData(new Category { CategoryId = 1, CategoryName = "Fruit pies" });
    modelBuilder.Entity<Category>().HasData(new Category { CategoryId = 2, CategoryName = "Cheese cakes" });
    modelBuilder.Entity<Category>().HasData(new Category { CategoryId = 3, CategoryName = "Seasonal pies" });

    //seed pies
    modelBuilder.Entity<Pie>().HasData(new Pie
    {
        PieId = 1,
        Name = "Apple Pie",
        Price = 12.95M,
        ShortDescription = "Our famous apple pies!",
        LongDescription =
            "Icing carrot cake jelly-o cheesecake. Sweet roll marzipan marshmallow toffee brownie brownie
candy tootsie roll. Chocolate cake gingerbread tootsie roll oat cake pie chocolate bar cookie dragée brownie.
Lollipop cotton candy cake bear claw oat cake. Dragée candy canes dessert tart. Marzipan dragée gummies lollipop
jujubes chocolate bar candy canes. Icing gingerbread chupa chups cotton candy cookie sweet icing bonbon gummies.
Gummies lollipop brownie biscuit danish chocolate cake. Danish powder cookie macaroon chocolate donut tart. Carrot
cake dragée croissant lemon drops liquorice lemon drops cookie lollipop toffee.",
        CategoryId = 1,
        ImageUrl = "https://gillcleerenpluralsight.blob.core.windows.net/files/applepie.jpg",
        InStock = true,
        IsPieOfTheWeek = true,
        ImageThumbnailUrl = "https://gillcleerenpluralsight.blob.core.windows.net/files/applepiesmall.jpg",
        AllergyInformation = ""
    });
```

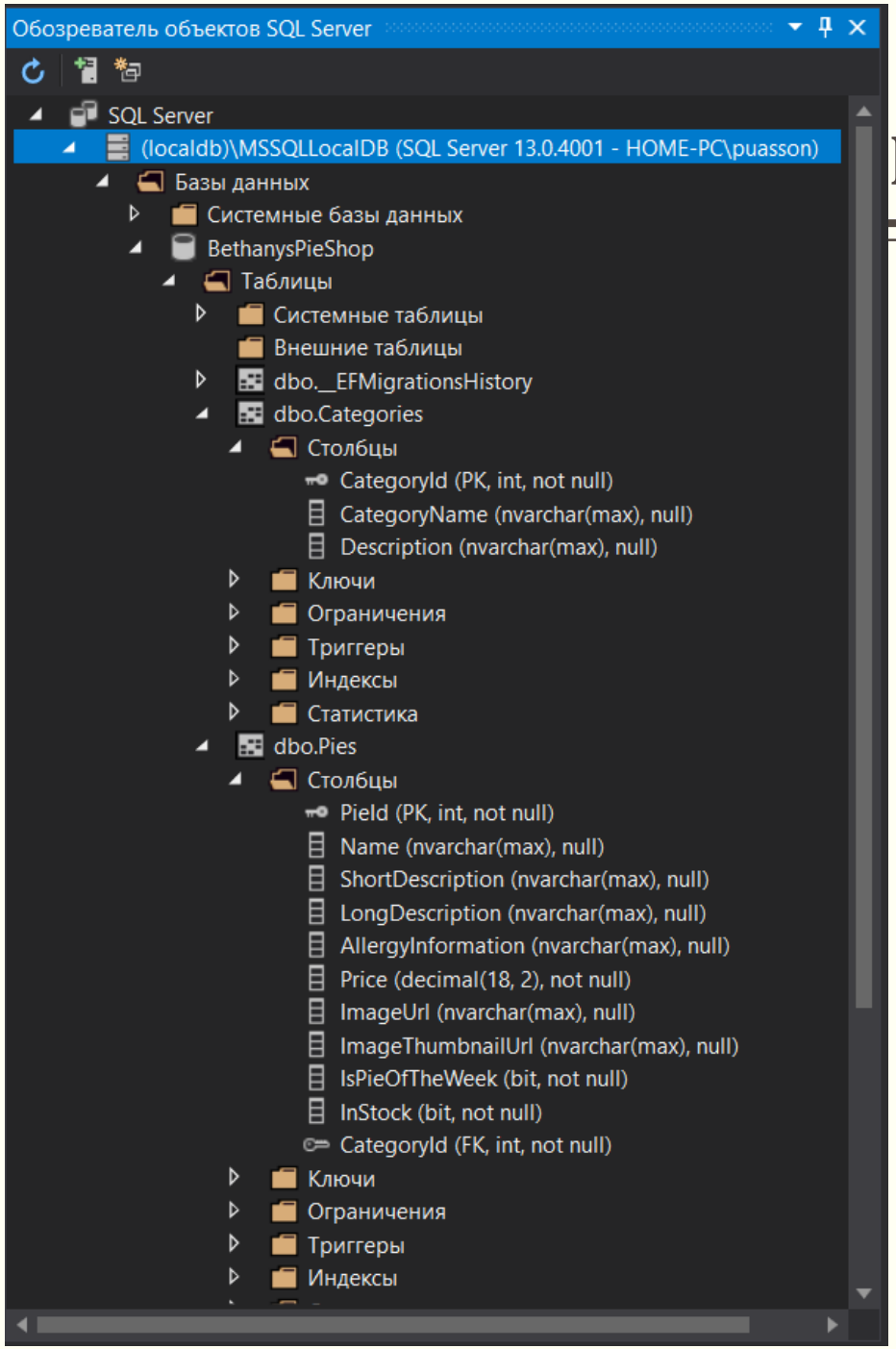


Ініціалізація бази даних

- Таким чином, додаємо кілька тортів та фіксуємо зміни, додавши нову міграцію (SeedDataAdded):

```
PM> add-migration SeedDataAdded
Microsoft.EntityFrameworkCore.Model.Validation[30000]
      No type was specified for the decimal column 'Price' on entity type 'Pie'. This will cause values to be si
precision and scale. Explicitly specify the SQL server column type that can accommodate all the values using 'Ha
Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 3.0.0-preview8.19405.11 initialized 'AppDbContext' using provider 'Microsoft.EntityF
To undo this action, use Remove-Migration.
```

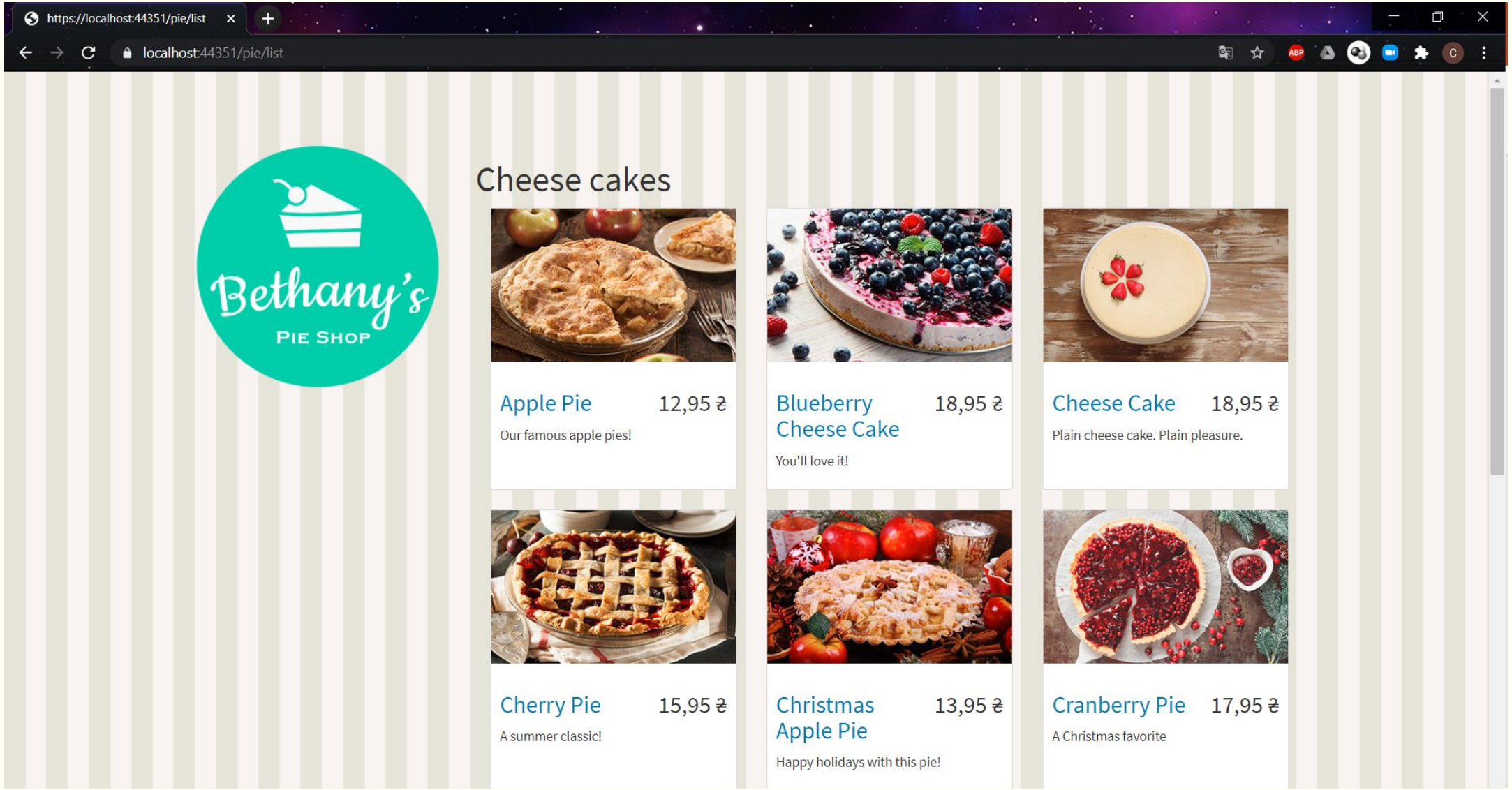
  - Автоматично згенерований код вставить дані про всі категорії та торти, проте все ще потрібно синхронізувати дані з БД за допомогою команди update-database.



Ініціалізація бази даних

```
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.InsertData(
                table: "Categories",
                columns: new[] { "CategoryId", "CategoryName", "Description" },
                values: new object[] { 1, "Fruit pies", null });

            migrationBuilder.InsertData(
                table: "Categories",
                columns: new[] { "CategoryId", "CategoryName", "Description" },
                values: new object[] { 2, "Cheese cakes", null });

            migrationBuilder.InsertData(
                table: "Categories",
                columns: new[] { "CategoryId", "CategoryName", "Description" },
                values: new object[] { 3, "Seasonal pies", null });

            migrationBuilder.InsertData(
                table: "Pies",
                columns: new[] { "PieId", "AllergyInformation", "CategoryId", "ImageThumbnailUrl", "ImageUrl",
                                 "InStock", "IsPieOfTheWeek", "LongDescription", "Name", "Price", "ShortDescription" },
                values: new object[,]
                {
                    { 1, "", 1, "https://gillcleerenpluralsight.blob.core.windows.net/files/applepiesmall.jpg",
"https://gillcleerenpluralsight.blob.core.windows.net/files/applepie.jpg", true, true, "Icing carrot cake jelly-o
cheesecake. Sweet roll marzipan marshmallow toffee brownie brownie candy tootsie roll. Chocolate cake gingerbread
tootsie roll oat cake pie chocolate bar cookie dragée brownie. Lollipop cotton candy cake bear claw oat cake.
Dragée candy canes dessert tart. Marzipan dragée gummies lollipop jujubes chocolate bar candy canes. Icing
gingerbread chupa chups cotton candy cookie sweet icing bonbon gummies. Gummies lollipop brownie biscuit danish
chocolate cake. Danish powder cookie macaroon chocolate donut tart. Carrot cake dragée croissant lemon drops
liquorice lemon drops cookie lollipop toffee. Carrot cake carrot cake liquorice sugar plum topping bonbon pie
muffin jujubes. Jelly pastry wafer tart caramels bear claw. Tiramisu tart pie cake danish lemon drops. Brownie
cupcake dragée gummies.", "Apple Pie", 12.95m, "Our famous apple pies!" },
                    { 4, "", 1, "https://gillcleerenpluralsight.blob.core.windows.net/files/cherrypiesmall.jpg",
"https://gillcleerenpluralsight.blob.core.windows.net/files/cherrypie.jpg", true, false, "Icing carrot cake jelly-o
cheesecake. Sweet roll marzipan marshmallow toffee brownie brownie candy tootsie roll. Chocolate cake gingerbread
tootsie roll oat cake pie chocolate bar cookie dragée brownie. Lollipop cotton candy cake bear claw oat cake.
Dragée candy canes dessert tart. Marzipan dragée gummies lollipop jujubes chocolate bar candy canes. Icing
gingerbread chupa chups cotton candy cookie sweet icing bonbon gummies. Gummies lollipop brownie biscuit danish
chocolate cake. Danish powder cookie macaroon chocolate donut tart. Carrot cake dragée croissant lemon drops
```

Результат запуску



Внесення змін у модель

- Теж здійснюється за допомогою міграцій.
  - Нехай доповнимо модель торта (клас Pie) автоматичну властивість Notes типу string.
- Додамо нову міграцію NotesAddedOnPie
  - Виконавши команду update-database, у БД з'явиться новий стовпчик таблиці Pies.

```
using Microsoft.EntityFrameworkCore.Migrations;

namespace BethanysPieShop.Migrations
{
    1 reference
    public partial class NotesAddedOnPie : Migration
    {
        2 references
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.AddColumn<string>(
                name: "Notes",
                table: "Pies",
                nullable: true);
        }

        2 references
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropColumn(
                name: "Notes",
                table: "Pies");
        }
    }
}
```