



**Gill Cleeren**

CTO XPIRIT BELGIUM

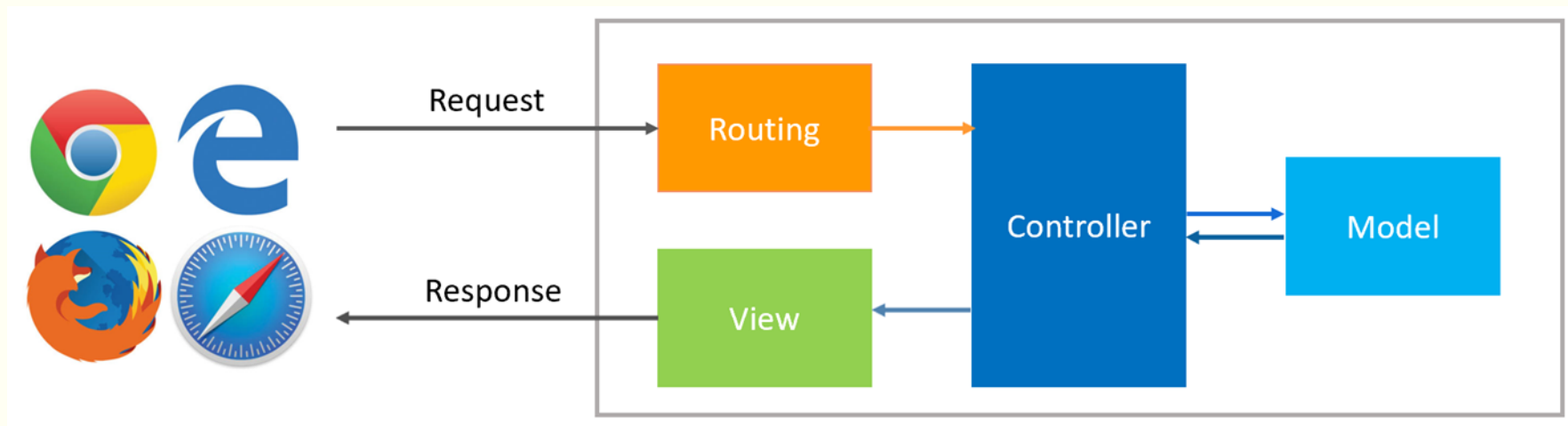
@gillcleeren [www.snowball.be](http://www.snowball.be)

# ФОРМУВАННЯ БАЗОВОЇ НАВІГАЦІЇ ПО САЙТУ

Питання 6.1 (відео 40-56)

# Демонстрація створення реального репозиторію даних

- Класична ситуація: запит на сервер відбувається щодо отримання реального файлу, який розміщується на сервері.
  - Далі файл повертається користувачу в якості відгуку.
- В ASP.NET Core MVC-проєктах послідовність інша.
  - Запити обробляються в відповідних action-методах на рівні відповідного контролера.
  - URL ефективно таргетує action-метод, що повертає файл cshtml.
  - Тут немає прямого зв'язку між файлом та запитом, що надходить.
  - Вибір потрібного Action-методу та контролера відбувається за допомогою процесу маршрутизації (routing), вбудованого в ASP.NET Core MVC.



# Процес маршрутизації

---

- Це MVC-специфічна можливість, яка відображає запит за деяким URI на кінцеву точку (endpoint) всередині додатку.
  - Дана можливість працює, оскільки було додано конвеєр запитів (request pipeline), а також викликалися методи UseRouting() та UseEndpoints() всередині методу Configure().
  - ASP.NET Core MVC також може генерувати посилання (links) на основі інформації щодо маршрутизації.
- Маршрутизація буває двох видів:
  - **Маршрутизація на основі угод (Convention-based routing)** – дозволяє визначати набір маршрутів, які «зіставляють» (match) URL-и запиту та їх сегменти з контролерами, діями (actions) та параметрами. Буде застосовуватись в подальшому.
  - Маршрутизація на основі атрибутів (Attribute-based routing) – дозволяє впроваджувати маршрутизацію, декоруючи атрибутами action-методи. Переважно використовується в різних API.
- З переходом до .NET Core 3 дещо змінились middleware-компоненти для маршрутизації.

# Шаблони та маршрутизація

---

- Шаблони дозволяють не вводити всі можливі шляхи для додатку вручну.
  - URL-и зіставляються з шаблонами, обирається шаблон з першим збігом.
- Шлях складається з сегментів: хоста, контролера, дії, значення



- Шаблон:
- Action-метод:

`{Controller}/{Action}`

```
public class PieController : Controller
{
    public ActionResult List()
    {
        return View();
    }
}
```

# Робота з сегментами

---

<http://www.bethanyspieshop.com/Pie/Details/1>

Controller

Action

Value

{Controller}/{Action}/{id}

- Відповідний action-метод:

```
public class PieController : Controller
{
    public ViewResult Details(int id)
    {
        //Do something
    }
}
```

# Маршрутизація в кодi

---

- Спочатку додаємо ПЗ проміжного прошарку для здійснення маршрутизації.

- Зразок коду з методу Configure():

```
app.UseRouting();  
app.UseEndpoints(endpoints =>  
{  
    endpoints.MapControllerRoute(  
        name: "default",  
        pattern: "{controller}/{action}");  
});
```

- Викликаючи метод UseEndpoints(), ми звертаємось до ASP.NET Core за конфігуруванням узагальненого шляху за умовчанням.
  - Впроваджуємо маршрутизацію на основі угод, викликаючи метод MapControllerRoute() з раніше розглянутим шаблоном: контролер та дія будуть виокремлені з загального URI та передані об'єкту-контролеру та його action-методам.
  - Якщо існує кілька маршрутів, що відповідають шаблону, потрібно ретельно продумати їх порядок при зіставленні, оскільки перший збіг завершує перевірку. Найбільш конкретні шаблони мають бути першими.

# Відмінності з ASP.NET Core 2.1

---

```
app.UseRouting();  
app.UseEndpoints(endpoints =>  
{  
    endpoints.MapControllerRoute(  
        name: "default",  
        pattern: "{controller}/{action}");  
});
```

■ ASP.NET Core 3

```
app.UseMvc(routes =>  
{  
    routes.MapRoute(  
        name: "default",  
        template: "{controller}/{action}");  
});
```

ASP.NET Core 2.1

## Уточнення шаблонів (route defaults)

---

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}");
});
```

- Допускається і передача значень:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id}");
});
```

```
public class PieController : Controller
{
    public ViewResult Details(int id)
    {
        //Do something
    }
}
```

- В ході прив'язування моделі (model binding) забезпечується надходження значення id в action-параметр id (назви повинні бути однаковими).



# Уточнення шаблонів (optional segments)

---

- Сегмент можна зробити необов'язковим, якщо додати символ '?'.

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

- Шаблон доречний для шляхів на зразок:
  - [www.betanyspieshop.com/Pie/List](http://www.betanyspieshop.com/Pie/List)
  - [www.betanyspieshop.com/Pie/Details/1](http://www.betanyspieshop.com/Pie/Details/1)
- Також можливо накладати обмеження (constraints) на значення:

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id:int?}");
});
```

# Додамо навігацію на сайт

---

- Потрібно згенерувати коректні лінки в результуючому HTML-коді.
  - В ASP.NET MVC використовувались HTML-хелпери, які впливають на процес генерування HTML-коду.
  - Основний недолік HTML-хелперів – їх опосередкованість, непрямолінійний синтаксис.
  - Вони все ще працюють в ASP.NET Core MVC, проте з'явилась нова можливість – тег-хелпери.
  - Тег-хелпери складаються з коду розмітки, який буде виконуватись на стороні сервера, запускати (trigger) виконання коду (тут – генерування коректних лінків).
  - Тег-хелпери переважно замінюють HTML-хелпери та формують чистіший Razor HTML-код.

```
@Html.ActionLink  
("View Pie List", "List",  
"Pie")
```

```
<a asp-controller="Pie"  
asp-action="List">  
View Pie List  
</a>
```

◀ HTML Helpers

◀ Tag Helpers

# Додамо навігацію на сайт

---

- Вбудованих тег-хелперів небагато, можливо створити власні. Вбудовані:

- asp-controller
- asp-action
- asp-route-\* (\* - назва параметру)
- asp-route (задає конкретний маршрут)

```
<a asp-controller="Pie" asp-action="List">  
    View Pie List  
</a>
```



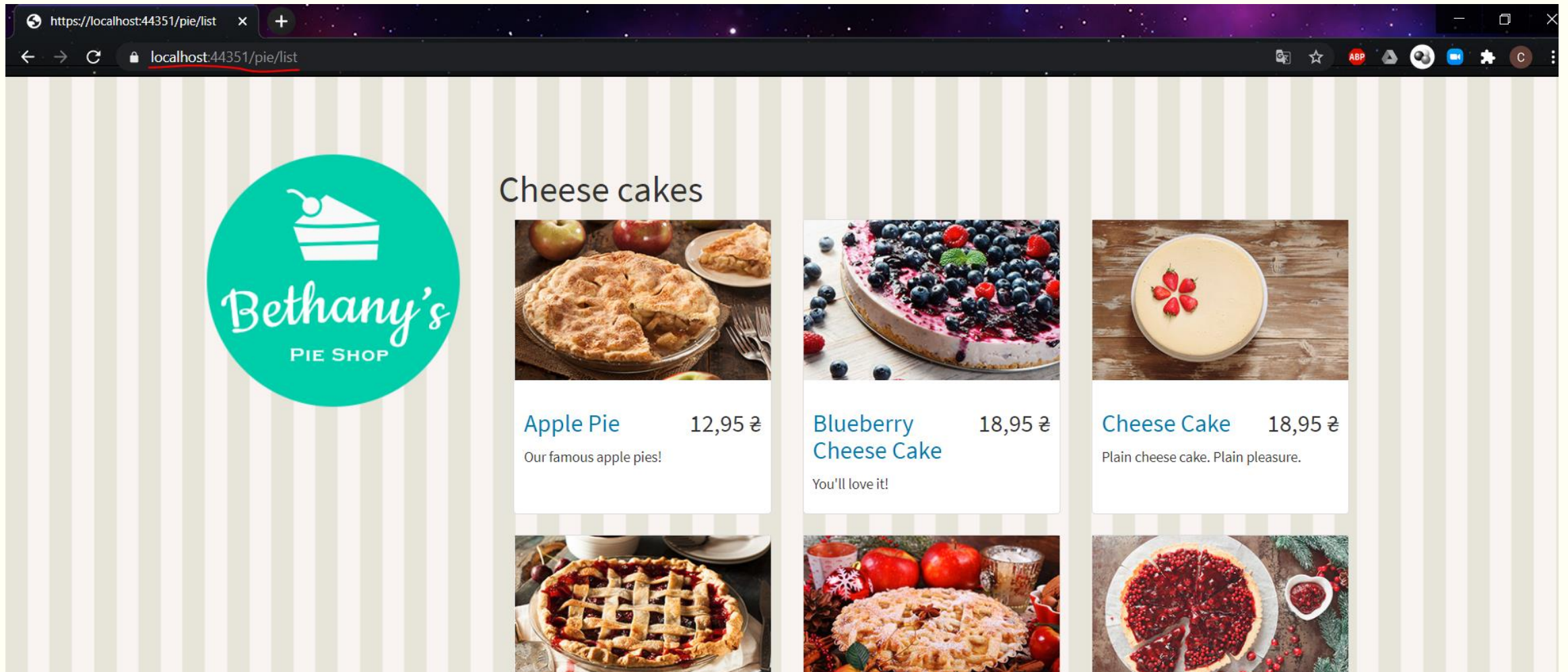
```
<a href="/Pie/List">View Pie List</a>
```

- Для загальної підтримки тег-хелперів потрібно додавати в кожний View-файл такий код:

```
@using BethanysPieShop.Models  
@using BethanysPieShop.ViewModels  
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

- Більш загальний підхід – додати цей код у файл \_ViewImports.cshtml (для всіх view в додатку).

# В поточній версії сайту для відображення веб-сторінки потрібно додатково уточнювати шлях



# Демонстрація створення реального репозиторію даних

---

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

    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

- Створимо нову дію в PieController для отримання даних щодо конкретного торта з репозиторію:

```
public IActionResult Details(int id)
{
    var pie = _pieRepository.GetPieById(id);
    if (pie == null)
        return NotFound();

    return View(pie);
}
```

- Також потрібне ще одне представлення для відображення цих даних.
  - Додамо в папку Views\Pie новий елемент Razor View з назвою Details.cshtml.

# Розмітка Details.cshtml

---

```
@model Pie
```

```
<h2>@Model.Name</h2>
```

```
<div class="thumbnail">
```

```
    
```

```
    <div class="caption-full">
```

```
        <h3 class="pull-right">@Model.Price</h3>
```

```
        <h3>
```

```
            <a href="#">@Model.Name</a>
```

```
        </h3>
```

```
        <h4>@Model.ShortDescription</h4>
```

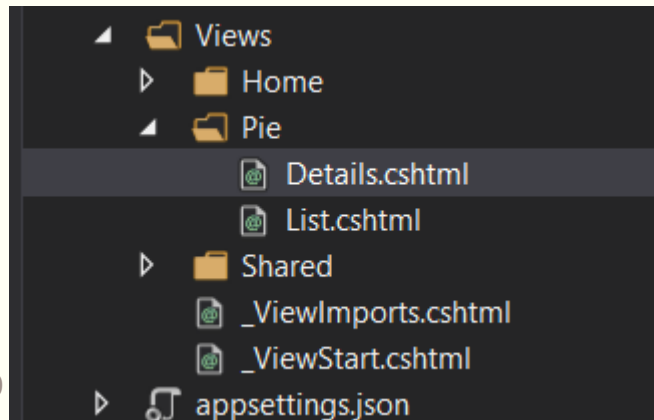
```
        <p>@Model.LongDescription</p>
```

```
    </div>
```

```
</div>
```

- Очевидно, що моделлю для цього представлення буде клас Pie.
- Оновимо файл \_ViewImports.cshtml:

```
1 @using BethanysPieShop.Models
2 @using BethanysPieShop.ViewModels
3
4 @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```





# Розмітка List.cshtml. Вводимо тег-хелпери

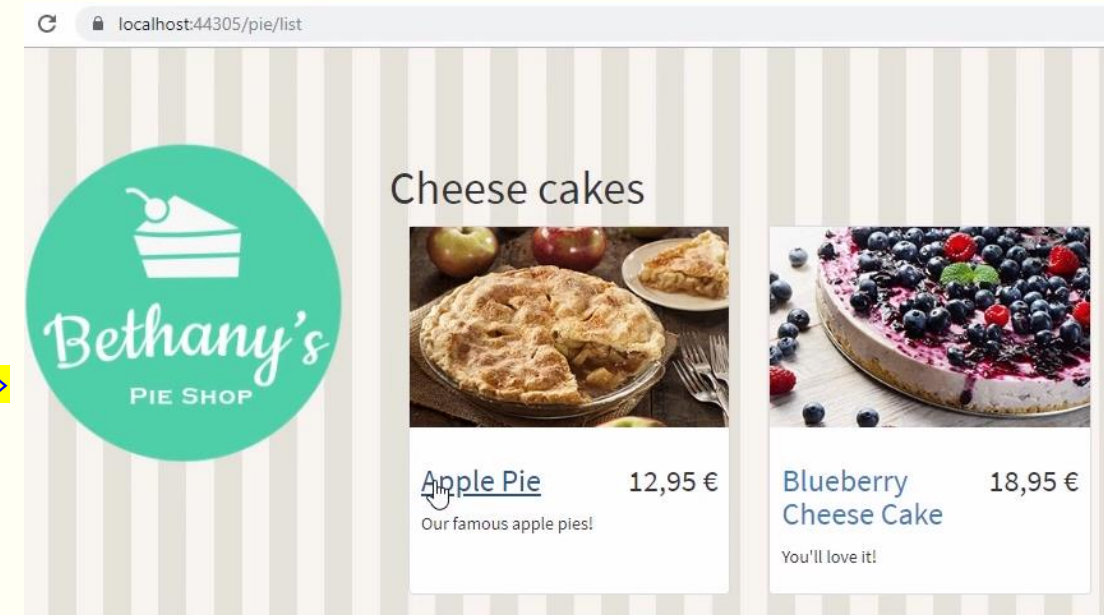
- Необов'язкову частину маршруту потрібно також прописати:

```
@model PiesListViewModel
```

```
<h1>@Model.CurrentCategory</h1>

@foreach (var pie in Model.Pies)
{
    <div class="col-sm-4 col-lg-4 col-md-4">
        <div class="thumbnail">
            
            <div class="caption">
                <h3 class="pull-right">@pie.Price.ToString("c")</h3>
                <h3>
                    <a asp-controller="Pie" asp-action="Details"
                      asp-route-id="@pie.PieId">@pie.Name</a>
                </h3>
                <p>@pie.ShortDescription</p>
            </div>
        </div>
    </div>
}
```

- Після цього посилання на окремі торти стануть клікабельними.



# Оновимо макет лендінга, додавши navbar

```
<body>
```

```
<div class="container">
```

```
<nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
```

```
<div class="container">
```

```
<div class="navbar-header">
```

```
<button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1">
```

```
<span class="sr-only">Toggle navigation</span>
```

```
<span class="icon-bar"></span>
```

```
<span class="icon-bar"></span>
```

```
<span class="icon-bar"></span>
```

```
</button>
```

```
</div>
```

```
<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
```

```
<ul class="nav navbar-nav">
```

```
<li><a asp-controller="Home" asp-action="Index">Home</a></li>
```

```
<li><a asp-controller="Pie" asp-action="List">Pies</a></li>
```

```
</ul>
```

```
</div>
```

```
</div>
```

```
</nav>
```

```
<div class="row">
```

```
<div class="col-md-3">
```

```
<p class="lead">
```

```
<img class="img-responsive" alt="Bethany's Pie Shop'" data-bbox="158 781 358 803"/>  
src="~/Images/bethanylogo.png"/>
```

```
</p>
```

```
</div>
```

```
<div class="col-md-9">
```

```
@RenderBody()
```

```
</div>
```

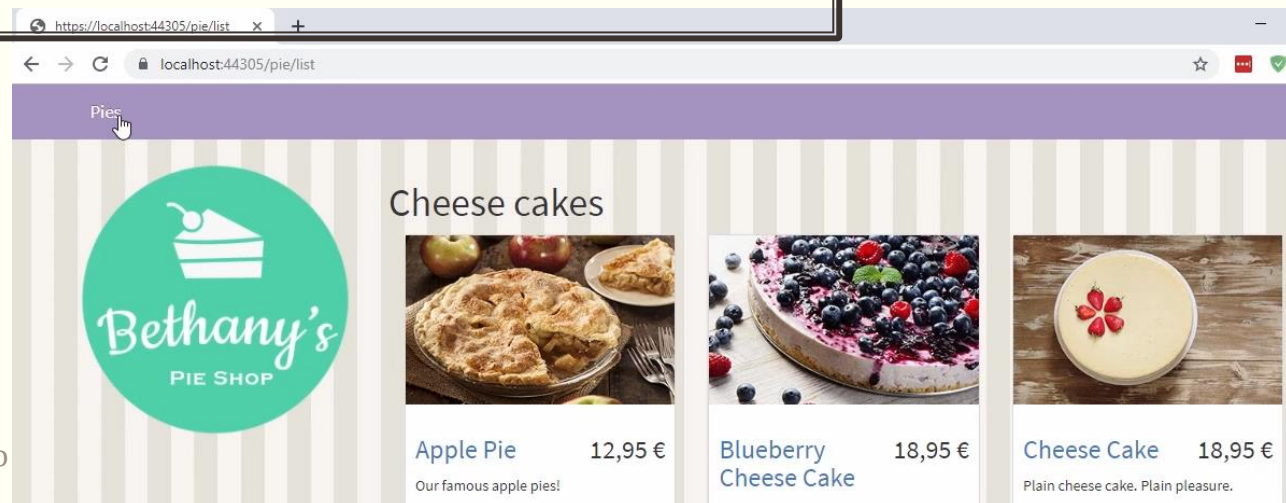
```
</div>
```

01.12.2020

@Марченко

```
</div>
```

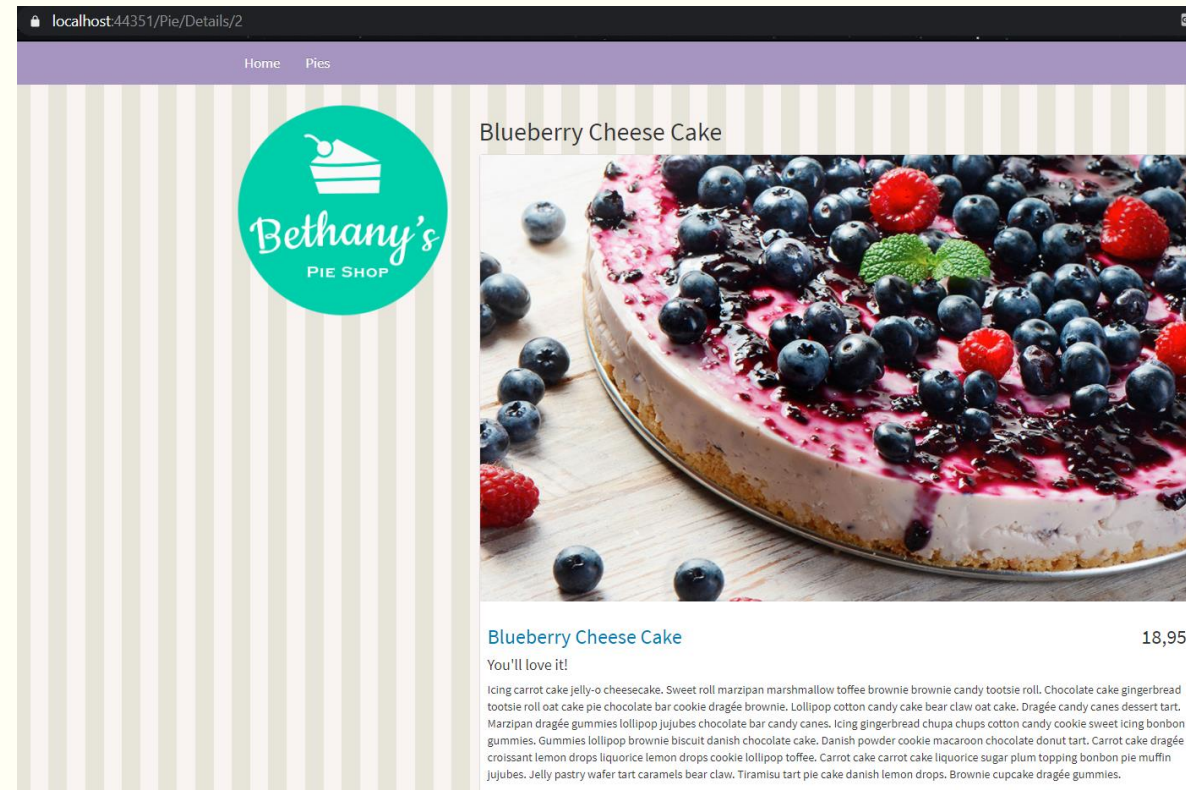
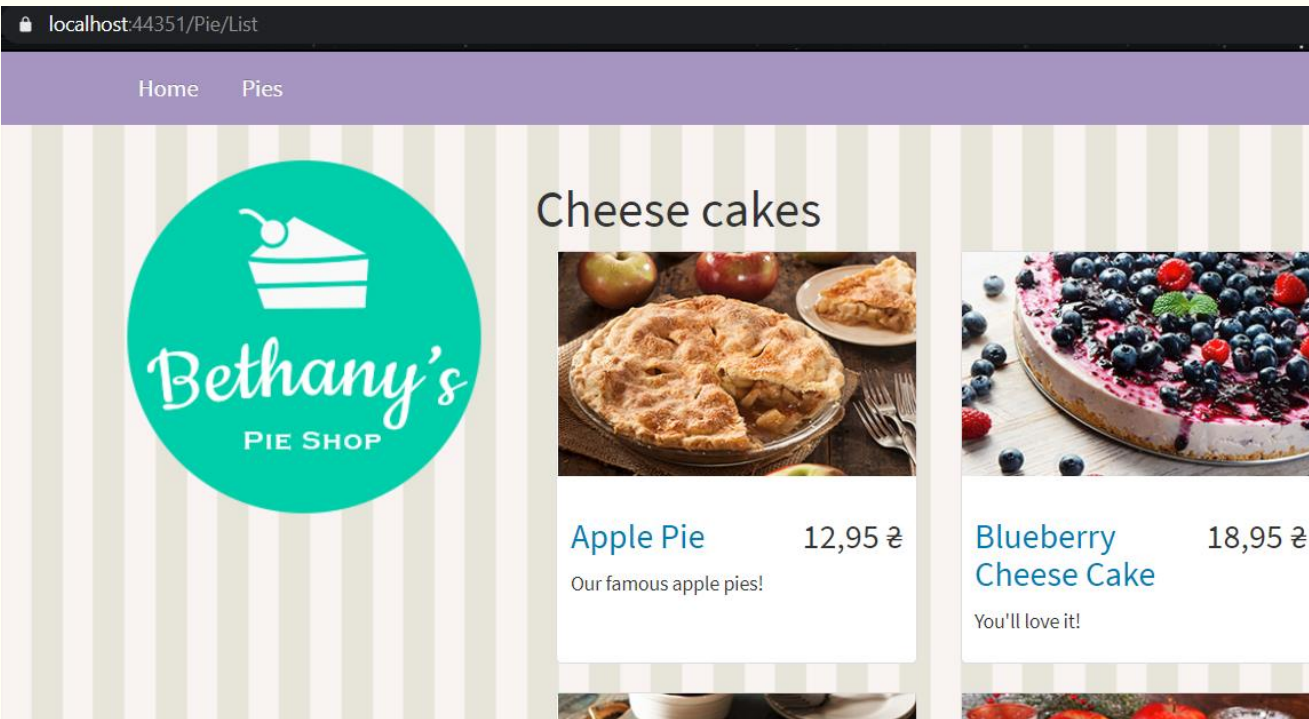
```
</body>
```





# Частинні представлення (Partial views)

- Зараз між двома представленнями на сайті є багато спільного:



- Спільні фрагменти коду можна об'єднати в частинні представлення.
  - За угодою, ASP.NET Core MVC надає всім частинним представленням назву, яка починається з символу «\_».
  - Також додамо окреме представлення для домашньої сторінки (Home), як на зображеннях

# Частинні представлення (Partial views)

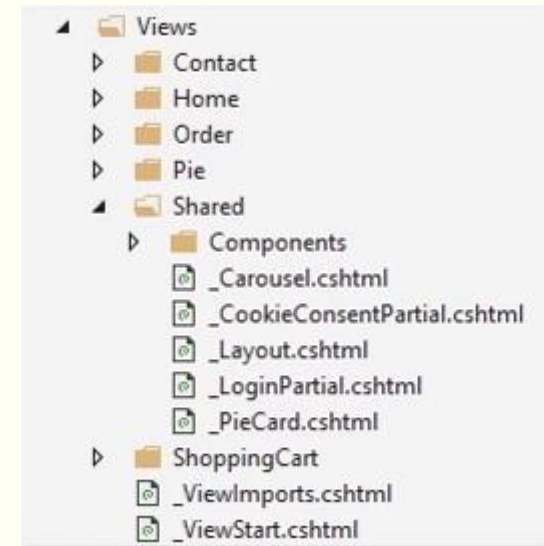
---

- Також використовується спеціальний тег-хелпер. Мінімальна форма запису:

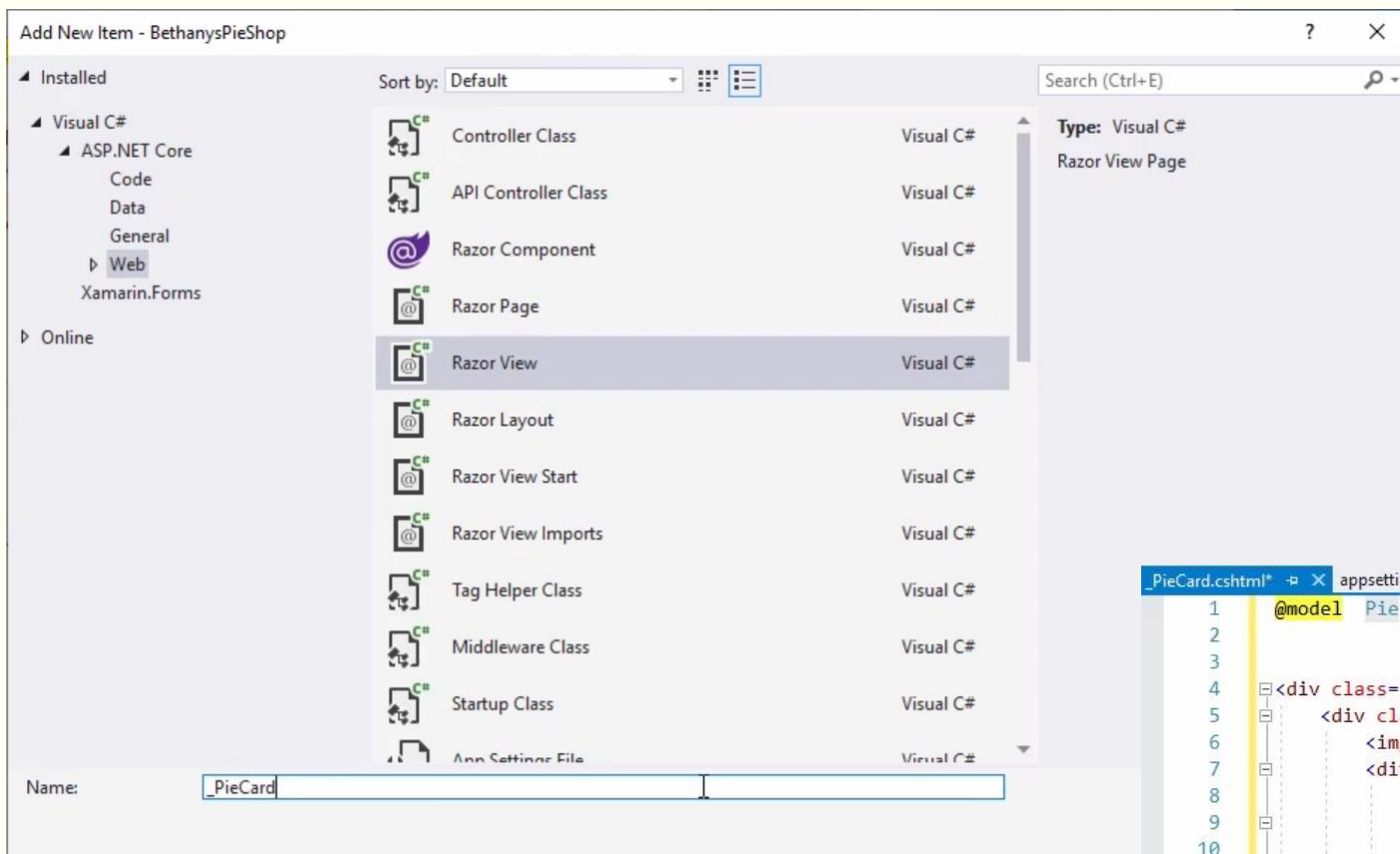
```
@foreach (var pie in Model.Pies)
{
    <partial name="_PieCard" model="pie" />
}
```

- Пошук частинних представлень здійснюється в поточній папці з контролерами та папці Shared.
- У кожній ітерації циклу foreach поточний об'єкт-торт ставиться у відповідність частинному представленню.

```
@model Pie
<div>
    <div class="thumbnail">
        
        <h3>@Model.Price.ToString("c")</h3>
    </div>
</div>
```



# Демонстрація створення частинних представлень



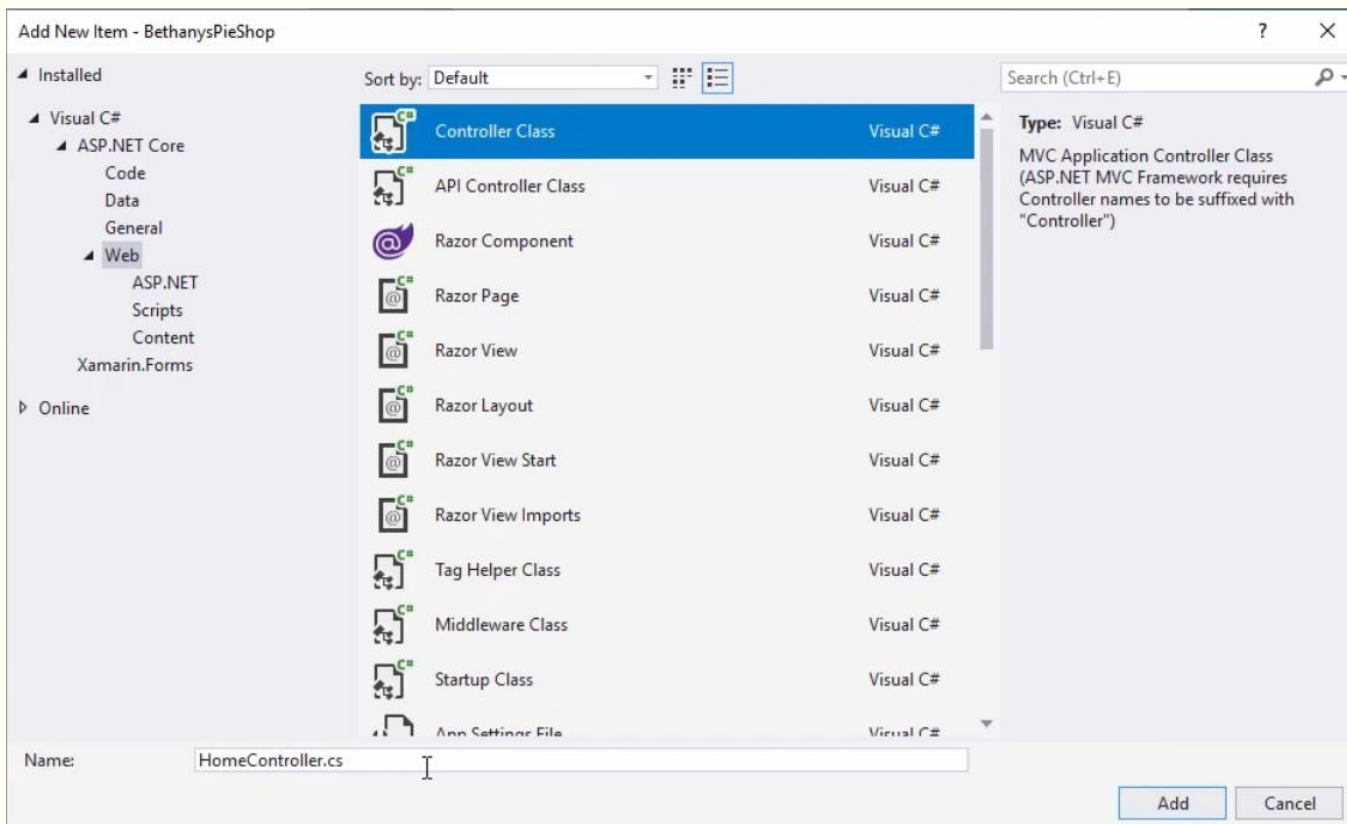
- Зазвичай файли з розміткою частинних представлень знаходяться в папці Shared.

- Спеціального шаблону для частинних представлень немає, оскільки за своєю суттю це Razor View.
- Додамо до проєкту частинне представлення \_PieCard.cshtml.
- Винесемо код з циклу foreach у List.cshtml в дане частинне представлення.

The screenshot shows the code for the \_PieCard.cshtml partial view. It starts with a model declaration `@model Pie`. The main content is a `<div>` with a class `col-sm-4 col-lg-4 col-md-4`. Inside, there's a `<div>` with class `thumbnail` containing an `<img>` tag with `src="@Model.ImageThumbnailUrl"` and an empty `alt` attribute. Below the image is a `<div>` with class `caption` containing an `<h3>` tag with class `pull-right` and text `@Model.Price.ToString("c")`. There's also an `<a>` tag with `asp-controller="Pie" asp-action="Details" asp-route-id="@Model.PieId">@Model.Name`. The `<div>` with class `caption` is closed, followed by the `<div>` with class `thumbnail`, and finally the main `<div>` is closed. The code ends with `</div>`.

# Додаємо домашню сторінку

- Створимо клас-контролер HomeController:



```
namespace BethanysPieShop.Controllers
{
    public class HomeController : Controller
    {
        private readonly IPieRepository _pieRepository;

        public HomeController(IPieRepository pieRepository)
        {
            _pieRepository = pieRepository;
        }

        public IActionResult Index()
        {
            var homeViewModel = new HomeViewModel
            {
                PiesOfTheWeek = _pieRepository.PiesOfTheWeek
            };

            return View(homeViewModel);
        }
    }
}
```

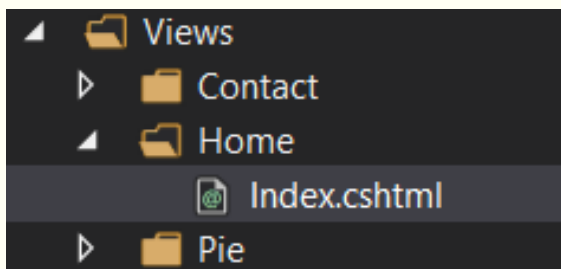
# Додаємо домашню сторінку

---

- Створимо клас HomeViewModel

```
public class HomeViewModel
{
    public IEnumerable<Pie> PiesOfTheWeek { get; set; }
}
```

- та представлення для домашньої сторінки



- Представлення містить частинне представлення \_Carousel, заготовку якого представлено на наступному слайді.

```
@model HomeViewModel
```

```
<partial name="_Carousel" />
```

```
<h2>Pies of the week</h2>
```

```
<h4>Our weekly selection - just for you!</h4>
```

```
<div class="row">
```

```
    @foreach (var pie in Model.PiesOfTheWeek)
```

```
    {
```

```
        <partial name="_PieCard" model="pie" />
```

```
    }
```

```
</div>
```



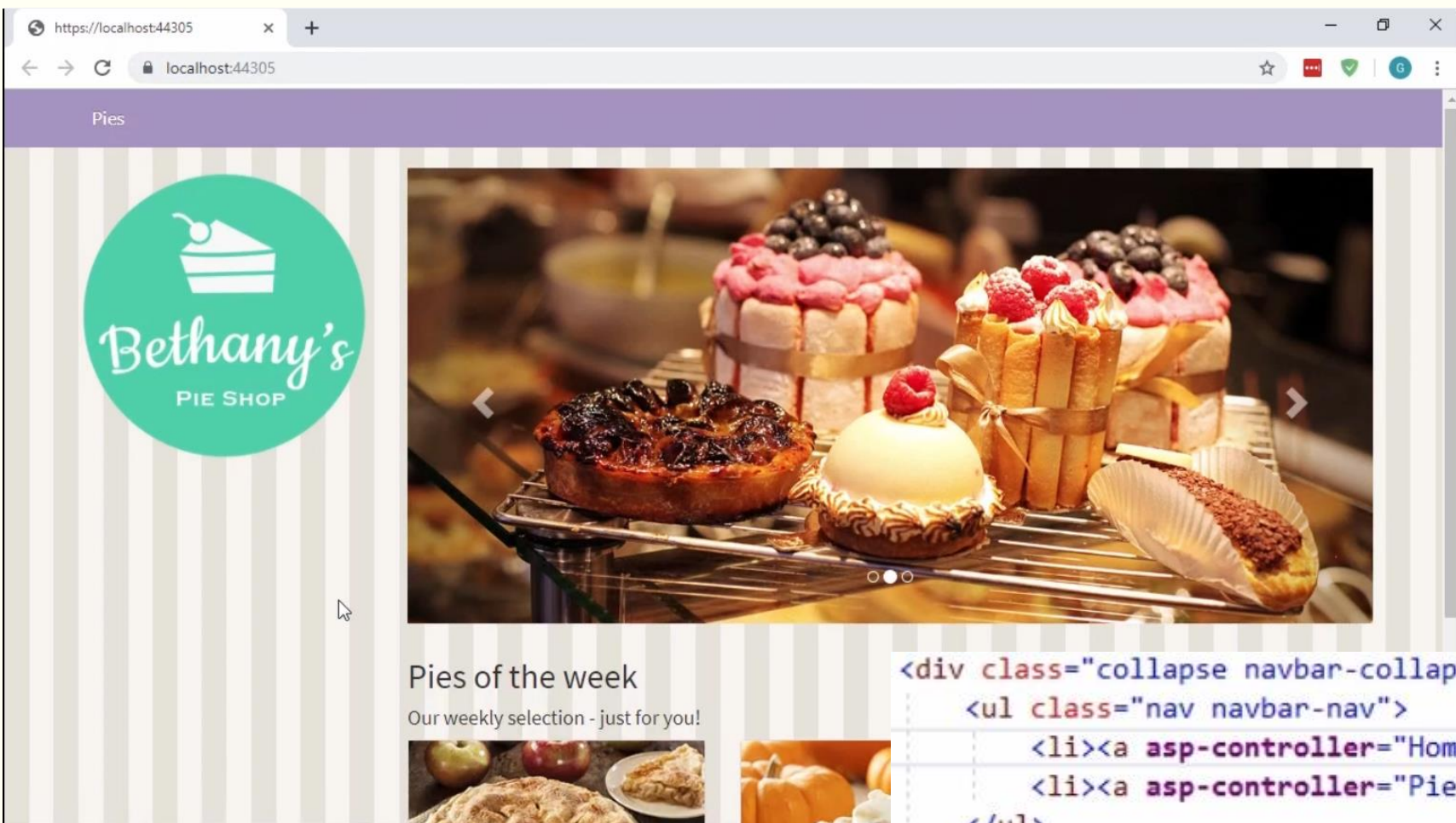
# Bootstrap-розмітка каруселі

---

```
<div class="row carousel-holder">
  <div class="col-md-12">
    <div id="carousel-example-generic" class="carousel slide" data-ride="carousel">
      <ol class="carousel-indicators">
        <li data-target="#carousel-example-generic" data-slide-to="0" class="active"></li>
        <li data-target="#carousel-example-generic" data-slide-to="1"></li>
        <li data-target="#carousel-example-generic" data-slide-to="2"></li>
      </ol>
      <div class="carousel-inner">
        <div class="item active">
          
        </div>
        <div class="item">
          
        </div>
        <div class="item">
          
        </div>
      </div>
      <a class="left carousel-control" href="#carousel-example-generic" data-slide="prev">
        <span class="glyphicon glyphicon-chevron-left"></span>
      </a>
      <a class="right carousel-control" href="#carousel-example-generic" data-slide="next">
        <span class="glyphicon glyphicon-chevron-right"></span>
      </a>
    </div>
  </div>
</div>
```

01.12.2020

# Демонстрація оновленого лендінгу



- На даний момент залишилось лише додати посилання на домашню сторінку в navbar:

```
<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">  
  <ul class="nav navbar-nav">  
    <li><a asp-controller="Home" asp-action="Index">Home</a></li>  
    <li><a asp-controller="Pie" asp-action="List">Pies</a></li>  
  </ul>  
</div>
```

# Сесії та розробка кошика для покупок

---

- Сесії – це фіча ASP.NET Core, яка дозволяє зберігати дані на сервері, поки користувач працює з додатком.
  - Сесія зберігатиме унікальний ідентифікатор картки для покупок та покладатиметься на cookie-файли, пов'язані з цим ідентифікатором.
- Підтримка сесій за умовчанням не ввімкнена. Для підключення потрібно викликати метод `AddSession()`.
  - Доступ до об'єкта-сесії можливий через так званий HTTP-контекст, що отримується з викликом методу `AddHttpContextAccessor()`.

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddHttpContextAccessor();
    services.AddSession();
}

public void Configure(IApplicationBuilder app,
IWebHostEnvironment env)
{
    app.UseSession();
}
```



# Створення корзини на сайті

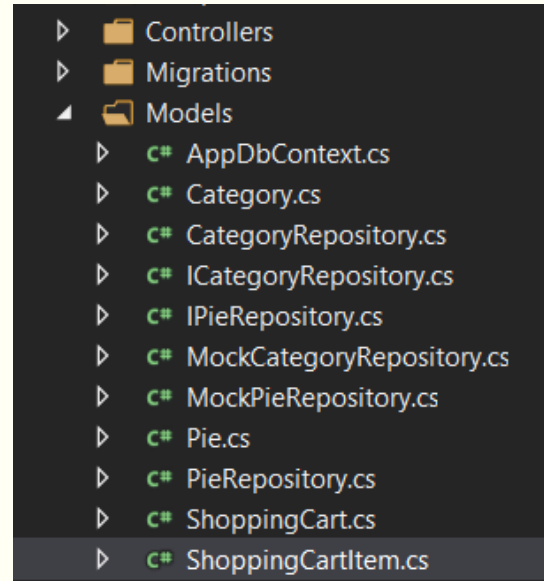
- Почнемо зі створення моделі даних для корзини.
  - Додамо клас ShoppingCartItem з описом одієї товарної позиції (торта) в корзині:

```
public class ShoppingCartItem
{
    public int ShoppingCartItemId { get; set; }
    public Pie Pie { get; set; }
    public int Amount { get; set; }
    public string ShoppingCartId { get; set; }
}
```

- Також доповнимо контекст БД новим DbSet-об'єктом:

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
    {
    }
    public DbSet<Category> Categories { get; set; }
    public DbSet<Pie> Pies { get; set; }

    public DbSet<ShoppingCartItem> ShoppingCartItems { get; set; }
}
```



# Створення корзини на сайті. Синхронізуємо нову схему БД

---

- Додаємо нову міграцію:

```
PM> add-migration ShoppingCartAdded
Microsoft.EntityFrameworkCore.Model.Validation[30000]
    No type was specified for the decimal column '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 using 'HasColumnType()'.
Microsoft.EntityFrameworkCore.Infrastructure[10403]
    Entity Framework Core 3.0.0-preview6.19304.10 initialized 'AppDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None
To undo this action, use Remove-Migration.
```

- Синхронізуємо схему за допомогою команди update-database.

# Клас для опису корзини (ShoppingCart.cs)

---

```
namespace BethanysPieShop.Models
{
    public class ShoppingCart
    {
        private readonly AppDbContext _appDbContext;

        public string ShoppingCartId { get; set; }

        public List<ShoppingCartItem> ShoppingCartItems { get; set; }

        private ShoppingCart(AppDbContext appDbContext)
        {
            _appDbContext = appDbContext;
        }

        public static ShoppingCart GetCart(IServiceProvider services)
        {
            ISession session = services.GetRequiredService<IHttpContextAccessor>()?
                .HttpContext.Session;

            var context = services.GetService<AppDbContext>();

            string cartId = session.GetString("CartId") ?? Guid.NewGuid().ToString();
            session.SetString("CartId", cartId);

            return new ShoppingCart(context) { ShoppingCartId = cartId };
        }
    }
}
```

01.12.2020

@Марченко С.В., ЧДБК, 2020

- Клас буде працювати з контекстом БД, тому в ньому присутні поле типу `AppDbContext` та конструктор з параметром цього ж типу.
  - Отримати об'єкт-корзину дозволяє метод `GetCart()`, якому передається `IServiceProvider`-об'єкт, що надає доступ до колекції сервісів, що керуються DI-контейнером.
  - Активна корзина буде представлятись в пам'яті за допомогою рядкового `Guid`-ідентифікатора, що зберігатиметься в сесії.
  - Наявність корзини перевірятиметься через наявність активної сесії та за її відсутності буде створюватись нова.
  - Через механізм кукісів ASP.NET Core зберігає інформацію щодо активної сесії користувача на сервері.

27

# Продовження класу (додавання та видалення товару)

---

```
public void AddToCart(Pie pie, int amount)
{
    var shoppingCartItem =
        _appDbContext.ShoppingCartItems.SingleOrDefault(
            s => s.Pie.PieId == pie.PieId
            && s.ShoppingCartId == ShoppingCartId);

    if (shoppingCartItem == null)
    {
        shoppingCartItem = new ShoppingCartItem
        {
            ShoppingCartId = ShoppingCartId,
            Pie = pie,
            Amount = 1
        };

        _appDbContext.ShoppingCartItems.Add(shoppingCartItem);
    }
    else
    {
        shoppingCartItem.Amount++;
    }
    _appDbContext.SaveChanges();
}
```

```
public int RemoveFromCart(Pie pie)
{
    var shoppingCartItem =
        _appDbContext.ShoppingCartItems.SingleOrDefault(
            s => s.Pie.PieId == pie.PieId
            && s.ShoppingCartId == ShoppingCartId);

    var localAmount = 0;

    if (shoppingCartItem != null)
    {
        if (shoppingCartItem.Amount > 1)
        {
            shoppingCartItem.Amount--;
            localAmount = shoppingCartItem.Amount;
        }
        else
        {
            _appDbContext.ShoppingCartItems.Remove(shoppingCartItem);
        }
    }

    _appDbContext.SaveChanges();

    return localAmount;
}
```

# Продовження класу (отримання переліку товарів, очистка корзини та обчислення загальної вартості)

---

```
public List<ShoppingCartItem> GetShoppingCartItems()
{
    return ShoppingCartItems ??
        (ShoppingCartItems =
            _appDbContext.ShoppingCartItems.Where(c => c.ShoppingCartId == ShoppingCartId)
                .Include(s => s.Pie)
                .ToList());
}

public void ClearCart()
{
    var cartItems = _appDbContext
        .ShoppingCartItems
        .Where(cart => cart.ShoppingCartId == ShoppingCartId);

    _appDbContext.ShoppingCartItems.RemoveRange(cartItems);

    _appDbContext.SaveChanges();
}

public decimal GetShoppingCartTotal()
{
    var total = _appDbContext.ShoppingCartItems.Where(c => c.ShoppingCartId == ShoppingCartId)
        .Select(c => c.Pie.Price * c.Amount).Sum();
    return total;
}
}
```

- Метод `GetShoppingCartItems()` перевірятиме, чи раніше товари в корзині вже переглядались та робитиме нову вибірку чи звертатиметься до вже існуючої.

# Додаткові налаштування конфігурації

---

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

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

    services.AddScoped<ShoppingCart>(sp => ShoppingCart.GetCart(sp));

    services.AddHttpContextAccessor();
    services.AddSession();

    services.AddControllersWithViews();//services.AddMvc(); також ще працює
}

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

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseSession();

    app.UseRouting();

    ...
}
```

- Будемо використовувати метод `GetCart()` при запиті на сайт.
  - Це дає можливість перевірити, чи вже є ID корзини в сесії та повернути корзину.
- Також додається підтримка `HttpContextAccessor` та сесій.
  - Для цього доведеться додавати також компоненти проміжного прошарку.
  - Дуже важливо, щоб метод `UseSession()` викликався перед методом `UseRouting()`, зберігаючи послідовність у конвеєрі запитів.

# Створюємо контролер для корзини

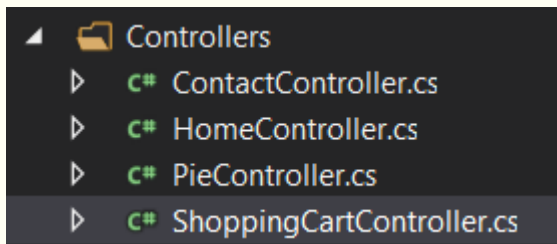
```
namespace BethanysPieShop.Controllers
{
    public class ShoppingCartController : Controller
    {
        private readonly IPieRepository _pieRepository;
        private readonly ShoppingCart _shoppingCart;

        public ShoppingCartController(IPieRepository pieRepository, ShoppingCart shoppingCart)
        {
            _pieRepository = pieRepository;
            _shoppingCart = shoppingCart;
        }

        public ViewResult Index()
        {
            var items = _shoppingCart.GetShoppingCartItems();
            _shoppingCart.ShoppingCartItems = items;

            var shoppingCartViewModel = new ShoppingCartViewModel
            {
                ShoppingCart = _shoppingCart,
                ShoppingCartTotal = _shoppingCart.GetShoppingCartTotal()
            };

            return View(shoppingCartViewModel);
        }
    }
}
```



```
public RedirectToActionResult AddToShoppingCart(int pieId)
{
    var selectedPie = _pieRepository.AllPies.FirstOrDefault(p => p.PieId == pieId);

    if (selectedPie != null)
    {
        _shoppingCart.AddToCart(selectedPie, 1);
    }
    return RedirectToAction("Index");
}

public RedirectToActionResult RemoveFromShoppingCart(int pieId)
{
    var selectedPie = _pieRepository.AllPies.FirstOrDefault(p => p.PieId == pieId);

    if (selectedPie != null)
    {
        _shoppingCart.RemoveFromCart(selectedPie);
    }
    return RedirectToAction("Index");
}
}
```

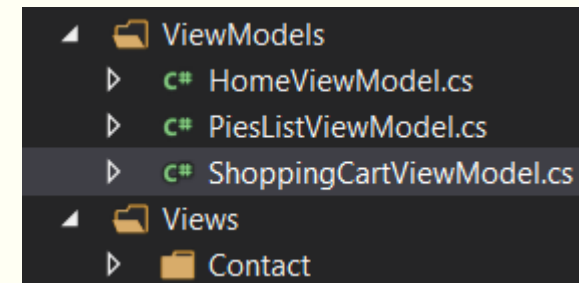
- ShoppinCart-об'єкт буде створений на scoped-рівні при конфігурації. Action-методи:
  - Index() – повертає представлення для корзини: отримуємо товари з корзини та звертаємось до відповідної в'юмоделі, яку ще потрібно створити.
  - AddToShoppingCart(), RemoveFromShoppingCart() повертають об'єкти типу RedirectToActionResult, які перенаправлятимуть користувача до action-методу Index().

# Доповнюємо проєкт в'юмоделлю

---

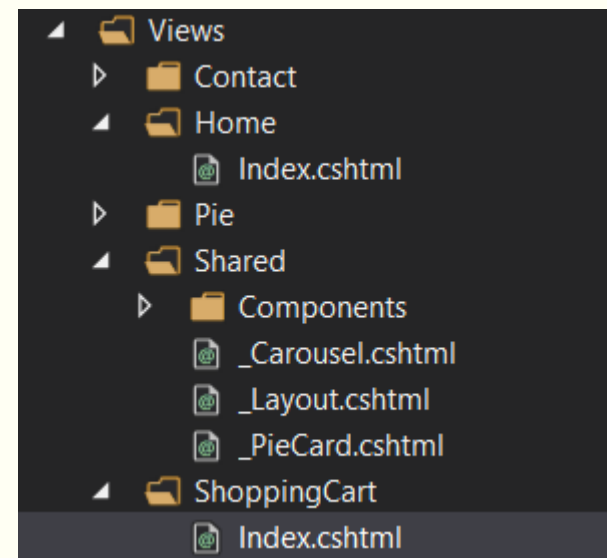
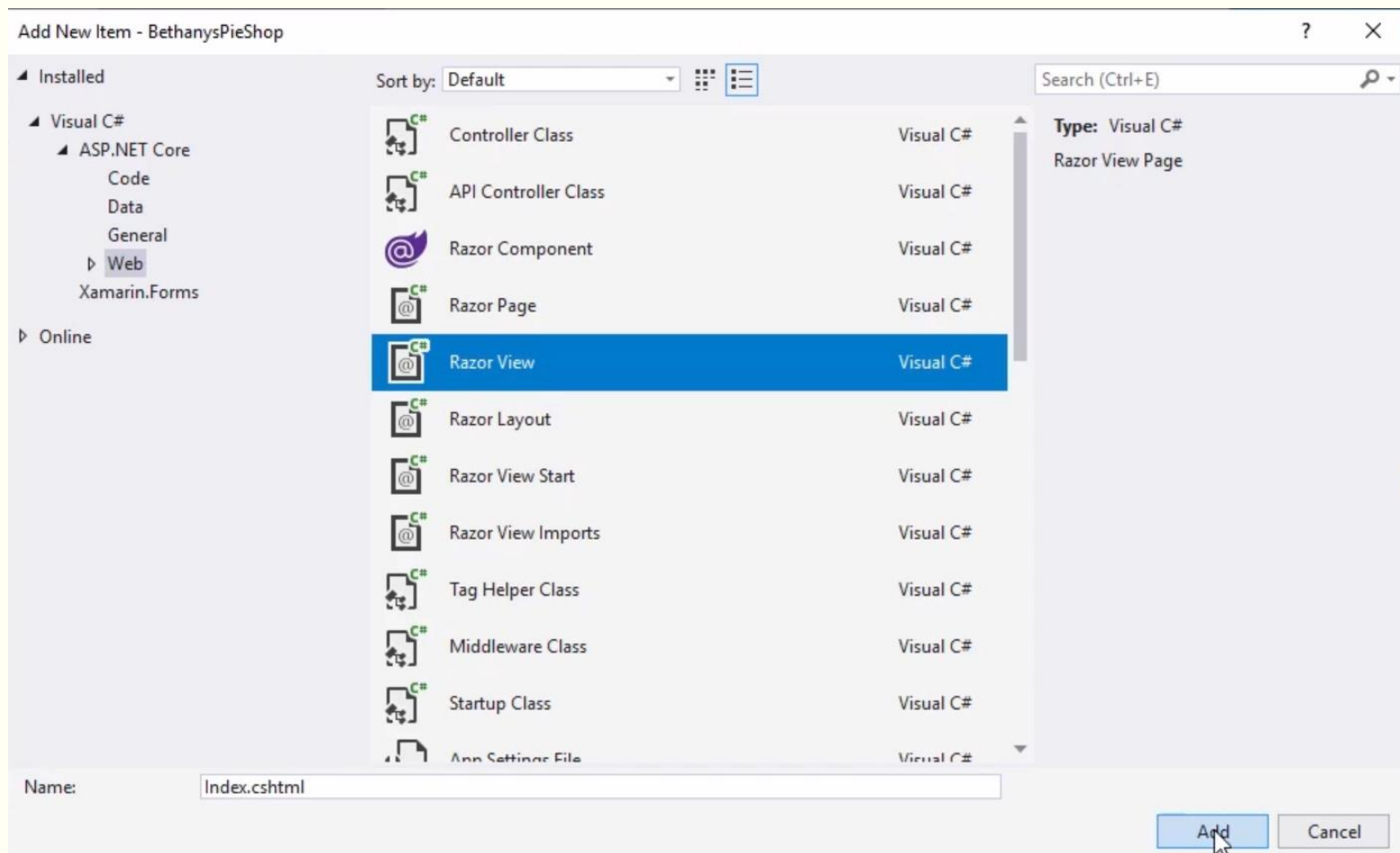
- Доступ на рівні в'юмоделі потрібний до корзини (ShoppingCart) та загальної вартості товарів (ShoppingCartTotal):

```
namespace BethanysPieShop.ViewModels
{
    public class ShoppingCartViewModel
    {
        public ShoppingCart ShoppingCart { get; set; }
        public decimal ShoppingCartTotal { get; set; }
    }
}
```





# Створюємо представлення для корзини



```
@model ShoppingCartViewModel
```

```
<h2>Your shopping cart</h2>
```

```
<h4>Here are the delicious pies in your shopping cart.</h4>
```

```
<table class="table table-bordered table-striped">
```

```
  <thead>
```

```
    <tr>
```

```
      <th>Selected amount</th>
```

```
      <th>Pie</th>
```

```
      <th class="text-right">Price</th>
```

```
      <th class="text-right">Subtotal</th>
```

```
    </tr>
```

```
  </thead>
```

```
  <tbody>
```

```
    @foreach (var line in Model.ShoppingCart.ShoppingCartItems)
```

```
    {
```

```
      <tr>
```

```
        <td class="text-center">@line.Amount</td>
```

```
        <td class="text-left">@line.Pie.Name</td>
```

```
        <td class="text-right">@line.Pie.Price.ToString("c")</td>
```

```
        <td class="text-right">
```

```
          @((line.Amount * line.Pie.Price).ToString("c"))
```

```
        </td>
```

```
      </tr>
```

```
    }
```

```
  </tbody>
```

```
  <tfoot>
```

```
    <tr>
```

```
      <td colspan="3" class="text-right">Total:</td>
```

```
      <td class="text-right">
```

```
        @Model.ShoppingCartTotal.ToString("c")
```

```
      </td>
```

```
    </tr>
```

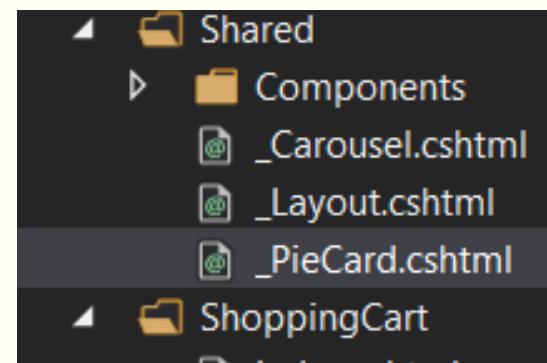
```
  </tfoot>
```

```
</table>
```

01.12.2020

## Створюємо представлення для корзини

- Розмітка передбачає виведення в циклі foreach усіх товарів у корзині, а також загальної їх вартості внизу.
  - Наступним кроком буде створення можливості додавати товари в корзину.
  - Будемо вносити зміни в частинне представлення \_PieCard.cshtml.



# Частинне представлення \_PieCard.cshtml

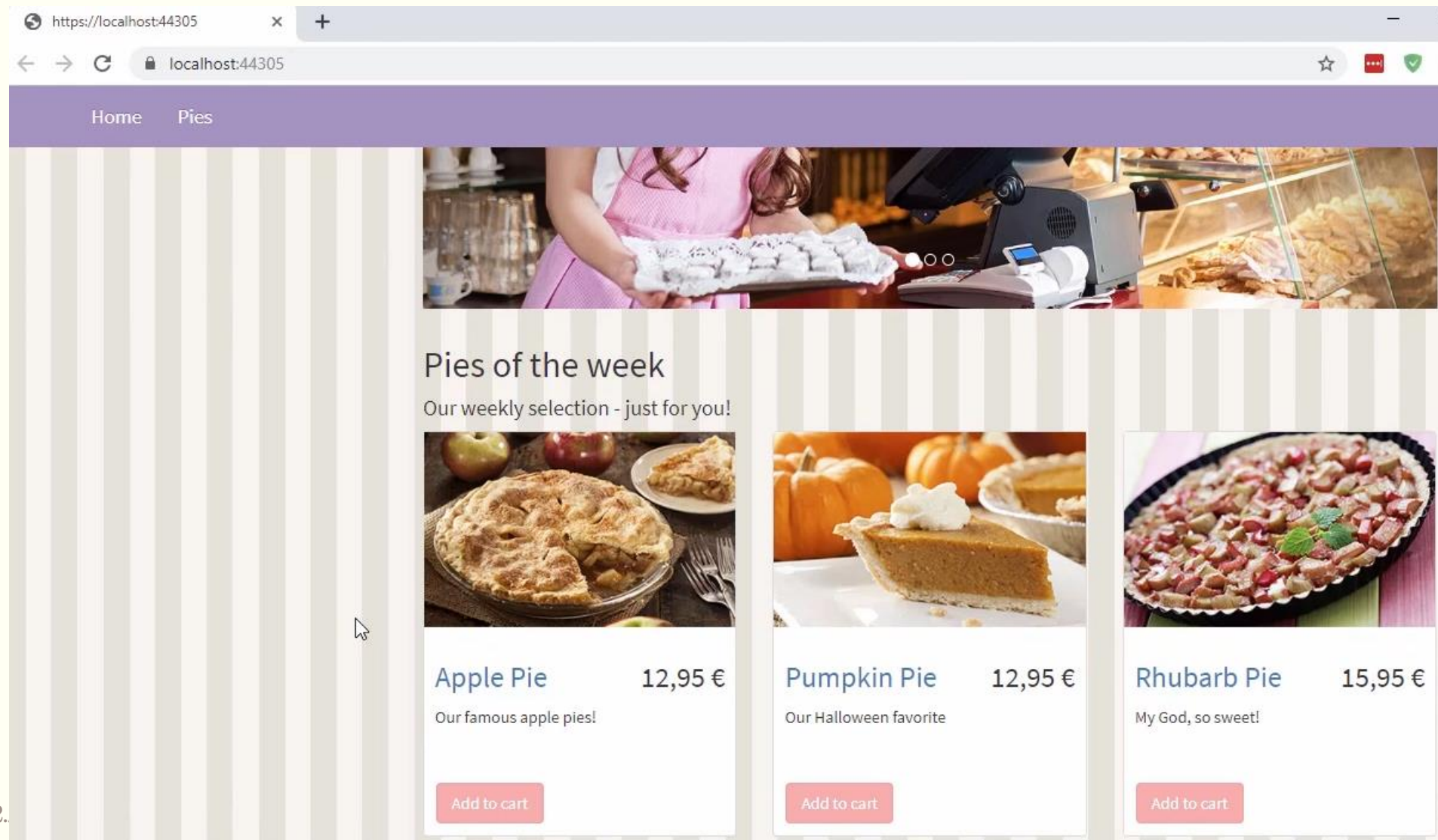
---

```
@model Pie
<div class="col-sm-4 col-lg-4 col-md-4">
  <div class="thumbnail">
    
    <div class="caption">
      <h3 class="pull-right">@Model.Price.ToString("c")</h3>
      <h3>
        <a asp-controller="Pie"
           asp-action="Details"
           asp-route-id="@Model.PieId">@Model.Name</a>
      </h3>
      <p>@Model.ShortDescription</p>
    </div>
    <div class="addToCart">
      <p class="button">
        <a class="btn btn-primary"
           asp-controller="ShoppingCart"
           asp-action="AddToShoppingCart"
           asp-route-pieId="@Model.PieId">Add to cart</a>
      </p>
    </div>
  </div>
</div>
```

- За допомогою Bootstrap введемо кнопку «Add to cart», яка додавати торт за його ID.
  - Викликається метод AddToShoppingCart з рівня контролера, який і приймає цей route-параметр.
- Аналогічний код додаємо і на представлення Details.cshtml:

```
<div class="addToCart">
  <p class="button">
    <a class="btn btn-primary" asp-controller="ShoppingCart"
       asp-action="AddToShoppingCart"
       asp-route-pieId="@Model.PieId">Add to cart</a>
  </p>
</div>
```

# Вигляд додатку на даний момент



# View-компоненти (View Components)

---

- Раніше розглянуті частинні представлення мають деяке обмеження – потребують даних ззовні (з в'юмоделі викликаючого представлення):

```
@model Pie
<div>
  <div>
    
    <h3>@Model.Price.ToString("c")</h3>
    <p>@Model.ShortDescription</p>
```

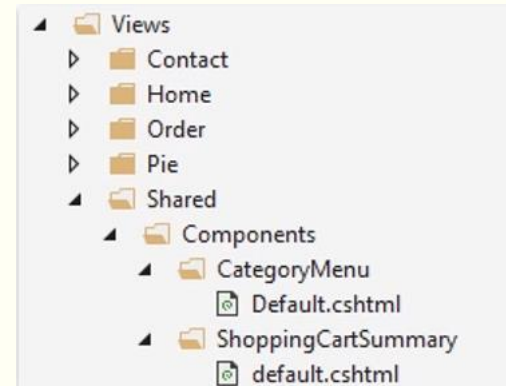
- Нова фіча ASP.NET Core – View-компоненти – багато в чому подібна до частинних представлень.
  - Якщо частинні компоненти застосовуються для спільного відображення частини представлення, то View-компоненти – для всього представлення.
  - View-компонент складається з класу та представлення, подібно до контролера з представленням.
  - Дані для View-компонента містяться в його C#-класі, також підтримується впровадження залежностей.
  - View-компонент не є абсолютно незалежним, а виявляється дочірнім для батьківського представлення.
  - Основна відмінність – здатність View-компонентом виконувати код.

# View-компоненти (View Components)

---

- Корисні при розробці standalone-компонентів сайту:
  - Панель авторизації (Login panel)
  - Меню динамічної навігації
  - Корзина
- Існують різні способи створення View-компонента.
  - Один з таких: успадкуватись від класу `ViewComponent`.
  - Новоутворений клас повинен бути відкритим, неабстрактним та невикладеним класом, аналогічно до контролера.
  - Потрібна реалізація включається в метод `Invoke()`
  - Представлення для View-компонента не є обов'язковим (можна повертати `string`), проте якщо воно є, то зазвичай розміщується в папці `Components` та називається `default.cshtml`:

```
public class ShoppingCartSummary : ViewComponent
{
    public IActionResult Invoke()
    {
        return View(model);
    }
}
```

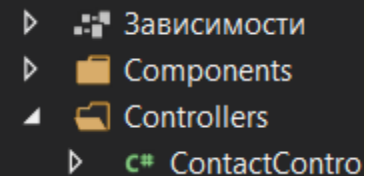


# Використання View-компонента

---

```
@await Component.InvokeAsync("ShoppingCartSummary")
```

- У якості демонстрації додамо 2 View-компонента:
  - Створимо представлення для навігації
  - Створимо View-компонент для корзини
- Зокрема, при роботі з корзиною відбувається багато звернень до БД, які можна мінімізувати, використовуючи View-компоненти.
  - Додамо папку Components до проєкту:





# Формування класу ShoppingCartSummary

---

```
namespace BethanysPieShop.Components
{
    public class ShoppingCartSummary: ViewComponent
    {
        private readonly ShoppingCart _shoppingCart;

        public ShoppingCartSummary(ShoppingCart shoppingCart)
        {
            _shoppingCart = shoppingCart;
        }

        public IViewComponentResult Invoke()
        {
            var items = _shoppingCart.GetShoppingCartItems();
            _shoppingCart.ShoppingCartItems = items;

            var shoppingCartViewModel = new ShoppingCartViewModel
            {
                ShoppingCart = _shoppingCart,
                ShoppingCartTotal = _shoppingCart.GetShoppingCartTotal()
            };
            return View(shoppingCartViewModel);
        }
    }
}
```

- Робота буде вестись на основі даних раніше створеного об'єкту типу ShoppingCart.
  - Застосуємо ін'єкцію конструктора.
  - Функціональність описується в методі Invoke() / InvokeAsync() без заміщення.
  - Дії дуже схожі на роботу контролера, проте метод повертатиме об'єкт типу IViewComponentResult (представлення).
  - Наступний крок – створити представлення.



# Створюємо представлення View-компонента

- Створимо в папці Shared ще одну папку Components, в якій буде папка з назвою компонента (тут – ShoppingCartSummary).

- Всередині буде представлення, описане в файлі Default.cshtml (Razor View)

```
@model ShoppingCartViewModel
```

```
@if (Model.ShoppingCart.ShoppingCartItems.Count > 0)
```

```
{
```

```
    <li>
```

```
        <a>
```

```
            <span class="glyphicon glyphicon-shopping-cart"></span>
```

```
            <span id="cart-status">
```

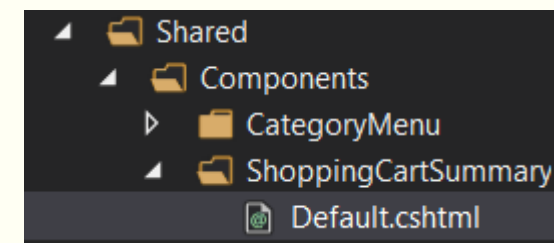
```
                @Model.ShoppingCart.ShoppingCartItems.Count
```

```
            </span>
```

```
        </a>
```

```
    </li>
```

```
}
```



- Оновимо макет у файлі \_Layout.cshtml:


```
<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
  <ul class="nav navbar-nav">
    <li><a asp-controller="Home" asp-action="Index">Home</a></li>
    <li><a asp-controller="Pie" asp-action="List">Pies</a></li>
    @await Component.InvokeAsync("ShoppingCartSummary")
  </ul>
</div>
```

# Демонстрація роботи оновленої версії додатку

https://localhost:44305/Shopping x +

localhost:44305/ShoppingCart

Home Pies 1



## Your shopping cart

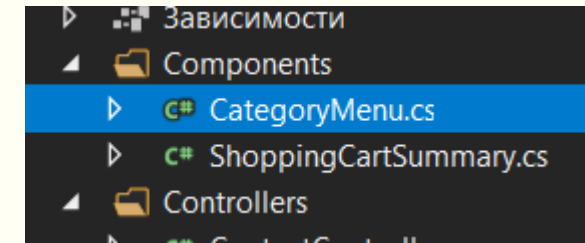
Here are the delicious pies in your shopping cart.

Selected amount	Pie	Price	Subtotal
1	Apple Pie	12,95 €	12,95 €
Total:			12,95 €

# Аналогічно додаємо View-компонент для навігації за категоріями тортів

```
namespace BethanysPieShop.Components
{
    public class CategoryMenu : ViewComponent
    {
        private readonly ICategoryRepository _categoryRepository;
        public CategoryMenu(ICategoryRepository categoryRepository)
        {
            _categoryRepository = categoryRepository;
        }

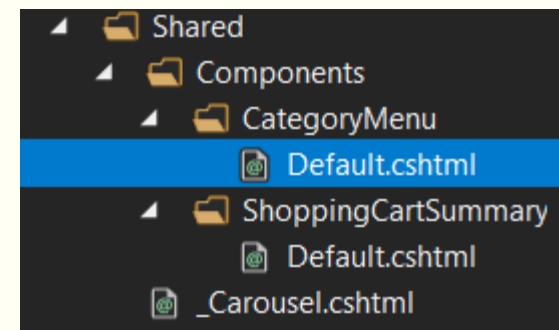
        public IViewComponentResult Invoke()
        {
            var categories = _categoryRepository.AllCategories.OrderBy(c => c.CategoryName);
            return View(categories);
        }
    }
}
```



# Представления для View-компонента

```
@model IEnumerable<Category>

<li class="dropdown">
  <a asp-controller="Pie"
    asp-action="Index"
    class="dropdown-toggle" data-toggle="dropdown">Pies<b class="caret"></b></a>
  <ul class="dropdown-menu">
    @foreach (var category in Model)
    {
      <li>
        <a asp-controller="Pie" asp-action="List"
          asp-route-category="@category.CategoryName">@category.CategoryName</a>
      </li>
    }
    <li class="divider"></li>
    <li>
      <a asp-controller="Pie" asp-action="List" asp-route-Category="">View all pies</a>
    </li>
  </ul>
</li>
```




# Оновлення контролера PieController для формування списку категорій

## Стара реалізація методу List

```
// GET: /<controller>/  
//public IActionResult List()  
//{  
//    //ViewBag.CurrentCategory = "Cheese cakes";  
  
//    //return View(_pieRepository.AllPies);  
//    PiesListViewModel piesListViewModel = new PiesListViewModel();  
//    piesListViewModel.Pies = _pieRepository.AllPies;  
  
//    piesListViewModel.CurrentCategory = "Cheese cakes";  
//    return View(piesListViewModel);  
//}
```

- Також підключаємо сформований View-компонент:

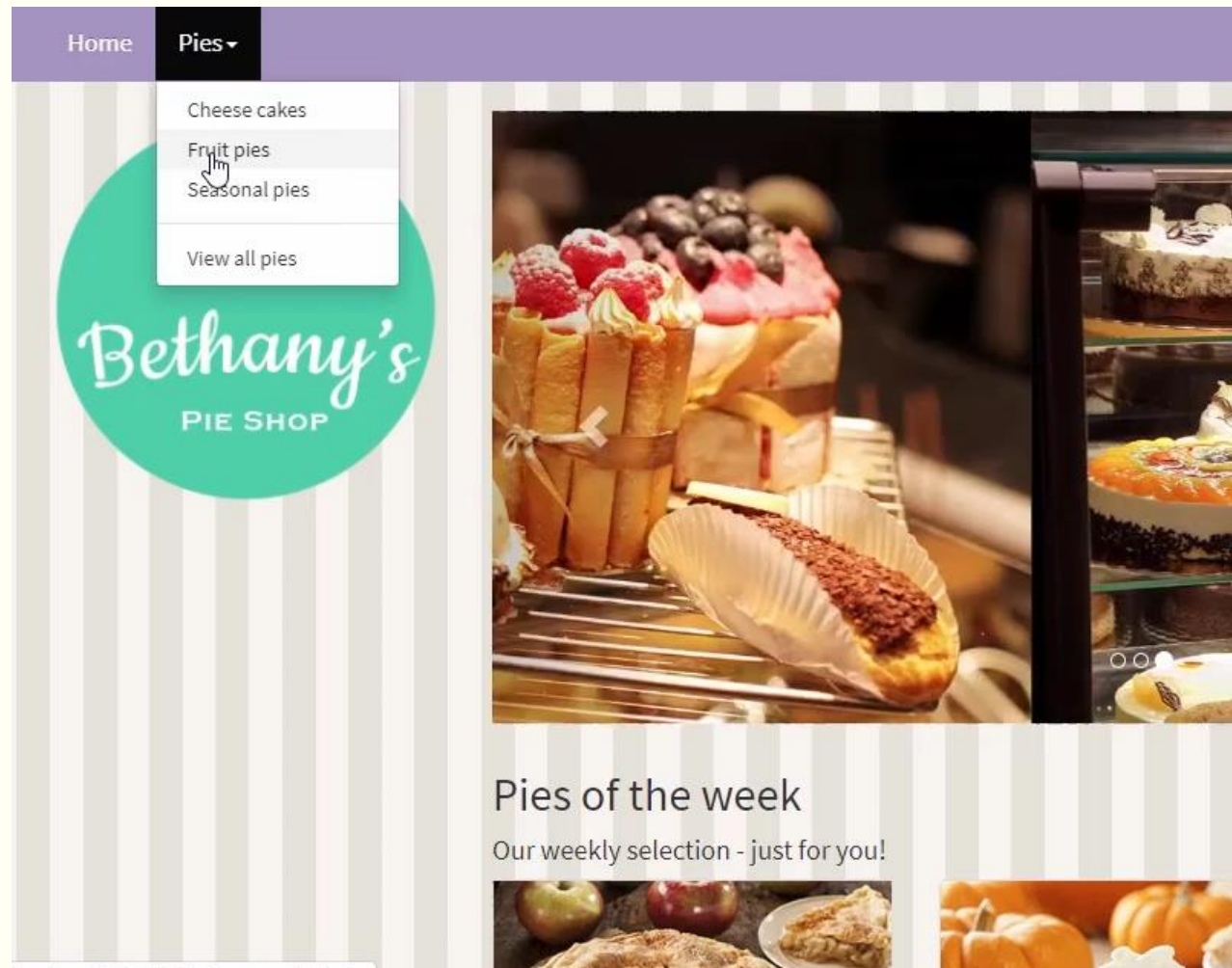


```
<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">  
  <ul class="nav navbar-nav">  
    <li><a asp-controller="Home" asp-action="Index">Home</a></li>  
    @await Component.InvokeAsync("CategoryMenu")  
    @await Component.InvokeAsync("ShoppingCartSummary")  
  </ul>  
</div>
```

## Нова реалізація методу List

```
public IActionResult List(string category)  
{  
    IEnumerable<Pie> pies;  
    string currentCategory;  
  
    if (string.IsNullOrEmpty(category))  
    {  
        pies = _pieRepository.AllPies.OrderBy(p => p.PieId);  
        currentCategory = "All pies";  
    }  
    else  
    {  
        pies = _pieRepository.AllPies.Where(p => p.Category.CategoryName == category)  
            .OrderBy(p => p.PieId);  
        currentCategory = _categoryRepository.AllCategories.FirstOrDefault(c => c.CategoryName == category)?.CategoryName;  
    }  
  
    return View(new PiesListViewModel  
    {  
        Pies = pies,  
        CurrentCategory = currentCategory  
    });  
}
```

# Демонстрація поточного вигляду сторінки



# Створення власних тег-хелперів

---

- Тег-хелпери містять всередині C#-код, який генеруватиме HTML-розмітку.
  - Якщо є невеликий спільний компонент для багатьох сторінок, доречним буде використання тег-хелперів.
- Власний тег-хелпер створюється на базі класу TagHelper.
  - Основний метод для реалізації – Process() – на цей раз заміщується.

```
public class EmailTagHelper: TagHelper
{
    public override void Process(
        TagHelperContext context, TagHelperOutput output)
    {
        ...
    }
}
```

- Застосування створеного тег-хелпера

```
<email
    address="info@@bethanyspieshop.com"
    content="Contact us">
</email>
```



# Створення власних тег-хелперів

---

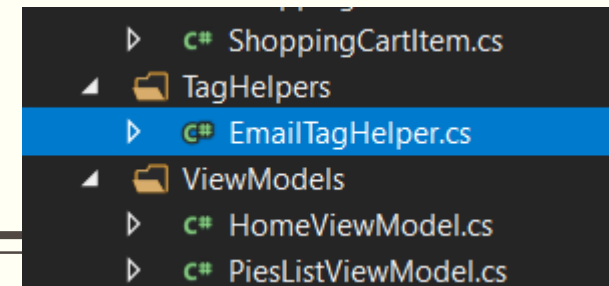
- Перед використанням те-хелперів їх потрібно зареєструвати, тому краще збирати їх усі в одному місці за допомогою файлів імпорту:

```
@using BethanysPieShop.Models
@using BethanysPieShop.ViewModels
@addTagHelper BethanysPieShop.TagHelpers.*, BethanysPieShop
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

- Продемонструємо розробку власних тег-хелперів на прикладі створення контактної сторінки.
  - Тег-хелпер прийматиме e-mail та текст для відображення, що буде надсилатись за цією поштою.
  - Створюємо заготовки контролера для контактів

```
namespace BethanysPieShop.Controllers
{
    References
    public class ContactController : Controller
    {
        // GET: /<controller>/
        References
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

# Створимо окрему папку TagHelpers



```
namespace BethanysPieShop.TagHelpers
{
    public class EmailTagHelper : TagHelper
    {
        public string Address { get; set; }
        public string Content { get; set; }

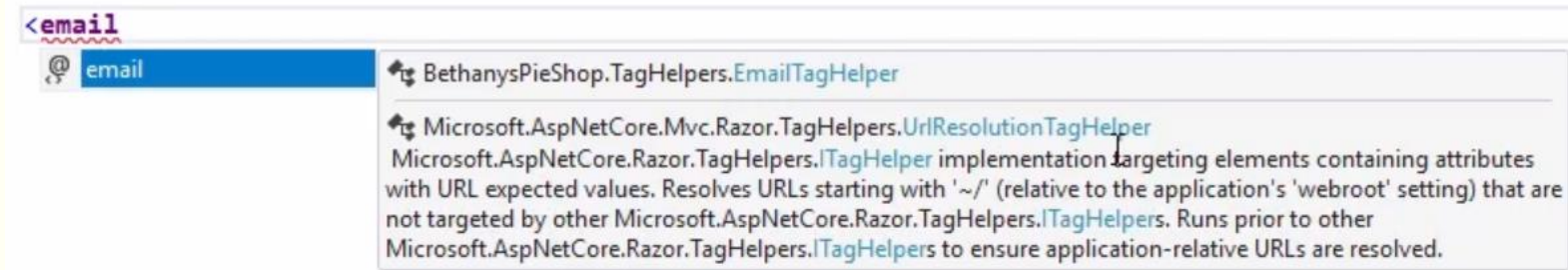
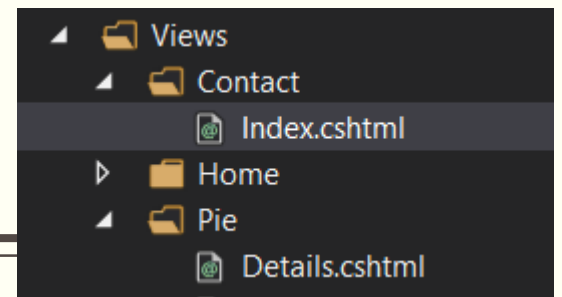
        public override void Process(TagHelperContext context, TagHelperOutput output)
        {
            output.TagName = "a";

            output.Attributes.SetAttribute("href", "mailto:" + Address);
            output.Content.SetContent(Content);
        }
    }
}
```

- У даному випадку використовуємо тільки output – генеруватимемо HTML-розмітку для місця використання тег-хелпера.
  - Для сповіщення додатка про появу нового тег-хелпера оновлюємо імпорти (\_ViewImports.cshtml):

```
1 @using BethanysPieShop.ViewModels
2 @using BethanysPieShop.Models
3 @addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
4 @addTagHelper BethanysPieShop.TagHelpers.*, BethanysPieShop
```

# Формуємо представлення для контактів



<h3>

Please contact us by sending an email using the button below

</h3>

<email address="info@@bethanyspieshop.com" content="Contact us"></email>

