Лабораторная работа №7

Razor Pages. Сессии. Middleware

1. Цель работы.

Знакомство со страницами Razor.

Изучение способов сохранения объектов в сессии.

Создание и регистрация своих сервисов.

Знакомство с механизмом логирования ASP.NET Core.

Знакомство с конвейером Middleware.

Создание своего Middleware

2. Общие сведения.

3. Выполнение работы

3.1. Исходные данные

Используйте проект из лабораторной работы №6.

3.2. Задание №1

При клике на меню «Администрирование» должен отобразиться список всех объектов с функциями добавления, редактирования и удаления. Раздел администрирования оформить в виде страниц Razor

Реализовать функции добавления, удаления и редактирования объектов.

При сохранении файла изображения в папку Images в качестве имени файла использовать id объекта.

При обращении к страницам Edit (редактирование) и Delete (удаление) id должен передаваться в качестве сегмента маршрута, а не в строке запроса, например: Admin/Edit/2

При удалении объекта предусмотреть удаление файла изображения.

3.2.1. Рекомендации к заданию №1

Воспользуйтесь автоматтическим генерированием страниц (Scaffold)

Страницы администрирования поместите в область (Area) «Admin» (см. параметры запроса, указанные в компоненте MenuViewComponent)

Для создания страниц воспользуйтесь Scaffold (автоматическим генерированием страниц) – выпадающее меню по клику правой клавиши мыши «Add->New Scaffolded Item»

(Необязательно) Для оформления выбора файла изображения используйте классы Bootstrap (https://getbootstrap.com/docs/4.3/components/forms/#file-browser). ***Это уже делалось на странице регистрации пользователя.*** При использовании класса bootstrap «custom-file» используется дополнительный скрипт «bs-custom-file-input.js».

Скачайте файл с сайта https://www.npmjs.com/package/bs-custom-file-input, включите его в проект, в папку scripts и подключите скрипт на страницах Create и Edit. Воспользуйтесь секцией Scripts страницы макета. Для активирования скрипта там же пропишите команду `bsCustomFileInput.init();`

3.3. Задание №2

Реализуйте функции работы с корзиной заказов.

При клике на кнопку «В корзину» выбранный объект добавляется в корзину заказов. ***При этом в меню пользователя должна правильно отображаться информация о корзине заказов***.

Добавление в корзину должно осуществляться только для пользователя, вошедшего в систему.

При клике на корзину в меню пользователя должен отобразиться список объектов в заказе с возможностью удаления объектов из заказа.

3.3.1. Рекомендации к заданию №2

Создайте контроллер Cart для работы с корзиной заказов.

Создайте класс, описывающий один элемент корзины со свойствами: объект (в примерах это Dish) и количество.

Опишите класс корзины заказов. Класс должен содержать список элементов корзины, предоставлять данные о количестве объектов в корзине и суммарную величину количественной характеристики объектов в корзине (в примерах – это сумма калорий), а также реализовывать добавление, удаление в корзину и очистку корзины.

Корзину заказов сохраняйте в сессии.

Для сохранения объектов в сессии создайте расширяющие методы, как описано в https://docs.microsoft.com/en-us/aspnet/core/fundamentals/app-state?view=aspnetcore-2.2#session-state.

3.4. Задание №3

Создайте сервис, который получает корзину заказа из сессии и сохраняет корзину в сессии. С помощью созданного сервиса реализуйте механизм внедрения зависимости для получения объекта корзины в контроллере Cart.

3.4.1. Рекомендации к заданию №3

Создайте сервис, который наследуется от класса Cart. Сделайте методы AddToCart, RemoveFromCart и ClearAll в классе Cart виртуальными. В сервисе переопределите эти методы с сохранением изменений в сессии.

Объект сессии храните в сервисе.

Для получения объекта HttpContext.Session понадобится объект HttpContextAccessor. Для возможности внедрения (Dependency Injection) в классе Startup, в методе ConfigureServices добавьте:

```
services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
```

В этом случае объект Session можно буде получить так:

```
provider.GetRequiredService<IHttpContextAccessor>()?
        .HttpContext
        .Session
```

где provider – объект IServiceProvider

Сделайте статический метод `GetCart(IServiceProvider provider)`, который вернет сервис в виде объекта Cart.

Добавьте сервис (Scoped service) в методе ConfigureServices класса Startup.

Внедрите объект Cart в конструктор контроллера.

Поскольку сервис сам сохраняет в/читает из сессии, то в контроллере уже не нужно будет выполнять эти действия.

3.5. Задание №4

Самостоятельно изучите систему логирования в ASP.Net Core

https://docs.microsoft.com/en-us/aspnet/core/fundamentals/logging/?view=aspnetcore-3.1

Самостоятельно изучите использование провайдера логирования serilog в ASP.Net для записи логов в файл

https://www.tutorialsteacher.com/core/aspnet-core-logging

Подключите serilog в проект.

Логер **не** должен записывать сообщения категории Microsoft.

Проверьте работу логера в методе Index контроллера Product: записывайте в файл информацию о переданных значениях group и page. После проверки уберите логирование в контроллере Product.

3.5.1. Рекомендации к заданию №4

Загрузите в проект NuGet пакет «serilog.extensions.logging.file»

Отменить логирование можно в классе Program:

```
public static IHostBuilder CreateHostBuilder(string[] args)
        {
            return Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                })
                .ConfigureLogging(lp =>
                {
                    lp.ClearProviders();
                    lp.AddFilter("Microsoft", LogLevel.None);
                });
}
```

Для проверки работы логера внедрите через конструктор контроллера объект ILogger.

3.6. Задание №5

Логирование должно выполняться **для всех** запросов, на которые получен ответ с кодом состояния, отличным от 200(OK). В файл логирования должно записывается:

- Url запроса;
- код состояния ответа

3.6.1. Рекомендации к заданию №5

Для логирования опишите Middleware, которое на входе будет получать текущее время, а на выходе проверять код состояния, и, если код отличен от 200, записывать заданную информацию в файл логирования.

Для получения логера в Middleware внедрите в метод InvokeAsync объект ILoggerFactory.

Для получения кода состояния используйте свойство объекта HttpContext `context.Response.StatusCode`.

Для получения Url запроса используйте свойства объекта HttpContext `context.Request.Path` и `context.Request.QueryString`.

Для первоначальной проверки работы созданного Middleware логируйте все запросы (без проверки кода состояния). Проверку кода состояния добавьте потом, когда убедитесь, что Middleware работает.

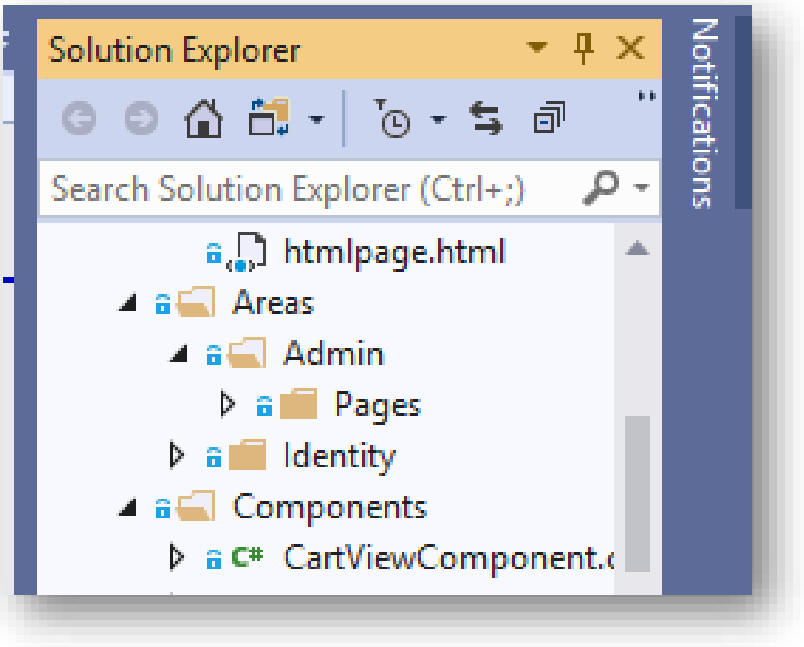
Для удобства добавления в конвейер созданного Middleware создайте расширяющий метод (например UseLogging) для IApplicationBuilder

4. Пример выполнения работы

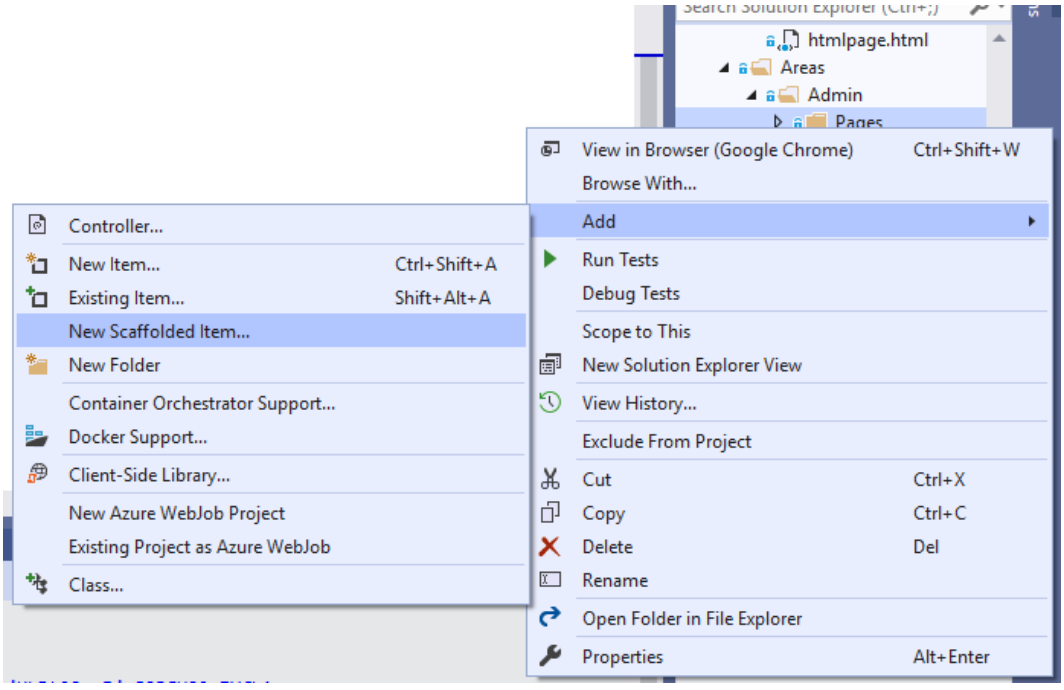
4.1. Страницы администрирования

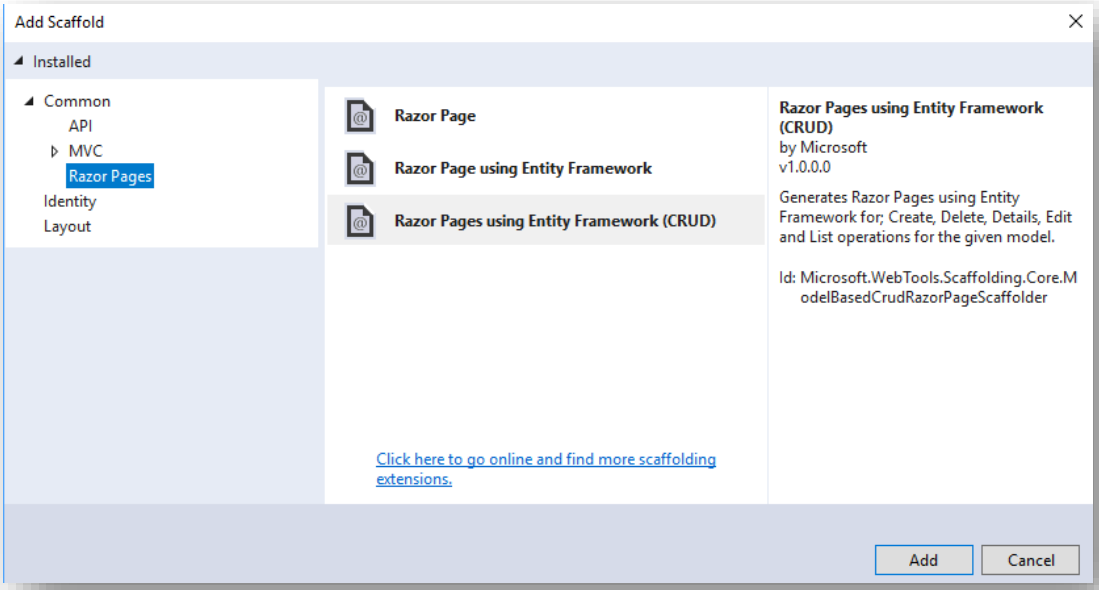
4.1.1. Генерирование страниц администрирования

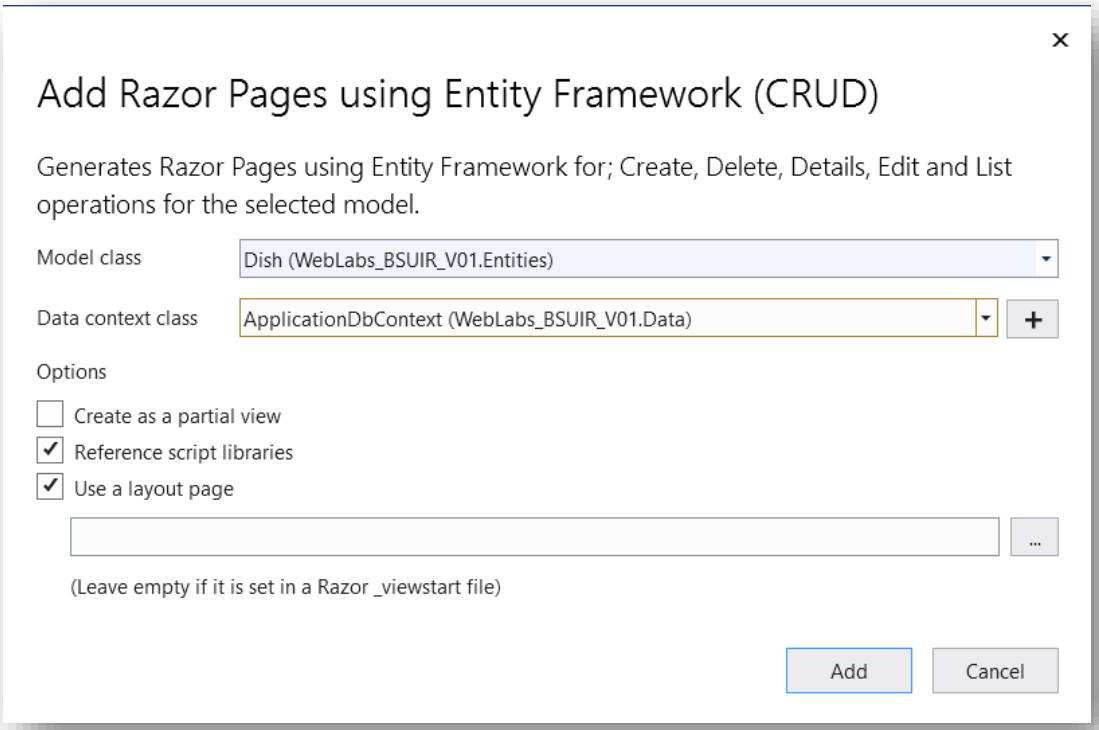
В папке Areas проекта создайте папки Admin->Pages



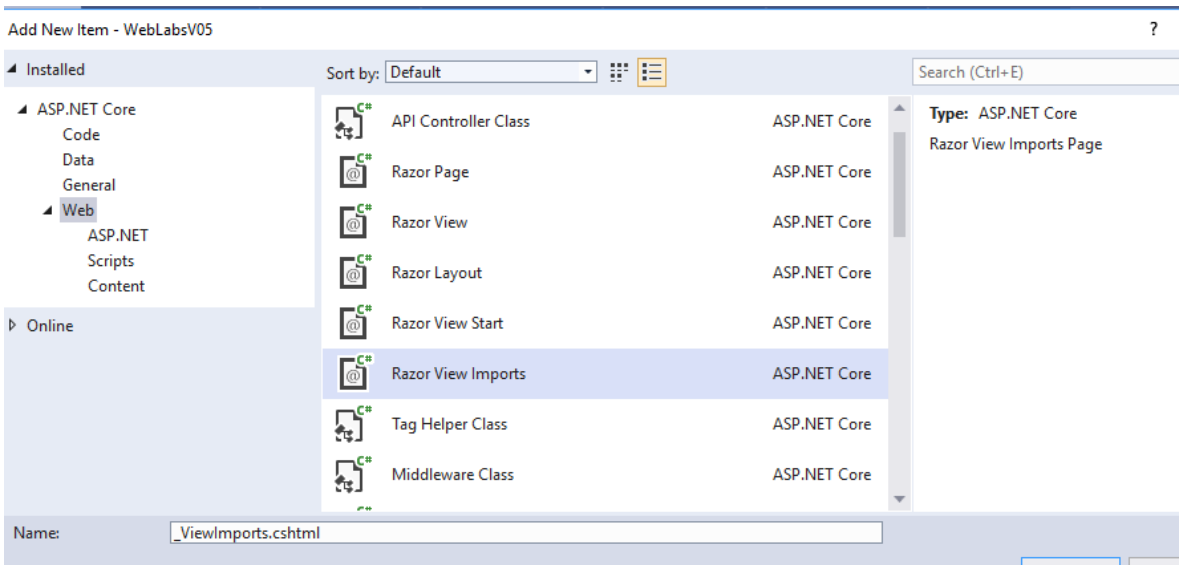
В папке Pages сгенерируйте страницы Razor с использованием EntityFramework (рис. ….). Укажите классы Dish и ApplicationDbContext в открывшемся диалоговом окне.







Добавьте в папку Pages файл _ViewImports.cshtml



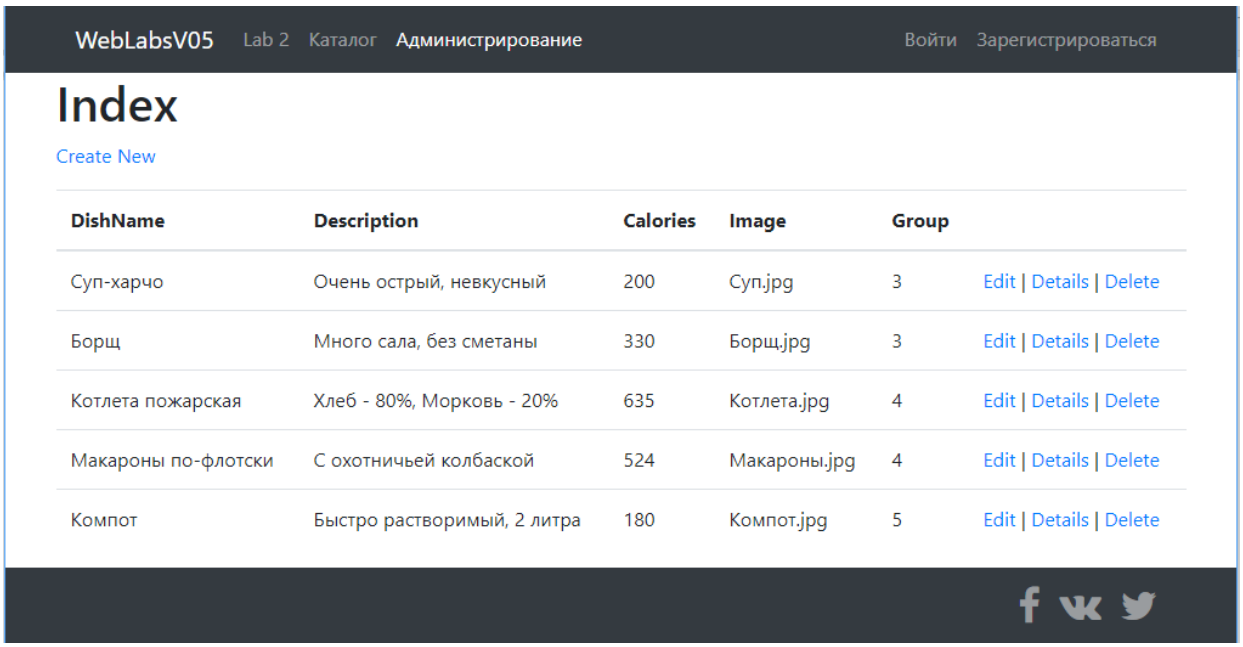
Содержимое файла должно быть следующим:

```
@using WebLabs_BSUIR_V01
@using WebLabs_BSUIR_V01.Data
@using WebLabs_BSUIR_V01.Entities
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Добавьте в папку Pages файл _ViewStart со следующим содержимым:

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

Запустите проект. Перейдите на страницу администрирования.

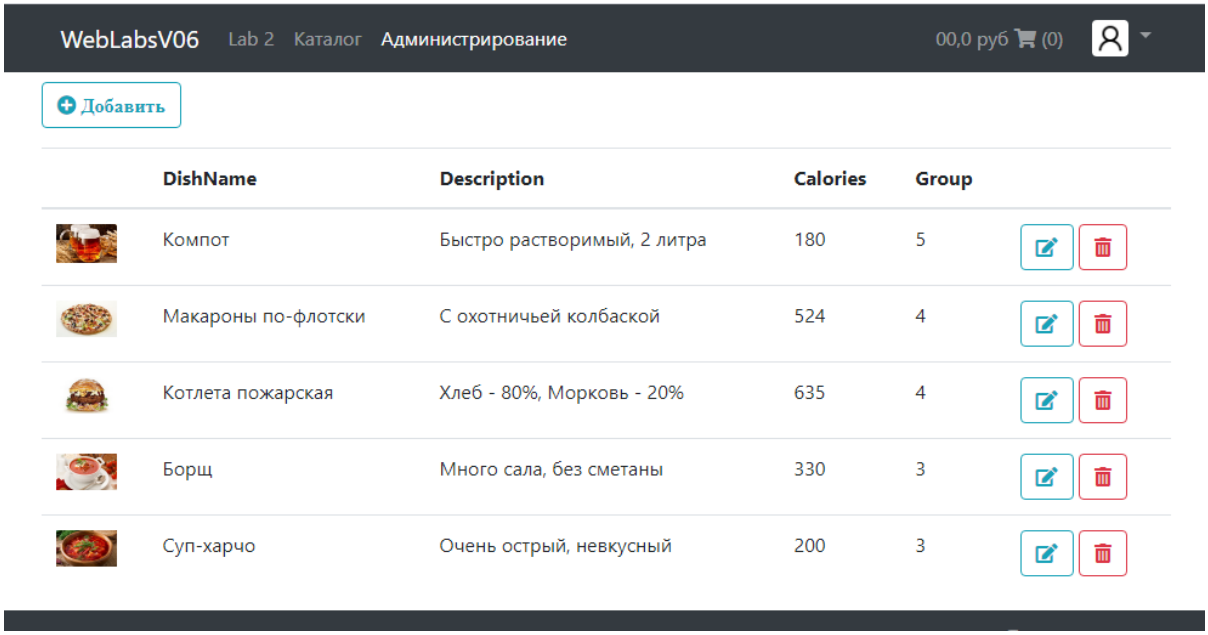


4.1.2. Оформление страницы index

Оформите список следующим образом:

- ссылку «Create New» оформите в виде кнопки «Добавить»;
- удалите ссылку «details»;
- ссылки «edit» и «delete» оформите в виде кнопок с иконками соответствующих действий (используйте классы fontawesome edit и trash);
- уберите колонку «image»;
- в первую колонку поместите изображение объекта

Пример показан на Рис.



Пример разметки страницы Index

```
@page
@model WebLabs_BSUIR_V01.Areas.Admin.Pages.IndexModel

@{
    ViewData["Title"] = "Index";
}
<h1>Index</h1>

<p class="mt-2">
    <a asp-page="Create" class="btn btn-outline-info">
        <span class="fa fa-plus-circle"> Добавить</span></a>
</p>
<table class="table">
    <thead>
        <tr>
            <th></th>
            <th>
                @Html.DisplayNameFor(model => model.Dish[0].DishName)
            </th>
```

```
            . . . (часть разметки не показана)
        </tr>
    </thead>
    <tbody>
        @foreach (var item in Model.Dish)
        {
            <tr>
                <td>
                    <img src="~/Images/@item.Image" alt="NoImage" width="50"/>
                </td>

                    . . . (часть разметки не показана)

                <td>
                    <a asp-page="./Edit" asp-route-id="@item.DishId"
                       class="btn btn-outline-info">
                            <span class="fas fa-edit"></span></a>
                    <a asp-page="./Delete" asp-route-id="@item.DishId"
                       class="btn btn-outline-danger">
                            <span class="fas fa-trash-alt"></span></a>
                </td>
            </tr>
        }
    </tbody>
</table>
```

4.1.3. Изменение маршрута ко страницам Edit и Delete

Чтобы id объекта для удаления или редактирования передавался в виде сегмента маршрута, укажите шаблон на страницах Edit и Delete в директиве @page:

```
@page "{id:int}"
```

4.1.4. Изменения на страницах Edit и Create

При добавлении или изменении объекта необходимо иметь возможность добавлять файл изображения, а также выбирать группу не по id, а по названию.

Для отображения в списке названий групп измените в коде модели страниц формирования ViewData[«DishGroupId»]:

```
ViewData["DishGroupId"] =
new SelectList(_context.DishGroups, "DishGroupId", "GroupName");
```

Для передачи файла изображения сделайте следующие изменения.

Добавьте существующий элемент – файл bs-custom-file-input.js в папку Scripts проекта.

Подключите этот файл скрипта в секцию Scripts на страницах Edit и Create, и там же активируйте скрипт:

```
@section Scripts{
    <script src="~/js/bs-custom-file-input.js"></script>
     <script>
          bsCustomFileInput.init();
     </script>
}
```

Вы уже делали так на странице Register

Добавьте разметку для загрузки файла изображения:

На странице Create:

```
<form method="post" enctype="multipart/form-data">

     . . .
<div class="form-group">
     <label asp-for="Dish.Image" class="control-label"></label>
     <div class="custom-file mt-3">
          <input type="file"  asp-for="Image"
               class="custom-file-input" id="customFile">
          <label class="custom-file-label"
               for="customFile">Выберите файл</label>
     </div>
</div>
```

В файле модели Create:

```
private readonly IWebHostEnvironment _environment;

public CreateModel(ApplicationDbContext context,
                   IWebHostEnvironment env)
{
     _context = context;
     _environment = env;
}

. . .
[BindProperty]
public Dish Dish { get; set; }
[BindProperty]
public IFormFile Image { get; set; }

public async Task<IActionResult> OnPostAsync()
{
     if (!ModelState.IsValid)
```

```
        {
            return Page();
        }

        _context.Dishes.Add(Dish);
        await _context.SaveChangesAsync();
        if(Image!=null)
        {
            var fileName = $"{Dish.DishId}" +
        Path.GetExtension(Image.FileName);
            Dish.Image = fileName;
            var path = Path.Combine(_environment.WebRootPath, "Images",
        fileName);
            using(var fStream = new FileStream(path, FileMode.Create))
            {
                await Image.CopyToAsync(fStream);
            }
            await _context.SaveChangesAsync();
        }
        return RedirectToPage("./Index");
    }
```

Запустите проект. Проверьте результат.

На странице Edit:

```
<form method="post" enctype="multipart/form-data">
    <div asp-validation-summary="ModelOnly"
                class="text-danger"></div>
    <input type="hidden" asp-for="Dish.DishId" />
        <div class="form-group">
            @if (!string.IsNullOrEmpty(Model.Dish.Image))
            {
                <img src=~/Images/@Model.Dish.Image
                    alt="NoSuchFile" width="150" />
            }
            else
            {
                <h4>Файл изображения не задан</h4>
            }
        <div class="custom-file mt-3">
            <input type="file" class="custom-file-input"
                asp-for="Image" id="customFile">
            <label class="custom-file-label" for="customFile">
                Выберите файл изображения</label>
        </div>
        . . .
```

В файле модели Edit:

```
public class EditModel : PageModel
{
    private readonly ApplicationDbContext _context;
```

```
        private IWebHostEnvironment _environment;

        public EditModel(ApplicationDbContext context,
                                IWebHostEnvironment env)
        {
                _context = context;
                _environment = env;
        }

        [BindProperty]
        public Dish Dish { get; set; }
        [BindProperty]
        public IFormFile Image { get; set; }

                . . .

public async Task<IActionResult> OnPostAsync()
{
        if (!ModelState.IsValid)
        {
                return Page();
        }
        if (Image != null)
        {
                var fileName = $"{Dish.DishId}" +
        Path.GetExtension(Image.FileName);
                Dish.Image = fileName;
                var path = Path.Combine(_environment.WebRootPath, "Images",
        fileName);
                using (var fStream = new FileStream(path, FileMode.Create))
                {
                        await Image.CopyToAsync(fStream);
                }
}

                }
                catch (DbUpdateConcurrencyException)
                {
                    . . .
                }

                return RedirectToPage("./Index");
                 }
```

Запустите проект, проверьте результат

4.2. Работа с корзиной заказа

4.2.1. Описание модели

В папку Models добавьте файл Cart. В файле опишите класс CartItem (описывает один элемент корзины) и класс Cart (описывает корзину заказов):

```
public class Cart
    {
        public Dictionary<int,CartItem> Items { get; set; }
        public Cart()
        {
            Items = new Dictionary<int, CartItem>();
        }
        /// <summary>
        /// Количество объектов в корзине
        /// </summary>
        public int Count
        {
            get
            {
                return Items.Sum(item => item.Value.Quantity);
            }
        }
        /// <summary>
        /// Количество калорий
        /// </summary>
        public int Calories
        {
            get
            {
                return Items.Sum(item => item.Value.Quantity *
item.Value.Dish.Calories);
            }
        }

        /// <summary>
        /// Добавление в корзину
        /// </summary>
        /// <param name="dish">добавляемый объект</param>
        public void AddToCart(Dish dish)
        {
           // если объект есть в корзине
           // то увеличить количество
            if (Items.ContainsKey(dish.DishId))
                Items[dish.DishId].Quantity++;
           // иначе - добавить объект в корзину
            else
                Items.Add(dish.DishId, new CartItem {
                    Dish = dish, Quantity = 1
                });
```

```
        }

        /// <summary>
        /// Удалить объект из корзины
        /// </summary>
        /// <param name="id">id удаляемого объекта</param>
        public void RemoveFromCart(int id)
        {
            Items.Remove(id);
        }

        /// <summary>
        /// Очистить корзину
        /// </summary>
        public void ClearAll()
        {
            Items.Clear();
        }
    }

    /// <summary>
    /// Клас описывает одну позицию в корзине
    /// </summary>
    public class CartItem
    {
        public Dish Dish { get; set; }
        public int Quantity { get; set; }
     }
```

4.2.2. Расширяющий метод сессии

Для возможности сохранения/чтения любого объекта в сессии опишите расширяющий метод, который сериализует объект в строку для сохранения в сессии и десериализует строку в объект при чтении из сессии. Для этого в папку Extensions добавьте файл SessionExtensions:

```
using Microsoft.AspNetCore.Http;
using System.Text.Json;
using System;

namespace WebLabs_BSUIR_V01.Extensions
{
    public static class SessionExtensions
    {
        public static void Set<T>(this ISession session,
                                  string key, T item)
        {
            var serializedItem = JsonSerializer.Serialize(item);
            session.SetString(key, serializedItem);
```

```
        }

        public static T Get<T>(this ISession session, string key)
        {
            var item = session.GetString(key);
            return item == null
                ? Activator.CreateInstance<T>() // или default(T)
                : JsonSerializer.Deserialize<T>(item);
        }
    }
}
```

4.2.3. Разрешить использование сессии в проекте

Измените код класса Startup.cs.

В методе ConfigureServices добавьте:

```
services.AddDistributedMemoryCache();
services.AddSession(opt =>
{
      opt.Cookie.HttpOnly = true;
      opt.Cookie.IsEssential = true;
});
```

В методе Configure:

```
app.UseAuthentication();
app.UseSession();
```

4.2.4. Создание контроллера

В папку Controllers добавьте контроллер Cart. Метод Index должен вывести список объектов в корзине. Метод Add должен добавить объект в корзину

```
public class CartController : Controller
    {
        private ApplicationDbContext _context;
        private string cartKey = "cart";

        public CartController(ApplicationDbContext context)
        {
            _context = context;
        }
        public IActionResult Index()
        {
           _cart = HttpContext.Session.Get<Cart>(cartKey);
```

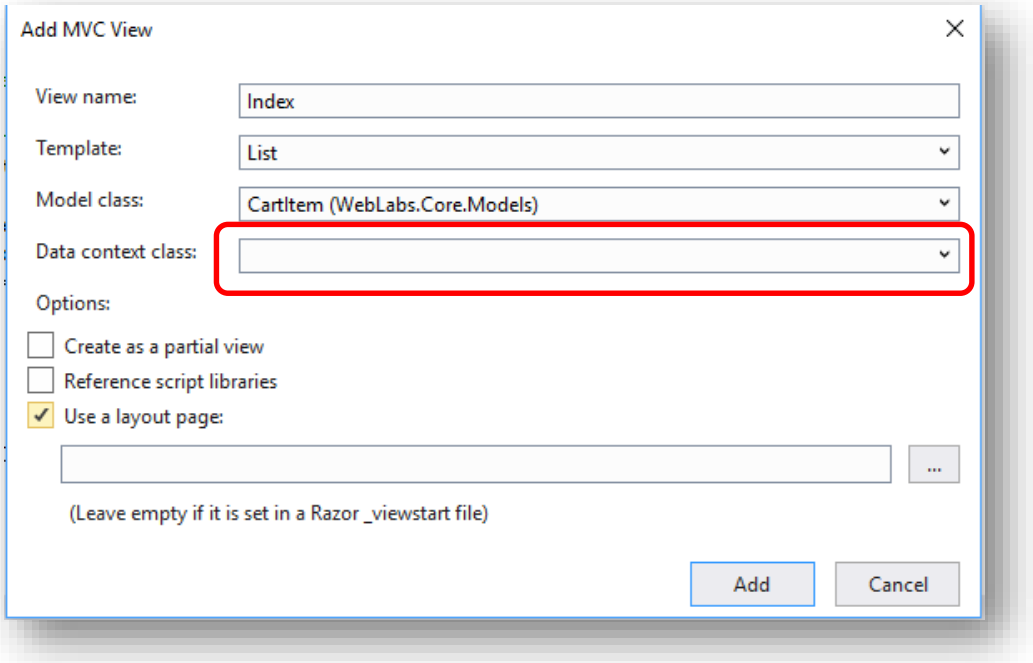
```
            return View(_cart.Items.Values);
        }

        [Authorize]
        public IActionResult Add(int id, string returnUrl)
        {
            _cart = HttpContext.Session.Get<Cart>(cartKey);
            var item = _context.Dishes.Find(id);
            if(item!=null)
            {
                _cart.AddToCart(item);
                HttpContext.Session.Set<Cart>(cartKey, _cart);
            }
            return Redirect(returnUrl);
        }
    }
```

4.2.5. Создание представления Index

Можно воспользоваться Scaffold – сгенерировать представление с использованием шаблона List. В качестве модели указать CartItem.

ОБЯЗАТЕЛЬНО: очистите поле контекста базы данных.



Отредактируйте представление, чтобы в списке отображалось только название, количество и изображение. Уберите ссылки добавления, редактирования и деталей. Ссылку удаления оформите в виде пиктограммы:

```
@model IEnumerable<CartItem>
```

```
@{
    ViewData["Title"] = "Index";
}

<h1>Корзина</h1>
<table class="table">
    <thead>
        <tr>
            <th></th>
            <th></th>
            <th>
                @Html.DisplayNameFor(model => model.Quantity)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>
                <img src="~/Images/@item.Dish.Image" alt="No image"
width="100"/>
            </td>
            <td>
                @Html.DisplayFor(m=>item.Dish.DishName)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Quantity)
            </td>
            <td>
                <a asp-controller="Cart"
                   asp-action="Delete"
                   asp-route-id="@item.Dish.DishId"
                   class="btn btn-outline-danger">
                    <i class="fas fa-trash-alt"></i>
                </a>
            </td>
        </tr>
}
    </tbody>
</table>
```

Запустите проект. На странице списка объектов кликните «В корзину». Кликните на информацию о корзине в меню пользователя – должен выводиться список объектов в корзине.

4.2.6. Изменение компонента CartViewComponent

В методе Invoke компонента:

```
public IViewComponentResult Invoke()
{
      var cart = HttpContext.Session.Get<Cart>("cart");
      return View(cart);
}
```

В представлении Default компонента:

```
@model Cart
<a asp-controller="Cart"
   asp-action="Index"
   class="navbar-text ml-auto">
    @Model.Calories калорий <i class="fas fa-shopping-cart nav-
color"></i>
    (@Model.Count)
</a>
```

Запустите проект. Проверьте результат.

4.3. Создание сервиса CartService

4.3.1. Предварительные действия

В классе Cart методы работы с корзиной сделайте виртуальными.

Для получения в создаваемом сервисе объекта класса HttpContent понадобится объект IHttpContextAccessor. Добавьте соответствующий сервис в файле Startup.cs:

```
services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
```

4.3.2. Описание класса CartService

В папку Services добавьте класс CartService:

```
namespace WebLabs_BSUIR_V01.Services
{
    public class CartService : Cart
    {
        private string sessionKey = "cart";
        /// <summary>
        /// Объект сессии
        /// Не записывается в сессию вместе с CartService
        /// </summary>
        [JsonIgnore]
        ISession Session { get; set; }
        /// <summary>
        /// Получение объекта класса CartService
```

```
        /// </summary>
        /// <param name="sp">объект IserviceProvider</param>
        /// <returns>объекта класса CartService, приведенный к типу
Cart</returns>
        public static Cart GetCart(IServiceProvider sp)
        {
            // получить объект сессии
            var session = sp
                .GetRequiredService<IHttpContextAccessor>()
                .HttpContext
                .Session;
            // получить CartService из сессии
            // или создать новый для возможности тестирования
            var cart = session?.Get<CartService>("cart")
                ?? new CartService();

            cart.Session = session;
            return cart;
        }
        // переопределение методов класса Cart
        // для сохранения результатов в сессии
        public override void AddToCart(Dish dish)
        {
            base.AddToCart(dish);
            Session?.Set<CartService>(sessionKey, this);
        }
        public override void RemoveFromCart(int id)
        {
            base.RemoveFromCart(id);
            Session?.Set<CartService>(sessionKey, this);
        }
        public override void ClearAll()
        {
            base.ClearAll();
            Session?.Set<CartService>(sessionKey, this);
        }
    }
}
```

4.3.3. Регистрация сервиса CartService

В классе Startup зарегистрируйте получение объекта класса Cart из сервисов:

```
services.AddScoped<Cart>(sp=>CartService.GetCart(sp));
```

4.3.4. Использование сервиса

Используйте внедрение зависимостей для получения объекта корзины заказов

В контроллере Cart:

```
public class CartController : Controller
    {
        private ApplicationDbContext _context;
        private Cart _cart;

        public CartController(ApplicationDbContext context, Cart cart)
        {
            _context = context;
            _cart = cart;

        }
        public IActionResult Index()
        {
            return View(_cart.Items.Values);
        }

        [Authorize]
        public IActionResult Add(int id, string returnUrl)
        {
            var item = _context.Dishes.Find(id);
            if(item!=null)
            {
                _cart.AddToCart(item);
            }
            return Redirect(returnUrl);
        }

        public IActionResult Delete(int id)
        {
            _cart.RemoveFromCart(id);
            return RedirectToAction("Index");
        }
}
```

При получении корзины из сервиса не нужно получать ее из сессии и не нужно сохранять результаты в сессии, т.к. это реализовано в сервисе CartService. При тестировании контроллера сессия также не будет использоваться, что упростит код теста.

В компоненте CartViewComponent:

```
public class CartViewComponent:ViewComponent
    {
        private Cart _cart;
        public CartViewComponent(Cart cart)
        {
            _cart = cart;
        }
        public IViewComponentResult Invoke()
```

```
        {
            return View(_cart);
        }
}
```

Запустите проект, проверьте результат.

4.4. Подключение логера

Загрузите в проект NuGet пакет Serilog.Extensions.Logging.File.

В классе Startup укажите использование логера:

```
public void Configure(IApplicationBuilder app,
                      IWebHostEnvironment env,
                      ApplicationDbContext context,
                      UserManager<ApplicationUser> userManager,
                      RoleManager<IdentityRole> roleManager,
                      ILoggerFactory logger)
        {
                    logger.AddFile("Logs/log-{Date}.txt");
      . . .
      }
```

Укажите фильтр логирования в файле program.cs:

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
             Host.CreateDefaultBuilder(args)
                 .ConfigureWebHostDefaults(webBuilder =>
                 {
                     webBuilder.UseStartup<Startup>();
                 })
             .ConfigureLogging(lp =>
             {
                 lp.ClearProviders();
                 lp.AddFilter("Microsoft", LogLevel.None);
                 }
```

4.4.1. Применение логера

В классе контроллера Product получите объект ILogger:

```
private ILogger _logger;

public ProductController(ApplicationDbContext context,
                 ILogger<ProductController> logger)
{
      _pageSize = 3;
      _context = context;
      _logger = logger;
}
```

В методе Index контроллера Product:

```
public IActionResult Index(int? group, int pageNo)
{
      var groupMame = group.HasValue
            ? _context.DishGroups.Find(group.Value)?.GroupName
            : "all groups";
      _logger.LogInformation($"info: group={group},  page={pageNo}");
      var dishesFiltered = _context.Dishes
            .Where(d => !group.HasValue || d.DishGroupId == group.Value);
      . . .
      }
```

Запустите проект, перейдите на страницу каталога. Переключайтесь между страницами и группами. Откройте файл логирования. Убедитесь, что в файл записывается требуемая информация, например:

```
info: group=all groups,  page=2
info: group=Супы,  page=1
info: group=Напитки,  page=1
```

Уберите логирование в контроллере Product

4.5. Создание Middleware

4.5.1. Описание класса LodMiddleware

Добавьте в проект папку Middleware.

Добавьте в созданную папку файл LogMiddleware.

```
namespace WebLabsV06.Middleware
{
    public class LogMiddleware
    {
        RequestDelegate _next;
        ILogger<LogMiddleware> _logger;
        public LogMiddleware(RequestDelegate next,
                             ILogger<LogMiddleware> logger)
        {
            _next = next;
            _logger = logger;
        }

        public async Task Invoke(HttpContext context)
        {
            await _next.Invoke(context);
            if(context.Response.StatusCode!=StatusCodes.Status200OK)
            {
                var path = context.Request.Path +
context.Request.QueryString;
```

```
                _logger.LogInformation($"Request {path} returns status
code {context.Response.StatusCode.ToString()}");
            }
        }
    }
}
```

4.5.2. Расширяющий метод UseLogging()

В папку Extensions добавьте файл appExtensions:

```
using Microsoft.AspNetCore.Builder;
using WebLabsV06.Middleware;

namespace WebLabsV06.Extensions
{
    public static class AppExtensions
    {
        public static IApplicationBuilder UseFileLogging(this
IApplicationBuilder app)
            => app.UseMiddleware<LogMiddleware>();
    }
}
```

4.5.3. Использование LogMiddleware в конвейере приложения

В классе startup.cs добавьте использование созданного компонента:

```
app.UseFileLogging ();
```

4.5.4. Проверка работы компонента

Закомментируйте в компоненте LogMiddleware строки, проверяющие код состояния запроса. Запустите приложение. Выполните переходы по страницам. Откройте файл логирования и убедитесь, что записывается верная информация, например:

```
24.07.2019-14:10:10: url: /Catalog/Page_1?group=3 returns 200
24.07.2019-14:10:10: url: /Image/GetAvatar returns 200
```

Верните (раскомментируйте) проверку кода состояния в компоненте LogMiddleware.