Лабораторная работа №8

Web API, Blazor (6 часов)

1. Цель работы.

Знакомство с логированием в ASP.Net Core.

Знакомство с API контроллерами.

Изучение проекта Blazor.

Получение навыков в создании компонентов Razor.

2. Общие сведения.

3. Выполнение работы

3.1. Исходные данные

Используйте проект из лабораторной работы №7.

3.2. Задание №1

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.

3.2.1. Рекомендации к заданию №1

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.

3.3. Задание №2

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

- Url запроса;
- код состояния ответа

3.3.1. Рекомендации к заданию №2

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`.

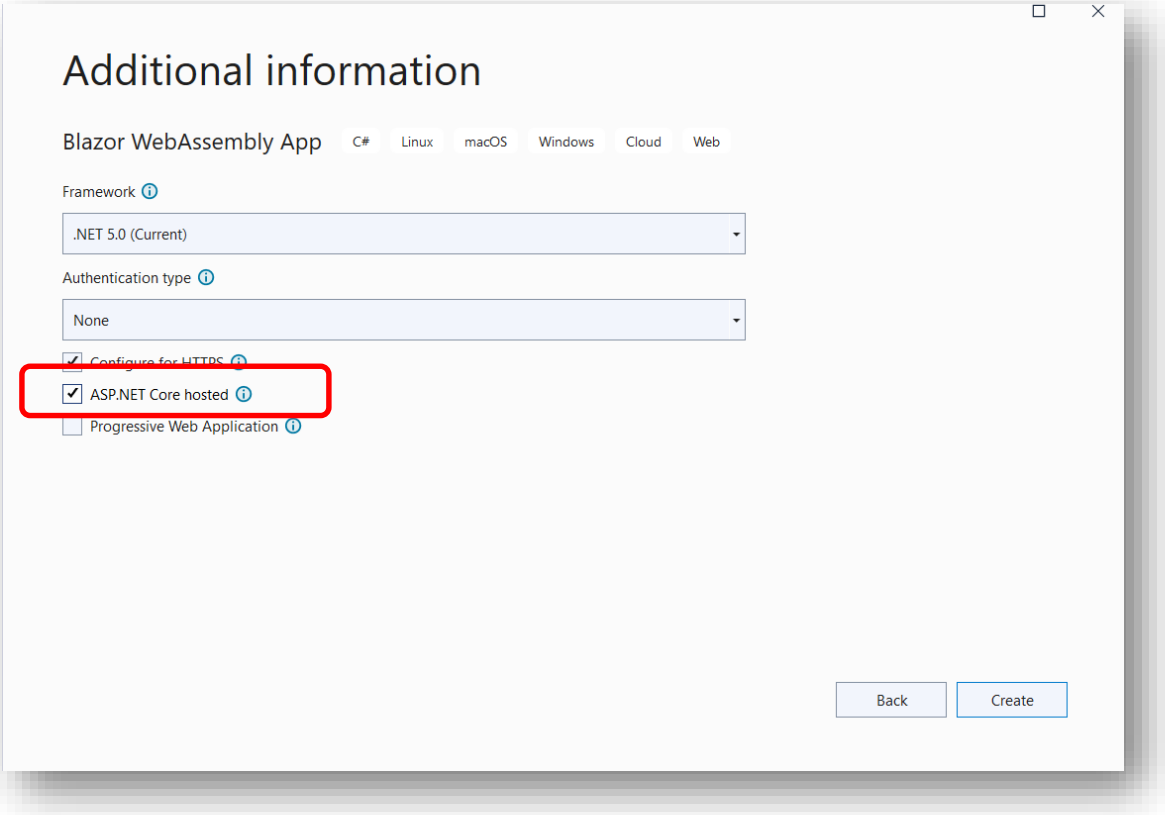
Для первоначальной проверки работы созданного Middleware логируйте все запросы (без проверки кода состояния). Проверку кода состояния добавьте потом, когда убедитесь, что Middleware работает.

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

3.4. Задание №3

Добавьте в решение новый проект – приложение Blazor WebAssembly (**Blazor WebAssembly App**). Назначьте проекту имя XXX.Blazor, где XXX – имя вашего решения.

При создании проекта в диалоге «Additional information» отметьте пункт «ASP.NET Core hosted»



В созданном проекте **XXX.Blazor.Client** найдите в файле Program.cs регистрацию компонента «app» и сервиса HttpClient.

Найдите корневой компонент приложения app.razor. Откройте его. Познакомьтесь с использованием компонента Router.

В папке wwwroot найдите корневую страницу приложения index.html. Откройте файл index.html и познакомьтесь с его содержимым. Найдите, где

размещен главный компонент приложения («app») и где подключается скрипт _framework/blazor.webassembly.js.

Найдите файл _Imports.razor и познакомьтесь с его содержимым.

Найдите страницу макета (MainLayout.razor) и познакомьтесь с ее содержимым. Найдите использование на макете компонента NavMenu. Найдите выражение @Body. Сюда будет размещена разметка страницы, использующей макет.

Найдите компонент NavMenu. Изучите его содержимое. Обратите внимание на использование компонента NavLink для переключения между страницами.

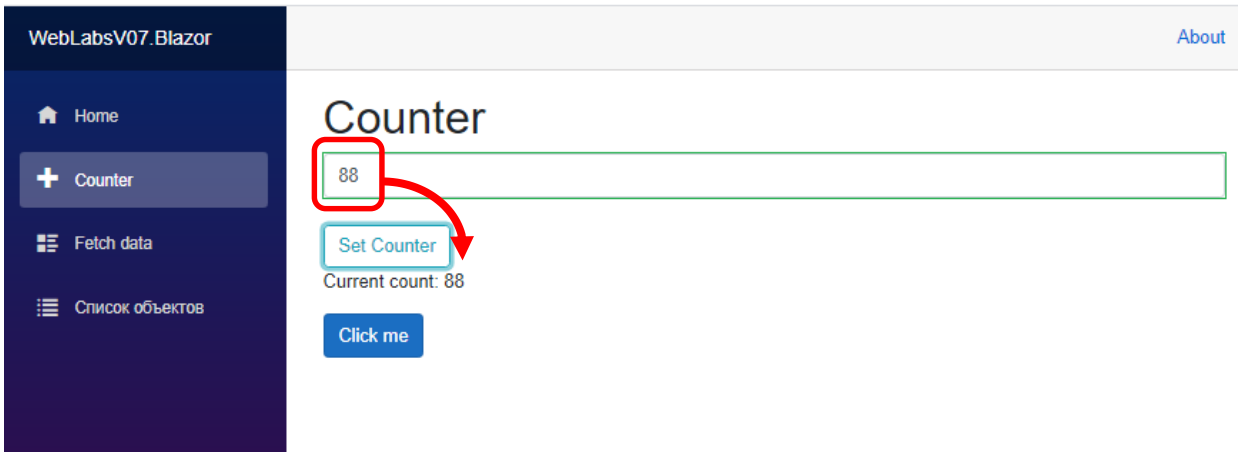
3.5. Задание №4

Сделайте проект **XXX.Blazor.Server** стартовым проектом.

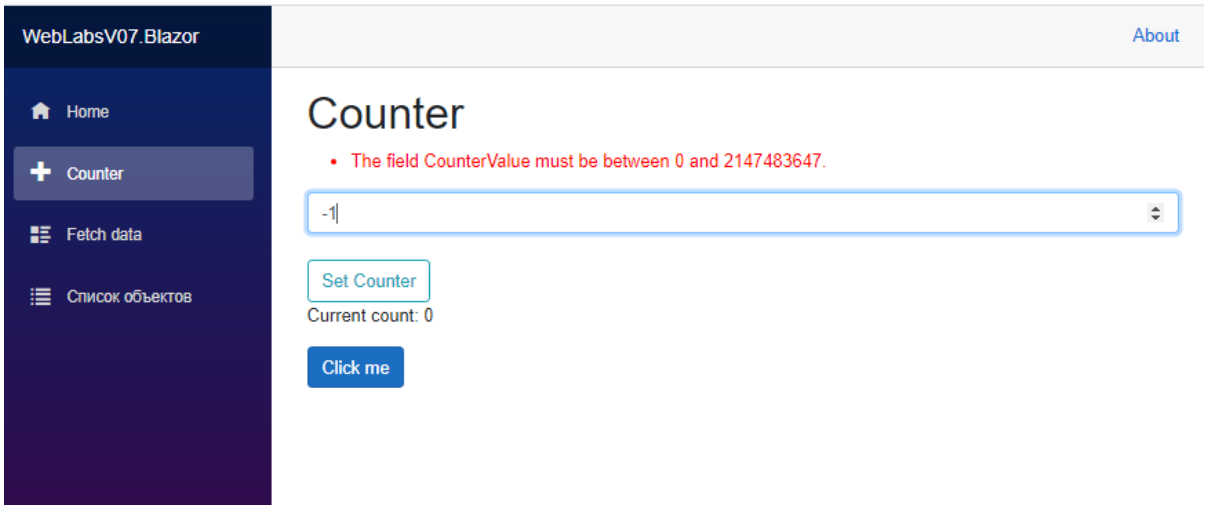
Запустите проект Blazor. В окне браузера запустите режим разработчика (клавиша F12). Перейдите к вкладке Network. Убедитесь, что при переключении страниц на сервер отправляются запросы Http только для чтения данных WeatherForecast.

3.6. Задание №5

На странице Counter поместите поле ввода и кнопку «Установить». При нажатии на кнопку счетчику должно присваиваться значение, введенное в поле ввода.



Введенное значение должно быть целым положительным числом. Предусмотреть валидацию с выводом соответствующего сообщения об ошибке.



3.6.1. Рекомендации к заданию №5

Используйте компонент EditForm. Инициализацию счетчика выполните в обработчике события формы OnValidSubmit.

Для ввода значения счетчика используйте компонент <InputNumber>

Для задания правил валидации опишите вспомогательный класс, содержащий свойство типа «int». Используйте атрибут [Range].

Для валидации введенного значения счетчика и вывода сообщения об ошибке используйте компоненты <DataAnnotationsValidator/> и <ValidationSummary/>.

3.7. Задание №6

В проекте **XXX.Blazor.Server** создайте контроллер API, реализующий CRUD функции для объектов вашей предметной области (в предлагаемых примерах – Dish).

Ознакомьтесь с содержимым с созданного контроллера.

В методе Get реализуйте возможность фильтрации по группе. Номер группы должен передаваться в строке запроса, например, «?group=3».

Обратитесь к созданному контроллеру из браузера. Убедитесь, что контроллер возвращает список объектов в формате Json.

3.7.1. Рекомендации к заданию №6

Класс контекста базы данных и классы сущностей используйте из проекта предыдущих лабораторных работ. Для этого достаточно в новом проекте сделать ссылку на проект из предыдущих работ.

Строку подключения используйте также из проекта предыдущих работ.

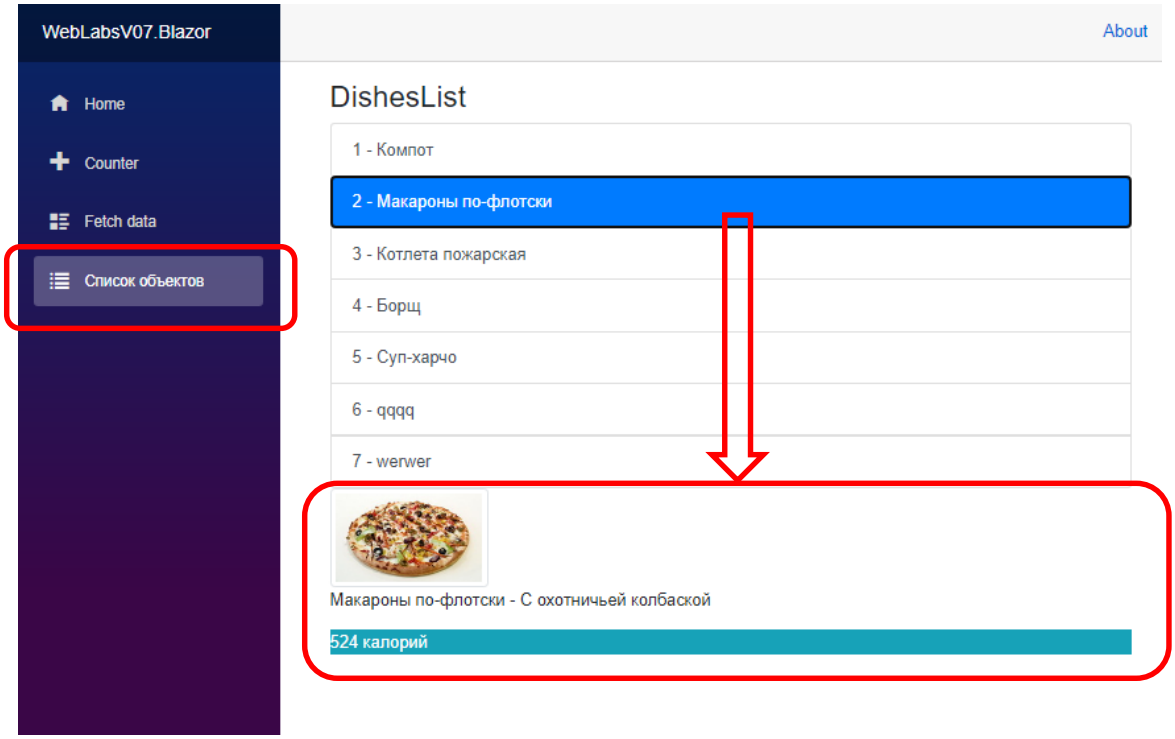
Зарегистрируйте контекст базы данных в качестве сервиса в классе Startup.

3.8. Задание №7

Создайте страницу (ApiDemo.razor), выводящую список объектов, полученных с помощью контроллера API, описанного в задании №6.

Для перехода на страницу добавьте в меню сайта пункт «Список объектов».

Список должен содержать порядковый номер и название объекта. При клике на объект необходимо вывести информацию о выбранном объекте: название, описание и количественную характеристику объекта, например:



Список объектов и подробная информация о выбранном объекте должны быть реализованы в виде отдельных компонентов.

Получение списка объектов и поиск объекта должны осуществляться на родительской странице (ApiDemo.razor).

3.8.1. Рекомендации к заданию №7

Для отправки запросов к API внедрите на страницу объект **`HttpClient`**.

Для десериализации данных Json, получаемых от API, используйте метод **GetFromJsonAsync** объекта HttpClient.

Для вывода информации опишите модели представления (модель для вывода списка объектов и модель для подробной информации об объекте).

Формат Json использует camel-style для именования свойств, т.е. все названия начинаются со строчной (маленькой) буквы. Свойства объектов C# именуются с прописной (заглавной) буквы. Для разрешения конфликта при десериализации воспользуйтесь одним из следующих методов:

- добавьте в метод `DeserializeAsync` параметр

```
new JsonSerializerOptions { PropertyNameCaseInsensitive=true }.
```

- в классе модели перед свойствами поставьте атрибуты JsonPropertyName, например `[JsonPropertyName("dishId")]`

Второй способ более производительный.

Для оформления списка объектов можно воспользоваться компонентом bootstrap list-group (см. *https://getbootstrap.com/docs/4.5/components/list-group/#links-and-buttons).*

Получение списка объектов и поиск объекта должны осуществляться на родительской странице (ApiDemo.razor). Выполните обмен данными между страницей ApiDemo.razor и дочерними компонентами (компонент вывода списка и компонент вывода подробной информации об объекте). Также в компоненте вывода списка объектов обработайте событие клика для передачи id выбранного объекта родительской странице.

Примечание:

Если при обращении к API сервер откажет в доступе, то в классе Startup, в методе ConfigureServices, нужно добавить политику, разрешающую любой доступ:

```
// CORS - политика "Разрешен любой доступ"
            services.AddCors(opt =>
                opt.AddPolicy("AllowAny",
                    builder => {
                        builder.AllowAnyMethod()
                        .AllowAnyHeader()
                        .AllowAnyOrigin();
                         }));
```

и в методе Configure добавить компонент CORS с созданной политикой:

```
app.UseCors("AllowAny");
```

4. Пример выполнения работы

4.1. Задание №1. 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.AddFilter("Microsoft", LogLevel.Error);
          lp.AddFilter("Microsoft.Hosting.Lifetime", LogLevel.None);
                }
```

4.1.1. 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

4.2. Задание №2. Создание Middleware

4.2.1. 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()}");
            }
        }
    }
}
```

4.2.2. 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>();
    }
}
```

4.2.3. Использование LogMiddleware в конвейере приложения

В классе startup.cs добавьте использование созданного компонента:

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

4.2.4. 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:

```
      2021-05-28T20:25:35.0977067+03:00                    [INF]        Request
/Cart/Add/1?returnUrl=%2FCatalog returns status code 302 (6cfb108d)
      2021-05-28T20:25:35.1157126+03:00                    [INF]        Request
/index.html?ReturnUrl=%2FCart%2FAdd%2F1%3FreturnUrl%3D%252FCatalog    returns
status code 404 (5498401b)
      2021-05-28T20:39:23.3863187+03:00                    [INF]        Request
/Cart/Add/1?returnUrl=%2FCatalog returns status code 302 (6cfb108d)
```

Верните (раскомментируйте) проверку кода состояния в компоненте LogMiddleware.

4.3. Задание №5. Инициализация счетчика (проект Blazor)

На странице Counter в блоке **@code** опишите класс **Input**, который будет использоваться для привязки формы, опишите поле «***Input input***» и проинициализируйте созданное поле input в методе **OnInitialized**:

```
@code {
    private int currentCount = 0;
    private FormModel formModel;

    protected override void OnInitialized()
    {
        formModel=new();
    }
    private void IncrementCount()
    {
        currentCount++;
    }

    class FormModel
    {
        [DataType("int")]
        [Range(0,int.MaxValue)]
        public int NewValue { get; set; }
    }
}
```

Свойство CounterValue будет использоваться для привязки к полю ввода начального значения счетчика.

Добавьте разметку для ввода начального значения счетчика:

```
<EditForm Model="formModel"  OnValidSubmit="SetCounter" >
    <DataAnnotationsValidator/>
    <ValidationSummary/>
    <InputNumber @bind-Value="formModel.NewValue"></InputNumber>
```

```
    <input type="submit"
           class="btn btn-outline-info mt-2 mb-2"
           value="Set counter">
</EditForm>
```

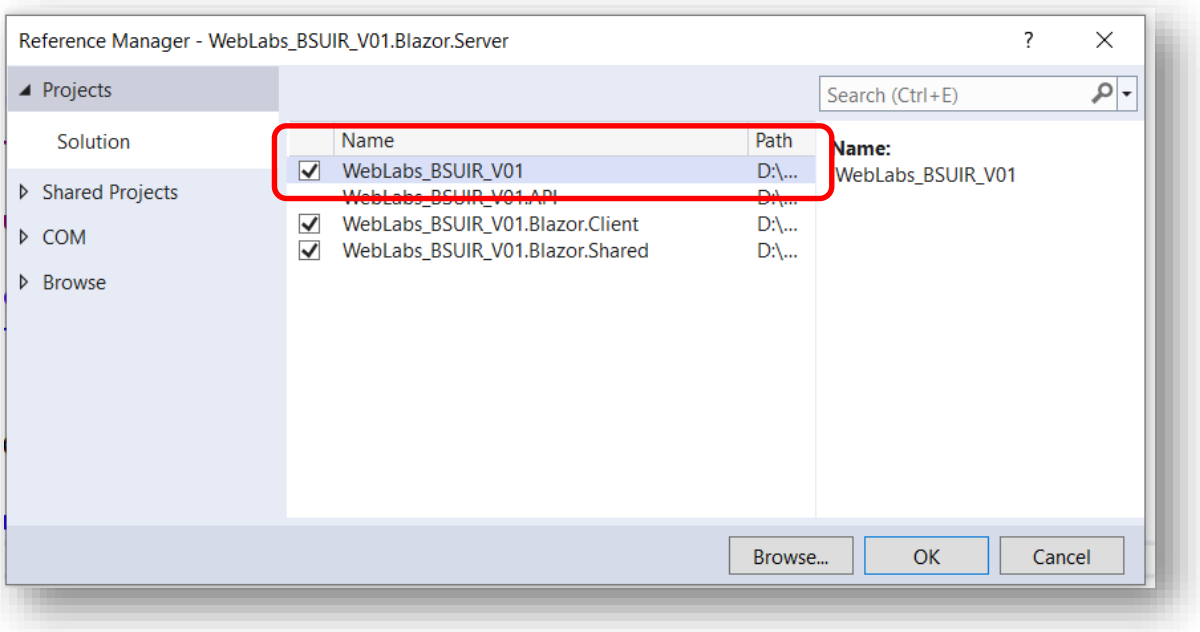
В блоке **@code** добавьте функцию-обработчик события отправки формы:

```
private void SetCounter()
    {
        currentCount = formModel.NewValue;
    }
```

Запустите проект. Проверьте результат:

4.4. Задание №6. Добавление контроллера API

Добавьте в проект **XXX.Blazor.Server** ссылку на проект из предыдущих лабораторных работ:

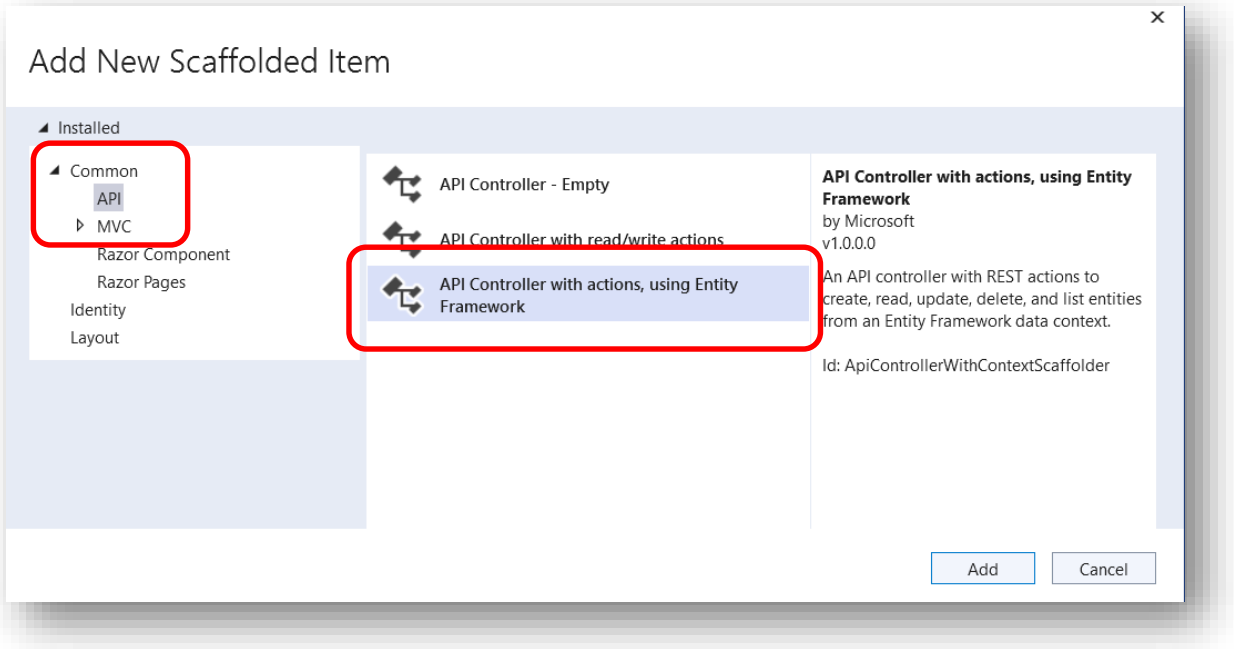


Зарегистрируйте контекст базы данных в качестве сервиса (класс Startup):

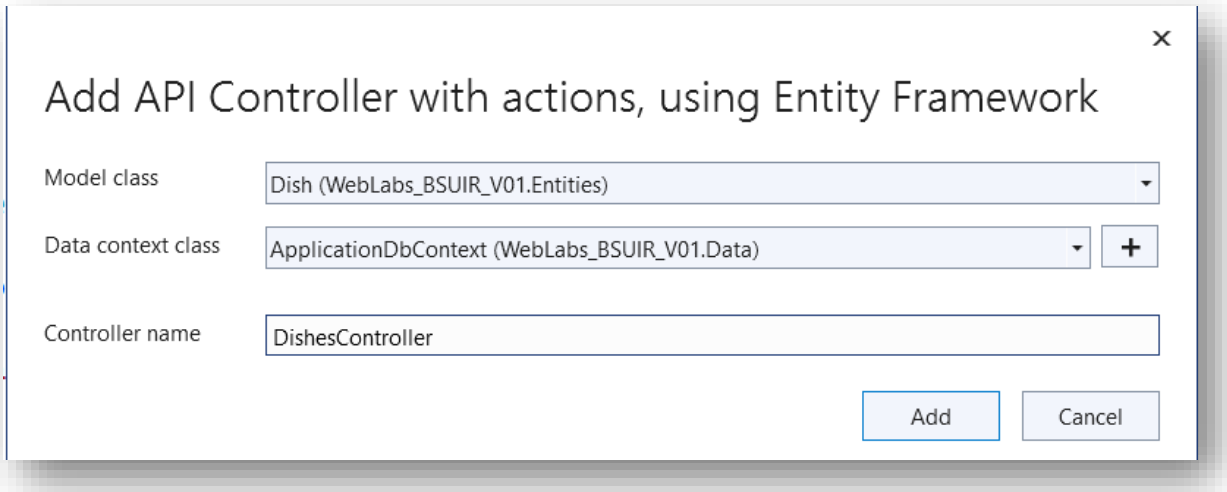
```
services.AddDbContext<ApplicationDbContext>(opt =>
     opt.UseSqlServer(Configuration
                          .GetConnectionString("DefaultConnection")));
```

Скопируйте раздел «ConnectionStrings» из файла appsettings.json проект предыдущих лабораторных работ в файл appsettings.json проекта **XXX.Blazor.Server.**

Добавьте в проект новый контроллер API с использованием EntityFramework



Укажите класс модели вашей предметной области и класс контекста базы данных:



Проверьте результат (пример запроса для проверки: https://localhost:44318/api/dishes) В браузере должны отобразиться данные в формате Json.

4.4.1. Добавление изображений

Добавьте в проект папку «wwwroot».
В папку wwwroot добавьте папку «Images».
Добавьте в папку Images файлы из соответствующей папки проекта предыдущих лабораторных работ.

4.5. Фильтрация по группе

Измените код метода GetXXX:

```
[HttpGet]
public async Task<ActionResult<IEnumerable<Dish>>> GetDishes(int?
group)
{
      var dishes = _context.Dishes.Where(
          d => !group.HasValue || d.DishGroupId == group.Value);

return await dishes.ToListAsync();
}
```

Проверьте результат (пример запроса для проверки: https://localhost:44318/api/dishes?group=4)

4.6. Задание №7. Работа с API

4.6.1. Описание моделей представления

В проект **XXX.Blazor.Client** добавьте папку Models.

Добавьте в папку Models классы:

– класс ListViewModel – содержит данные для отображения списка объектов

– класс DetailsViewModel – содержит подробную информацию о выбранном объекте

```
using System.Text.Json.Serialization;
```

```
    public class ListViewModel
    {
        [JsonPropertyName("dishId")]
        public int DishId { get; set; } // id блюда
        [JsonPropertyName("dishName")]
```

```
        public string DishName { get; set; } // название блюда

    }

    public class DetailsViewModel
    {
        [JsonPropertyName("dishName")]
        public string DishName { get; set; } // название блюда
        [JsonPropertyName("description")]
        public string Description { get; set; } // описание блюда
        [JsonPropertyName("calories")]
        public int Calories { get; set; } // кол. калорий на порцию
        [JsonPropertyName("image")]
        public string Image { get; set; } // имя файла изображения
    }
```

Укажите использование пространства имен XXX.Blazor.Client.Models в файле _imports.razor.

4.6.2. Добавление страницы ApiDemo

Добавьте в папу Pages компонент razor. Укажите название компонента ApiDemo.razor.

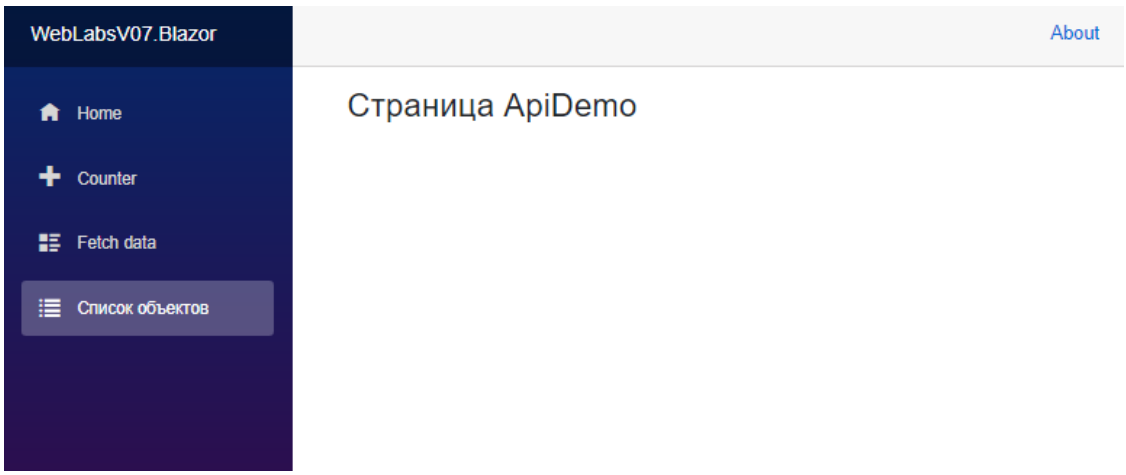
```
@page "/apidemo"
@inject HttpClient client

    <div class="container">
        <h3>Страница ApiDemo</h3>
    </div>

     @code {    }
```

Внедренный объект client понадобится для формирования запросов к Api сервису.

Откройте компонент NavMenu и добавьте в него вызов созданной страницы ApiDemo:

```
<li class="nav-item px-3">
            <NavLink class="nav-link" href="apidemo">
                <span class="oi oi-list" aria-hidden="true"></span>
Список объектов
            </NavLink>
</li>
```

Запустите проект, перейдите на созданную страницу. Убедитесь, что страница отображается:



4.6.3. Компонент вывода информации об объекте

Добавьте в проект папку Components.

В созданную папку добавьте компонент DishDetails.razor:

```
@if(Dish!=null)
{
    <img src="@imageSrc" class="img-thumbnail" width="150"/>
    <div>
        <p>@Dish.DishName - @Dish.Description</p>
        <div class="badge badge-info ">@Dish.Calories
калорий</div>
    </div>
}

@code {
    [Parameter]
    public DetailsViewModel Dish { get; set; }
    [Parameter]
    public EventCallback<DetailsViewModel> DishChanged {get; set;}
    string imageSrc
    {
        get
        {
            return $"images/{Dish.Image}";
        }
    }
}
```

Примечание:

Параметр Dish будет использоваться для обмена с родительской страницей. Для уведомлении об изменении параметра Dish добавлен параметр DishChanged.

Поле imageSrc используется для формирования адреса размещения файла изображения. Файл расположен в папке Images.

4.6.4. Компонент для вывода списка объектов

В папку Components добавьте компонент DishesList

```
<h3>DishesList</h3>

@if (Dishes == null)
{
    <p>Загрузка ...</p>
}
else
{
<div class="list-group">
    @{
        var i = 1;
        foreach (var dish in Dishes)
        {
            <button type="button" class="list-group-item list-group-
item-action">
                @(i++) - @dish.DishName
            </button>
        }
    }
</div>
}

@code {
    [Parameter]
    public IEnumerable<ListViewModel> Dishes { get; set; }
    [Parameter]
    public EventCallback<IEnumerable<ListViewModel>> DishesChanged {
get; set; }
}
```

На данном этапе компонент только выводит список объектов.

4.6.5. Проверка компонента списка объектов

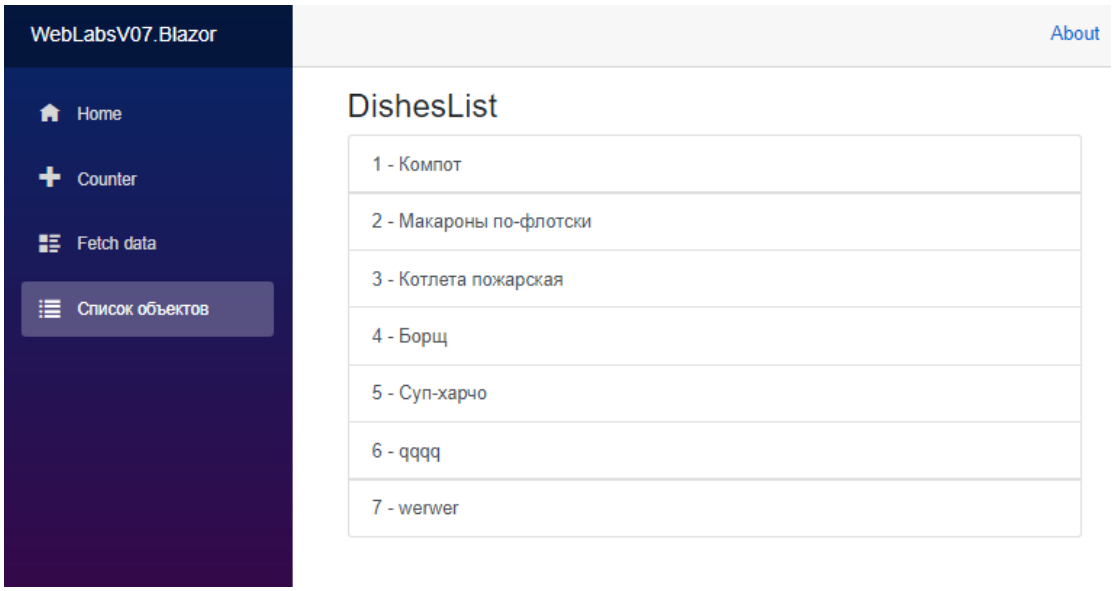
На странице ApiDemo получите список объектов из Api сервиса.

Разместите компонент DishesList на странице ApiDemo и выполните привязку списка объектов:

```
@using WebLabs_BSUIR_V01.Blazor.Client.Components

@page "/apidemo"
@inject HttpClient client
```

```
    <div class="container">
        <DishesList @bind-Dishes="@Dishes"></DishesList>
    </div>


@code {
    [Parameter]
    public IEnumerable<ListViewModel> Dishes{ get; set; }

    protected override async Task OnInitializedAsync()
    {
        Dishes = await
client.GetFromJsonAsync<IEnumerable<ListViewModel>>("api/dishes");
    }
}
```

Запустите основной проект. Перейдите на страницу «Список объектов». Пример полученного списка приведен на рисунке:



4.6.6. Доработка компонента DishesList.

Добавьте в компонент DishesList.razor обработку клика кнопки:

```
<div class="list-group">
    @{
        var i = 1;
        foreach (var dish in Dishes)
        {
            <button type="button"                                    1
                    class="list-group-item list-group-item-action
                    @(SelectedId == dish.DishId ? "active" : "")"
```

```
                @onclick="@(e => Selected(e, dish.DishId))">
                @(i++) - @dish.DishName
            </button>
        }
    }
</div>
}

@code {
    [Parameter]
    public IEnumerable<ListViewModel> Dishes { get; set; }
    [Parameter]
    public EventCallback<IEnumerable<ListViewModel>> DishesChanged {
get; set; }

    private int SelectedId = 0;
    [Parameter]
    public EventCallback<int> SelectedObjectChanged { get; set; }

    private void Selected(MouseEventArgs e, int id)
    {
        SelectedId = id;
        SelectedObjectChanged.InvokeAsync(id);
    }
}
```

Примечание:

1. *Если текущий элемент списка совпадает с выбранным, то данному элементу списка присваивается класс «active»*
2. *По событию «click» вызывается обработчик события – метод Selected. Метод принимает Id выбранного объекта*
3. *`SelectedObjectChanged` - это делегат-обработчик события выбора объекта в списке. Делегат будет передан компоненту из родительской страницы (ApiDemo.razor).*

4.6.7. Вывод информации о выбранном объекте

На странице ApiDemo добавьте обработку клика в компоненте DishesList

```
<DishesList @bind-Dishes="dishes"
SelectedObjectChanged="ShowDetails"></DishesList>
...
```

```
    ...
[Parameter]
public DetailsViewModel SelectedDish { get; set; }

private async Task ShowDetails(int id)
{
  SelectedDish = await
       client.GetFromJsonAsync<DetailsViewModel>($"api/dishes/{id}");
}
```

На странице ApiDemo добавьте компонент DishDetails:

```
<div class="container">
      <DishesList @bind-Dishes="dishes"
SelectedObjectChanged="ShowDetails"></DishesList>
      <DisDetails @bind-Dish="SelectedDish"></DisDetails>
</div>
```

Запустите основной проект и проект Blazor. На странице «Список объектов» при клике на элементе списка должна выводится подробная информация о выбранном объекте: