Лабораторная работа №4

Работа с REST API

1. Цель работы.

Знакомство с принципом работы сервисов REST.

2. Задача работы

Научиться создавать REST API сервисы и взаимодействовать с ними из приложений.

Время выполнения работы: 2 часа

3. Выполнение работы

3.1. Создание проекта

Добавьте в решение проект ASP NET Core Web API.

Название проекта XXX.API, где XXX – название вашего решения.

Добавьте в проект ссылку на проект XXX.Domain.

В данном проекте будем использовать порты **7002** и **5002**. Сделайте соответствующие изменения в файле launchSettings.json.

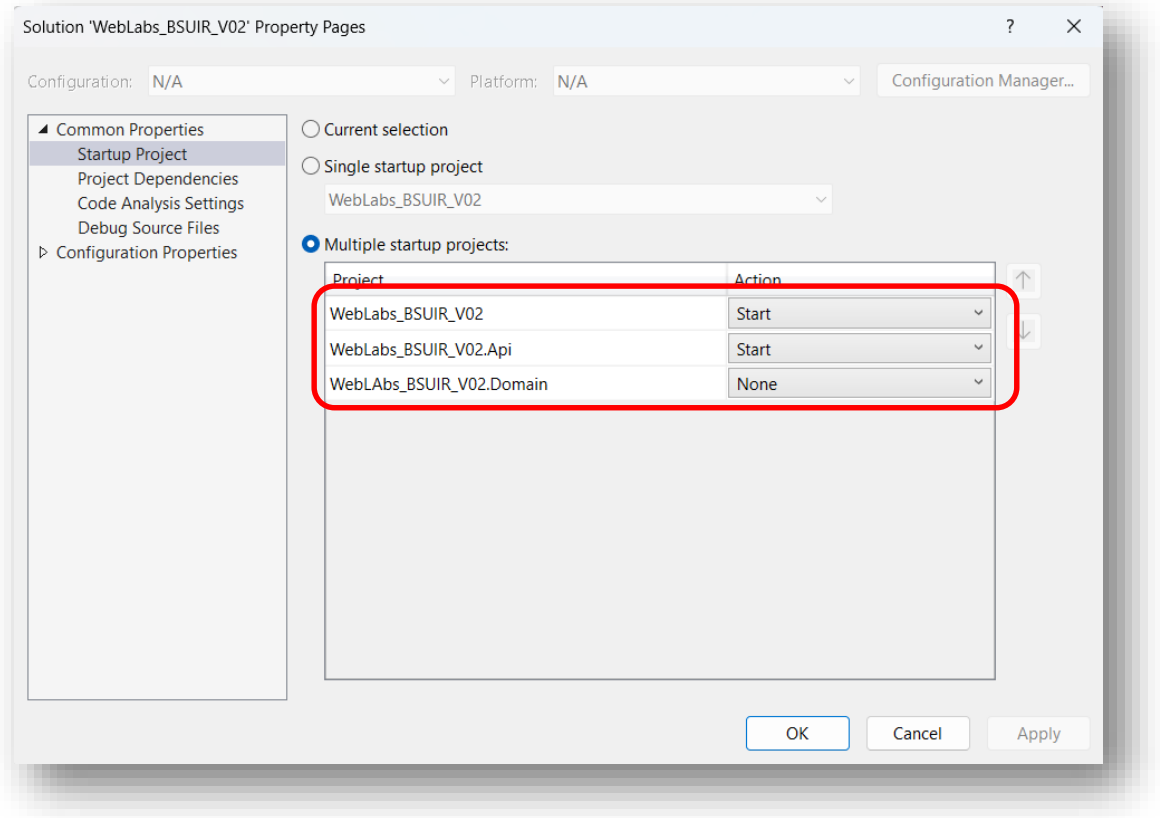
Добавьте следующие пакеты NuGet:

- Microsoft.EntityFrameworkCore
- Microsoft.EntityFrameworkCore.Sqlite (для использования SqLite. Если вы используете другую СУБД, загрузите соответствующего поставщика)
- Microsoft.EntityFrameworkCore.Tools

Добавьте в проект папку wwwroot/Images. В этой папке будут храниться файлы изображений. Скопируйте в нее файлы из соответствующей папки основного проекта.

Внимание: Для доступа к файлам изображений в классе program добавьте в конвейер Middleware компонент статических файлов.

В настройках решения укажите, чтобы запускались сразу оба проекта (такая возможность есть только в Visual Studio). В VScode проекты нужно будет запускать отдельно:



Чтобы при запуске проекта XXX.API не открывался браузер, в файле launchSettings.json закомментируйте строку

```
"launchBrowser": true,
```

3.2. Создание контекста БД

В проект XXX.API добавьте папку Data

В папку Data добавьте класс контекста базы данных с именем AppDbContext. Конструктор класса должен принимать объект DbContextOptions<AppDbContext>.

Опишите в контексте свойства DbSet, содержащие коллекции сущностей предметной области из библиотеки XXX.Domain.

3.3. Начальная миграция БД

Добавьте в файл appsettings.json секцию, описывающую строку подключения к БД. В данном примере используется база данных SQLite:

```
"ConnectionStrings": {
    "Default": "Data Source = MenuDb.db"
}
```

Зарегистрируйте созданный контекст базы данных в качестве сервиса в файле program.cs. Строку подключения к БД получите с помощью объекта builder.Configuration.

Создайте начальную миграцию.

Выполните команду Update для создания базы данных.

Убедитесь, что в проекте появился файл созданной базы данных (если использовалась база данных MS SQL, MySQL и др., убедитесь в создании базы данных с помощью соответствующих инструментов)

3.4. Заполнение базы начальными данными

В папку Data добавьте класс DbInitializer.

Опишите в классе DbInitializer статический метод для заполнения базы исходными данными:

```
public static async Task SeedData(WebApplication app)
```

Для заполнения данными можно использовать код из класса MemoryProductSerbvice основного проекта, за исключением следующего:

– данные должны записываться не в коллекции, а в контекст БД с последующим сохранением в БД

– при создании объектов не должен указываться Id объекта, т.к. он будет назначен базой данных

– изображения будут сохраняться в папке wwwroot/Images проекта XXX.Api.

3.4.1. Рекомендации к заданию 3.4.

– Контекст БД нужно получить из сервисов с помощью параметра `WebApplication app`, передаваемого в метод.

– Контекст БД – это «Scoped» сервис.

– Перед заполнением БД выполните миграцию

Пример:

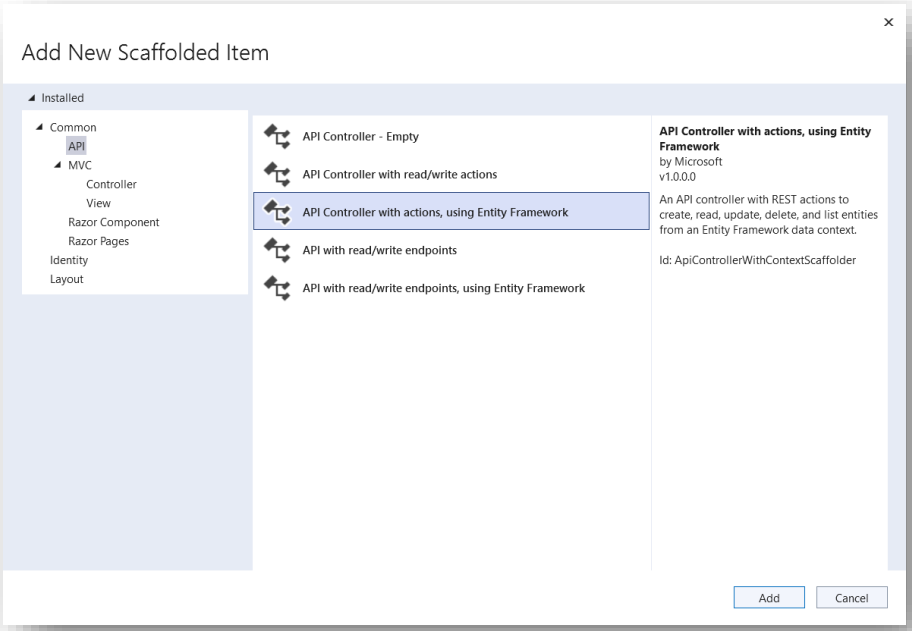
```
// Получение контекста БД
using var scope = app.Services.CreateScope();
var context =
    scope.ServiceProvider.GetRequiredService<AppDbContext>();
// Выполнение миграций
await context.Database.MigrateAsync();
```

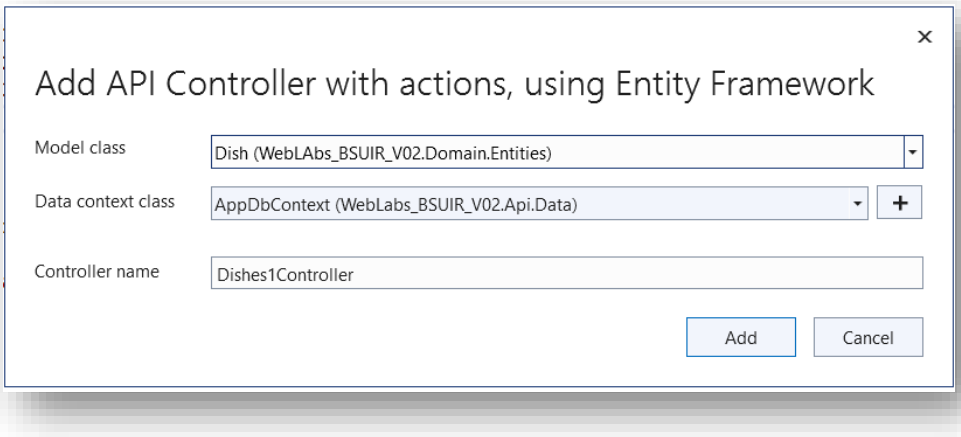
– Адрес изображения будет содержать Url приложения XXX.Api (в данном проекте это https://localhost:7002 – см. п.3.1). Добавьте в файл **appsettings.json** секцию с данным адресом. В методе SeedData получите адрес из этой секции с помощью **app.Configuration**.

3.5. Создание контроллеров API

В папку Controllers проекта XXX.Api добавьте REST Api контроллеры для ваших объектов и для категорий. Для этого:

В VisualStudio выберите Add-New scaffolded item или Add-Controller, в открывшемся окне выберите API – API controller with actions, using EntityFramework.





В VS Code или в VisualStudio for MAC:

Добавьте пакеты NuGet

dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design -v 7.0.0

dotnet add package Microsoft.EntityFrameworkCore.Design -v 7.0.0

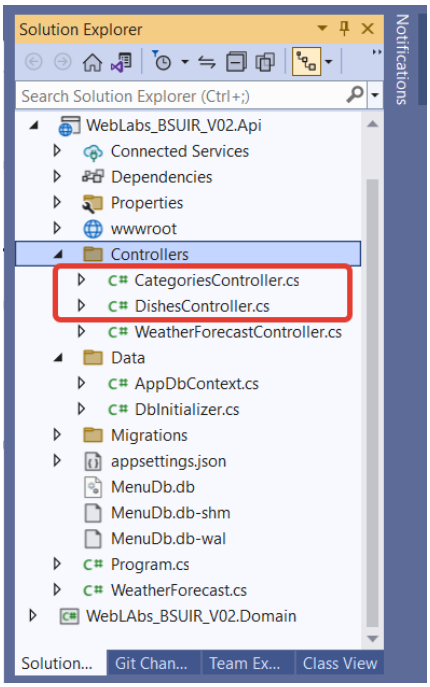
dotnet add package Microsoft.EntityFrameworkCore.SqlServer -v 7.0.0

dotnet tool uninstall -g dotnet-aspnet-codegenerator

dotnet tool install -g dotnet-aspnet-codegenerator

Выполните команду:

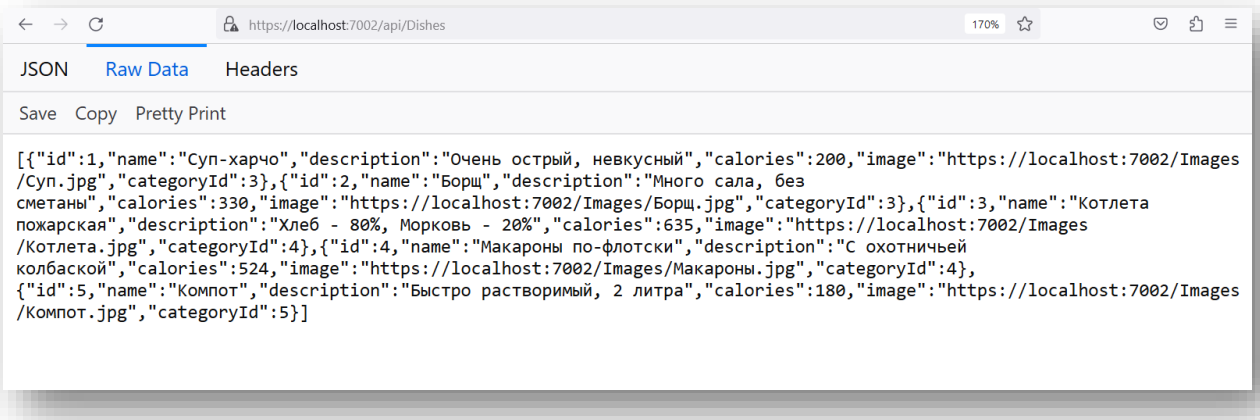
```
dotnet aspnet-codegenerator controller -name [ИМЯ КОНТРОЛЛЕРА] -async -api -m [КЛАСС МОДЕЛИ] -dc [КОНТЕКСТ БД] -outDir Controllers
```



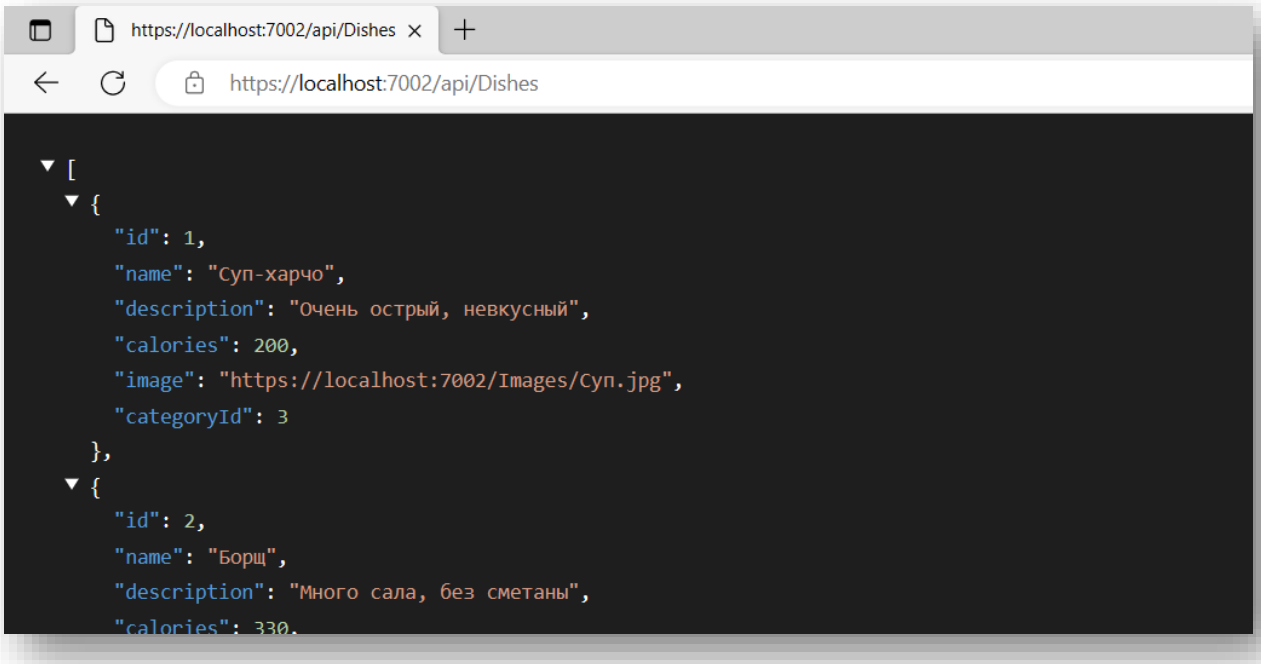
3.6. Проверка API

Запустите проект.

В адресной строке браузера наберите адреса Api объектов и категорий. Убедитесь, что браузер получает данные в формате Json:



Или:



Для проверки функций добавления/удаления/редактирования можно воспользоваться программами Postman, Insomnia и др.

3.7. Регистрация сервисов

Задание: контроллеры API не должны взаимодействовать с контекстом базы данных напрямую. Данные должны получаться с помощью сервисов.

В проект XXX.Api добавьте папку Services. В папке Services опишите интерфейсы сервисов, аналогичные сервисам из основного проекта.

```
public interface ICategoryService
{
    /// <summary>
    /// Получение списка всех категорий
    /// </summary>
    /// <returns></returns>
    public Task<ResponseData<List<Category>>> GetCategoryListAsync();
}


public interface IProductService
{
    /// <summary>
    /// Получение списка всех объектов
    /// </summary>
    /// <param name="categoryNormalizedName">нормализованное имя категории для
фильтрации</param>
    /// <param name="pageNo">номер страницы списка</param>
    /// <param name="pageSize">количество объектов на странице</param>
    /// <returns></returns>
    public Task<ResponseData<ListModel<Dish>>> GetProductListAsync(
                                                string? categoryNormalizedName,
```

```
                                        int pageNo=1,
                                        int pageSize=3);

    /// <summary>
    /// Поиск объекта по Id
    /// </summary>
    /// <param name="id">Идентификатор объекта</param>
    /// <returns></returns>
    public Task<ResponseData<Dish>> GetProductByIdAsync(int id);

    /// <summary>
    /// Обновление объекта
    /// </summary>
    /// <param name="id">Id изменяемомго объекта</param>
    /// <param name="product">объект с новыми параметрами</param>
    /// <returns></returns>
    public Task UpdateProductAsync(int id, Dish product);
    /// <summary>
    /// Удаление объекта
    /// </summary>
    /// <param name="id">Id удаляемомго объекта</param>
    /// <returns></returns>
    public Task DeleteProductAsync(int id);
    /// <summary>
    /// Создание объекта
    /// </summary>
    /// <param name="product">Новый объект</param>
    /// <returns>Созданный объект</returns>
    public Task<ResponseData<Dish>> CreateProductAsync(Dish product);
    /// <summary>
    /// Сохранить файл изображения для объекта
    /// </summary>
    /// <param name="id">Id объекта</param>
    /// <param name="formFile">файл изображения</param>
    /// <returns>Url к файлу изображения</returns
    public Task<ResponseData<string>> SaveImageAsync(int id, IFormFile formFile);
}
```

Добавьте файлы CategoryService и ProductService, реализующие интерфейсы ICategoryService и IProductService соответственно.

В классе ProductService предусмотрите ограничение на размер страницы:

```
// максимальный размер страницы
private readonly int _maxPageSize = 20;
```

Зарегистрируйте сервисы в классе Program проекта XXX.API

Пример реализации метода получения списка объектов:

```
public async Task<ResponseData<ListModel<Dish>>> GetProductListAsync(
        string? categoryNormalizedName,
        int pageNo = 1,
```

```
        int pageSize = 3)
    {
        if (pageSize > _maxPageSize)
            pageSize = _maxPageSize;


        var query = _context.Dishes.AsQueryable();
        var dataList = new ListModel<Dish>();

        query = query
            .Where(d=> categoryNormalizedName==null
                       ||
d.Category.NormalizedName.Equals(categoryNormalizedName));
        // количество элементов в списке
        var count = await query.CountAsync(); //.Count();
        if(count==0)
        {
            return ResponseData<ListModel<Dish>>.Success(dataList);

        }
        // количество страниц
        int totalPages = (int)Math.Ceiling(count / (double)pageSize);

        if (pageNo > totalPages)
            return ResponseData<ListModel<Dish>>.Error("No such page");

        dataList.Items = await query
                            .OrderBy(d=>d.Id)
                            .Skip((pageNo - 1) * pageSize)
                            .Take(pageSize)
                            .ToListAsync();
        dataList.CurrentPage = pageNo;
        dataList.TotalPages = totalPages;

        return ResponseData<ListModel<Dish>>.Success(dataList);
    }
```

3.8. Доработка кода контроллеров API

Измените код контроллеров CategoryController и ProductController так, чтобы контроллеры получали необходимые данные из сервисов.

Методы контроллеров должны возвращать объекты класса **ResponseData** (см. п.3.2. лабораторной работы №3)

> **Примечание:** в рамках лабораторных заданий в контроллере Category достаточно реализовать только метод получения списка категорий

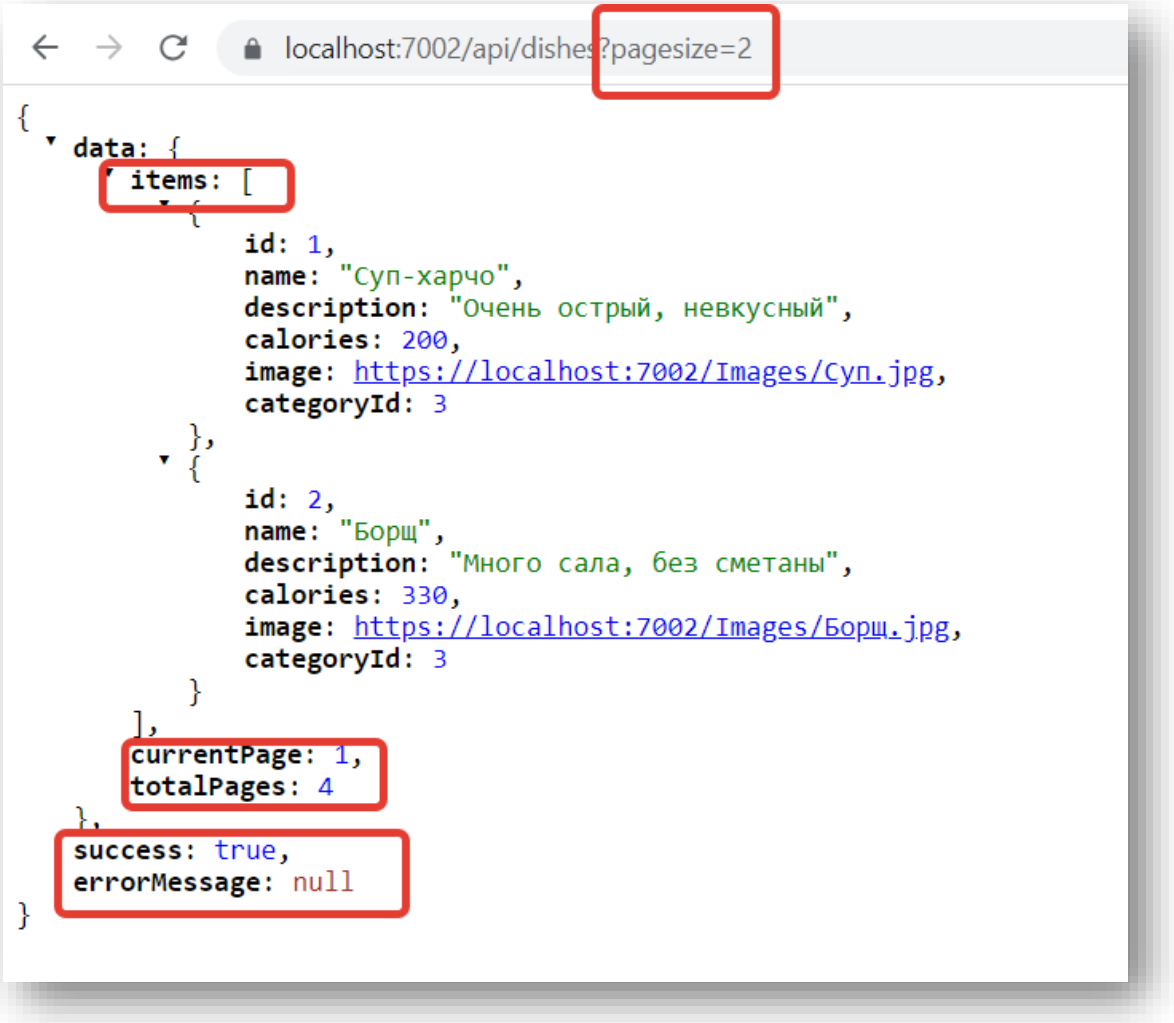
Пример метода GetProducts контроллера Product:

```
// GET: api/Dishes
[HttpGet]
public async Task<ActionResult<ResponseData<List<Dish>>>> GetDishes(
                                        string? category,
                                        int pageNo = 1,
                                        int pageSize = 3)
{
       return Ok(await _productService.GetProductListAsync(
                                    category,
                                    pageNo,
                                    pageSize));
}
```

3.9. Проверка API

Запустите проект.

В адресной строке браузера наберите адреса Api объектов и категорий. Убедитесь, что браузер получает данные в формате Json:



3.10. Регистрация маршрута

Зарегистрируйте маршрут к методу Get, чтобы имя категории объекта передавалось как сегмент маршрута:

https://localhost:7001/menu/main-dishes?pageno=2

3.11. Получение данных с API в основном проекте

3.11.1. Подготовка проекта

Для обращения к Api понадобится базовый адрес Api-сервиса. Для хранения этой информации опишите **в проекте XXX.UI** класс UriData:

```
public class UriData
{
	public string ApiUri { get; set; }=string.Empty;
}
```

ApiUri – адрес Api – сервиса;

В файле appsettings.json добавьте раздел с адресами сервисов:

```
"UriData": {
    "ApiUri": "https://localhost:7002/api/"
}
```

В классе Program получите **UriData** из объекта IConfiguration.

3.11.2. Создание сервиса для запросов к Api

В проекте XXX.UI опишите класс ApiCategoryService и ApiProductService, реализующие интерфейсы ICategoryService и IProductService посредством взаимодействия с API.

Для отправки запросов понадобится объект IHttpClient. Зарегистрируйте в классе program клиента для IProductService и ICategoryService, например:

```
builder.Services
     .AddHttpClient<IProductService, ApiProductService>(opt=>
               opt.BaseAddress=new Uri(UriData.ApiUri));
```

Пример конструктора:

```
public ApiProductService(HttpClient httpClient,
                         IConfiguration configuration,
                         ILogger<ApiProductService> logger)
{
       _httpClient = httpClient;
       _pageSize = configuration.GetSection("ItemsPerPage").Value;
       _serializerOptions = new JsonSerializerOptions()
            {
                PropertyNamingPolicy = JsonNamingPolicy.CamelCase
            };
       _logger = logger;
}
```

Пример чтения данных:

```
public async Task<ResponseData<ListModel<Dish>>> GetProductListAsync(
                                 string? categoryNormalizedName,
                                 int pageNo = 1)
{
     // подготовка URL запроса
     var urlString
          = new
     StringBuilder($"{_httpClient.BaseAddress.AbsoluteUri}dishes/");
     // добавить категорию в маршрут
     if (categoryNormalizedName != null)
     {
          urlString.Append($"{categoryNormalizedName}/");
```

```
        };
        // добавить номер страницы в маршрут
        if (pageNo > 1)
        {
            urlString.Append($"page{pageNo}");
        };
        // добавить размер страницы в строку запроса
        if (!_pageSize.Equals("3"))
        {
            urlString.Append(QueryString.Create("pageSize", _pageSize));
        }

        // отправить запрос к API
        var response = await _httpClient.GetAsync(
                                        new Uri(urlString.ToString()));

        if(response.IsSuccessStatusCode)
        {
            try
            {
                return await response
                    .Content
                    .ReadFromJsonAsync<ResponseData<ListModel<Dish>>>
                                                (_serializerOptions);
            }
            catch(JsonException ex)
            {
                _logger.LogError($"-----> Ошибка: {ex.Message}");

                return ResponseData<ListModel<Dish>>
                            .Error($"Ошибка: {ex.Message}")
            }
        }
        _logger.LogError($"-----> Данные не получены от сервера. Error:
{response.StatusCode.ToString()}");
        return ResponseData<ListModel<Dish>>
            .Error($"Данные не получены от сервера. Error:
        {response.StatusCode.ToString()}")
}
```

Пример записи данных:

```
public async Task<ResponseData<Dish>> CreateProductAsync(
                                                Dish product,
                                                IFormFile? formFile)
{
        var uri = new Uri(_httpClient.BaseAddress.AbsoluteUri + "Dishes");

        var response = await _httpClient.PostAsJsonAsync(
                                                uri,
                                                product,
                                                _serializerOptions);
        if (response.IsSuccessStatusCode)
           {
            var data = await response
```

```
                .Content
                .ReadFromJsonAsync<ResponseData<Dish>>
                (_serializerOptions);

            return data; // dish;
        }
        _logger
              .LogError($"-----> object not created. Error:
        {response.StatusCode.ToString()}");
        return ResponseData<Dish>.Error($"Объект не добавлен. Error:
{response.StatusCode.ToString()}")
}
```

Запустите проект. Убедитесь, что данные на странице «Каталог» отображаются правильно.

4. Контрольные вопросы

1. Чем контроллер API отличается от обычного контроллера?
2. Как осуществляется выбор метода (Action) контроллера API при обработке запроса Http?
3. Где в запросе и в каком виде передаются данные в контроллер API?
4. Как в коде получить Scoped сервис?
5. Что могут возвращать методы контроллера API?
6. Что такое Minimal API?
7. Как зарегистрировать конечную точку для Minimal API?
8. Как зарегистрировать группу конечных точек для Minimal API?