Оглавление

[Техническое задание «Разработка веб-API для сервиса приема заказов покупателей». 2](#_Toc167177700)

[Контроллеры 6](#_Toc167177701)

[Создание сущностей. 10](#_Toc167177702)

[Создание миграции 11](#_Toc167177703)

[Создание интерфейсов 13](#_Toc167177704)

[Методы для работы с базой данных RDB 14](#_Toc167177705)

[DBContext 18](#_Toc167177706)

[Файл настройки соединения с базой данных 18](#_Toc167177707)

[Program (запуск приложения) 18](#_Toc167177708)

[Ссылка на GitHub 19](#_Toc167177709)

# Техническое задание «Разработка веб-API для сервиса приема заказов покупателей».

1. Формулировка задания.

**Товар** представляет собой объект товара, который можно заказать на нашем сайте. Товар имеет поля «Название», «Стоимость», «Присутствует ли в наличии на складе».

**Товар В заказе** представляет собой объект, ссылающийся на товар и на заказ, а также содержащий поле «кол-во товаров в заказе»

**Заказ** представляет собой заявку на покупку товара. Включает в себя поля «уникальный шифр заказа», «имя клиента», «дата формирования заказа», «список товаров заказа».

api должен позволять:

* добавлять/удалять/редактировать/получать список товаров
* получать товар по id
* получать товар по названию
* формировать заказы на товары
* получать список всех заказов
* получать заказ по id
* удалять заказ по id

2. Формат общения

api осуществляет общение с клиентом посредством JSON, следовательно является JSON-api. Используется маршрутизация на основе url и методов запросов.

3. Стек технологий

C# + ASP

EF

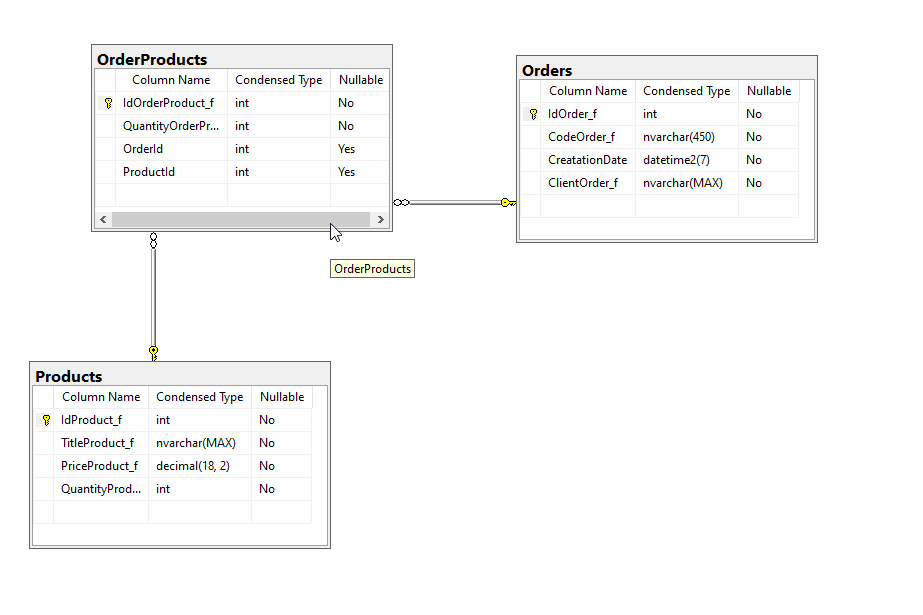
MS SQL Server / MySQL / PostgreSQL

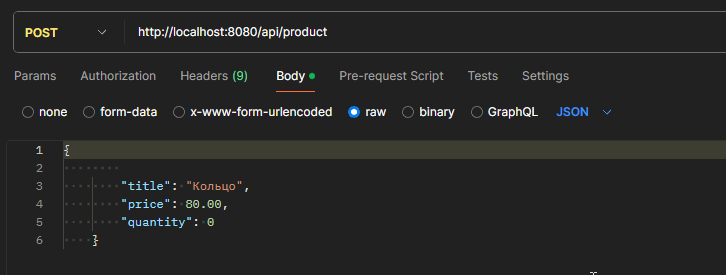
**Порядок сдачи.**

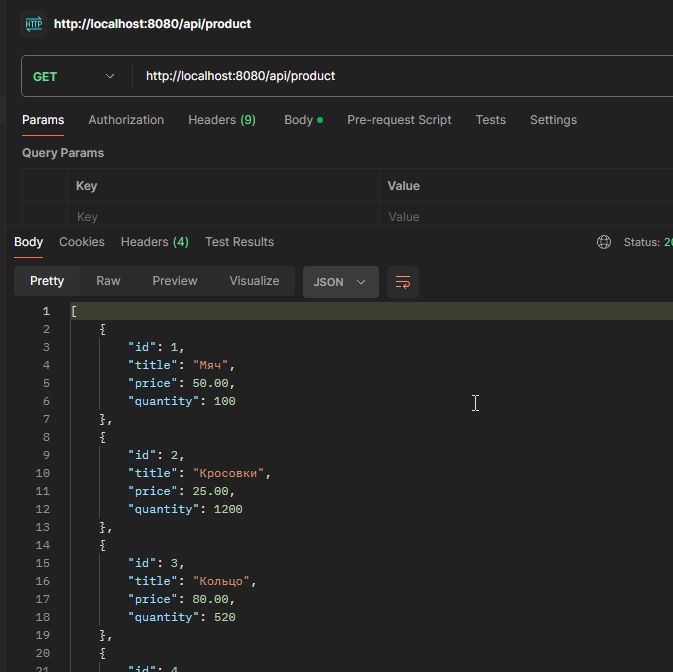
1. Работа выполняется в течение двух часов (120 минут)

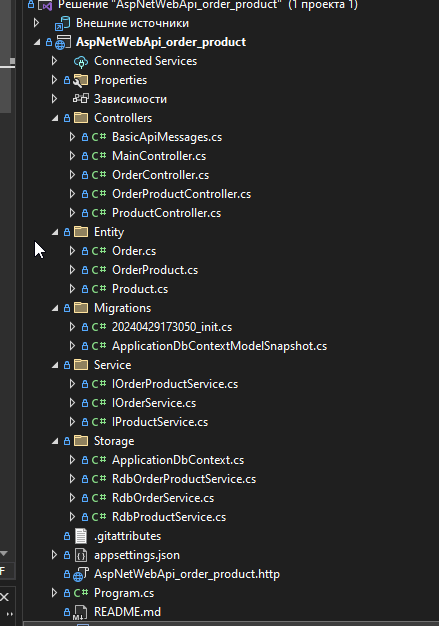
2. Студент предоставляет выполненную работу, демонстрируя программу, исходный код, БД

3. Студент отвечает на вопросы защиты









# Контроллеры

1. Проверка работы сервера

using Microsoft.AspNetCore.Mvc;

using static AspNetWebApi\_order\_product.Controllers.BasicApiMessages;

namespace AspNetWebApi\_order\_product.Controllers

[Route("api")]

[ApiController]

public class MainController : ControllerBase

{

[HttpGet]

public StringMessage Index()

{

int? port = HttpContext.Request.Host.Port;

return new StringMessage(message: $"server is running on port {port}");

}

[HttpGet("ping")]

public StringMessage Ping()

{

return new StringMessage(message: "pong");

}

}

}

1. Контроллер заказов

using AspNetWebApi\_order\_product.Entity;

using AspNetWebApi\_order\_product.Service;

using Microsoft.AspNetCore.Mvc;

namespace AspNetWebApi\_order\_product.Controllers

{

[Route("api/order")]

[ApiController]

public class OrderController : ControllerBase

{

private readonly IOrderService \_orderService;

public OrderController(IOrderService orderService)

{

\_orderService = orderService;

}

[HttpGet]

public async Task<List<Order>> ListAll()

{

return await \_orderService.ListAll();

}

[HttpGet("{id:int}")]

public async Task<Order?> GetById(int id)

{

Order? order = await \_orderService.GetById(id);

if (order == null)

{

HttpContext.Response.StatusCode = StatusCodes.Status404NotFound;

}

return order;

}

[HttpPost]

public async Task<Order?> Add(Order order)

{

Order? result = await \_orderService.Add(order);

if (result == null)

{

HttpContext.Response.StatusCode = StatusCodes.Status400BadRequest;

}

return result;

}

[HttpDelete("{id:int}")]

public async Task<Order?> RemoveById(int id)

{

Order? order = await \_orderService.RemoveById(id);

if (order == null)

{

HttpContext.Response.StatusCode = StatusCodes.Status404NotFound;

}

return order;

}

[HttpPatch("{id:int}")]

public async Task<Order?> UpdateById(int id, Order order)

{

Order? updated = await \_orderService.UpdateById(id, order);

if (updated == null)

{

HttpContext.Response.StatusCode = StatusCodes.Status400BadRequest;

}

return updated;

}

}

}

1. Контроллер продуктов

using AspNetWebApi\_order\_product.Entity;

using AspNetWebApi\_order\_product.Service;

using Microsoft.AspNetCore.Mvc;

namespace AspNetWebApi\_order\_product.Controllers

{

[Route("api/product")]

[ApiController]

public class ProductController : ControllerBase

{

private readonly IProductService \_productService;

public ProductController(IProductService productService)

{

\_productService = productService;

}

[HttpGet]

public async Task<List<Product>> ListAll()

{

return await \_productService.ListAll();

}

[HttpGet("{id:int}")]

public async Task<Product?> GetById(int id)

{

Product? product = await \_productService.GetById(id);

if (product == null)

{

HttpContext.Response.StatusCode = StatusCodes.Status404NotFound;

}

return product;

}

[HttpPost]

public async Task<Product?> Add(Product product)

{

Product? result = await \_productService.Add(product);

if (result == null)

{

HttpContext.Response.StatusCode = StatusCodes.Status400BadRequest;

}

return result;

}

[HttpDelete("{id:int}")]

public async Task<Product?> RemoveById(int id)

{

Product? product = await \_productService.RemoveById(id);

if (product == null)

{

HttpContext.Response.StatusCode = StatusCodes.Status404NotFound;

}

return product;

}

[HttpPatch("{id:int}")]

public async Task<Product?> UpdateById(int id, Product product)

{

Product? updated = await \_productService.UpdateById(id, product);

if (updated == null)

{

HttpContext.Response.StatusCode = StatusCodes.Status400BadRequest;

}

return updated;

}

}

}

1. Таблица связи заказы – продукты

using AspNetWebApi\_order\_product.Entity;

using AspNetWebApi\_order\_product.Service;

using Microsoft.AspNetCore.Mvc;

namespace AspNetWebApi\_order\_product.Controllers

{

[Route("api/orderproduct")]

[ApiController]

public class OrderProductController : ControllerBase

{

private readonly IOrderProductService \_orderProductService;

public OrderProductController(IOrderProductService orderProductService)

{

// IoC-контейнер при создании контроллера

// автоматически добавит нужную зависимость (если она есть)

\_orderProductService = orderProductService;

}

[HttpGet]

public async Task<List<OrderProduct>> ListAll()

{

return await \_orderProductService.ListAll();

}

[HttpGet("{id:int}")]

public async Task<OrderProduct?> GetById(int id)

{

OrderProduct? orderProduct = await \_orderProductService.GetById(id);

if (orderProduct == null)

{

HttpContext.Response.StatusCode = StatusCodes.Status404NotFound;

}

return orderProduct;

}

[HttpPost]

public async Task<OrderProduct?> Add(OrderProduct orderProduct)

{

OrderProduct? result = await \_orderProductService.Add(orderProduct);

if (result == null)

{

HttpContext.Response.StatusCode = StatusCodes.Status400BadRequest;

}

return result;

}

[HttpDelete("{id:int}")]

public async Task<OrderProduct?> RemoveById(int id)

{

OrderProduct? orderProduct = await \_orderProductService.RemoveById(id);

if (orderProduct == null)

{

HttpContext.Response.StatusCode = StatusCodes.Status404NotFound;

}

return orderProduct;

}

[HttpPatch("{id:int}")]

public async Task<OrderProduct?> UpdateById(int id, OrderProduct orderProduct)

{

OrderProduct? updated = await \_orderProductService.UpdateById(id, orderProduct);

if (updated == null)

{

HttpContext.Response.StatusCode = StatusCodes.Status400BadRequest;

}

return updated;

}

}

}

# Создание сущностей.

1. Заказы

using Microsoft.EntityFrameworkCore;

using System.ComponentModel.DataAnnotations.Schema;

using System.Net.Mail;

using System.Text.Json.Serialization;

namespace AspNetWebApi\_order\_product.Entity

{

[Index(nameof(СodeOrder), IsUnique = true)]

public class Order

{

[Column("IdOrder\_f")]

public int Id { get; set; }

[Column("СodeOrder\_f")]

public string СodeOrder { get; set; }

[Column("CreatationDate")]

public DateTime CreatationDate { get; set; } = DateTime.Now;

[Column("ClientOrder\_f")]

public string Client { get; set; }

[JsonIgnore]

public ICollection<OrderProduct>? OrderProducts { get; set; }

public Order()

{

Client = string.Empty;

}

public override string ToString()

{

return $"{Id} - {СodeOrder} - {CreatationDate} - {Client}";

}

}

}

1. Продукты

using System.ComponentModel.DataAnnotations.Schema;

using System.Text.Json.Serialization;

namespace AspNetWebApi\_order\_product.Entity

{

public class Product

{

[Column("IdProduct\_f")]

public int Id { get; set; }

[Column("TitleProduct\_f")]

public string Title { get; set; }

[Column("PriceProduct\_f")]

public decimal Price { get; set; }

[Column("QuantityProduct\_f")]

public int Quantity { get; set; }

[JsonIgnore]

public ICollection<OrderProduct>? OrderProducts { get; set; }

public Product()

{

Title = string.Empty;

Price = 0;

Quantity = 0;

}

public override string ToString()

{

return $"{Id} - {Title} - {Price} - {Quantity}";

}

}

}

1. Заказы – Продукты

using System.ComponentModel.DataAnnotations.Schema;

using System.Text.Json.Serialization;

namespace AspNetWebApi\_order\_product.Entity

{

public class OrderProduct

{

[Column("IdOrderProduct\_f")]

public int Id { get; set; }

[Column("QuantityOrderProduct\_f")]

public int Quantity { get; set; }

[JsonIgnore]

public Order? Order { get; set; }

public Product? Product { get; set; }

public override string ToString()

{

return $"{Id} - {Quantity}";

}

}

}

# Создание миграции

using System;

using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace AspNetWebApi\_order\_product.Migrations

{

/// <inheritdoc />

public partial class init : Migration

{

/// <inheritdoc />

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.CreateTable(

name: "Orders",

columns: table => new

{

IdOrder\_f = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

СodeOrder\_f = table.Column<string>(type: "nvarchar(450)", nullable: false),

CreatationDate = table.Column<DateTime>(type: "datetime2", nullable: false),

ClientOrder\_f = table.Column<string>(type: "nvarchar(max)", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Orders", x => x.IdOrder\_f);

});

migrationBuilder.CreateTable(

name: "Products",

columns: table => new

{

IdProduct\_f = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

TitleProduct\_f = table.Column<string>(type: "nvarchar(max)", nullable: false),

PriceProduct\_f = table.Column<decimal>(type: "decimal(18,2)", nullable: false),

QuantityProduct\_f = table.Column<int>(type: "int", nullable: false)

},

constraints: table =>

{

table.PrimaryKey("PK\_Products", x => x.IdProduct\_f);

});

migrationBuilder.CreateTable(

name: "OrderProducts",

columns: table => new

{

IdOrderProduct\_f = table.Column<int>(type: "int", nullable: false)

.Annotation("SqlServer:Identity", "1, 1"),

QuantityOrderProduct\_f = table.Column<int>(type: "int", nullable: false),

OrderId = table.Column<int>(type: "int", nullable: true),

ProductId = table.Column<int>(type: "int", nullable: true)

},

constraints: table =>

{

table.PrimaryKey("PK\_OrderProducts", x => x.IdOrderProduct\_f);

table.ForeignKey(

name: "FK\_OrderProducts\_Orders\_OrderId",

column: x => x.OrderId,

principalTable: "Orders",

principalColumn: "IdOrder\_f");

table.ForeignKey(

name: "FK\_OrderProducts\_Products\_ProductId",

column: x => x.ProductId,

principalTable: "Products",

principalColumn: "IdProduct\_f");

});

migrationBuilder.CreateIndex(

name: "IX\_OrderProducts\_OrderId",

table: "OrderProducts",

column: "OrderId");

migrationBuilder.CreateIndex(

name: "IX\_OrderProducts\_ProductId",

table: "OrderProducts",

column: "ProductId");

migrationBuilder.CreateIndex(

name: "IX\_Orders\_СodeOrder\_f",

table: "Orders",

column: "СodeOrder\_f",

unique: true);

}

/// <inheritdoc />

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.DropTable(

name: "OrderProducts");

migrationBuilder.DropTable(

name: "Orders");

migrationBuilder.DropTable(

name: "Products");

}

}

}

# Создание интерфейсов

1. Заказы

using AspNetWebApi\_order\_product.Entity;

namespace AspNetWebApi\_order\_product.Service

{

public interface IOrderService

{

Task<Order?> Add(Order order);

Task<List<Order>> AddRange(List<Order> orders);

Task<Order?> GetById(int id);

Task<List<Order>> ListAll();

Task<Order?> RemoveById(int id);

Task<Order?> UpdateById(int id, Order order);

}

}

1. Продукты

using AspNetWebApi\_order\_product.Entity;

namespace AspNetWebApi\_order\_product.Service

{

public interface IProductService

{

Task<Product?> Add(Product product);

Task<Product?> GetById(int id);

Task<List<Product>> ListAll();

Task<Product?> RemoveById(int id);

Task<Product?> UpdateById(int id, Product product);

}

}

1. Заказы – Продукты

using AspNetWebApi\_order\_product.Entity;

namespace AspNetWebApi\_order\_product.Service

{

public interface IOrderProductService

{

Task<OrderProduct?> Add(OrderProduct orderProduct);

Task<OrderProduct?> GetById(int id);

Task<List<OrderProduct>> ListAll();

Task<OrderProduct?> RemoveById(int id);

Task<OrderProduct?> UpdateById(int id, OrderProduct orderProduct);

}

}

# Методы для работы с базой данных RDB

1. Заказы

using AspNetWebApi\_order\_product.Entity;

using AspNetWebApi\_order\_product.Service;

using Microsoft.EntityFrameworkCore;

namespace AspNetWebApi\_order\_product.Storage

{

public class RdbOrderService : IOrderService

{

private readonly ApplicationDbContext \_db;

public RdbOrderService(ApplicationDbContext db)

{

\_db = db;

}

public async Task<Order?> Add(Order order)

{

\_db.Orders.Add(order);

await \_db.SaveChangesAsync();

return order;

}

public async Task<List<Order>> AddRange(List<Order> orders)

{

await \_db.Orders.AddRangeAsync(orders);

await \_db.SaveChangesAsync();

return orders;

}

public async Task<Order?> GetById(int id)

{

return await \_db.Orders.FirstOrDefaultAsync(order => order.Id == id);

}

public async Task<List<Order>> ListAll()

{

return await \_db.Orders.ToListAsync();

}

public async Task<Order?> RemoveById(int id)

{

Order? removed = await \_db.Orders.FirstOrDefaultAsync(order => order.Id == id);

if (removed != null)

{

\_db.Orders.Remove(removed);

await \_db.SaveChangesAsync();

}

return removed;

}

public async Task<Order?> UpdateById(int id, Order order)

{

Order? updated = await \_db.Orders.FirstOrDefaultAsync(order => order.Id == id);

if (updated != null)

{

updated.СodeOrder = order.СodeOrder;

await \_db.SaveChangesAsync();

}

return updated;

}

}

}

1. Продукты

using AspNetWebApi\_order\_product.Entity;

using AspNetWebApi\_order\_product.Service;

using Microsoft.EntityFrameworkCore;

namespace AspNetWebApi\_order\_product.Storage

{

public class RdbProductService : IProductService

{

private readonly ApplicationDbContext \_db;

public RdbProductService(ApplicationDbContext db)

{

\_db = db;

}

public async Task<Product?> Add(Product product)

{

\_db.Products.Add(product);

await \_db.SaveChangesAsync();

return product;

}

public Task<Product?> GetById(int id)

{

return \_db.Products.FirstOrDefaultAsync(product => product.Id == id);

}

public async Task<List<Product>> ListAll()

{

return await \_db.Products.ToListAsync();

}

public async Task<Product?> RemoveById(int id)

{

Product? removed = await \_db.Products.FirstOrDefaultAsync(product => product.Id == id);

if (removed != null)

{

\_db.Products.Remove(removed);

await \_db.SaveChangesAsync();

}

return removed;

}

public async Task<Product?> UpdateById(int id, Product product)

{

Product? update = await \_db.Products.FirstOrDefaultAsync(product => product.Id == id);

if (update != null)

{

update.Price = product.Price;

update.Quantity = product.Quantity;

update.Title = product.Title;

update.Quantity = product.Quantity;

await \_db.SaveChangesAsync();

}

return update;

}

}

}

1. Заказы – Продукты

using AspNetWebApi\_order\_product.Entity;

using AspNetWebApi\_order\_product.Service;

using Microsoft.EntityFrameworkCore;

namespace AspNetWebApi\_order\_product.Storage

{

public class RdbOrderProductService : IOrderProductService

{

private readonly ApplicationDbContext \_db;

public RdbOrderProductService(ApplicationDbContext db)

{

\_db = db;

}

public async Task<OrderProduct?> Add(OrderProduct orderProduct)

{

\_db.OrderProducts.Add(orderProduct);

await \_db.SaveChangesAsync();

return orderProduct;

}

public async Task<OrderProduct?> GetById(int id)

{

return await \_db.OrderProducts.FirstOrDefaultAsync(OrderProduct => OrderProduct.Id == id);

}

public async Task<List<OrderProduct>> ListAll()

{

return await \_db.OrderProducts.ToListAsync();

}

public async Task<OrderProduct?> RemoveById(int id)

{

OrderProduct? removed = await \_db.OrderProducts.FirstOrDefaultAsync(OrderProduct => OrderProduct.Id == id);

if (removed != null)

{

\_db.OrderProducts.Remove(removed);

await \_db.SaveChangesAsync();

}

return removed;

}

public async Task<OrderProduct?> UpdateById(int id, OrderProduct orderProduct)

{

OrderProduct? updated = await \_db.OrderProducts.FirstOrDefaultAsync(OrderProduct => orderProduct.Id == id);

if (updated != null)

{

updated.Quantity = orderProduct.Quantity;

await \_db.SaveChangesAsync();

}

return updated;

}

}

}

# DBContext

using AspNetWebApi\_order\_product.Entity;

using Microsoft.EntityFrameworkCore;

namespace AspNetWebApi\_order\_product.Storage

{

public class ApplicationDbContext:DbContext

{

public DbSet<Product> Products { get; set; }

public DbSet<Order> Orders { get; set; }

public DbSet<OrderProduct> OrderProducts { get; set; }

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

IConfigurationRoot configuration = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json")

.Build();

string useConnection = configuration.GetSection("UseConnection").Value ?? "DefaultConnection";

string? connectionString = configuration.GetConnectionString(useConnection);

optionsBuilder.UseSqlServer(connectionString);

}

}

}

# Файл настройки соединения с базой данных

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"UseConnection": "LocalDbConnection",

"ConnectionStrings": {

"LocalDbConnection": "Data Source=FISHMAN\\SQLEXPRESS; Initial Catalog=client\_order\_db; Integrated Security=SSPI; Timeout=5; TrustServerCertificate=true"

}

}

# Program (запуск приложения)

using AspNetWebApi\_order\_product.Service;

using AspNetWebApi\_order\_product.Storage;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddControllers(); // добавление классов контроллеров в IoC-контейнер

builder.Services.AddDbContext<ApplicationDbContext>(); // добавим DbContext

builder.Services.AddTransient<IOrderService, RdbOrderService>();

builder.Services.AddTransient<IProductService, RdbProductService>();

builder.Services.AddTransient<IOrderProductService, RdbOrderProductService>();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseAuthorization();

app.MapControllers();

app.Run();

# Ссылка на GitHub

https://github.com/fishman123456/AspNetWebApi\_order\_product