Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Ивановский государственный энергетический университет имени В.И. Ленина»

Кафедра программного обеспечения компьютерных систем

Отчёт по лабораторной работе №3

Конструирование интернет-приложений

Подключение к базе данных.

Выполнила студент гр. 3-42 _____ Шарабанов Н.А.

Проверил _____ Садыков А.М.

Иваново 2022

Цель лабораторной работы: создать базу данных по объектной модели использованием технологии доступа к данным Entity Framework Core.

Задания:

1. Добавить подключение к базе данных

2. Добавить подключение базы данных Microsoft SQL Server

3. Создать базу данных

4. Добавить инициализацию базы данных с предопределенными данными

5. Добавить загрузку связанных данных

5.1 Исключить ошибки зацикливания для связанных объектов

5.2. Добавить загрузку связанных данных в методе контроллера

6. Реконструировать модель по существующей БД*

1. Результаты добавления подключения к базе данных

За основу был взят проект веб-API Лаб.работы №2. В файле appsettings.json была добавлена срока подключения к базе данных:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DefaultConnection": "Data Source=DESKTOP-T0L4JP9;Database=InternetShopWebBD;Integrated Security=True;Connect Timeout=30;Encrypt=False;Trust Server Certificate=False;Application Intent=ReadWrite;Multi Subnet Failover=False"
  }
}
```

2. Результат добавления подключения базы данных Microsoft SQL Server

Были установлены зависимость Microsoft.EntityFrameworkCore.SqlServer. После этого в контекст внесены изменения, добавлена информация о подключении к БД. (Рис.1)

```
using InternetShopWebApp.Models;
using Microsoft.EntityFrameworkCore;

namespace InternetShopWebApp.Context
{
    public class ShopContext : DbContext
    {
        protected readonly IConfiguration Configuration;
        #region Constructors
        //public Context(DbContextOptions<Context> options) : base(options) { }
        public ShopContext(IConfiguration configuration)
        {
            Configuration = configuration;
        }
        #endregion
        protected override void OnConfiguring(DbContextOptionsBuilder options)
        {
            options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));
        }
        public virtual DbSet<ProductModel> Product { get; set; }
        public virtual DbSet<CathegoryModel> Cathegory { get; set; }
        public virtual DbSet<OrderItemModel> OrderItem { get; set; }
        protected override void OnModelCreating(ModelBuilder
        modelBuilder)
        {
            modelBuilder.Entity<ProductModel>(entity =>
            {
                //entity.Property(e => e.Product_Code).IsRequired();
                entity.HasOne(d => d.ProductOrderItem)
                .WithMany(p => p.Products)
                .HasForeignKey(d => d.OrderItem_Code);
            });
```

Рисунок 2 – Информация о подключении к БД в классе контекста

В program.cs изменена строка добавления контекста БД. Результат показан на рисунке 2.

```
using InternetShopWebApp.Context;
using InternetShopWebApp.Data;
using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

//builder.Services.AddDbContext<Context>(opt =>
//opt.UseInMemoryDatabase("Shop"));

builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
//builder.Services.AddDbContext<ProductContext>(opt =>
//opt.UseModel(ProductModel));
builder.Services.AddDbContext<ShopContext>();
builder.Services.AddControllers().AddJsonOptions(x =>
x.JsonSerializerOptions.ReferenceHandler =
ReferenceHandler.IgnoreCycles);

var app = builder.Build();

using (var scope = app.Services.CreateScope())
{
    var shopInternetContext =
    scope.ServiceProvider.GetRequiredService<ShopContext>();
    await ShopContextSeed.SeedAsync(shopInternetContext);
}
```

Рисунок 2 – Информация о подключении к БД в классе контекста

3. Результаты создания базы данных

Была обновлена БД InternetShopWebBD. Результат показан на рисунке 3.

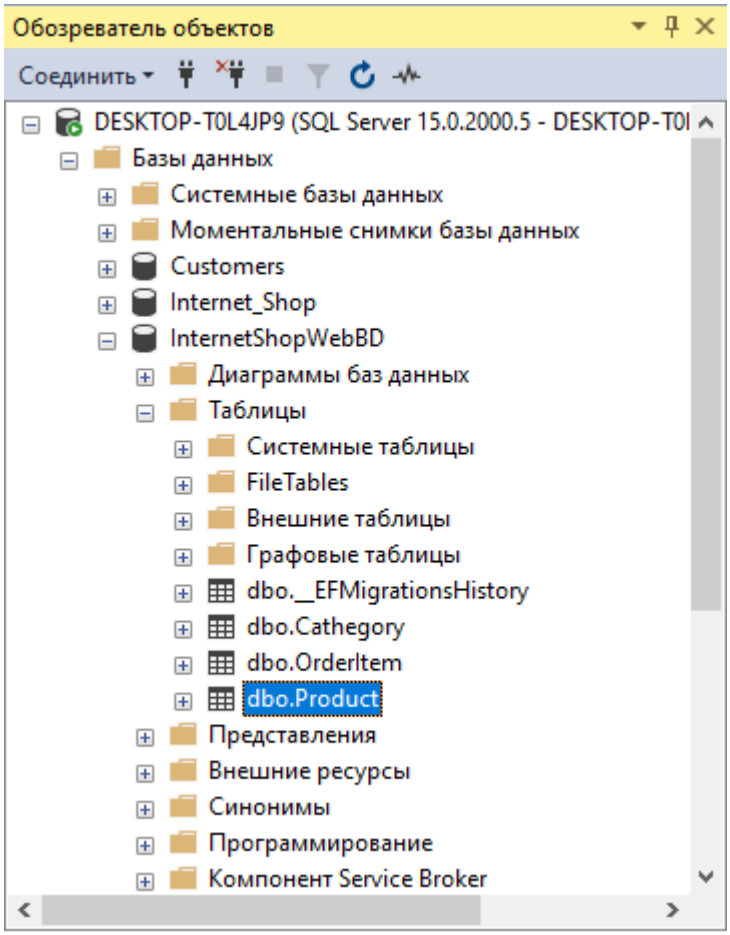

Рисунок 3 – Результат добавления и создания БД

4. Результаты добавления инициализации базы данных с предопределенными данными

Для того, чтобы выполнялась проверка на наличие записей в таблице заказов или в таблице актов списаний, был создан класс ShopContextSeed с кодом, который показан на рисунке 4.

Если таблицы пустые, то создается новый продукт, категория и строка заказа. Результат показан на рисунке 4.

```
13          public static async Task SeedAsync(ShopContext context)
14          {
15              try
16              {
17                  context.Database.EnsureCreated();
18                  if (context.Product.Any())
19                  {
20                      return;
21                  }
22
23                  var cathegories = new CathegoryModel[]
24                  {
25                      new CathegoryModel {
26                      Cathegory_Name = "Smartphones",
27                      Parent_ID=1
28                      },
29
30                      new CathegoryModel
31                      {
32                      Cathegory_Name = "Consoles",
33                      Parent_ID=2
34                      }
35                  };
36                  foreach (CathegoryModel cathegory in cathegories)
37                  {
38                      context.Cathegory.Add(cathegory);
39                  }
40                  await context.SaveChangesAsync();
41
42                  var orderitems = new OrderItemModel[]
43                  {
44                      new OrderItemModel {
45                      Order_Sum = 100,
```

Рисунок 5 – Содержание класса ShopContextSeed

После запуска приложения в БД создадутся данные. Результат показан на рисунке 6.

	Product_Code	OrderItem_Code	NumberInStock	CategoryID	DateOfManufacture	Description	PurchasePrice	MarketPrice	BestBeforeDate	Name
1	1	1	0	1	0001-01-01 00:00:00.0000000	asdasd	0	0	0	Nokia
2	2	2	0	1	0001-01-01 00:00:00.0000000	safasgag	0	0	0	Samsung

Рисунок 6 – Результат запуска приложения

5. Результаты загрузки связанных данных

На данном этапе были исключены ошибки зацикливания для связанных объектов. Для этого в program.cs был изменен код:

```
builder.Services.AddControllers();
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
//builder.Services.AddDbContext<ProductContext>(opt =>
//opt.UseModel(ProductModel));
builder.Services.AddDbContext<ShopContext>();
builder.Services.AddControllers().AddJsonOptions(x =>
x.JsonSerializerOptions.ReferenceHandler =
ReferenceHandler.IgnoreCycles);
```

Рисунок 7 – Изменения в program.cs

Далее была добавлена загрузка связанных данных в методе контроллера:

```
// GET: api/OrderItems
[HttpGet]
public async Task<ActionResult<IEnumerable<OrderItemModel>>> GetAllOrderItem()
{
    return await _context.OrderItem.Include(p=>p.Products).ToListAsync();
}
```

Рисунок 8 – Изменения в контроллере

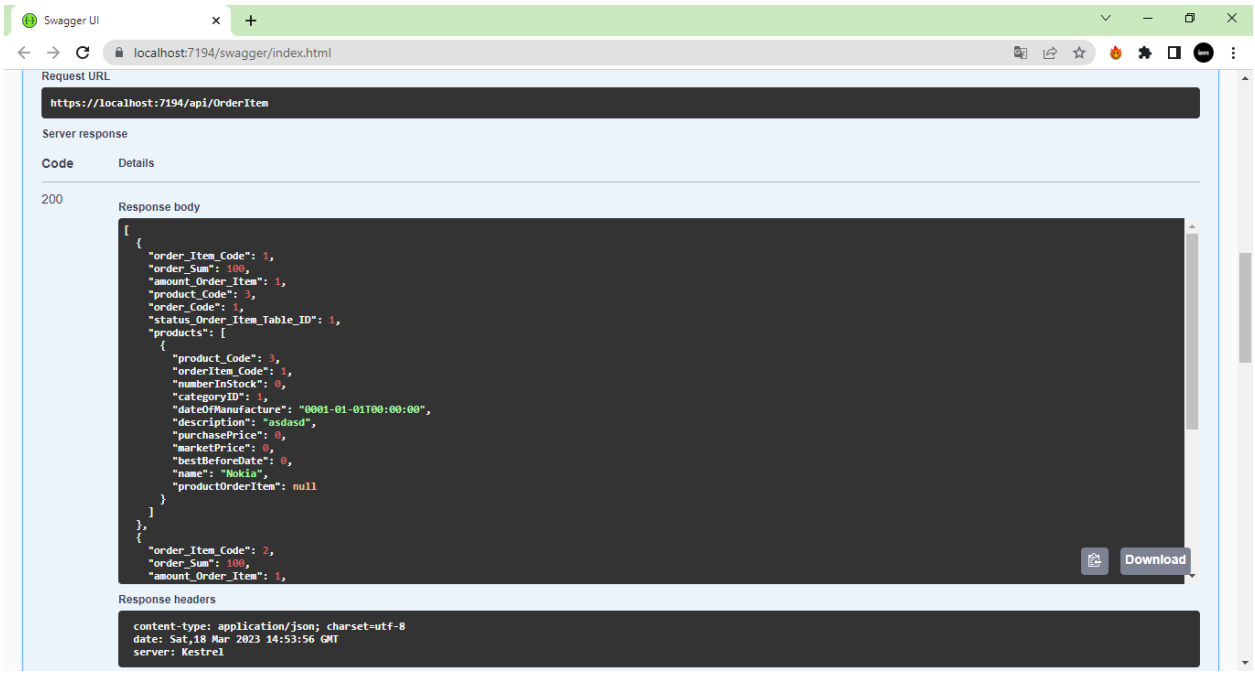

Рисунок 9 – Результат внесения изменений

6. Результаты реконструкции модели по существующей базе данных

Так как БД у меня уже существовала, я реконструировал модель. Для этого в консоли диспетчера пакетов была введена строка: Scaffold-DbContext "Data Source=DESKTOP-T0L4JP9;Database=InternetShopWebBD;Integrated Security=True;Connect Timeout=30;Encrypt=False;Trust Server Certificate=False;Application Intent=ReadWrite;Multi Subnet Failover=False" Microsoft.EntityFrameworkCore.SqlServer -OutputDirModels1. Результат показан на рисунке 10.

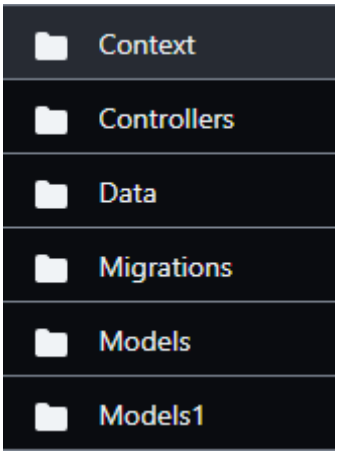

Рисунок 10 – Результат реконструкции бд

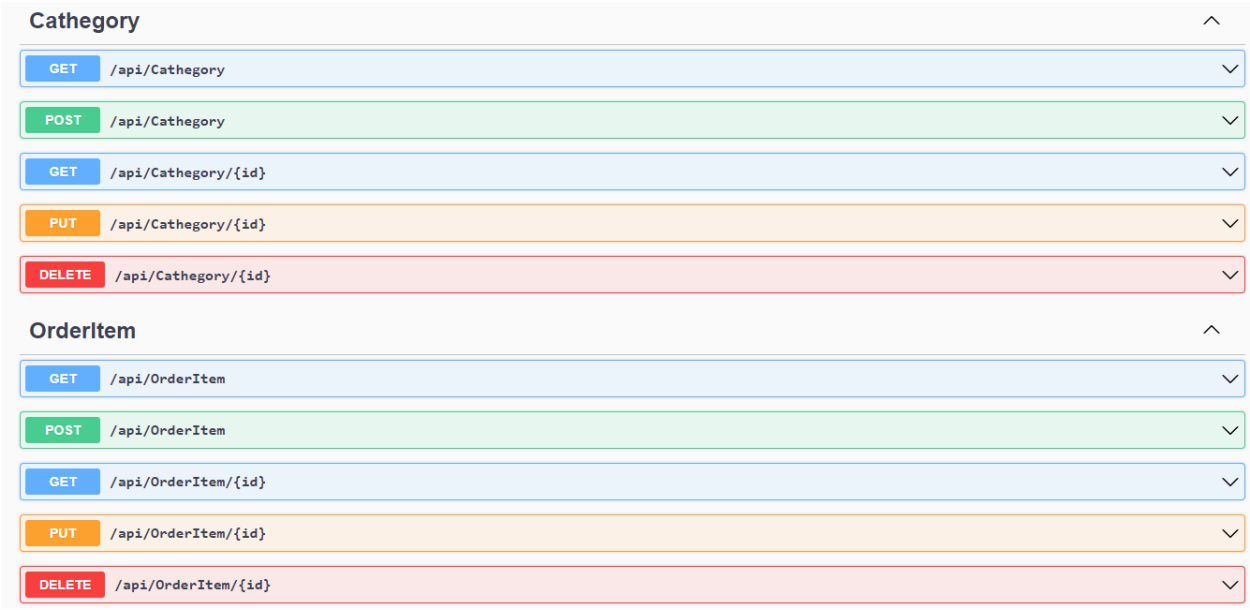

Рисунок 11 – Результат запуска приложения

Вывод

В ходе лабораторной работы создал базу данных по объектной модели с использованием технологии доступа к данным Entity Framework Core.