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 №2

Конструирование интернет-приложений

Разработка веб-API. Реализация функциональности CRUD.

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

Проверил _____ Садыков А.М.

Иваново 2022

Цель лабораторной работы: создать контроллер веб-API с методами получения, создания,

обновления и удаления данных.

create, read, update, and delete (CRUD)

Задания:

1. Создать проект веб-API

2. Создать модели данных сущностей

3. Добавить в проект поставщика EntityFrameworkCore.InMemory

4. Создать контекст базы данных

5. Выполнить регистрацию контекста базы данных

6. Создать контроллер Продуктов

6.1. Добавить новый контроллер Продуктов

6.2. Добавить методы действий получения элементов Продуктов

6.3. Добавить метод действия создания Продукта

6.4. Добавить метод действия обновления Продукта

6.5. Добавить метод действия удаления Продукта

1. Создать проект веб-API

В Visual Studio 2022 был создан проект веб-API:

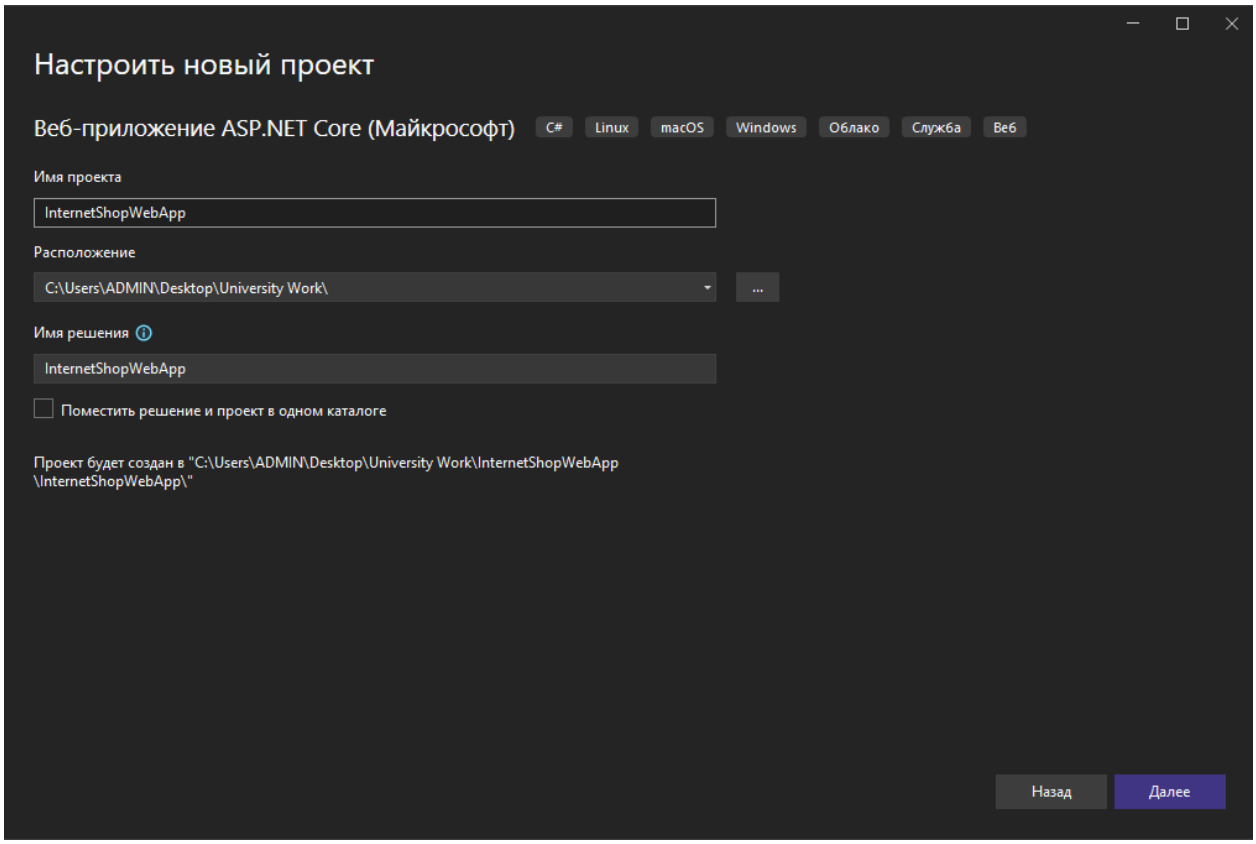

Рисунок 1 – Процесс создания приложения веб-API

2. Создать модели данных сущностей

Моя тема – «Интернет магазин».

Для реализации этой темы я создал модель данных продукта.

Все необходимые сущности были созданы в папке Models. Товар связан с категорией, как много к одному. Помимо того, товар лежит в строке заказа. Первичным ключом будет являться номер товара.

```
using System.ComponentModel.DataAnnotations;


namespace InternetShopWebApp.Models
{
    public class ProductModel
    {

        public ProductModel()
        {
            Cathegories = new HashSet<CathegoryModel>();
        }

        [Key]
        public int Product_Code { get; set; }
        public int NumberInStock { get; set; }
        public int CategoryID { get; set; }
        public System.DateTime DateOfManufacture { get; set; }
```

```
        public string Description { get; set; }
        public int PurchasePrice { get; set; }
        public int MarketPrice { get; set; }
        public float BestBeforeDate { get; set; }
        public string Name { get; set; }
        public virtual ICollection<CathegoryModel> Cathegories { get; set; }
    }
}
```

В строке заказа лежат товары. Они связаны, как много к 1. Для строки заказа была содана модель. Первичным ключом будет являться номер строки заказа.

```
using System.ComponentModel.DataAnnotations;

namespace InternetShopWebApp.Models
{
    public class OrderItemModel
    {
        public OrderItemModel()
        {
            Products = new HashSet<ProductModel>();
        }

        public virtual ICollection<ProductModel> Products { get; set; }
        [Key]
        public int Order_Item_Code { get; set; }
        public Nullable<int> Order_Sum { get; set; }
        public Nullable<int> Amount_Order_Item { get; set; }
        public Nullable<int> Product_Code { get; set; }
        public Nullable<int> Order_Code { get; set; }
        public int Status_Order_Item_Table_ID { get; set; }
    }
}
```

Категория связана сама с собой. У неё может быть родительская категория. Первичным ключом является номер категории.

```
using System.ComponentModel.DataAnnotations;

namespace InternetShopWebApp.Models
{
    public class CathegoryModel
    {
        public CathegoryModel()
        {
        }

        [Key]
        public int Category_ID { get; set; }
        public string Category_Name { get; set; }
        public int Parent_ID { get; set; }
        public virtual CathegoryModel ParentCathegory { get; set; }
    }
}
```

3. Добавить в проект поставщика EntityFrameworkCore.InMemory

EntityFrameworkCore.InMemory добавлена в проект с помощью команды Install-Package Microsoft.EntityFrameworkCore.InMemory. Она была введена в командной консоли NuGet.

4. Создать контекст базы данных

Контекст базы данных — это основной класс, который координирует функциональные возможности Entity Framework для заданной модели данных и обеспечивает взаимодействие с хранилищем данных.

Я создал новую папку для контекста. Потом был создан Context.cs в папке Context, а также в данном классе были прописаны классы созданных сущностей.

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

namespace InternetShopWebApp.Context
{
    public class Context : DbContext
    {
        #region Constructor
        public Context(DbContextOptions<Context> options) : base(options) { }
        #endregion
        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();
            });
            modelBuilder.Entity<CathegoryModel>(entity =>
            {
                entity.HasOne(e => e.ParentCathegory).WithOne(a =>
a.ParentCathegory);
            });
            modelBuilder.Entity<OrderItemModel>(entity =>
            {
                entity.Property(e => e.Order_Item_Code).IsRequired();
            });
        }
    }
}
```

5. Выполнить регистрацию контекста

Контекст базы данных регистрируется с помощью контейнера внедрения зависимостей, который позволяет сделать объекты слабосвязанными. Была выполнена регистрация контекста базы данных в Program.cs.

```
using InternetShopWebApp.Context;
using Microsoft.EntityFrameworkCore;

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();

var app = builder.Build();


// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();
```

6. Создать контроллер Продуктов

6.1. Добавить новый контроллер Продуктов

Контроллер представляет собой центральное звено, которому маршрутизация передает данные запроса и который занимается обработкой ответом.

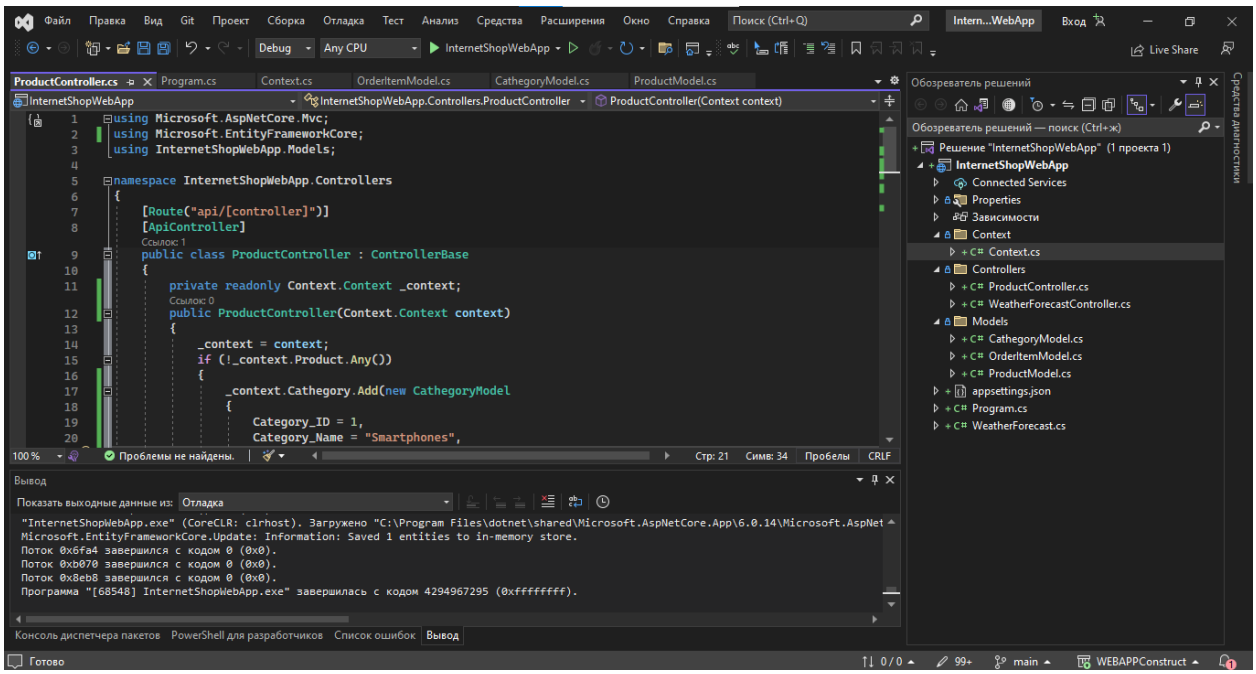

Рисунок 2 – Результат добавления Контроллера

6.2. Добавить методы действий получения элементов Продуктов

В контроллере заказа были написаны функции методов действий получения элементов заказов. Функция GetAllProducts() позволяет получить сведения обо всех продуктах. Функция GetProduct() позволяет получить сведения о продукте по введенному номеру.

```
// GET: api/Products
        [HttpGet]
```

```
        public async Task<ActionResult<IEnumerable<ProductModel>>> GetAllProduct()
        {
            return await _context.Product.ToListAsync();
        }
        // GET: api/Products/5
        [HttpGet("{id}")]
        public async Task<ActionResult<ProductModel>> GetProduct(int id)
        {
            var blog = await _context.Product.FindAsync(id);
            if (blog == null)
            {
                return NotFound();
            }
            return blog;
        }
```

6.3. Добавить метод действия создания Продукта

В контроллере товаров написан метод создания нового продукта.

```
// POST: api/Product
        [HttpPost]
        public async Task<ActionResult<ProductModel>> NewProduct(ProductModel
Product)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }
            _context.Product.Add(Product);
            await _context.SaveChangesAsync();
            return CreatedAtAction("GetProduct", new { id = Product.Product_Code },
Product);
        }
```

6.4. Добавить метод действия обновления Продукта

Добавлен метод обновления существующего продукта.

```
// PUT: api/Product/5
        [HttpPut("{id}")]
        public async Task<IActionResult> PutProduct(int id, ProductModel Product)
        {
            if (id != Product.Product_Code)
            {
                return BadRequest();
            }
            _context.Entry(Product).State = EntityState.Modified;
            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!ProductExists(id))
                {
                    return NotFound();
                }
                else
                {
                    throw;
                }
            }
```

```
            return NoContent();
        }

        private bool ProductExists(int id)
        {
            return _context.Product.Any(e => e.Product_Code == id);
        }
```

6.5. Добавить метод действия удаления Продукта

Добавлен метод удаления продукта.

```
// DELETE: api/Product/5
        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteProduct(int id)
        {
            var blog = await _context.Product.FindAsync(id);
            if (blog == null)
            {
                return NotFound();
            }
            _context.Product.Remove(blog);
            await _context.SaveChangesAsync();
            return NoContent();
        }
```

Аналогичным образом была проделана работа с сущности акта списания и строки акта списания.

Далее проект был запущен.

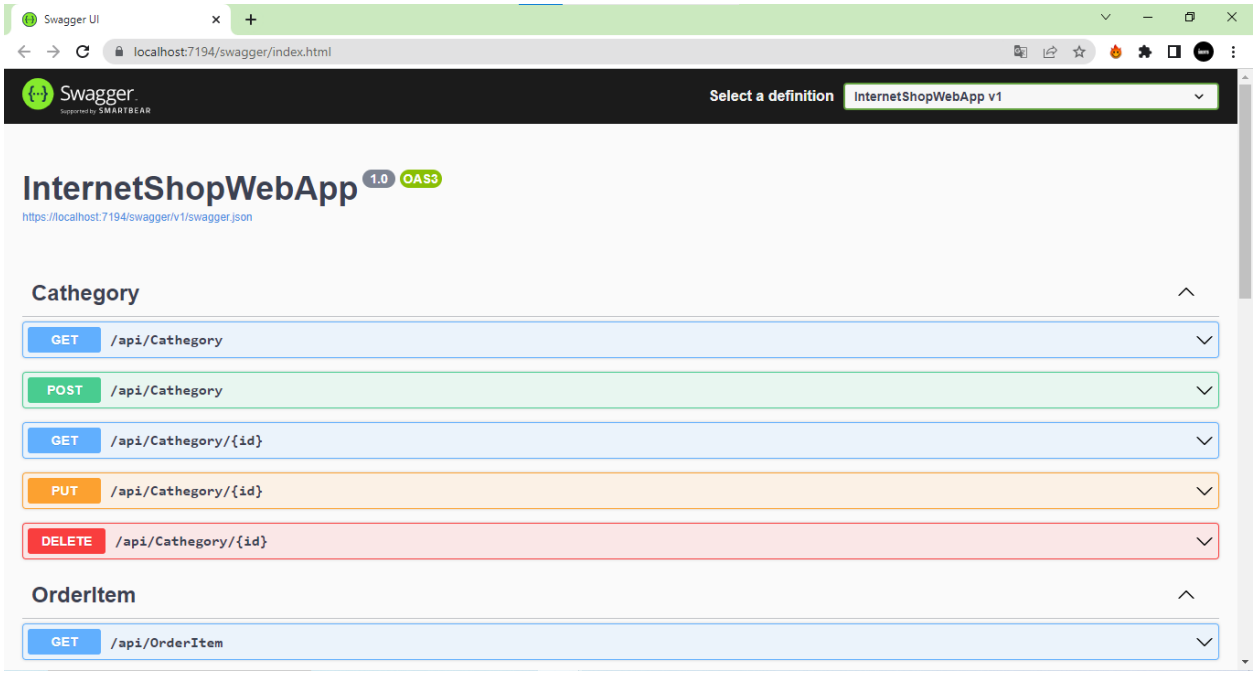

Рисунок 3 – Результат запуска

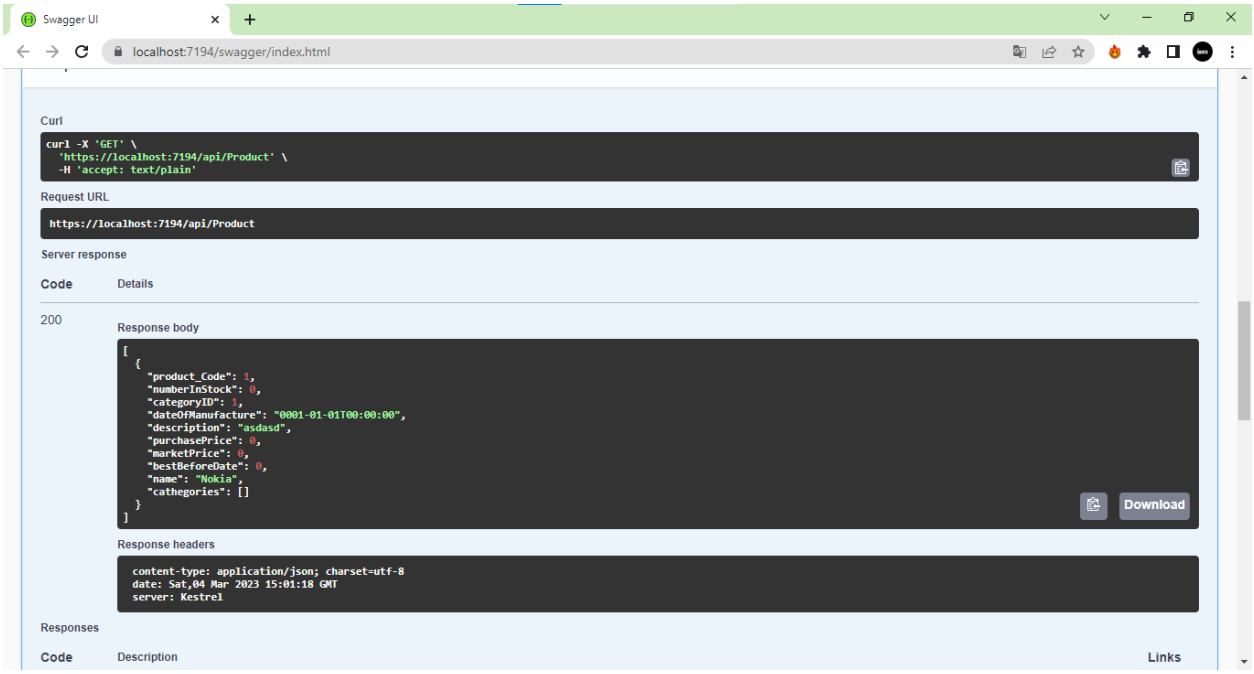

Рисунок 4 – Результат get запроса

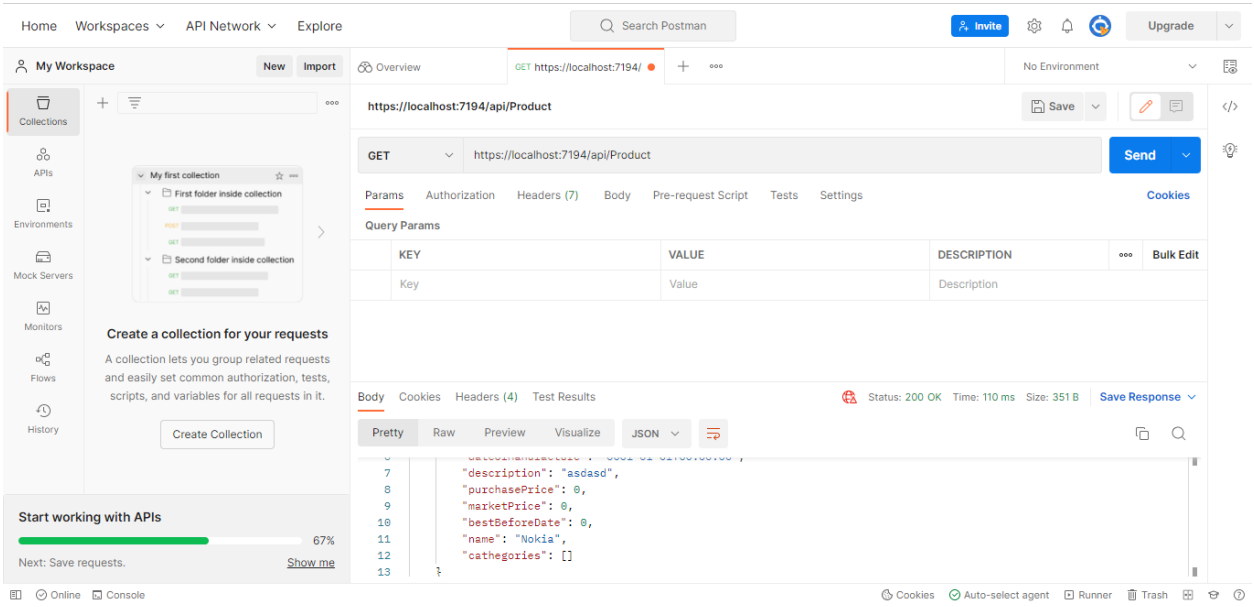

Рисунок 5 – Результат get запроса в Postman

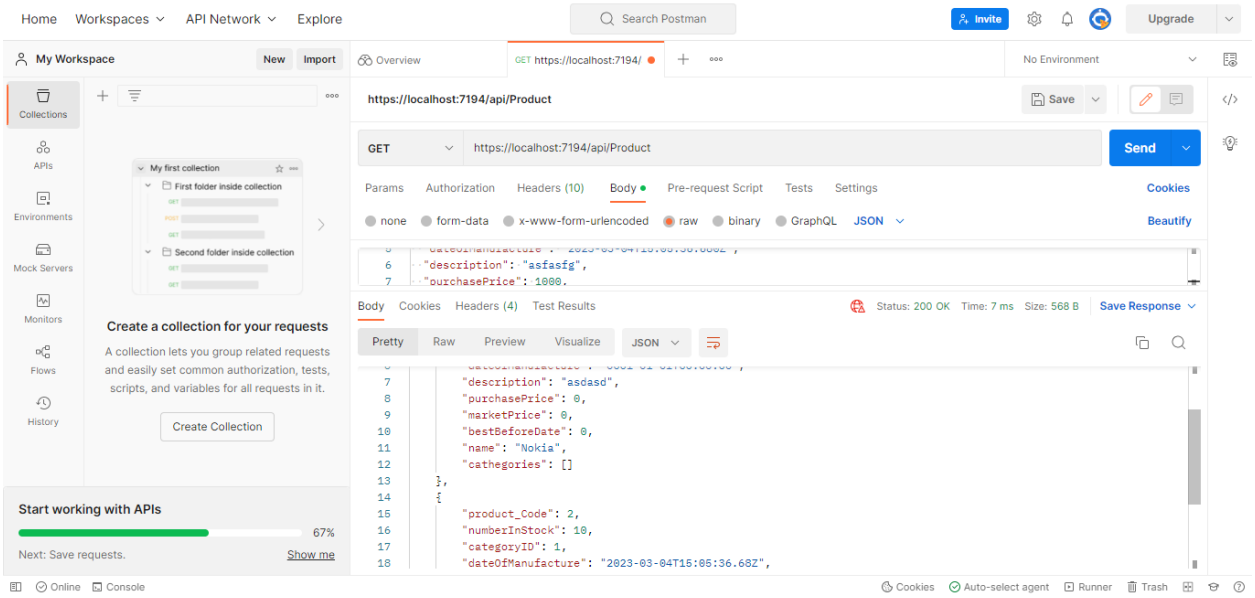

Рисунок 6 – Результат post запроса в Postman

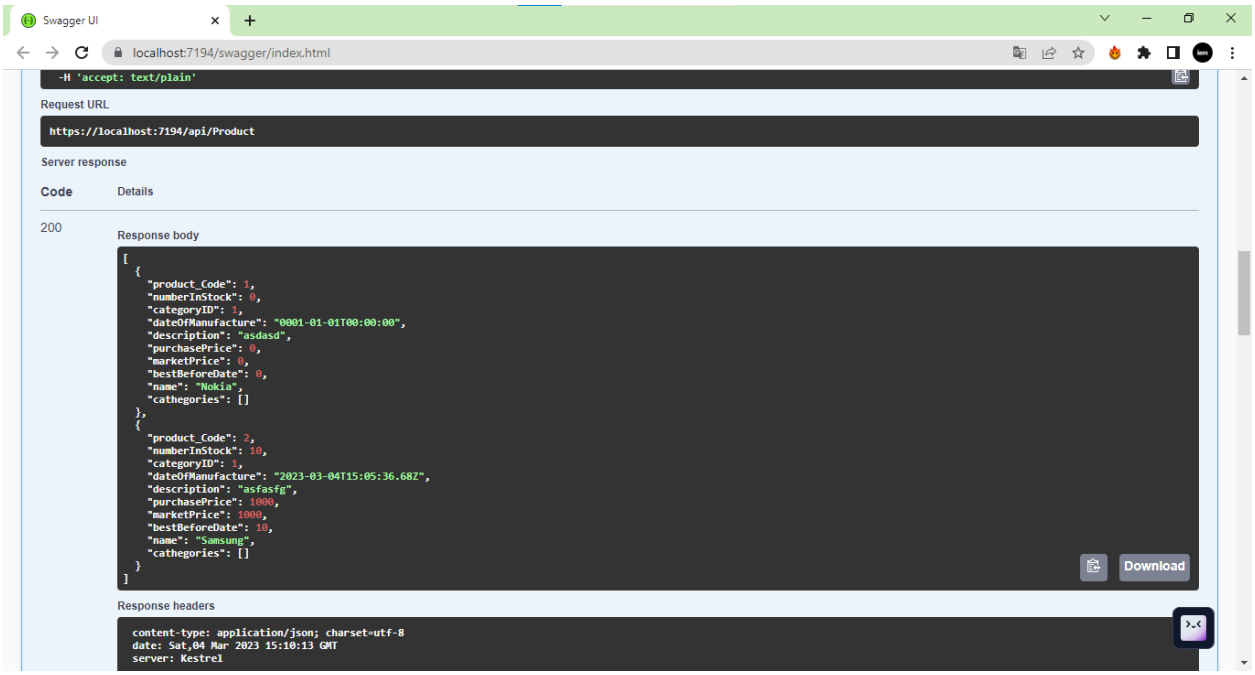

Рисунок 7 – Результат post запроса

Рисунок 8 – Результат delete запроса

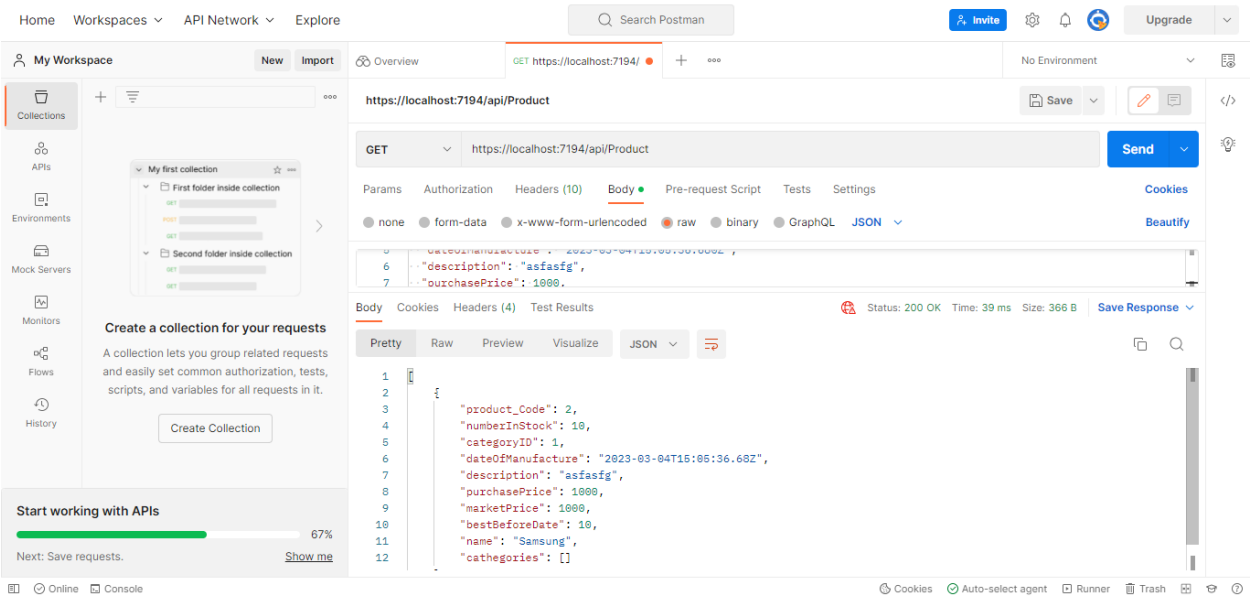

Рисунок 9 – Результат delete запроса в Postman

Рисунок 10 – Результат put запроса

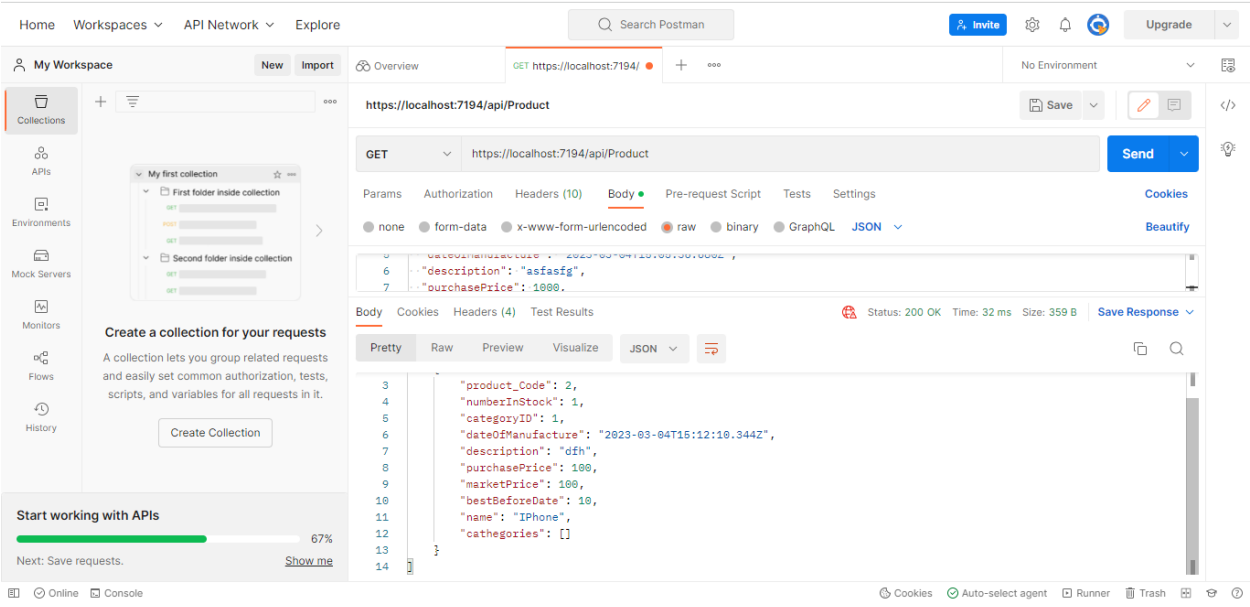

Рисунок 10 – Результат put запроса в Postman

Вывод

В ходе лабораторной работы научилась создавать контроллер веб-API с методами получения, создания, обновления и удаления данных.