Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №5

по дисциплине «Программные технологии проектирования программного обеспечения вычислительных средств»

на тему «Создание простого Web-приложения»

Выполнили: студенты группы 22ВВП1

Беляев Д. И.

Демин М. С.

Приняли:

Патунин Д. В.

Деев М. В.

Пенза 2025

**Цель работы**

Создание простого Web-приложения

**Задание**

1. 1. Реализовать класс ApplicationContext с наследованием от класса DbContext входящий в состав EntityFramework.

2. Внутри DbContext должна храниться таблица DbSet<Product> Products.

3. База данных должна создаваться при первом запуске приложения. Если база уже существует, то пересоздавать не требуется. Внимательно читайте про EnsureCreated.

4. Строку подключения к базе данных передавать через IConfiguration (сама строка подключения должна хранится в файле конфигурации).

5. Внести изменения в класс Product. Поле Id должно быть помечено, как индексируемое (читайте про атрибуты).

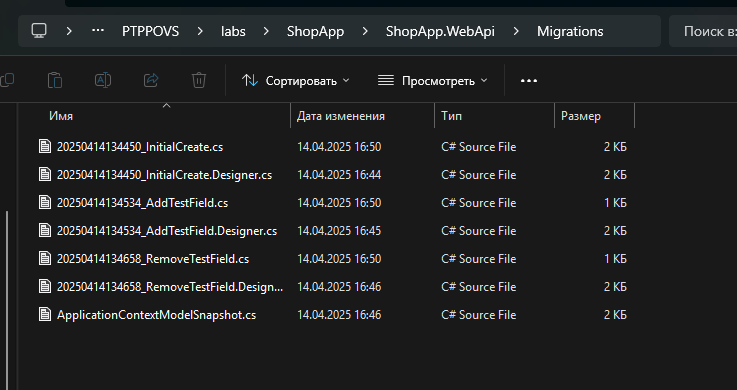
6. Реализовать класс DataBaseProductService реализующий интерфейс IProductService. DataBaseProductService для взаимодействия с базой данных должен использовать класс ApplicationContext.

7. Внести изменения в класс Program. Связать интерфейс IProductService с классом DataBaseProductService.

8. Добавить в класс Product новое поле. Произвести миграцию.

**Ход работы**

Сделали миграции



**Листинг**

using Microsoft.EntityFrameworkCore;

namespace ShopApp.Core.Models

{

/// <summary>

/// Represents a product in the store.

/// </summary>

[Index(nameof(Id))]

public class Product : ICloneable

{

/// <summary>

/// Gets or sets the unique identifier of the product.

/// </summary>

public Guid Id { get; set; }

/// <summary>

/// Gets or sets the description of the product.

/// </summary>

public string? Description { get; set; }

/// <summary>

/// Gets or sets the price of the product.

/// </summary>

public double Price { get; set; }

/// <summary>

/// Gets or sets the image path of the product.

/// </summary>

public string? Image { get; set; } = null;

/// <summary>

/// Creates a deep copy of the current product instance.

/// </summary>

/// <returns>A new <see cref="Product"/> object with the same values.</returns>

public object Clone()

{

return new Product()

{

Id = Id,

Description = Description,

Price = Price,

Image = Image

};

}

}

}

using Microsoft.EntityFrameworkCore;

using ShopApp.Core.Models;

using ShopApp.WebApi.Data;

using ShopApp.WebApi.Services;

namespace ShopApp.Tests

{

/// <summary>

/// NUnit tests for the DataBaseProductService class.

/// </summary>

[TestFixture]

public class DataBaseProductServiceTests

{

/// <summary>

/// Creates a new set of DbContextOptions with an InMemory database for isolated tests.

/// </summary>

/// <returns>A new instance of DbContextOptions for ApplicationContext.</returns>

private DbContextOptions<ApplicationContext> CreateNewContextOptions()

{

return new DbContextOptionsBuilder<ApplicationContext>()

.UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())

.Options;

}

/// <summary>

/// Tests that adding a product returns true and the product is correctly saved in the database.

/// The test data is provided via TestCase attributes.

/// </summary>

/// <param name="description">Product description.</param>

/// <param name="price">Product price.</param>

/// <param name="image">Product image path.</param>

[TestCase("Test Product A", 9.99, "imageA.jpg")]

[TestCase("Test Product B", 19.99, "imageB.jpg")]

public async Task Add\_Product\_ShouldReturnTrue(string description, double price, string image)

{

// Arrange

DbContextOptions<ApplicationContext> options = CreateNewContextOptions();

using ApplicationContext context = new(options);

DataBaseProductService service = new(context);

Product product = new()

{

Id = Guid.NewGuid(),

Description = description,

Price = price,

Image = image,

};

// Act

bool result = await service.Add(product);

// Assert

Assert.That(result, Is.True, "Product should be added successfully.");

// Verify that the product is saved in the database.

Product? savedProduct = await context.Products.FirstOrDefaultAsync(p => p.Id == product.Id);

Assert.That(savedProduct, Is.Not.Null, "Saved product should not be null.");

Assert.That(savedProduct?.Description, Is.EqualTo(description), "The product description does not match.");

Assert.That(savedProduct?.Price, Is.EqualTo(price), "The product price does not match.");

Assert.That(savedProduct?.Image, Is.EqualTo(image), "The product image path does not match.");

}

/// <summary>

/// Tests that removing a product returns the removed product and deletes it from the database.

/// The test data is provided via a TestCase attribute.

/// </summary>

/// <param name="description">Product description.</param>

/// <param name="price">Product price.</param>

/// <param name="image">Product image path.</param>

[TestCase("Test Product Remove", 10.0, "remove.jpg")]

public async Task Remove\_Product\_ShouldReturnProductAndRemoveIt(string description, double price, string image)

{

// Arrange

DbContextOptions<ApplicationContext> options = CreateNewContextOptions();

Product product = new()

{

Id = Guid.NewGuid(),

Description = description,

Price = price,

Image = image,

};

// Save the product in the database.

using (ApplicationContext context = new(options))

{

\_ = context.Products.Add(product);

\_ = await context.SaveChangesAsync();

}

// Act & Assert

using (ApplicationContext context = new(options))

{

DataBaseProductService service = new(context);

Product? removedProduct = await service.Remove(product.Id);

Assert.That(removedProduct, Is.Not.Null, "Removed product should not be null.");

Assert.That(removedProduct.Id, Is.EqualTo(product.Id), "The removed product ID should match the original product ID.");

// Verify that the product has been removed.

bool exists = await context.Products.AnyAsync(p => p.Id == product.Id);

Assert.That(exists, Is.False, "Product should no longer exist in the database.");

}

}

/// <summary>

/// Tests that editing an existing product returns the updated product.

/// Test data is provided via a TestCase attribute.

/// </summary>

/// <param name="originalDesc">Original product description.</param>

/// <param name="updatedDesc">Updated product description.</param>

/// <param name="originalPrice">Original product price.</param>

/// <param name="updatedPrice">Updated product price.</param>

/// <param name="originalImg">Original product image path.</param>

/// <param name="updatedImg">Updated product image path.</param>

[TestCase("Original Description", "Updated Description", 10.0, 15.50, "original.jpg", "updated.jpg")]

public async Task Edit\_Product\_ShouldReturnUpdatedProduct(

string originalDesc, string updatedDesc,

double originalPrice, double updatedPrice,

string originalImg, string updatedImg)

{

// Arrange

DbContextOptions<ApplicationContext> options = CreateNewContextOptions();

Product product = new()

{

Id = Guid.NewGuid(),

Description = originalDesc,

Price = originalPrice,

Image = originalImg,

};

// Save the original product.

using (ApplicationContext context = new(options))

{

\_ = context.Products.Add(product);

\_ = await context.SaveChangesAsync();

}

// Modify the product.

product.Description = updatedDesc;

product.Price = updatedPrice;

product.Image = updatedImg;

// Act & Assert

using (ApplicationContext context = new(options))

{

DataBaseProductService service = new(context);

Product? updatedProduct = await service.Edit(product);

Assert.That(updatedProduct, Is.Not.Null, "Updated product should not be null.");

Assert.That(updatedProduct.Description, Is.EqualTo(updatedDesc), "Product description was not updated correctly.");

Assert.That(updatedProduct.Price, Is.EqualTo(updatedPrice), "Product price was not updated correctly.");

Assert.That(updatedProduct.Image, Is.EqualTo(updatedImg), "Product image path was not updated correctly.");

}

}

/// <summary>

/// Tests that searching for an existing product returns the correct product.

/// Test data is provided via a TestCase attribute.

/// </summary>

/// <param name="description">Product description.</param>

/// <param name="price">Product price.</param>

/// <param name="image">Product image path.</param>

[TestCase("Search Test Product", 5.00, "search.jpg")]

public async Task Search\_Product\_ShouldReturnCorrectProduct(string description, double price, string image)

{

// Arrange

DbContextOptions<ApplicationContext> options = CreateNewContextOptions();

Product product = new()

{

Id = Guid.NewGuid(),

Description = description,

Price = price,

Image = image,

};

using (ApplicationContext context = new(options))

{

\_ = context.Products.Add(product);

\_ = await context.SaveChangesAsync();

}

// Act & Assert

using (ApplicationContext context = new(options))

{

DataBaseProductService service = new(context);

Product? foundProduct = await service.Search(product.Id);

Assert.That(foundProduct, Is.Not.Null, "Found product should not be null.");

Assert.That(foundProduct.Id, Is.EqualTo(product.Id), "The found product ID should match the searched product ID.");

Assert.That(foundProduct.Description, Is.EqualTo(description), "The product description does not match.");

Assert.That(foundProduct.Price, Is.EqualTo(price), "The product price does not match.");

Assert.That(foundProduct.Image, Is.EqualTo(image), "The product image path does not match.");

}

}

}

}

using Microsoft.EntityFrameworkCore;

using ShopApp.Core.Models;

namespace ShopApp.WebApi.Data

{

/// <summary>

/// Represents the application's database context.

/// </summary>

public class ApplicationContext : DbContext

{

/// <summary>

/// Gets or sets the Products table.

/// </summary>

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

/// <summary>

/// Initializes a new instance of the <see cref="ApplicationContext"/> class.

/// Ensures the database is created if it does not exist.

/// </summary>

/// <param name="options">The options for configuring the context.</param>

public ApplicationContext(DbContextOptions<ApplicationContext> options)

: base(options)

{

}

}

}

using ShopApp.Core.Models;

using ShopApp.Core.Services;

using ShopApp.WebApi.Data;

namespace ShopApp.WebApi.Services

{

/// <summary>

/// Provides database operations for products.

/// </summary>

public class DataBaseProductService : IProductsService<Product>

{

private readonly ApplicationContext \_context;

/// <summary>

/// Initializes a new instance of the <see cref="DataBaseProductService"/> class.

/// </summary>

/// <param name="context">The application database context.</param>

public DataBaseProductService(ApplicationContext context)

{

\_context = context;

}

/// <summary>

/// Adds a new product to the database.

/// </summary>

/// <param name="product">The product to add.</param>

/// <returns>True if the product was added; otherwise, false.</returns>

public async Task<bool> Add(Product product)

{

\_ = \_context.Products.Add(product);

return await \_context.SaveChangesAsync() > 0;

}

/// <summary>

/// Removes a product from the database by its identifier.

/// </summary>

/// <param name="productId">The identifier of the product to remove.</param>

/// <returns>The removed product if successful; otherwise, null.</returns>

public async Task<Product?> Remove(Guid productId)

{

Product? product = await \_context.Products.FindAsync(productId);

if (product != null)

{

\_ = \_context.Products.Remove(product);

\_ = await \_context.SaveChangesAsync();

}

return product;

}

/// <summary>

/// Edits an existing product in the database.

/// </summary>

/// <param name="product">The updated product.</param>

/// <returns>The updated product if successful; otherwise, null.</returns>

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

{

Product? entity = await \_context.Products.FindAsync(product.Id);

if (entity == null)

{

return null;

}

// Update individual properties

entity.Description = product.Description;

entity.Price = product.Price;

entity.Image = product.Image;

\_ = await \_context.SaveChangesAsync();

return entity;

}

/// <summary>

/// Searches for a product in the database by its identifier.

/// </summary>

/// <param name="productId">The identifier of the product to search for.</param>

/// <returns>The found product if exists; otherwise, null.</returns>

public async Task<Product?> Search(Guid productId)

{

return await \_context.Products.FindAsync(productId);

}

/// <summary>

/// Releases the unmanaged resources used by the service.

/// </summary>

public void Dispose()

{

\_context.Dispose();

}

}

}

using Microsoft.EntityFrameworkCore;

using ShopApp.Core.Models;

using ShopApp.Core.Services;

using ShopApp.WebApi.Data;

using ShopApp.WebApi.Services;

WebApplicationBuilder builder = WebApplication.CreateBuilder(args);

// Retrieve connection string from configuration.

string connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? "Data Source=shopapp.db";

// Register the application's DbContext using SQLite.

builder.Services.AddDbContext<ApplicationContext>(options => options.UseSqlite(connectionString));

// Register the product service for dependency injection.

builder.Services.AddScoped<IProductsService<Product>, DataBaseProductService>();

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(options =>

{

string basePath = AppContext.BaseDirectory;

string xmlPath = Path.Combine(basePath, "ShopApp.WebApi.xml");

options.IncludeXmlComments(xmlPath);

});

WebApplication app = builder.Build();

using (IServiceScope scope = app.Services.CreateScope())

{

ApplicationContext context = scope.ServiceProvider.GetRequiredService<ApplicationContext>();

context.Database.Migrate();

}

if (app.Environment.IsDevelopment())

{

\_ = app.UseSwagger();

\_ = app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}");

app.Run();

**Вывод**

Создали простое Web-приложение