Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №3

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**

В ходе выполнения лабораторной работы вам будет необходимо выполнить следующие пункты:

1. Добавить в решение проект модульного тестирования (Выбрать шаблон проекта NUnit)

2. Затем в созданный проект добавить ссылки на существующие проекты.

3. Реализовать два класса для тестирования:

a. TestsProductController – класс, в котором будет происходить тестирование методов контроллера. (Все методы котроллера доступные для вызова публичным API должны быть покрыты тестами)

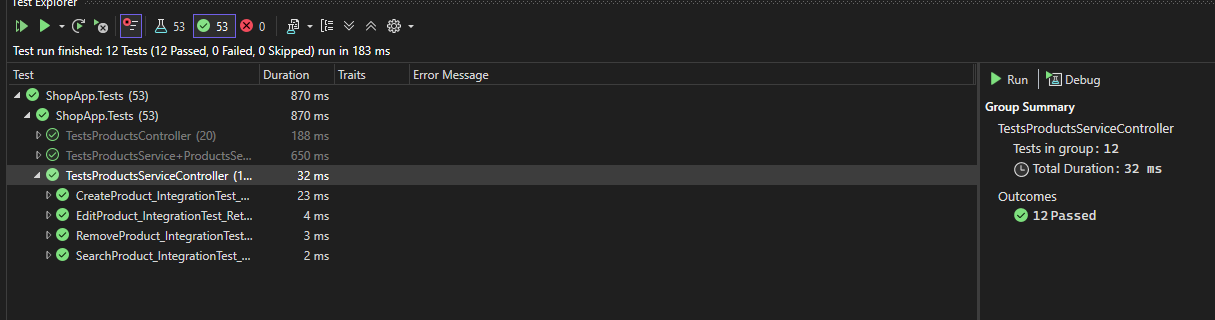
b. TestsProductService – класс, в котором будет происходить тестирование методов класса. (Все доступные для вызова методы класса должны быть покрыты тестами).

4. Провести интеграционный тест. Внедрить зависимость внутрь контроллера используя явную реализацию интерфейса.

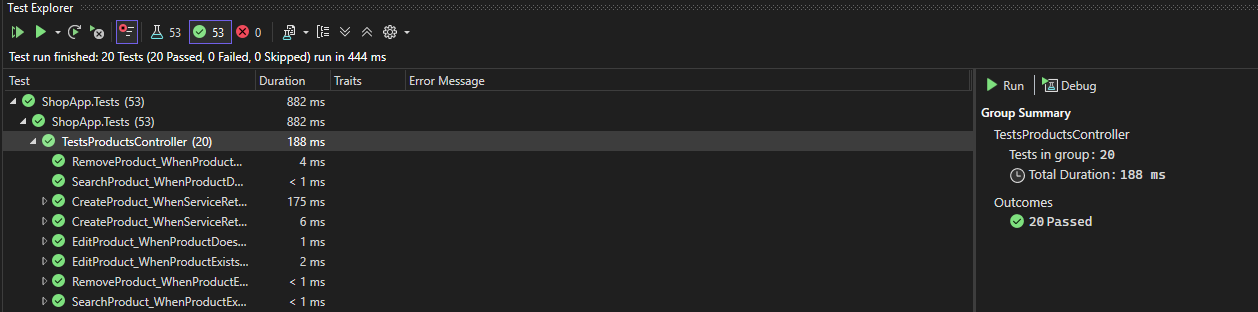
5. Провести модульный тест контроллера с помощью Stub-ов и Mock-ов.

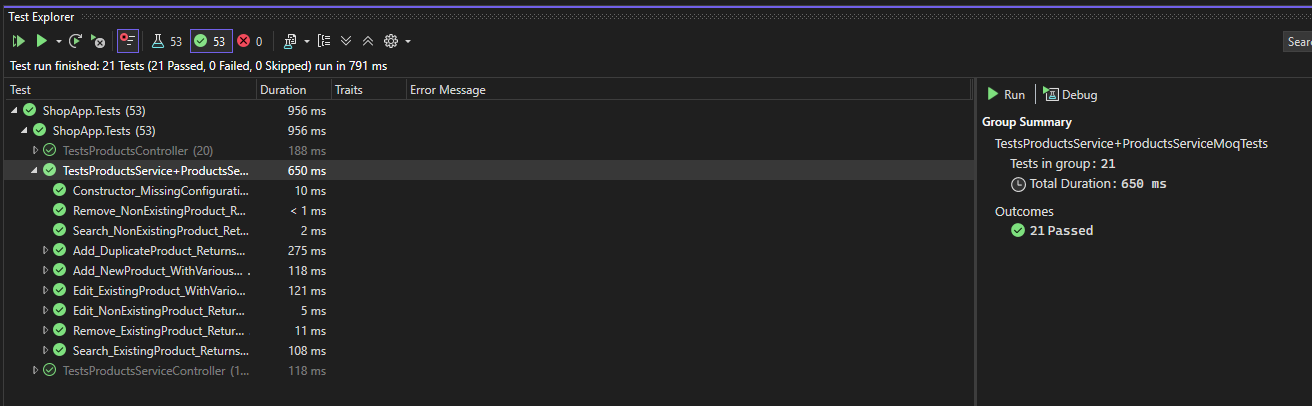
**Ход работы**

Интеграционные тесты



Модульные тесты





**Листинг**

*TestsProductsServiceController.cs*

using Microsoft.Extensions.Configuration;

using Moq;

using ShopApp.Core.Models;

using ShopApp.WebApi.Services;

namespace ShopApp.Tests

{

[TestFixture]

internal class TestsProductsService

{

[TestFixture]

public class ProductsServiceMoqTests

{

private string \_tempFilePath;

private Mock<IConfiguration> \_mockConfiguration;

private ProductsService \_service;

[SetUp]

public void SetUp()

{

// Generate a unique temporary file path using a GUID.

\_tempFilePath = Path.Combine(Path.GetTempPath(), $"{Guid.NewGuid()}.json");

// Configure the mock for IConfiguration.

\_mockConfiguration = new Mock<IConfiguration>();

\_ = \_mockConfiguration.Setup(config => config["DataBaseFilePath"]).Returns(\_tempFilePath);

// Initialize the service with the mocked configuration.

\_service = new ProductsService(\_mockConfiguration.Object);

}

[TearDown]

public void TearDown()

{

\_service.Dispose();

//// TEMP

const int maxRetries = 10;

const int delayMs = 100;

for (int retry = 0; retry < maxRetries; retry++)

{

try

{

if (File.Exists(\_tempFilePath))

{

File.Delete(\_tempFilePath);

}

break;

}

catch (IOException)

{

Thread.Sleep(delayMs);

}

}

}

[Test]

public void Constructor\_MissingConfiguration\_ThrowsException()

{

// Simulate a missing configuration for the database file path.

Mock<IConfiguration> emptyConfig = new();

\_ = emptyConfig.Setup(config => config["DataBaseFilePath"]).Returns(null as string);

\_ = Assert.Throws<Exception>(() => new ProductsService(emptyConfig.Object),

"Constructor should throw an exception when the database file path is missing.");

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void Add\_NewProduct\_WithVariousData\_ReturnsTrue(string description, double price)

{

// Arrange

Product product = new()

{

Id = Guid.NewGuid(),

Description = description,

Price = price

};

// Act

bool result = \_service.Add(product);

// Assert

Assert.That(result, Is.True, $"Adding a new product with description '{description}' and price {price} should return true.");

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void Add\_DuplicateProduct\_ReturnsFalse(string descriprion, double price)

{

// Arrange

Product product = new()

{

Id = Guid.NewGuid(),

Description = descriprion,

Price = price

};

// Act

\_ = \_service.Add(product);

bool secondAdd = \_service.Add(product);

// Assert

Assert.That(secondAdd, Is.False, "Adding a duplicate product should return false.");

}

[TestCase("Original Product A", 10.0, "Updated Product A", 15.0)]

[TestCase("Original Product B", 20.0, "Updated Product B", 25.0)]

[TestCase("Original Product C", 30.0, "Updated Product C", 35.0)]

public void Edit\_ExistingProduct\_WithVariousData\_ReturnsUpdatedProduct(

string originalDescription, double originalPrice,

string updatedDescription, double updatedPrice)

{

// Arrange

Product product = new()

{

Id = Guid.NewGuid(),

Description = originalDescription,

Price = originalPrice

};

\_ = \_service.Add(product);

// Update product values.

Product newProduct = (Product)product.Clone();

newProduct.Description = updatedDescription;

newProduct.Price = updatedPrice;

// Act

Product? updatedProduct = \_service.Edit(newProduct);

// Assert

Assert.That(updatedProduct, Is.Not.Null, "Editing an existing product should return the updated product.");

Assert.That(updatedProduct.Description, Is.EqualTo(updatedDescription), "The product description should be updated.");

Assert.That(updatedProduct.Price, Is.EqualTo(updatedPrice), "The product price should be updated.");

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void Edit\_NonExistingProduct\_ReturnsNull(string descriprion, double price)

{

// Arrange

Product product = new()

{

Id = Guid.NewGuid(),

Description = descriprion,

Price = price

};

// Act

Product? result = \_service.Edit(product);

// Assert

Assert.That(result, Is.Null, "Editing a non-existing product should return null.");

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void Remove\_ExistingProduct\_ReturnsRemovedProduct(string descriprion, double price)

{

// Arrange

Product product = new()

{

Id = Guid.NewGuid(),

Description = descriprion,

Price = price

};

\_ = \_service.Add(product);

// Act

Product? removedProduct = \_service.Remove(product.Id);

// Assert

Assert.That(removedProduct, Is.Not.Null, "Removing an existing product should return the removed product.");

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

}

[Test]

public void Remove\_NonExistingProduct\_ReturnsNull()

{

// Act

Product? removedProduct = \_service.Remove(Guid.NewGuid());

// Assert

Assert.That(removedProduct, Is.Null, "Removing a non-existing product should return null.");

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void Search\_ExistingProduct\_ReturnsProduct(string descriprion, double price)

{

// Arrange

Product product = new()

{

Id = Guid.NewGuid(),

Description = descriprion,

Price = price

};

\_ = \_service.Add(product);

// Act

Product? foundProduct = \_service.Search(product.Id);

// Assert

Assert.That(foundProduct, Is.Not.Null, "Searching for an existing product should return the product.");

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

}

[Test]

public void Search\_NonExistingProduct\_ReturnsNull()

{

// Act

Product? foundProduct = \_service.Search(Guid.NewGuid());

// Assert

Assert.That(foundProduct, Is.Null, "Searching for a non-existing product should return null.");

}

}

}

}

*TestsProductsServiceController.cs*

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.Logging;

using Microsoft.Extensions.Logging.Abstractions;

using ShopApp.WebApi.Controllers;

using ShopApp.WebApi.Services;

namespace ShopApp.Tests

{

[TestFixture]

public class TestsProductsServiceController

{

private string \_tempFilePath;

private IConfiguration \_configuration;

private ProductsService \_productService;

private ProductsController \_controller;

private ILogger<ProductsController> \_logger;

[SetUp]

public void SetUp()

{

// Generate a unique temporary file path using a GUID.

\_tempFilePath = Path.Combine(Path.GetTempPath(), $"{Guid.NewGuid()}.json");

// Build an in-memory configuration with the database file path.

Dictionary<string, string> configDictionary = new()

{

{ "DataBaseFilePath", \_tempFilePath }

};

\_configuration = new ConfigurationBuilder()

.AddInMemoryCollection(configDictionary)

.Build();

// Instantiate the real ProductsService using the configuration.

\_productService = new ProductsService(\_configuration);

// Use a NullLogger for the controller.

\_logger = NullLogger<ProductsController>.Instance;

// Inject the real dependency into the controller.

\_controller = new ProductsController(\_logger, \_productService);

}

[TearDown]

public void TearDown()

{

\_productService.Dispose();

//// TEMP

const int maxRetries = 10;

const int delayMs = 100;

for (int retry = 0; retry < maxRetries; retry++)

{

try

{

if (File.Exists(\_tempFilePath))

{

File.Delete(\_tempFilePath);

}

break;

}

catch (IOException)

{

Thread.Sleep(delayMs);

}

}

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void CreateProduct\_IntegrationTest\_ReturnsNonEmptyGuid(string description, double price)

{

// Act

Guid productId = \_controller.CreateProduct(description, price);

// Assert

Assert.That(productId, Is.Not.EqualTo(Guid.Empty), "CreateProduct should return a non-empty Guid.");

// Verify that the product can be searched afterwards.

Core.Models.Product? product = \_controller.SearchProduct(productId);

Assert.That(product, Is.Not.Null, "Product should be found after creation.");

Assert.That(product.Description, Is.EqualTo(description), "Product description should match.");

Assert.That(product.Price, Is.EqualTo(price), "Product price should match.");

}

[TestCase("Original Product A", 10.0, "Updated Product A", 15.0)]

[TestCase("Original Product B", 20.0, "Updated Product B", 25.0)]

[TestCase("Original Product C", 30.0, "Updated Product C", 35.0)]

public void EditProduct\_IntegrationTest\_ReturnsUpdatedProduct(

string initialDescription, double initialPrice,

string updatedDescription, double updatedPrice)

{

// Arrange

Guid productId = \_controller.CreateProduct(initialDescription, initialPrice);

Assert.That(productId, Is.Not.EqualTo(Guid.Empty));

// Act

Core.Models.Product? editedProduct = \_controller.EditProduct(productId, updatedDescription, updatedPrice);

// Assert

Assert.That(editedProduct, Is.Not.Null, "EditProduct should return the updated product.");

Assert.That(editedProduct.Description, Is.EqualTo(updatedDescription), "Product description should be updated.");

Assert.That(editedProduct.Price, Is.EqualTo(updatedPrice), "Product price should be updated.");

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void RemoveProduct\_IntegrationTest\_ReturnsRemovedProduct(string description, double price)

{

// Arrange

Guid productId = \_controller.CreateProduct(description, price);

Assert.That(productId, Is.Not.EqualTo(Guid.Empty));

// Act

Core.Models.Product? removedProduct = \_controller.RemoveProduct(productId);

// Assert

Assert.That(removedProduct, Is.Not.Null, "RemoveProduct should return the removed product.");

Assert.That(removedProduct.Id, Is.EqualTo(productId), "Removed product should have the same Id.");

// Verify that searching for the product now returns null.

Core.Models.Product? productAfterRemoval = \_controller.SearchProduct(productId);

Assert.That(productAfterRemoval, Is.Null, "Product should not be found after removal.");

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void SearchProduct\_IntegrationTest\_ReturnsProduct(string description, double price)

{

// Arrange

Guid productId = \_controller.CreateProduct(description, price);

Assert.That(productId, Is.Not.EqualTo(Guid.Empty));

// Act

Core.Models.Product? product = \_controller.SearchProduct(productId);

// Assert

Assert.That(product, Is.Not.Null, "SearchProduct should return the product if it exists.");

Assert.That(product.Description, Is.EqualTo(description), "Product description should match.");

Assert.That(product.Price, Is.EqualTo(price), "Product price should match.");

}

}

}

*TestsProductsController.cs*

using Microsoft.Extensions.Logging;

using Moq;

using ShopApp.Core.Models;

using ShopApp.Core.Services;

using ShopApp.WebApi.Controllers;

namespace ShopApp.Tests

{

[TestFixture]

internal class TestsProductsController

{

private Mock<ILogger<ProductsController>> \_mockLogger;

private Mock<IProductsService<Product>> \_mockProductService;

private ProductsController \_controller;

[SetUp]

public void SetUp()

{

// Create mocks for dependencies

\_mockLogger = new Mock<ILogger<ProductsController>>();

\_mockProductService = new Mock<IProductsService<Product>>();

// Initialize controller with mocked dependencies

\_controller = new ProductsController(\_mockLogger.Object, \_mockProductService.Object);

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void CreateProduct\_WhenServiceReturnsTrue\_ReturnsNonEmptyGuid(string description, double price)

{

\_ = \_mockProductService.Setup(s => s.Add(It.IsAny<Product>())).Returns(true);

// Act

Guid result = \_controller.CreateProduct(description, price);

// Assert

Assert.That(result, Is.Not.EqualTo(Guid.Empty), "CreateProduct should return a non-empty Guid if the product is successfully created.");

\_mockProductService.Verify(s => s.Add(It.Is<Product>(p => p.Description == description && p.Price == price)), Times.Once);

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void CreateProduct\_WhenServiceReturnsFalse\_ReturnsEmptyGuid(string description, double price)

{

\_ = \_mockProductService.Setup(s => s.Add(It.IsAny<Product>())).Returns(false);

// Act

Guid result = \_controller.CreateProduct(description, price);

// Assert

Assert.That(result, Is.EqualTo(Guid.Empty), "CreateProduct should return Guid.Empty if product creation fails.");

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void RemoveProduct\_WhenProductExists\_ReturnsProduct(string description, double price)

{

// Arrange

Product product = new()

{

Id = Guid.NewGuid(),

Description = description,

Price = price

};

\_ = \_mockProductService.Setup(s => s.Remove(product.Id)).Returns(product);

// Act

Product? result = \_controller.RemoveProduct(product.Id);

// Assert

Assert.That(result, Is.Not.Null, "RemoveProduct should return the product if removal is successful.");

Assert.That(result.Id, Is.EqualTo(product.Id), "Removed product should have the same Id.");

}

[Test]

public void RemoveProduct\_WhenProductDoesNotExist\_ReturnsNull()

{

// Arrange

Guid productId = Guid.NewGuid();

\_ = \_mockProductService.Setup(s => s.Remove(productId)).Returns(null as Product);

// Act

Product? result = \_controller.RemoveProduct(productId);

// Assert

Assert.That(result, Is.Null, "RemoveProduct should return null if the product is not found.");

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void EditProduct\_WhenProductExists\_ReturnsEditedProduct(string newDescription, double newPrice)

{

// Arrange

Product editedProduct = new()

{

Id = Guid.NewGuid(),

Description = newDescription,

Price = newPrice

};

\_ = \_mockProductService.Setup(s => s.Edit(It.Is<Product>(p => p.Id == editedProduct.Id && p.Description == newDescription && p.Price == newPrice)))

.Returns(editedProduct);

// Act

Product? result = \_controller.EditProduct(editedProduct.Id, newDescription, newPrice);

// Assert

Assert.That(result, Is.Not.Null, "EditProduct should return the updated product if editing is successful.");

Assert.That(result.Description, Is.EqualTo(newDescription), "Product description should be updated.");

Assert.That(result.Price, Is.EqualTo(newPrice), "Product price should be updated.");

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void EditProduct\_WhenProductDoesNotExist\_ReturnsNull(string newDescription, double newPrice)

{

// Arrange

Guid productId = Guid.NewGuid();

\_ = \_mockProductService.Setup(s => s.Edit(It.IsAny<Product>())).Returns(null as Product);

// Act

Product? result = \_controller.EditProduct(productId, newDescription, newPrice);

// Assert

Assert.That(result, Is.Null, "EditProduct should return null if the product is not found for editing.");

}

[TestCase("Product A", 10.0)]

[TestCase("Product B", 20.0)]

[TestCase("Product C", 30.0)]

public void SearchProduct\_WhenProductExists\_ReturnsProduct(string newDescription, double newPrice)

{

// Arrange

Product product = new()

{

Id = Guid.NewGuid(),

Description = newDescription,

Price = newPrice

};

\_ = \_mockProductService.Setup(s => s.Search(product.Id)).Returns(product);

// Act

Product? result = \_controller.SearchProduct(product.Id);

// Assert

Assert.That(result, Is.Not.Null, "SearchProduct should return the product if it exists.");

Assert.That(result.Id, Is.EqualTo(product.Id), "Found product should have the correct Id.");

}

[Test]

public void SearchProduct\_WhenProductDoesNotExist\_ReturnsNull()

{

// Arrange

Guid productId = Guid.NewGuid();

\_ = \_mockProductService.Setup(s => s.Search(productId)).Returns(null as Product);

// Act

Product? result = \_controller.SearchProduct(productId);

// Assert

Assert.That(result, Is.Null, "SearchProduct should return null if the product is not found.");

}

}

}

**Вывод**

Создали простое Web-приложение