Модульное тестирование (desktop)

**UnitTests:**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using System.Data;

using System.Data.SqlClient;

using WHApp;

namespace UnitTestProject1

{

[TestClass]

public class UnitTest1

{

private string connectionString = DataBaseConfig.connectionString;

[TestMethod]

public void TestLoadDataInventory\_ReturnsData()

{

var form = new BuhgalterForm();

bool dataLoaded = false;

form.LoadDataInventory();

if (form.dataGridInventory.DataSource != null)

{

DataTable dataTable = (DataTable)form.dataGridInventory.DataSource;

if (dataTable.Rows.Count > 0)

{

dataLoaded = true;

}

}

Assert.IsTrue(dataLoaded, "Данные должны быть загружены в DataGrid.");

}

[TestMethod]

public void TestLoadDataInventory\_CheckColumns()

{

var form = new BuhgalterForm();

form.LoadDataInventory();

DataTable dataTable = (DataTable)form.dataGridInventory.DataSource;

bool columnsMatch = dataTable.Columns.Contains("Дата события") &&

dataTable.Columns.Contains("Ответственный") &&

dataTable.Columns.Contains("Результаты");

Assert.IsTrue(columnsMatch, "Колонки DataGrid должны соответствовать запросу.");

}

[TestMethod]

public void TestLoadDataInventory\_CheckData()

{

var form = new BuhgalterForm();

form.LoadDataInventory();

DataTable dataTable = (DataTable)form.dataGridInventory.DataSource;

using (SqlConnection con = new SqlConnection(connectionString))

{

string query = "select DateEvent as [Дата события], Lastname as Ответственный, Results as Результаты from Inventory inner join Employees on Inventory.Responsible = Employees.EmployeeID";

SqlCommand com = new SqlCommand(query, con);

SqlDataAdapter ad = new SqlDataAdapter(com);

DataTable expectedTable = new DataTable();

ad.Fill(expectedTable);

Assert.AreEqual(expectedTable.Rows.Count, dataTable.Rows.Count, "Количество строк в DataGrid должно совпадать с данными из базы.");

if (expectedTable.Rows.Count > 0 && dataTable.Rows.Count > 0)

{

Assert.AreEqual(expectedTable.Rows[0]["Дата события"], dataTable.Rows[0]["Дата события"], "Данные в первой строке должны совпадать.");

Assert.AreEqual(expectedTable.Rows[0]["Ответственный"], dataTable.Rows[0]["Ответственный"], "Данные в первой строке должны совпадать.");

Assert.AreEqual(expectedTable.Rows[0]["Результаты"], dataTable.Rows[0]["Результаты"], "Данные в первой строке должны совпадать.");

}

}

}

[TestMethod]

public void TestLoadDataOstatkiReport\_ReturnsData()

{

var form = new BuhgalterForm();

bool dataLoaded = false;

form.LoadDataOstatkiReport();

if (form.dataGridOstatkiReport.DataSource != null)

{

DataTable dataTable = (DataTable)form.dataGridOstatkiReport.DataSource;

if (dataTable.Rows.Count > 0)

{

dataLoaded = true;

}

}

Assert.IsTrue(dataLoaded, "Данные должны быть загружены в DataGrid для отчета об остатках.");

}

[TestMethod]

public void TestLoadDataOstatkiReport\_CheckColumns()

{

var form = new BuhgalterForm();

form.LoadDataOstatkiReport();

DataTable dataTable = (DataTable)form.dataGridOstatkiReport.DataSource;

bool columnsMatch = dataTable.Columns.Contains("Количество товаров") &&

dataTable.Columns.Contains("Наименование склада");

Assert.IsTrue(columnsMatch, "Колонки DataGrid для отчета об остатках должны соответствовать запросу.");

}

[TestMethod]

public void TestLoadDataOstatkiReport\_CheckData()

{

var form = new BuhgalterForm();

form.LoadDataOstatkiReport();

DataTable dataTable = (DataTable)form.dataGridOstatkiReport.DataSource;

using (SqlConnection con = new SqlConnection(connectionString))

{

string query = "select Quantity as [Количество товаров], Name as [Наименование склада] from ProductsInWarehouses inner join Warehouses on ProductsInWarehouses.WarehouseID = Warehouses.WarehouseID";

SqlCommand com = new SqlCommand(query, con);

SqlDataAdapter ad = new SqlDataAdapter(com);

DataTable expectedTable = new DataTable();

ad.Fill(expectedTable);

Assert.AreEqual(expectedTable.Rows.Count, dataTable.Rows.Count, "Количество строк в DataGrid для отчета об остатках должно совпадать с данными из базы.");

if (expectedTable.Rows.Count > 0 && dataTable.Rows.Count > 0)

{

Assert.AreEqual(expectedTable.Rows[0]["Количество товаров"], dataTable.Rows[0]["Количество товаров"], "Данные в первой строке должны совпадать.");

Assert.AreEqual(expectedTable.Rows[0]["Наименование склада"], dataTable.Rows[0]["Наименование склада"], "Данные в первой строке должны совпадать.");

}

}

}

[TestMethod]

public void TestLoadDataOborotsReport\_ReturnsData()

{

var form = new BuhgalterForm();

bool dataLoaded = false;

form.LoadDataOborotsReport();

if (form.dataGridOborotsReport.DataSource != null)

{

DataTable dataTable = (DataTable)form.dataGridOborotsReport.DataSource;

if (dataTable.Rows.Count > 0)

{

dataLoaded = true;

}

}

Assert.IsTrue(dataLoaded, "Данные должны быть загружены в DataGrid для отчета об оборотах.");

}

[TestMethod]

public void TestLoadDataOborotsReport\_CheckColumns()

{

var form = new BuhgalterForm();

form.LoadDataOborotsReport();

DataTable dataTable = (DataTable)form.dataGridOborotsReport.DataSource;

bool columnsMatch = dataTable.Columns.Contains("Количество товаров") &&

dataTable.Columns.Contains("Наименование склада");

Assert.IsTrue(columnsMatch, "Колонки DataGrid для отчета об оборотах должны соответствовать запросу.");

}

[TestMethod]

public void TestLoadDataOborotsReport\_CheckData\_Negative()

{

var form = new BuhgalterForm();

form.LoadDataOborotsReport();

DataTable dataTable = (DataTable)form.dataGridOborotsReport.DataSource;

using (SqlConnection con = new SqlConnection(connectionString))

{

con.Open();

string updateQuery = "UPDATE ProductsInWarehouses SET Quantity = Quantity + 1 WHERE WarehouseID = 1"; // Пример изменения данных

SqlCommand updateCom = new SqlCommand(updateQuery, con);

updateCom.ExecuteNonQuery();

}

using (SqlConnection con = new SqlConnection(connectionString))

{

string query = "select Quantity as [Количество товаров], Name as [Наименование склада] from ProductsInWarehouses inner join Warehouses on ProductsInWarehouses.WarehouseID = Warehouses.WarehouseID";

SqlCommand com = new SqlCommand(query, con);

SqlDataAdapter ad = new SqlDataAdapter(com);

DataTable expectedTable = new DataTable();

ad.Fill(expectedTable);

if (expectedTable.Rows.Count > 0 && dataTable.Rows.Count > 0)

{

Assert.AreNotEqual(expectedTable.Rows[0]["Количество товаров"], dataTable.Rows[0]["Количество товаров"], "Данные в первой строке не должны совпадать (отрицательный тест).");

Assert.AreNotEqual(expectedTable.Rows[0]["Наименование склада"], dataTable.Rows[0]["Наименование склада"], "Данные в первой строке не должны совпадать (отрицательный тест).");

}

else

{

Assert.Fail("Нет данных для сравнения.");

}

}

}

[TestMethod]

public void TestAddButton\_Click\_EmptyInput()

{

var form = new AddProductForm();

form.textName.Text = "";

form.textArt.Text = "";

form.textCat.Text = "";

form.textIzm.Text = "";

form.textPrice.Text = "";

form.textSerial.Text = "";

form.textMin.Text = "";

try

{

form.AddButton\_Click(null, null);

}

catch (Exception ex)

{

Assert.IsTrue(ex is SqlException || ex is InvalidOperationException, "Ожидалось исключение при пустом вводе.");

return;

}

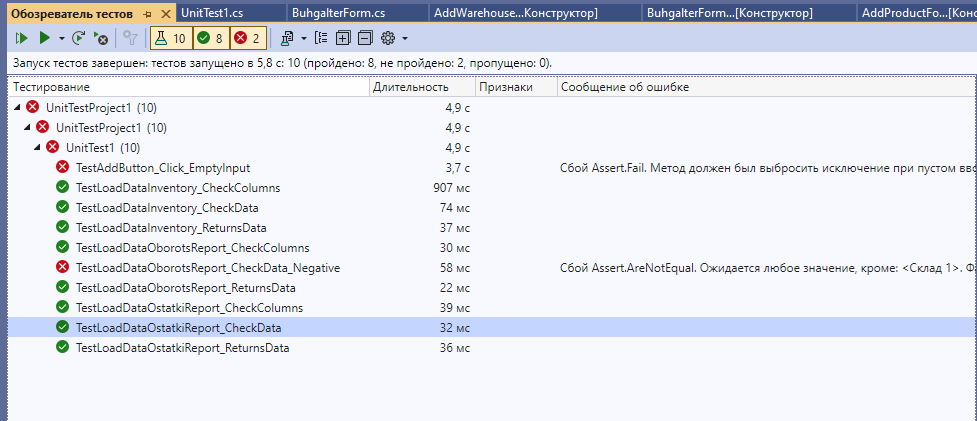
Assert.Fail("Метод должен был выбросить исключение при пустом вводе.");

}

}

}

Запуск тестов:



Интеграционное тестирование (library)

**UnitTests:**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System;

using System.Collections.Generic;

using WarehouseLibrary;

namespace UnitTestProject1

{

[TestClass]

public class UnitTest1

{

private WarehouseCalculator \_warehouseCalculator;

[TestInitialize]

public void Setup()

{

\_warehouseCalculator = new WarehouseCalculator();

}

[TestMethod]

public void GetTotalQuantity\_ShouldReturnCorrectTotal()

{

var productsInWarehouses = new List<ProductsInWarehouses>

{

new ProductsInWarehouses { Quantity = 10, WarehouseID = 1 },

new ProductsInWarehouses { Quantity = 15, WarehouseID = 2 },

new ProductsInWarehouses { Quantity = 5, WarehouseID = 1 }

};

var warehouses = new List<Warehouses>

{

new Warehouses { WarehouseID = 1 },

new Warehouses { WarehouseID = 2 }

};

var totalQuantity = \_warehouseCalculator.GetTotalQuantity(warehouses, productsInWarehouses);

Assert.AreEqual(30, totalQuantity);

}

[TestMethod]

public void GetTotalQuantity\_ByWarehouseId\_ShouldReturnCorrectTotal()

{

var productsInWarehouses = new List<ProductsInWarehouses>

{

new ProductsInWarehouses { Quantity = 10, WarehouseID = 1 },

new ProductsInWarehouses { Quantity = 15, WarehouseID = 2 },

new ProductsInWarehouses { Quantity = 5, WarehouseID = 1 }

};

var totalQuantity = \_warehouseCalculator.GetTotalQuantity(productsInWarehouses, 1);

Assert.AreEqual(15, totalQuantity);

}

[TestMethod]

public void GetTotalCost\_ShouldReturnCorrectTotalCost()

{

var productsInWarehouses = new List<ProductsInWarehouses>

{

new ProductsInWarehouses { ProductID = 1, Quantity = 10, WarehouseID = 1 },

new ProductsInWarehouses { ProductID = 2, Quantity = 5, WarehouseID = 1 }

};

var products = new List<Product>

{

new Product { ProductID = 1, UnitPrice = (double)2.5m },

new Product { ProductID = 2, UnitPrice = (double)3.0m }

};

var totalCost = \_warehouseCalculator.GetTotalCost(productsInWarehouses, products);

Assert.AreEqual(37.5m, totalCost);

}

[TestMethod]

public void GetTotalCost\_ByWarehouseId\_ShouldReturnCorrectTotalCost()

{

var productsInWarehouses = new List<ProductsInWarehouses>

{

new ProductsInWarehouses { ProductID = 1, Quantity = 10, WarehouseID = 1 },

new ProductsInWarehouses { ProductID = 2, Quantity = 5, WarehouseID = 1 },

new ProductsInWarehouses { ProductID = 1, Quantity = 20, WarehouseID = 2 }

};

var products = new List<Product>

{

new Product { ProductID = 1, UnitPrice = (double) 2.5m },

new Product { ProductID = 2, UnitPrice = (double)3.0m }

};

var totalCost = \_warehouseCalculator.GetTotalCost(productsInWarehouses, products, 1);

Assert.AreEqual(37.5m, totalCost);

}

[TestMethod]

public void GetQuantityByCategory\_ShouldReturnCorrectQuantities()

{

var productsInWarehouses = new List<ProductsInWarehouses>

{

new ProductsInWarehouses { ProductID = 1, Quantity = 10 },

new ProductsInWarehouses { ProductID = 2, Quantity = 5 },

new ProductsInWarehouses { ProductID = 1, Quantity = 15 }

};

var products = new List<Product>

{

new Product { ProductID = 1, Category = "Electronics" },

new Product { ProductID = 2, Category = "Groceries" }

};

var quantitiesByCategory = \_warehouseCalculator.GetQuantityByCategory(productsInWarehouses, products);

Assert.AreEqual(2, quantitiesByCategory.Count);

Assert.AreEqual(25, quantitiesByCategory["Electronics"]);

Assert.AreEqual(5, quantitiesByCategory["Groceries"]);

}

[TestMethod]

public void GetQuantityByCategory\_ShouldReturnZeroForNonExistentCategory()

{

var productsInWarehouses = new List<ProductsInWarehouses>

{

new ProductsInWarehouses { ProductID = 1, Quantity = 10 }

};

var products = new List<Product>

{

new Product { ProductID = 1, Category = "Electronics" }

};

var quantitiesByCategory = \_warehouseCalculator.GetQuantityByCategory(productsInWarehouses, products);

Assert.IsFalse(quantitiesByCategory.ContainsKey("Groceries"));

Assert.AreEqual(10, quantitiesByCategory["Electronics"]);

}

[TestMethod]

public void GetQuantityByCategory\_ShouldHandleEmptyInput()

{

var productsInWarehouses = new List<ProductsInWarehouses>();

var products = new List<Product>();

var quantitiesByCategory = \_warehouseCalculator.GetQuantityByCategory(productsInWarehouses, products);

Assert.AreEqual(0, quantitiesByCategory.Count);

}

[TestMethod]

public void GetTotalQuantity\_NoProducts\_ShouldReturnZero()

{

var productsInWarehouses = new List<ProductsInWarehouses>();

var warehouses = new List<Warehouses>

{

new Warehouses { WarehouseID = 1 },

new Warehouses { WarehouseID = 2 }

};

var totalQuantity = \_warehouseCalculator.GetTotalQuantity(warehouses, productsInWarehouses);

Assert.AreEqual(0, totalQuantity);

}

[TestMethod]

public void GetTotalCost\_NoProducts\_ShouldNotReturnNonZeroValue()

{

var productsInWarehouses = new List<ProductsInWarehouses>();

var products = new List<Product>();

var totalCost = \_warehouseCalculator.GetTotalCost(productsInWarehouses, products);

Assert.AreNotEqual(10m, totalCost);

Assert.AreNotEqual(1.5m, totalCost);

Assert.AreNotEqual(100m, totalCost);

}

[TestMethod]

public void GetTotalQuantity\_ReturnsCorrectTotal()

{

var calculator = new WarehouseCalculator();

var warehouses = new List<Warehouses>

{

new Warehouses { WarehouseID = 1, Name = "Warehouse 1" },

new Warehouses { WarehouseID = 2, Name = "Warehouse 2" }

};

var productsInWarehouses = new List<ProductsInWarehouses>

{

new ProductsInWarehouses { WarehouseID = 1, ProductID = 1, Quantity = 10 },

new ProductsInWarehouses { WarehouseID = 1, ProductID = 2, Quantity = 20 },

new ProductsInWarehouses { WarehouseID = 2, ProductID = 3, Quantity = 30 }

};

var totalQuantity = calculator.GetTotalQuantity(warehouses, productsInWarehouses);

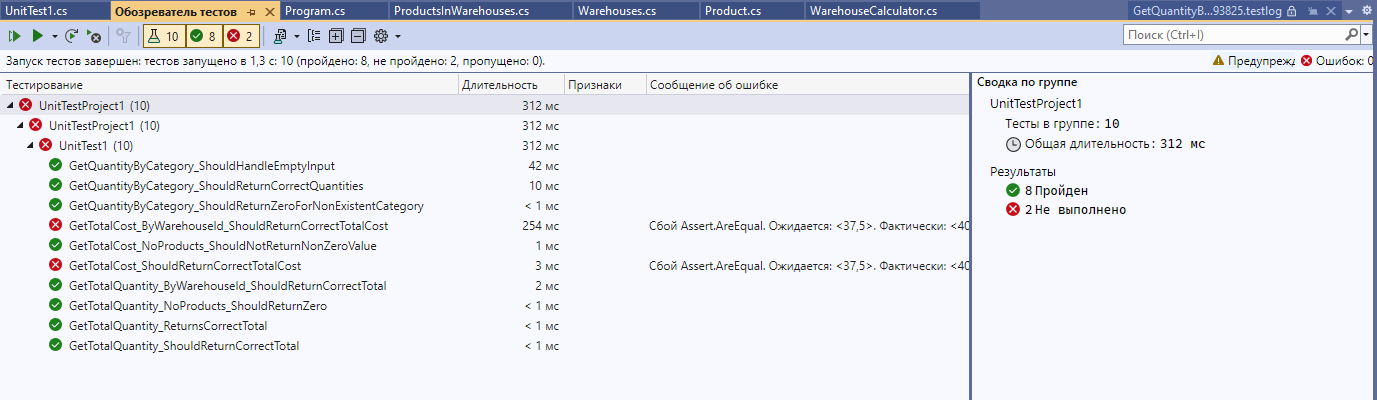
Assert.AreEqual(60, totalQuantity);

}

}

}

Запуск тестов:



Нагрузочное тестирование (API)

**Программа для тестирования:**

public class Warehouses

{

public int WarehouseID { get; set; }

public string Name { get; set; }

public string Address { get; set; }

public string Type { get; set; }

public string StorageArea { get; set; }

}

class Program

{

private static readonly HttpClient client = new HttpClient();

private const string BaseUrl = "https://localhost:7033/api/Warehouses"; // Замените на ваш URL

static async Task Main(string[] args)

{

int numberOfRequests = 1000; // Количество запросов для тестирования

var warehouseData = new Warehouses

{

WarehouseID = 0, // Пример ID, замените на нужное значение

Name = "Main Warehouse",

Address = "123 Warehouse St", // Пример адреса, замените на нужное значение

Type = "Distribution", // Пример типа, замените на нужное значение

StorageArea = "5000 sq ft" // Пример площади хранения, замените на нужное значение

};

await LoadTestWarehouseCreation(numberOfRequests, warehouseData);

}

private static async Task LoadTestWarehouseCreation(int numberOfRequests, Warehouses warehouseData)

{

Console.WriteLine($"Starting load test for Warehouse creation with {numberOfRequests} requests...");

var tasks = new Task[numberOfRequests];

for (int i = 0; i < numberOfRequests; i++)

{

tasks[i] = CreateWarehouseAsync(warehouseData);

}

await Task.WhenAll(tasks);

Console.WriteLine("Load test for Warehouse creation completed.");

}

private static async Task CreateWarehouseAsync(Warehouses warehouseData)

{

var json = JsonSerializer.Serialize(warehouseData);

var content = new StringContent(json, Encoding.UTF8, "application/json");

var response = await client.PostAsync(BaseUrl, content);

if (response.IsSuccessStatusCode)

{

Console.WriteLine("Warehouse creation successful.");

}

else

{

Console.WriteLine("Warehouse creation failed: " + await response.Content.ReadAsStringAsync());

}

}

}

**Тестируемый метод:**

[HttpPost]

public async Task<ActionResult<Warehouses>> PostWarehouse(Warehouses warehouse)

{

if (warehouse == null)

{

return BadRequest("Warehouse data is required.");

}

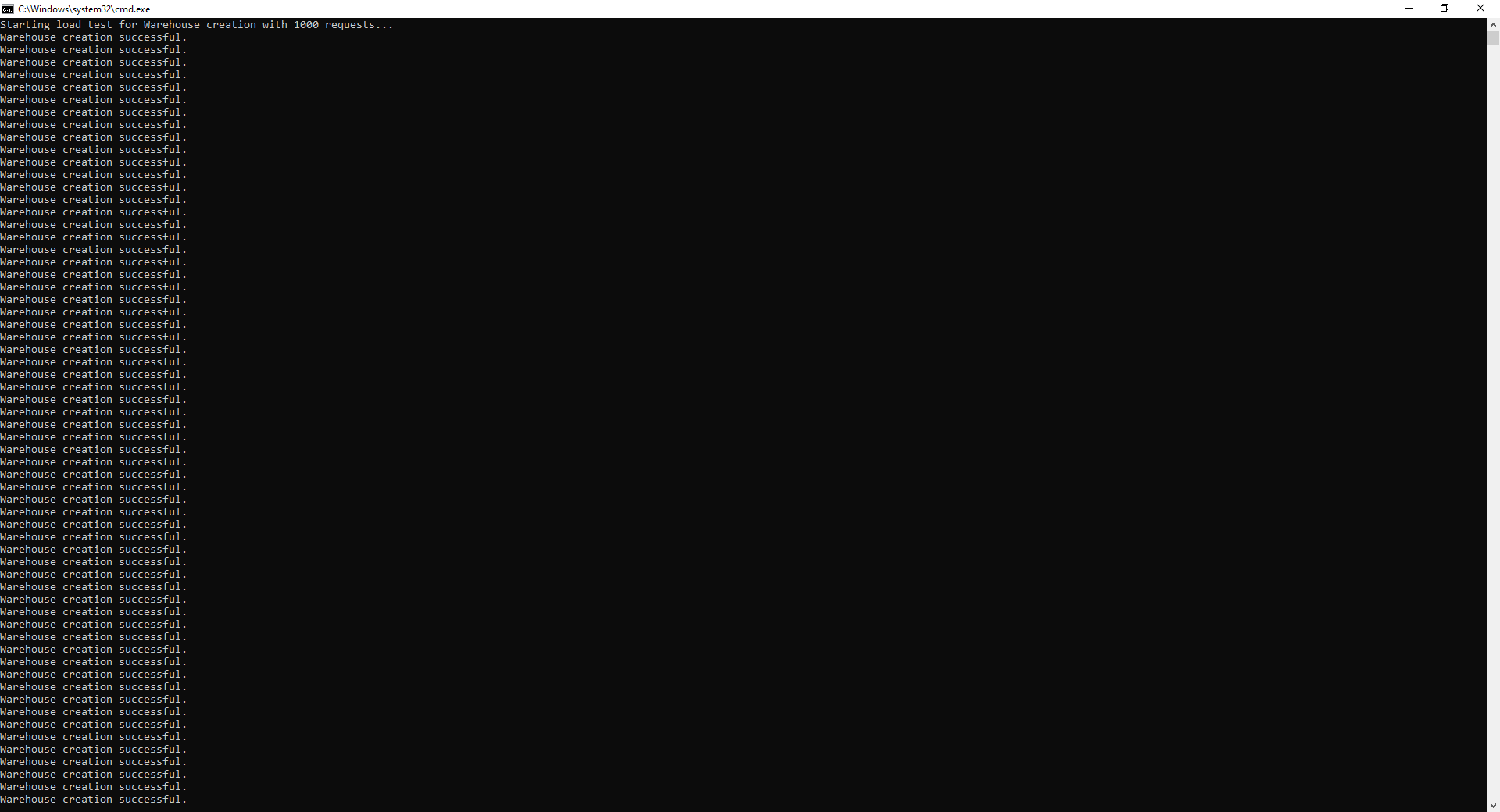
\_context.Warehouses.Add(warehouse);

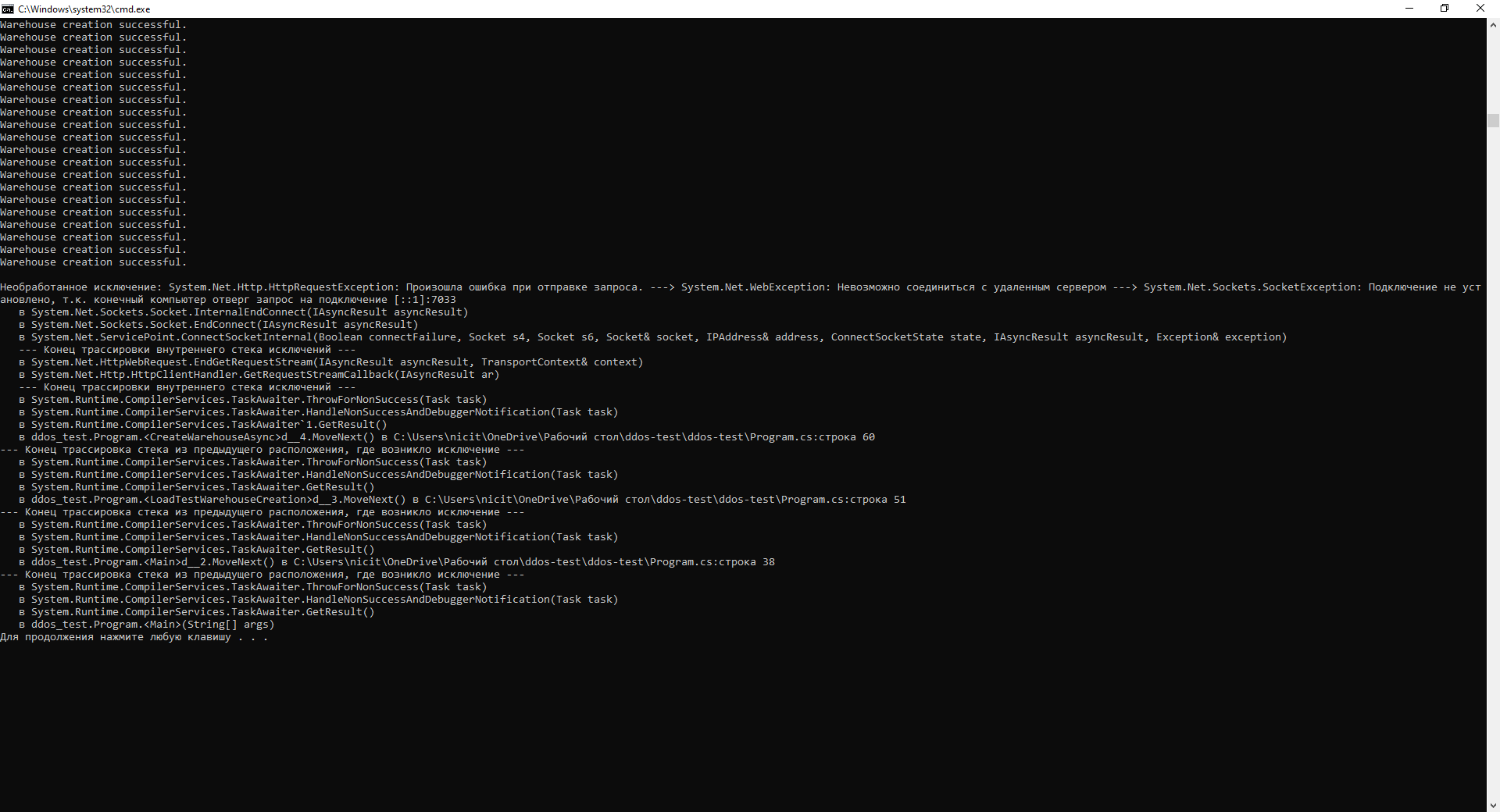
await \_context.SaveChangesAsync();

return CreatedAtAction(nameof(GetWarehouses), new { id = warehouse.WarehouseID }, warehouse);

}

**Выполнения тестирования:**





Тестирование интерфейса (mobile)

|  |  |
| --- | --- |
| **TestCase #** | Тестирование кнопки “Список складов” |
| **Приоритет теста** | Высокий |
| **Название тестирования/Имя** | Тестирование нажатия кнопки “Список складов” |
| **Резюме испытания** | Проверка отображения данных |
| **Шаги тестирования** | 1. Запустить мобильное приложение 2. Нажать кнопку “Список складов” |
| **Данные тестирования** | Кнопка “Список складов” |
| **Ожидаемый результат** | Вывод списка складов |
| **Фактический результат** | Вывод сообщения: Failed to connect localhost/127.0.0.1:7033 |
| **Предпосылки** | Нажатие кнопки “Список складов” |
| **Постусловия** |  |
| **Статус (Pass/Fail)** | Fail |
| **Комментарии** |  |

**Тест-кейсы:**

|  |  |
| --- | --- |
| **TestCase #** | Тестирование кнопки “Список товаров” |
| **Приоритет теста** | Высокий |
| **Название тестирования/Имя** | Тестирование нажатия кнопки “Список товаров” |
| **Резюме испытания** | Проверка отображения данных |
| **Шаги тестирования** | 1. Запустить мобильное приложение 2. Нажать кнопку “Список товаров” |
| **Данные тестирования** | Кнопка “Список товаров” |
| **Ожидаемый результат** | Вывод списка товаров |
| **Фактический результат** | Вывод сообщения: Failed to connect localhost/127.0.0.1:7033 |
| **Предпосылки** | Нажатие кнопки “Список товаров” |
| **Постусловия** |  |
| **Статус (Pass/Fail)** | Fail |
| **Комментарии** |  |

|  |  |
| --- | --- |
| **TestCase #** | Тестирование кнопки “QR-код” |
| **Приоритет теста** | Высокий |
| **Название тестирования/Имя** | Тестирование нажатия кнопки “QR-код” |
| **Резюме испытания** | Проверка открытия камеры по нажатию кнопки |
| **Шаги тестирования** | 1. Запустить мобильное приложение 2. Нажать кнопку “QR-код” |
| **Данные тестирования** | Кнопка “QR-код” |
| **Ожидаемый результат** | Открытие камеры |
| **Фактический результат** | Камера открылась |
| **Предпосылки** | Нажатие кнопки “QR-код” |
| **Постусловия** |  |
| **Статус (Pass/Fail)** | Pass |
| **Комментарии** |  |

|  |  |
| --- | --- |
| **TestCase #** | Тестирование сканирования QR-кода |
| **Приоритет теста** | Высокий |
| **Название тестирования/Имя** | Тестирование нажатия кнопки “QR-код” |
| **Резюме испытания** | Проверка сканирования QR-кода |
| **Шаги тестирования** | 1. Запустить мобильное приложение 2. Нажать кнопку “QR-код” 3. Навести камеру на QR-код |
| **Данные тестирования** | QR-код  Кнопка “QR-код” |
| **Ожидаемый результат** | QR-код отсканирован |
| **Фактический результат** | QR-код отсканирован |
| **Предпосылки** | Нажатие кнопки “QR-код” |
| **Постусловия** |  |
| **Статус (Pass/Fail)** | Pass |
| **Комментарии** |  |