

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования  
Кафедра проектирования информационно-компьютерных систем

## **ОТЧЕТ**

### **ПО ИНДИВИДУАЛЬНОЙ ПРАКТИЧЕСКОЙ РАБОТЕ №3**

по дисциплине

«Современные технологии проектирования информационных систем»

специальности 1-40 05 01-10 «Информационные системы и технологии (в  
бизнес-менеджменте)»

Выполнил:  
студент группы №994351  
Богомаз Д. Л.

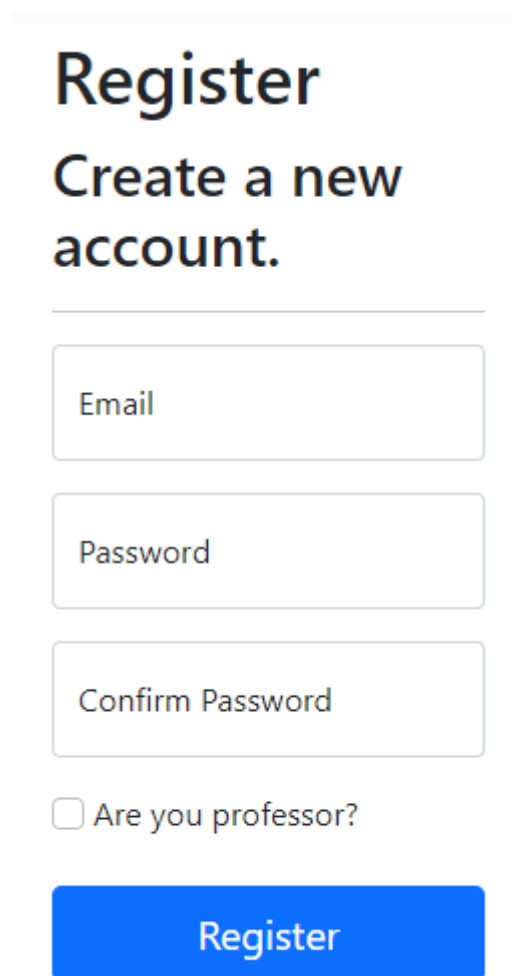
Проверила: Левченя Ж.Б.

Минск 2023

Цель: практическая разработка unit-тестов на примере разрабатываемого приложения.

В данной работе выполняется вариант номер 3: Программные средства мониторинга и консультативной поддержки процесса изучения дисциплин вуза.

В ходе работы прошлого семестра было создано приложение на dotnet 7, mvc тип, где в качестве frontend выбран razor pages. Базовая логика создана с помощью MVC паттерна проектирования, в качестве логина использован scaffold identity default с изменением ролей для преподавателей, также были удалены неиспользованный функционал, как работа с почтой. Также на форме регистрации добавлен выбор режима регистрации:

A screenshot of a web registration form. At the top, the text 'Register' is in a large, bold, dark blue font, followed by 'Create a new account.' in a slightly smaller, bold, dark blue font. Below this is a horizontal line. There are three input fields stacked vertically: 'Email', 'Password', and 'Confirm Password'. Each field has a light blue border and a light blue placeholder text. Below the input fields is a checkbox with the text 'Are you professor?'. At the bottom is a solid blue button with the text 'Register' in white.

В качестве базы данных используется sql lite, как orm используется entity framework, automapper не используется, так как является излишним для подобного формата работы (mappings реализованы как статический класс).

Ссылка на проект: <https://github.com/bogomazdmitry/LmsCopy>

## Теоретические сведения

С Unit-тестирование — это методология разработки программного обеспечения, которая позволяет проверять отдельные компоненты (или "юниты") программы на корректность их работы. Unit-тесты представляют собой автоматизированные тесты, которые проверяют, выполняется ли ожидаемое поведение каждой функции или метода в изоляции от других компонентов системы.

Основные плюсы unit-тестирования:

- Раннее выявление ошибок: Unit-тесты позволяют обнаружить ошибки еще на ранних стадиях разработки, когда их исправление гораздо проще и дешевле.
- Улучшение качества кода: Написание тестов принуждает программиста к более четкому определению требований и спецификации компонента.
- Упрощение рефакторинга: При наличии хорошо написанных тестов программист может быть уверен, что изменения, внесенные в код во время рефакторинга, не привели к появлению новых ошибок.
- Документация: Unit-тесты могут служить в качестве документации, позволяя другим разработчикам быстро понять, как использовать и как работает компонент.

Некоторые минусы unit-тестирования:

- Затраты на написание тестов: Написание и поддержка тестов требует времени и ресурсов, особенно при разработке сложных систем.
- Недостаточное покрытие: Не всегда возможно написать полное покрытие всех возможных сценариев и ветвлений в коде.
- Зависимость от реализации: Иногда тесты могут быть сильно связаны с конкретной реализацией, что делает их хрупкими и требует изменений при каждом изменении кода.

Unit-тесты следует использовать в следующих случаях:

- При разработке новых компонентов: Unit-тестирование помогает убедиться, что новый код работает правильно и не вызывает ошибок.
- При внесении изменений в существующий код: Unit-тесты позволяют проверить, что изменения не сломали уже существующий функционал.
- В командной разработке: Unit-тесты способствуют интеграции кода от разных разработчиков и помогают избежать конфликтов.

NUnit — это один из популярных фреймворков для unit-тестирования в языке программирования C#. Он предоставляет множество функций и возможностей для удобного написания и запуска тестов.

Особенности и возможности NUnit:

- Атрибуты: NUnit использует атрибуты для определения тестовых методов, фикстур (наборов тестов) и настроек.
- Утверждения (Assertions): NUnit предоставляет разнообразные методы утверждений, которые позволяют проверять ожидаемые результаты.
- Фикстуры (Fixtures): Фикстуры позволяют группировать связанные тесты и управлять состоянием инициализации и очистки для наборов тестов.
- Параметризованные тесты: NUnit поддерживает параметризованные тесты, позволяя запустить один и тот же тест с разными наборами параметров.
- Маркирование тестов: NUnit позволяет помечать тесты атрибутами, чтобы управлять их запуском и выполнять специфические действия перед или после каждого теста.
- Покрываемость кода: NUnit интегрируется с инструментами для анализа покрытия кода, позволяя оценить, насколько хорошо тесты охватывают код.

NUnit обеспечивает простой и эффективный способ написания и выполнения unit-тестов в C#, помогая разработчикам достичь высокого качества кода и быстрой обратной связи о его работоспособности.

### Листинг кода

CsvFileReportTests:

namespace LmsCopy.Web.Services.Tests;

[TestFixture]

public class CsvFileReportTests

{

    [Test]

    public void GenerateReport\_ReturnsFileModelWithCorrectContentType()

    {

        // Arrange

        var report = new CsvFileReport();

        var items = new List<int> { 1, 2, 3 };

        // Act

        var fileModel = report.GenerateReport(items);

        // Assert

```
Assert.AreEqual("text/plain", fileModel.ContentType);  
}
```

```
[Test]
```

```
public void GenerateReport_ReturnsFileModelWithCorrectFileName()  
{
```

```
    // Arrange
```

```
    var report = new CsvFileReport();
```

```
    report.FileName = "Report";
```

```
    var items = new List<decimal> { 1.23m, 4.56m, 7.89m };
```

```
    // Act
```

```
    var fileModel = report.GenerateReport(items);
```

```
    // Assert
```

```
    Assert.AreEqual("Report.csv", fileModel.FileName);
```

```
}
```

```
}
```

```
JsonFileReportTests:
```

```
using NUnit.Framework;
```

```
using System.Collections.Generic;
```

```
using System.Text;
```

```
using System.Text.Json;
```

```
namespace LmsCopy.Web.Services.Tests;
```

```
[TestFixture]
```

```
public class JsonFileReportTests
```

```
{
```

```
    [Test]
```

```
    public void GenerateReport_ReturnsFileModelWithCorrectContentType()
```

```
    {
```

```
        // Arrange
```

```
        var report = new JsonFileReport();
```

```
        var items = new List<int> { 1, 2, 3 };
```

```
        // Act
```

```
        var fileModel = report.GenerateReport(items);
```

```
    // Assert
    Assert.AreEqual("text/plain", fileModel.ContentType);
}
```

```
[Test]
public void GenerateReport_ReturnsFileModelWithCorrectFileName()
{
    // Arrange
    var report = new JsonFileReport();
    report.FileName = "Report";
    var items = new List<decimal> { 1.23m, 4.56m, 7.89m };
    var expectedFileName = "Report.json";

    // Act
    var fileModel = report.GenerateReport(items);

    // Assert
    Assert.AreEqual(expectedFileName, fileModel.FileName);
}
}
```

XmlFileReportTests:

```
using System.Text;
using System.Xml.Serialization;
```

```
namespace LmsCopy.Web.Services.Tests;
```

```
[TestFixture]
public class XmlFileReportTests
{
    [Test]
    public void GenerateReport_ReturnsFileModelWithCorrectContentType()
    {
        // Arrange
        var report = new XmlFileReport();
        var items = new List<int> { 1, 2, 3 };

        // Act
        var fileModel = report.GenerateReport(items);
    }
}
```

```

        // Assert
        Assert.AreEqual("text/plain", fileModel.ContentType);
    }

[Test]
public void GenerateReport_ReturnsFileModelWithCorrectFileName()
{
    // Arrange
    var report = new XmlFileReport();
    report.FileName = "Report";
    var items = new List<decimal> { 1.23m, 4.56m, 7.89m };
    var expectedFileName = "Report.xml";

    // Act
    var fileModel = report.GenerateReport(items);

    // Assert
    Assert.AreEqual(expectedFileName, fileModel.FileName);
}

// Helper method to serialize items to XML
private string GetSerializedXml<T>(List<T> items, XmlSerializer
serializer)
{
    using var writer = new StringWriter();
    serializer.Serialize(writer, items);
    return writer.ToString();
}
}

```

Вывод: тестирование является неотъемлемой частью разработки программного обеспечения, позволяющей обнаружить ошибки, улучшить качество кода и обеспечить надежность функционала. Покрывание тестами для различных форматов файлов (Xml, Json, Csv) гарантирует правильность работы соответствующих компонентов, повышая надежность и качество вашего кода.