UNIT TESTING WITH NUnit – MY FIRST EXPERIENCE

1. What is Unit Testing?

Unit testing means testing one small part of your code (a unit) to make sure it works as expected.
For example, I created a Calculator class. To check if the addition method works, I tested: 5 + 10 = 15?

This is how we verify our code before using it in bigger projects.

2. Unit Testing vs. Functional Testing

Unit Testing:
- Tests a small piece of code (like a method)
- Written by developers
- Very fast
- Uses mocks

Functional Testing:
- Tests an entire feature or flow
- Often done by testers
- Slower
- Tests from user perspective

3. Mocking Dependencies

If my method depends on a database or API, I use a fake/mock version of it.
This makes the test simple, fast, and independent of other systems.

4. Types of Testing I Learned

- Unit Testing
- Functional Testing
- Automated Testing
- Performance Testing

5. Why Automated Testing is Useful

- Fast and repeatable
- No human errors
- Finds bugs early

- Works well with CI/CD pipelines

6. Loosely Coupled & Testable Design

To make your code testable:
- Don’t create dependencies inside your class
- Use dependency injection (pass them in from outside)
- Depend on interfaces, not specific classes

7. My First NUnit Test Program

Steps I followed:

Step 1: Create a test project
- Added a new NUnit Test Project (.NET Framework)

Step 2: Add reference to my Calculator project

Step 3: Wrote this test code:

```
[TestFixture]
public class CalculatorTests {
    private SimpleCalculator _calculator;

    [SetUp]
    public void Setup() => _calculator = new SimpleCalculator();

    [TearDown]
    public void Teardown() => _calculator = null;

    [Test]
    public void Addition_TwoPositiveNumbers_ReturnsCorrectSum() {
        Assert.That(_calculator.Addition(5, 10), Is.EqualTo(15));
    }

    [TestCase(2, 3, 5)]
    [TestCase(1.5, 2.5, 4.0)]
    public void Addition_WithVariousInputs_ReturnsExpected(double a, double b, double
expected) {
        Assert.That(_calculator.Addition(a, b), Is.EqualTo(expected));
    }

    [Test]
    [Ignore("Test disabled for now.")]
```

```
    public void Subtraction_IgnoredTest() {
        Assert.Fail("This should not run.");
    }
}
```

8. NUnit Attributes I Used

[TestFixture] - Declares this class as a test class
[SetUp] - Runs before each test
[TearDown] - Runs after each test
[Test] - Marks a test method
[TestCase] - Runs test with different values
[Ignore] - Skips a test

9. Running the Tests

- Built the project
- Opened Test Explorer
- Clicked “Run All”
- Green = Passed, Yellow = Ignored

10. What I Learned

- Unit testing helps me catch bugs early
- NUnit is beginner-friendly
- Now I feel confident testing real code!