Hands-On Exercise: Unit Testing with NUnit and Moq in C#

Submitted by- Richa Goyal(6365275)

1. Objective Overview

This hands-on exercise is designed to implement **Unit Testing using NUnit and Moq** in C#. Through this we:

- Understand how Mocking supports Test-Driven Development (TDD).
- Learn to isolate dependencies using Mocks, Fakes, and Stubs.
- Apply Dependency Injection to create testable designs. Write testable code for a mock email-sending module.
- Mock external services like databases and file systems.

2. Key Concepts

What is Mocking in Unit Testing?

-Mocking is the act of simulating the behavior of real objects in controlled ways using mock objects.

Why Use Mocks in Unit Testing?

- Isolate dependencies
- Avoid reliance on external systems
- Simulate behaviors and test edge cases
- Speed up testing and ensure reliability

Test Doubles: Mock vs Stub vs Fake

- **Mock**: Pre-programmed with expectations which form a test oracle.
- **Stub**: Returns predefined responses but does not fail the test.
- **Fake**: Has working implementations but is not suitable for production.

Benefits of TDD (Test-Driven Development)

- Promotes better design and loosely coupled code
- Identifies bugs early
- Encourages modular development

3. Dependency Injection in Unit Testing

Dependency Injection (DI) is the practice of passing dependencies to a class, rather than hard-coding them inside.

 Constructor Injection: Dependencies are provided via constructor.
Method Injection: Dependencies are passed to methods.

DI helps in mocking the dependencies for test purposes.

4. Task 1: Create Testable Code with Mockable Dependency

Step 1: Create the Class Library dotnet new classlib -n CustomerCommLib dotnet sln add CustomerCommLib/CustomerCommLib.csproj

Step 2: Edit the Code

Open MailSender.cs and write:

```
using System.Net;
using System.Net.Mail;

namespace CustomerCommLib
{
   public interface IMailSender
   {
      bool SendMail(string toAddress, string message);
   }

   public class MailSender : IMailSender
   {
      public bool SendMail(string toAddress, string message)
      {
         MailMessage mail = new MailMessage();
         SmtpClient smtpServer = new SmtpClient("smtp.gmail.com")
         {
            Port = 587,
            Credentials = new NetworkCredential("username", "password"),
            EnableSsl = true
         };
```

```
            mail.From = new MailAddress("your_email_address@gmail.com");
            mail.To.Add(toAddress); mail.Subject = "Test Mail"; mail.Body =
            message;

            smtpServer.Send(mail);
            return true;
        }
    }

    public class CustomerComm
    {
        private readonly IMailSender _mailSender;

        public CustomerComm(IMailSender mailSender)
        {
            _mailSender = mailSender;
        }

        public bool SendMailToCustomer()
        {
            return _mailSender.SendMail("cust123@abc.com", "Some Message");
        }
    }
}
```

5. Task 2: Create Unit Test Project Using NUnit and Moq

Step 1: Create Test Project dotnet new
nunit -n CustomerComm.Tests
dotnet sln add CustomerComm.Tests/CustomerComm.Tests.csproj
dotnet add CustomerComm.Tests reference CustomerCommLib/CustomerCommLib.cspr
oj

Step 2: Add Packages cd

CustomerComm.Tests

dotnet add package Moq

Step 3: Write the Unit Test

Create a test file CustomerCommTests.cs:

```
using NUnit.Framework;
using Moq; using
CustomerCommLib;

namespace CustomerComm.Tests
{
   [TestFixture]
   public class CustomerCommTests
   {
      private Mock<IMailSender> mockMailSender;
      private CustomerComm customerComm;

      [OneTimeSetUp]
      public void Init()
      {
         mockMailSender = new Mock<IMailSender>();
         mockMailSender.Setup(m => m.SendMail(It.IsAny<string>(), It.IsAny<string>())).R
eturns(true);

         customerComm = new CustomerComm(mockMailSender.Object);
      }

      [TestCase]
      public void SendMailToCustomer_ShouldReturnTrue_WhenMocked()
      {
         bool result = customerComm.SendMailToCustomer();
         Assert.That(result, Is.True);
      }
   }
}
```

6. Run the Tests

```
cd .. dotnet test
```

Expected Output:

```
Passed! - Failed: 0, Passed: 1
```

7. Conclusion

In this exercise, we:

- Created a testable class using constructor injection.
- Used Moq to simulate an external dependency (mail sender).
- Ensured that unit testing could be done without actually sending emails.