NUnit Implementation Report with Mocking

Moq and Test-Driven Development Exercises

1. Objectives

- Understand how mocking supports Test-Driven Development (TDD).
- Explain mocking in Unit Testing with real-world dependencies.
- Explore Dependency Injection: constructor and method injection.
- Demonstrate testable code using Moq (mail, file system, database).

2. Task 1 – Mocking MailSender using Moq

Step 1: Create `CustomerCommLib` class library

IMailSender.cs

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

MailSender.cs

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

namespace CustomerCommLib
{
    public class MailSender : IMailSender
    {
        public bool SendMail(string toAddress, string message)
        {
            MailMessage mail = new MailMessage();
            SmtpClient smtpServer = new SmtpClient("smtp.gmail.com");

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

            smtpServer.Port = 587;
            smtpServer.Credentials = new NetworkCredential("username", "
                ↪ password");
            smtpServer.EnableSsl = true;

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

CustomerComm.cs

```
namespace CustomerCommLib
{
    public class CustomerComm
    {
        private readonly IMailSender _mailSender;
```

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

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

Step 2: Create `CustomerComm.Tests` with Moq

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>()))
                        .Returns(true);
            _customerComm = new CustomerComm(_mockMailSender.Object);
        }

        [Test]
        public void SendMailToCustomer_ReturnsTrue()
        {
            var result = _customerComm.SendMailToCustomer();
            Assert.IsTrue(result);
        }
    }
}
```

Expected Output:

```
Test Run Summary:
-------------------------
SendMailToCustomer_ReturnsTrue: Passed
Result: Passed 1 test
```

3. Task 2 – Mocking File System Access

Step 1: Create `MagicFilesLib`

IDirectoryExplorer.cs

```
public interface IDirectoryExplorer
{
    ICollection<string> GetFiles(string path);
}
```

DirectoryExplorer.cs

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

namespace MagicFilesLib
{
    public class DirectoryExplorer : IDirectoryExplorer
    {
        public ICollection<string> GetFiles(string path)
        {
            return Directory.GetFiles(path);
        }
    }
}
```

Step 2: Create `DirectoryExplorer.Tests`

DirectoryExplorerTests.cs

```
using NUnit.Framework;
using Moq;
using MagicFilesLib;
using System.Collections.Generic;

namespace DirectoryExplorer.Tests
{
    [TestFixture]
    public class DirectoryExplorerTests
    {
        private Mock<IDirectoryExplorer> _mockExplorer;
        private readonly string _file1 = "file.txt";
        private readonly string _file2 = "file2.txt";

        [OneTimeSetUp]
        public void Init()
        {
            _mockExplorer = new Mock<IDirectoryExplorer>();
            _mockExplorer.Setup(d => d.GetFiles(It.IsAny<string>()))
                         .Returns(new List<string> { _file1, _file2 });
        }

        [Test]
        public void GetFiles_ReturnsExpectedFiles()
        {
            var files = _mockExplorer.Object.GetFiles("dummyPath");

            Assert.IsNotNull(files);
            Assert.AreEqual(2, files.Count);
            CollectionAssert.Contains(files, _file1);
        }
    }
```

```
}
```

Expected Output:

```
Test Run Summary:
-------------------------
GetFiles_ReturnsExpectedFiles: Passed
Result: Passed 1 test
```

4. Task 3 – Mocking Database with Moq

Step 1: Create `PlayersManagerLib`

IPlayerMapper.cs

```
public interface IPlayerMapper
{
    bool IsPlayerNameExistsInDb(string name);
    void AddNewPlayerIntoDb(string name);
}
```

PlayerMapper.cs

```
using System.Data.SqlClient;

namespace PlayersManagerLib
{
    public class PlayerMapper : IPlayerMapper
    {
        private readonly string _connectionString =
            "Data␣Source=(local);Initial␣Catalog=GameDB;Integrated␣Security=
                ↪ True";

        public bool IsPlayerNameExistsInDb(string name)
        {
            using var conn = new SqlConnection(_connectionString);
            conn.Open();
            using var cmd = conn.CreateCommand();
            cmd.CommandText = "SELECT␣COUNT(*)␣FROM␣Player␣WHERE␣Name␣=␣@name";
            cmd.Parameters.AddWithValue("@name", name);
            return (int)cmd.ExecuteScalar() > 0;
        }

        public void AddNewPlayerIntoDb(string name)
        {
            using var conn = new SqlConnection(_connectionString);
            conn.Open();
            using var cmd = conn.CreateCommand();
            cmd.CommandText = "INSERT␣INTO␣Player␣([Name])␣VALUES␣(@name)";
            cmd.Parameters.AddWithValue("@name", name);
            cmd.ExecuteNonQuery();
        }
    }
}
```

Player.cs

```
using System;

namespace PlayersManagerLib
{
    public class Player
    {
        public string Name { get; }
        public int Age { get; }
        public string Country { get; }
        public int NoOfMatches { get; }

        public Player(string name, int age, string country, int noOfMatches)
        {
            Name = name;
            Age = age;
            Country = country;
```

```
            NoOfMatches = noOfMatches;
        }

        public static Player RegisterNewPlayer(string name, IPlayerMapper
            ↪ playerMapper = null)
        {
            playerMapper ??= new PlayerMapper();

            if (string.IsNullOrWhiteSpace(name))
                throw new ArgumentException("Player␣name␣ c a n t ␣be␣empty.");

            if (playerMapper.IsPlayerNameExistsInDb(name))
                throw new ArgumentException("Player␣name␣already␣exists.");

            playerMapper.AddNewPlayerIntoDb(name);
            return new Player(name, 23, "India", 30);
        }
    }
}
```

Step 2: Create `PlayerManager.Tests`

PlayerTests.cs

```
using NUnit.Framework;
using Moq;
using PlayersManagerLib;
using System;

namespace PlayerManager.Tests
{
    [TestFixture]
    public class PlayerTests
    {
        private Mock<IPlayerMapper> _mockMapper;

        [OneTimeSetUp]
        public void Init()
        {
            _mockMapper = new Mock<IPlayerMapper>();
        }

        [Test]
        public void RegisterNewPlayer_ValidName_ReturnsPlayer()
        {
            _mockMapper.Setup(m => m.IsPlayerNameExistsInDb(It.IsAny<string>())
                ↪ ).Returns(false);
            _mockMapper.Setup(m => m.AddNewPlayerIntoDb(It.IsAny<string>()));

            var player = Player.RegisterNewPlayer("Rohit", _mockMapper.Object);

            Assert.AreEqual("Rohit", player.Name);
            Assert.AreEqual(23, player.Age);
            Assert.AreEqual("India", player.Country);
            Assert.AreEqual(30, player.NoOfMatches);
        }

        [Test]
        public void RegisterNewPlayer_EmptyName_ThrowsException()
        {
            var ex = Assert.Throws<ArgumentException>(() =>
                Player.RegisterNewPlayer("", _mockMapper.Object));
```

```
            Assert.That(ex.Message, Is.EqualTo("Player␣name␣ c a n t ␣be␣empty.")
                ↪ );
        }

        [Test]
        public void RegisterNewPlayer_DuplicateName_ThrowsException()
        {
            _mockMapper.Setup(m => m.IsPlayerNameExistsInDb(It.IsAny<string>())
                ↪ ).Returns(true);
            var ex = Assert.Throws<ArgumentException>(() =>
                Player.RegisterNewPlayer("Rohit", _mockMapper.Object));
            Assert.That(ex.Message, Is.EqualTo("Player␣name␣already␣exists."));
        }
    }
}
```

Expected Output:

```
Test Run Summary:
-------------------------
RegisterNewPlayer_ValidName_ReturnsPlayer: Passed
RegisterNewPlayer_EmptyName_ThrowsException: Passed
RegisterNewPlayer_DuplicateName_ThrowsException: Passed
Result: Passed 3 tests
```