Superset ID- 6364957

1. Write Testable Code with Moq

Scenario

You are tasked to write a unit test code for the below scenario.
The application in which you are teamed up with, deals with a mail server communication in which your application tries to send mail to its users upon every transaction. Your role is to write unit testing the module that contains send mail functionality. You wanted to perform testing the module without sending any email.
After investigating the problem scenario, you found a solution and that is creating **mock** objects of these external dependencies in the unit testing project so that you can achieve speedier test execution and loose coupling of code.
Note: Duration to complete this exercise is **30 min**.

Task1

In this task, you will create a class library that will be used for unit testing.

- Create a **Class Library (Language C#)** project using Visual Studio IDE, and name it as **CustomerCommLib.**

- Rename the default **Class1** class name as **MailSender.**

- Include the following namespaces with 'using' directive.

    - **System.Net**
    - **System.Net.Mail**

- Define an interface as follow.

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

- And provide implementation of **IMailSender** in the **MailSender** class as seen below.

```
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_address@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);

            }
        }
}
```

The above class can’t be unit testing since the code access the STMP mail server.

- Create another class called **CustomeComm** which is the **class under test** in the given scenario.

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

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

                public bool SendMailToCustomer()
                {
                        //Actual logic goes here
                        //define message and mail address

                        _mailSender.SendMail(cust123@abc.com,”Some Message”);

                        return true;
                }
        }
}
```

In the above code we **injected the dependency** (IMailSender) through **constructor** of **CustomerComm** class so that we can **pass the mock object** of the dependency wherever it is necessary.

We have successfully created a class that's written in such a way that we can run a unit test against it and an exception won't be thrown. We achieve this by mocking the call to IMailSender.SendMail() and adding a mocked return value of true to it.

- Finally **build** your project and be ready for the unit testing with NUnit and Moq.

Task2

In this task, you will create unit test project which make use of NUnit framework and Moq.

- Create a new class library project called **CustomerComm.Tests** and add the following external dependencies to it using **NuGet Package Manager.**

    - NUnit
    - NUnit Test Adapter
    - Moq

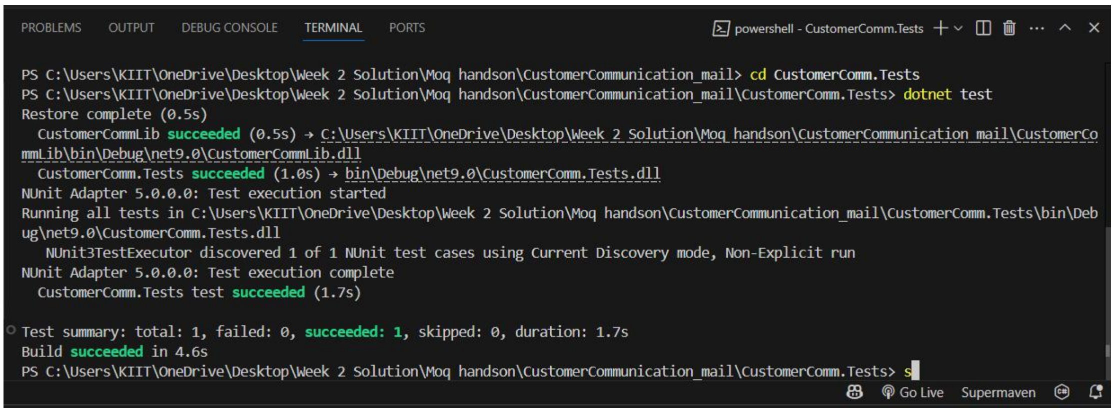
- Add the references of assemblies as appropriate including **CustomerCommLib.**

- Write unit test code and **mock** the **MailSender (IMailSender)** class.

- Use **TestFixture**, **OneTimeSetUp** and **TestCase** attribute classes on top of test class, init method and test method respectively.
- **Configure** the mock object in such away that **SendMail()** method will accept any two string arguments and always return true when **SendMailToCustomer()** gets invoked.
- Finally **assert** the return value to “true”.

CustomerCommTests.cs Code:

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

namespace CustomerCommLib.Tests
{
    [TestFixture]
    public class CustomerCommTests
    {
        private Mock<IMailSender> _mailSenderMock;
        private CustomerComm _customerComm;
        [OneTimeSetUp]
        public void GlobalSetup()
        {
            // Initializing the mock and the class under test once for all test cases
            _mailSenderMock = new Mock<IMailSender>();
            // Configuring the mock to always return true for any string arguments
            _mailSenderMock
                .Setup(m => m.SendMail(It.IsAny<string>(), It.IsAny<string>()))
                .Returns(true);
            _customerComm = new CustomerComm(_mailSenderMock.Object);
        }
        [TestCase]
        public void SendMailToCustomer_ShouldReturnTrue()
        {
            // Act
            var result = _customerComm.SendMailToCustomer();
               Assert.That(result, Is.True);
        }
    }
}
```



```
PS C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\CustomerCommunication_mail> cd CustomerComm.Tests
PS C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\CustomerCommunication_mail\CustomerComm.Tests> dotnet test
Restore complete (0.5s)
  CustomerCommLib succeeded (0.5s) → C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\CustomerCommunication_mail\CustomerCo
mmLib\bin\Debug\net9.0\CustomerCommLib.dll
  CustomerComm.Tests succeeded (1.0s) → bin\Debug\net9.0\CustomerComm.Tests.dll
NUnit Adapter 5.0.0.0: Test execution started
Running all tests in C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\CustomerCommunication_mail\CustomerComm.Tests\bin\Deb
ug\net9.0\CustomerComm.Tests.dll
   NUnit3TestExecutor discovered 1 of 1 NUnit test cases using Current Discovery mode, Non-Explicit run
NUnit Adapter 5.0.0.0: Test execution complete
  CustomerComm.Tests test succeeded (1.7s)

Test summary: total: 1, failed: 0, succeeded: 1, skipped: 0, duration: 1.7s
Build succeeded in 4.6s
PS C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\CustomerCommunication_mail\CustomerComm.Tests> s
```

2. Mock file object for Unit Tests

Scenario

You are tasked to write a unit test code for the below scenario.
The application in which you are teamed up with, deals with the file system and it searches for files and retrieves files under the specified path. In the existing system, **Directory.GetFiles()** method has been used. You found that it’s not good idea to use Directory.GetFiles from the System.IO being its **static** and **unable to unit tes**t such methods.
After investigating the problem scenario, you found a solution and that is refactoring the code. Instead of using directly the static method Directory.GetFiles, you decided to create your own implementation to the method so that be able to **mock** files in the Unit Tests.
Note: Duration to complete this exercise is **30 min**.

Task1

- Create a **Class Library (Language C#)** project using Visual Studio IDE, and name it as **MagicFilesLib.**

- Rename the default **Class1** class name as **DirectoryExplorer** and include the following code snippet into it.

- Include the following namespaces with ‘using’ directive.

    - **System.Collections.Generic**
    - **System.IO**

- Define an interface as follow.

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

- And provide implementation of **IDirectoryExplorer** in the **DirectoryExplorer** class as seen below.

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

Finally **build** your project and be ready for the unit testing with NUnit and Moq.

Task2

- Create a new class library project called **DirectoryExplorer.Tests** and add the following external dependencies to it using **NuGet Package Manager.**

    - NUnit
    - NUnit Test Adapter
    - Moq

- Add the references of assemblies as appropriate including **MagicFilesLib.**

- Write unit test code and **mock the DirectoryExplorer (IDirectoryExplorer),** which is the class under test, with some hard coded file names.

- Use **TestFixture**, **OneTimeSetUp** and **TestCase** attribute classes on top of test class, init method and test method respectively.

- Add the following declarations in the test class.

  private readonly string _file1 = "file.txt";
  private readonly string _file2 = "file2.txt";

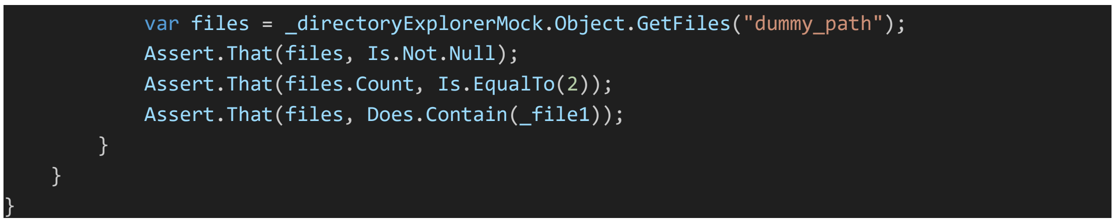
- In the test method, **assert** the following so that,

  the collection is not null
  the collection count is equal to 2
  the collection contains _file1

3.

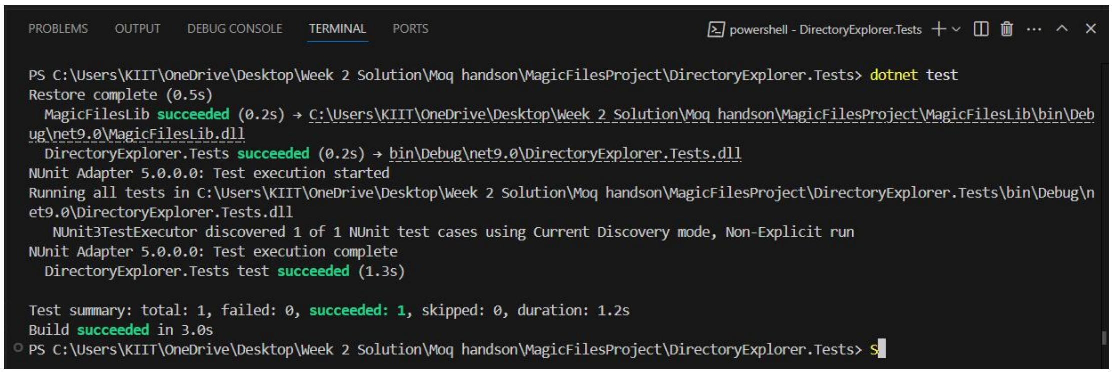
Code for testing **MagicFilesProject**:

```
using Moq;
using NUnit.Framework;
using System.Collections.Generic;
using MagicFilesLib;
namespace DirectoryExplorer.Tests
{
    [TestFixture]
    public class DirectoryExplorerTests
    {
        private Mock<IDirectoryExplorer> _directoryExplorerMock;
        private readonly string _file1 = "file.txt";
        private readonly string _file2 = "file2.txt";
        [OneTimeSetUp]
        public void Setup()
        {
            _directoryExplorerMock = new Mock<IDirectoryExplorer>();
            _directoryExplorerMock
                .Setup(de => de.GetFiles(It.IsAny<string>()))
                .Returns(new List<string> { _file1, _file2 });
        }
        [TestCase]
        public void GetFiles_ShouldReturnMockedFiles()
        {
```



```
            var files = _directoryExplorerMock.Object.GetFiles("dummy_path");
            Assert.That(files, Is.Not.Null);
            Assert.That(files.Count, Is.EqualTo(2));
            Assert.That(files, Does.Contain(_file1));
        }
    }
}
```

Output of Testing :



```
PS C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\MagicFilesProject\DirectoryExplorer.Tests> dotnet test
Restore complete (0.5s)
  MagicFilesLib succeeded (0.2s) → C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\MagicFilesProject\MagicFilesLib\bin\Deb
ug\net9.0\MagicFilesLib.dll
  DirectoryExplorer.Tests succeeded (0.2s) → bin\Debug\net9.0\DirectoryExplorer.Tests.dll
NUnit Adapter 5.0.0.0: Test execution started
Running all tests in C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\MagicFilesProject\DirectoryExplorer.Tests\bin\Debug\n
et9.0\DirectoryExplorer.Tests.dll
   NUnit3TestExecutor discovered 1 of 1 NUnit test cases using Current Discovery mode, Non-Explicit run
NUnit Adapter 5.0.0.0: Test execution complete
  DirectoryExplorer.Tests test succeeded (1.3s)

Test summary: total: 1, failed: 0, succeeded: 1, skipped: 0, duration: 1.2s
Build succeeded in 3.0s
PS C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\MagicFilesProject\DirectoryExplorer.Tests> S
```

3.Mock database for Unit Tests

Scenario

You are tasked to write a unit test code for the below scenario.
The application in which you are teamed up with, deals with a network database in which your application stores the record or certain players. It involves storing and retrieval of player details. Your role is to write unit testing the player module which involves an external dependency. You can't proceed with unit testing.
After investigating the problem scenario, you found a solution and that is creating **mock** objects of these external dependencies in the unit testing project so that you can achieve speedier test execution and loose coupling of code.
Note: Duration to complete this exercise is **60 min**.

Task1

In this task, you will create a class library that will be used for unit testing.

- Create a **Class Library (Language C#)** project using Visual Studio IDE, and name it as **PlayersManagerLib.**

- Rename the default **Class1** class name as **PlayerManager.**

- Include the following namespaces with 'using' directive.

    - **System.Data**
    - **System.Data.SqlClient**

- Define an interface as follow.

  public interface IPlayerMapper
  {

```
        bool IsPlayerNameExistsInDb(string name);
        Void AddNewPlayerIntoDb(string name);
    }
```

- And provide implementation of **IPlayerMapper** in the **PlayerMapper** class as seen below.

```
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(SqlConnection connection = new SqlConnection(_connectionString))
      {
        connection.Open();

        using(SqlCommand command = connection.CreateCommand())
        {
          command.CommandText = "SELECT count(*) FROM Player WHERE
'Name' = @name";

          command.Parameters.AddWithValue("@name", name);

          // Get the number of player with this name
          var existingPlayersCount = (int) command.ExecuteScalar();

          // Result is 0, if no player exists, or 1, if a player already exists
          return existingPlayersCount > 0;
        }
      }
    }

    public void AddNewPlayerIntoDb(string name)
    {
      using(SqlConnection connection = new SqlConnection(_connectionString))
      {
        connection.Open();

        using(SqlCommand command = connection.CreateCommand())
        {
          command.CommandText = "INSERT INTO Player ([Name]) VALUES
(@name)";

          command.Parameters.AddWithValue("@name", name);

          command.ExecuteNonQuery();
        }
      }
    }
  }
}
```

The above class can’t be unit testing since the code access the database.

- Create another class called **Player** and add the following codes.

```
public class Player
```

```
{
    public string Name { get; private set; }
    public int Age { get; private set; }
    public string Country { get; private set; }
    public int NoOfMatches {get; private set;}

    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)
    {
       // If a PlayerMapper was not passed in, use a real one.
       // This allows us to pass in a "mock" PlayerMapper (for testing),
       // but use a real PlayerMapper, when running the program.
       if(playerMapper == null)
       {
          playerMapper = new PlayerMapper();
       }

       if(string.IsNullOrWhiteSpace(name))
       {
          throw new ArgumentException("Player name can’t be empty.");
       }

       // Throw an exception if there is already a player with this name in the
database.
       if(playerMapper.IsPlayerNameExistsInDb (name))
       {
          throw new ArgumentException("Player name already exists.");
       }

       // Add the player to the database.
       playerMapper. AddNewPlayerIntoDb (name);

       return new Player(name, 23, “India”,30);
    }
  }
```

Finally **build** your project and be ready for the unit testing with NUnit and Moq.

Task2

In this task, you will create unit test project which make use of NUnit framework and Moq.

- Create a new class library project called **PlayerManager.Tests** and add the following external dependencies to it using **NuGet Package Manager.**
    - NUnit
    - NUnit Test Adapter
    - Moq
- Add the references of assemblies as appropriate including **PlayersManagerLib.**
- Write unit test code and **mock** the **PlayerMapper (IPlayerMapper)** class.

- Use **TestFixture**, **OneTimeSetUp** and **TestCase** attribute classes on top of test class, init method and test method respectively.

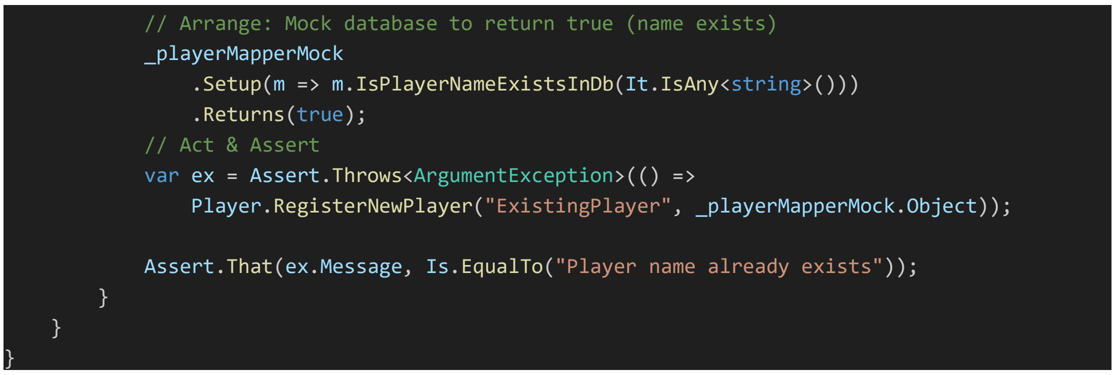
- Use **ExpectedException** attribute to specify that the execution of a test will throw an exception.

- When the **RegisterNewPlayer** function calls **IsPlayerNameExistsInDb**, you need to make sure that the mock object to return **“false”.**

- In the test method, **assert** various player attributes.

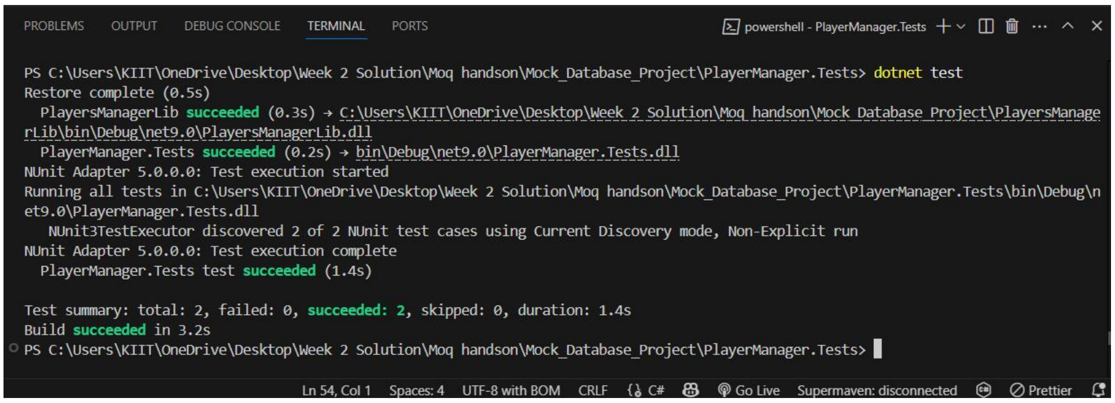
Testing Code for **Mock_Database_Project** :-

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

namespace PlayerManager.Tests
{
    [TestFixture]
    public class PlayerTests
    {
        private Mock<IPlayerMapper> _playerMapperMock;
        [OneTimeSetUp]
        public void Setup()
        {
            _playerMapperMock = new Mock<IPlayerMapper>();
        }
        [Test]
        public void RegisterNewPlayer_WithNewName_ReturnsPlayer()
        {
            // Arrange: Mock database to return false (name doesn't exist)
            _playerMapperMock
                .Setup(m => m.IsPlayerNameExistsInDb(It.IsAny<string>()))
                .Returns(false);
            // Act
            var player = Player.RegisterNewPlayer("NewPlayer", _playerMapperMock.Object);
            // Assert
            Assert.That(player, Is.Not.Null);
            Assert.That(player.Name, Is.EqualTo("NewPlayer"));
            Assert.That(player.Age, Is.EqualTo(23));
            Assert.That(player.Country, Is.EqualTo("India"));
            Assert.That(player.NoOfMatches, Is.EqualTo(30));
        }
        [Test]
        public void RegisterNewPlayer_WithExistingName_ThrowsException()
        {
```



```
            // Arrange: Mock database to return true (name exists)
            _playerMapperMock
                .Setup(m => m.IsPlayerNameExistsInDb(It.IsAny<string>()))
                .Returns(true);
            // Act & Assert
            var ex = Assert.Throws<ArgumentException>(() =>
                Player.RegisterNewPlayer("ExistingPlayer", _playerMapperMock.Object));

            Assert.That(ex.Message, Is.EqualTo("Player name already exists"));
        }
    }
}
```

Output of TEST:



```
PS C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\Mock_Database_Project\PlayerManager.Tests> dotnet test
Restore complete (0.5s)
  PlayersManagerLib succeeded (0.3s) → C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\Mock_Database_Project\PlayersManage
rLib\bin\Debug\net9.0\PlayersManagerLib.dll
  PlayerManager.Tests succeeded (0.2s) → bin\Debug\net9.0\PlayerManager.Tests.dll
NUnit Adapter 5.0.0.0: Test execution started
Running all tests in C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\Mock_Database_Project\PlayerManager.Tests\bin\Debug\n
et9.0\PlayerManager.Tests.dll
   NUnit3TestExecutor discovered 2 of 2 NUnit test cases using Current Discovery mode, Non-Explicit run
NUnit Adapter 5.0.0.0: Test execution complete
  PlayerManager.Tests test succeeded (1.4s)

Test summary: total: 2, failed: 0, succeeded: 2, skipped: 0, duration: 1.4s
Build succeeded in 3.2s
PS C:\Users\KIIT\OneDrive\Desktop\Week 2 Solution\Moq handson\Mock_Database_Project\PlayerManager.Tests>
```