**WEEK -2 Moq Mandatory Hands-On**

**1. Moq-HandsOn**

**Given :**

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.

**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

\_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.

**SOLUTION :**

**1. MailSender.cs :**

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");

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);

return true;

}

}

}

**2. CustomerComm.cs :**

namespace CustomerCommLib

{

public class CustomerComm

{

private readonly IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

string toAddress = "cust123@abc.com";

string message = "Some Message";

return \_mailSender.SendMail(toAddress, message);

}

}

}

**3. 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\_ShouldReturnTrue()

{

var result = \_customerComm.SendMailToCustomer();

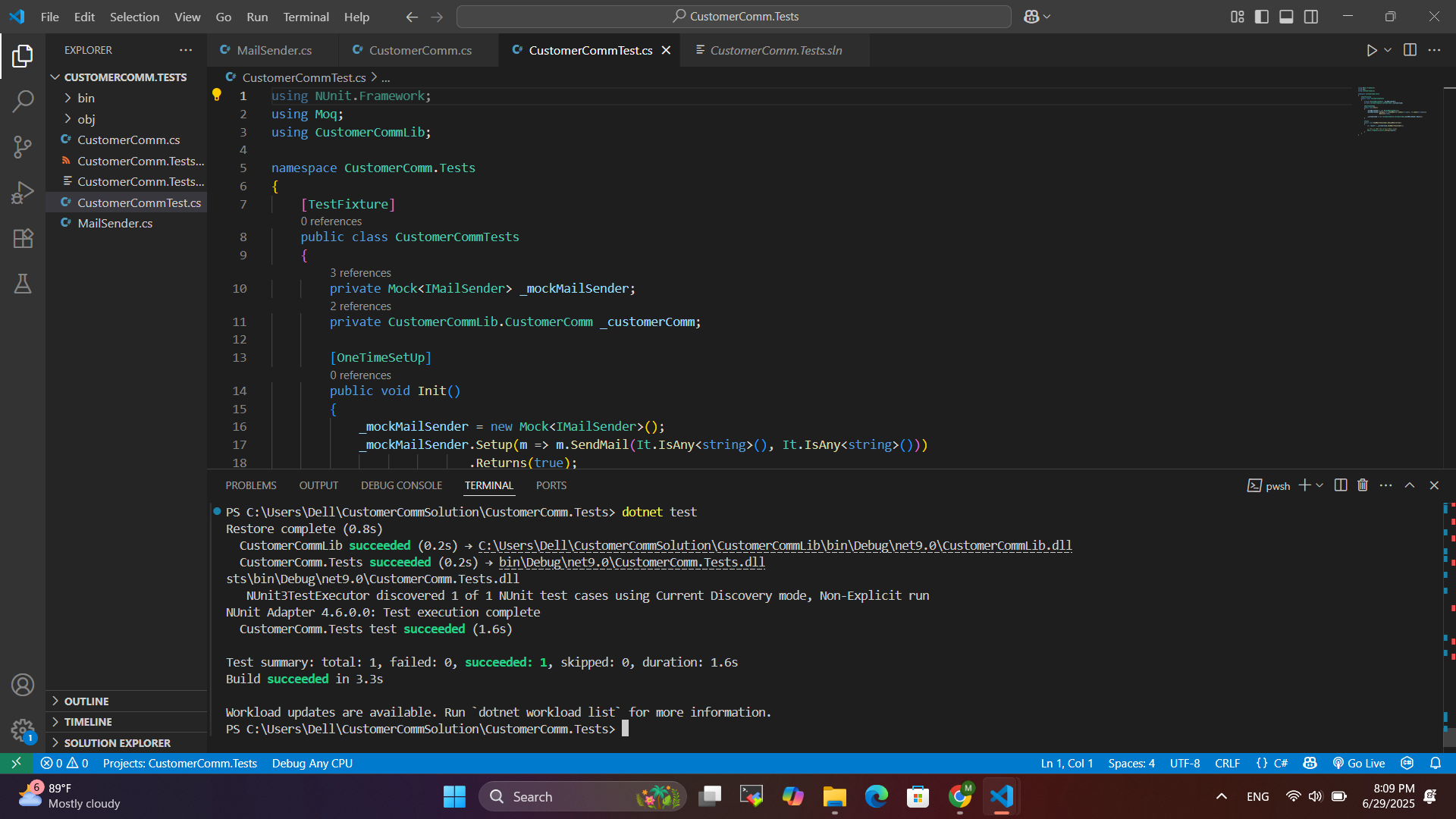
Assert.IsTrue(result);

}

}

}

**OUTPUT :**

****

**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”.

**SOLUTION :**

**1. Define An Interface For Data Access :**

namespace CustomerCommLib

{

public interface ICustomerRepository

{

string GetCustomerEmail(int customerId);

}

}

**2. Create A Class That Uses This Dependency :**namespace CustomerCommLib

{

public class CustomerDbComm

{

private readonly ICustomerRepository \_repository;

private readonly IMailSender \_mailSender;

public CustomerDbComm(ICustomerRepository repository, IMailSender mailSender)

{

\_repository = repository;

\_mailSender = mailSender;

}

public bool NotifyCustomer(int customerId, string message)

{

string email = \_repository.GetCustomerEmail(customerId);

return \_mailSender.SendMail(email, message);

}

}

}

**3. Add A New Test File In CustomerComm.Tests :**

using NUnit.Framework;

using Moq;

using CustomerCommLib;

namespace CustomerComm.Tests

{

[TestFixture]

public class CustomerDbCommTest

{

private Mock<ICustomerRepository> \_mockRepo;

private Mock<IMailSender> \_mockMailSender;

private CustomerDbComm \_customerDbComm;

[SetUp]

public void Setup()

{

\_mockRepo = new Mock<ICustomerRepository>();

\_mockMailSender = new Mock<IMailSender>();

\_mockRepo.Setup(r => r.GetCustomerEmail(It.IsAny<int>()))

.Returns("mockuser@example.com");

\_mockMailSender.Setup(m => m.SendMail(It.IsAny<string>(), It.IsAny<string>()))

.Returns(true);

\_customerDbComm = new CustomerDbComm(\_mockRepo.Object, \_mockMailSender.Object);

}

[Test]

public void NotifyCustomer\_ShouldSendEmail\_AndReturnTrue()

{

var result = \_customerDbComm.NotifyCustomer(1, "Welcome!");

Assert.IsTrue(result);

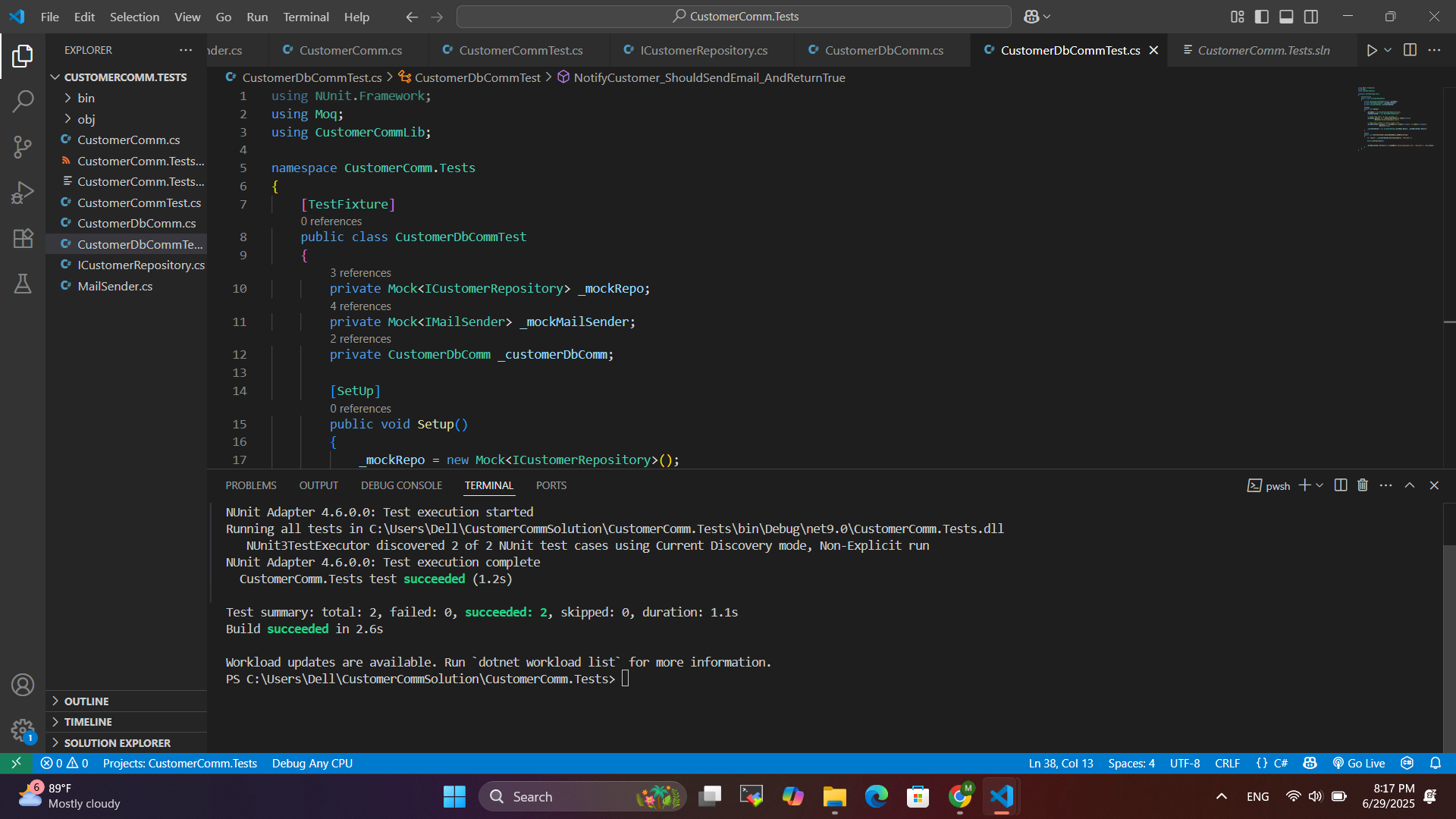
\_mockMailSender.Verify(m => m.SendMail("mockuser@example.com", "Welcome!"), Times.Once);

}

}

}

**OUTPUT :**

****