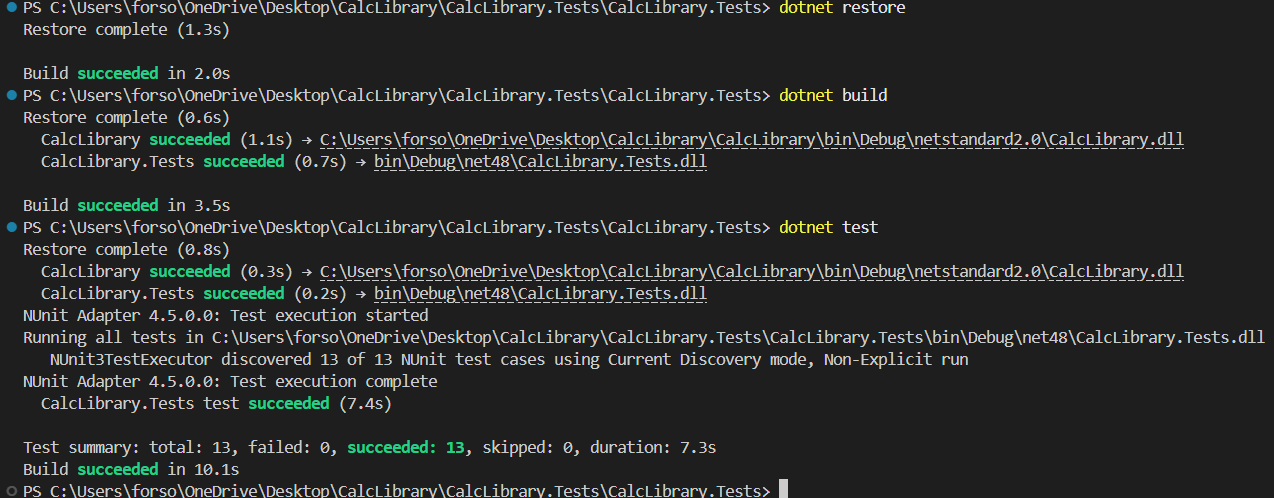
**NUnit-Handson**

Superset ID-  6365411

**TestFixture & Test**

* Create a Unit test project(.Net Framework) in the solution provided.
* Add the CalcLibrary project as reference
* Create a class “CalculatorTests” to write all the test cases for the methods in the solution
* Use the ‘TestFixture’, ‘SetUp’ and ‘TearDown’ attributes, to declare, initialize and cleanup activities respectively
* Create a Test method to check the addition functionality
* Use the ‘TestCase’ attribute to send the inputs and the expected result
* Use Assert.That to check the actual and expected result match

**OUTPUT:**



**CODE:**

**MathLibrary.cs**

using System;

namespace CalcLibrary

{

    interface IMathLibrary

    {

        double Addition(double a, double b);

        double Subtraction(double a, double b);

        double Multiplication(double a, double b);

        double Division(double a, double b);

    }

    public class SimpleCalculator : IMathLibrary

    {

        double result = 0;

        public double Addition(double a, double b)

        {

            result = a + b;

            return result;

        }

        public double Subtraction(double a, double b)

        {

            result = a - b;

            return result;

        }

        public double Multiplication(double a, double b)

        {

            result = a \* b;

            return result;

        }

        public double Division(double a, double b)

        {

            if (b == 0)

                throw new ArgumentException("Second Parameter Can't be Zero");

            result = a / b;

            return result;

        }

        public void AllClear()

        {

            result = 0;

        }

        public double GetResult

        {

            get { return result; }

        }

    }

}

**Moq-Handson**

**Write Testable Code with Moq**

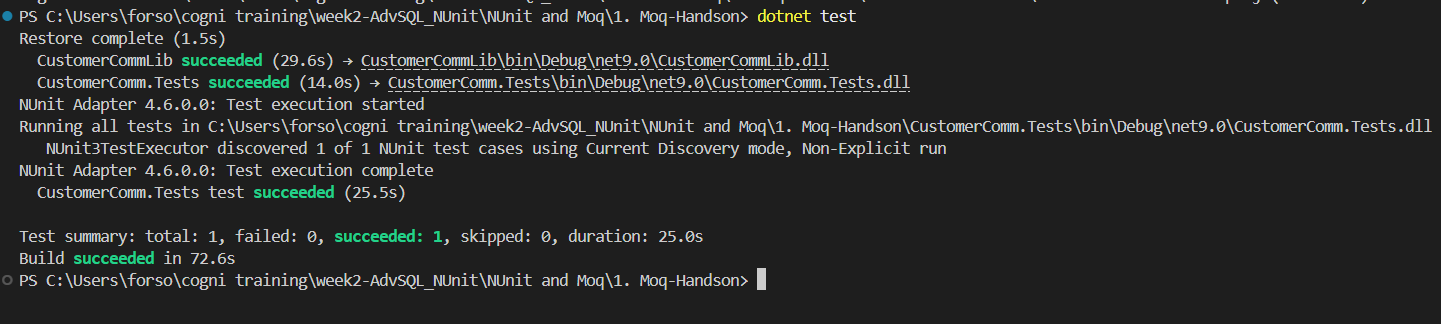
**Scenario**

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.

**OUTPUT:**

****

**CODE:**

CustomerCommTests.cs

using Moq;

using NUnit.Framework;

using CustomerCommLib;

namespace CustomerComm.Tests

{

    [TestFixture]

    public class CustomerCommTests

    {

        [Test]

        public void SendMailToCustomer\_ShouldReturnTrue()

        {

            var mockMailSender = new Mock<IMailSender>();

            mockMailSender.Setup(ms => ms.SendMail(It.IsAny<string>(), It.IsAny<string>()))

                          .Returns(true);

            var customerComm = new CustomerCommLib.CustomerComm(mockMailSender.Object);

            bool result = customerComm.SendMailToCustomer();

            Assert.That(result, Is.True);

            mockMailSender.Verify(ms => ms.SendMail(It.IsAny<string>(), It.IsAny<string>()), Times.Once);

        }

    }

}

CustomerComm.cs

namespace CustomerCommLib

{

    public class CustomerComm

    {

        private readonly IMailSender \_mailSender;

        public CustomerComm(IMailSender mailSender)

        {

            \_mailSender = mailSender;

        }

        public bool SendMailToCustomer()

        {

            string email = "22052571@kiit.ac.in";

            string message = "Hello!";

            return \_mailSender.SendMail(email, message);

        }

    }

}

IMailSender.cs

namespace CustomerCommLib

{

    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("sample@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;

            return true;

        }

    }

}

**Directory Structure:**

