MOQ

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.

TASK 1

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;

}

}

}

TASK 2

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.

**IEmailSerivces.cs:**

namespace EmailApp

{

public interface IEmailService

{

bool SendEmail(string to, string message);

}

}

**NotificationService.cs:**

namespace EmailApp

{

public class NotificationService

{

private readonly IEmailService \_emailService;

public NotificationService(IEmailService emailService)

{

\_emailService = emailService;

}

public bool NotifyUser()

{

string email = "user@example.com";

string message = "Transaction successful!";

return \_emailService.SendEmail(email, message);

}

}

}

**NotificationServicesTests:**

using NUnit.Framework;

using Moq;

using EmailApp;

namespace EmailApp.Tests

{

public class NotificationServiceTests

{

private Mock<IEmailService> \_mockEmailService;

private NotificationService \_notificationService;

[SetUp]

public void Setup()

{

\_mockEmailService = new Mock<IEmailService>();

\_notificationService = new NotificationService(\_mockEmailService.Object);

}

[Test]

public void NotifyUser\_ShouldReturnTrue\_WhenEmailIsSentSuccessfully()

{

// Arrange

\_mockEmailService

.Setup(x => x.SendEmail(It.IsAny<string>(), It.IsAny<string>()))

.Returns(true);

// Act

var result = \_notificationService.NotifyUser();

// Assert

Assert.IsTrue(result);

}

[Test]

public void NotifyUser\_ShouldReturnFalse\_WhenEmailFailsToSend()

{

// Arrange

\_mockEmailService

.Setup(x => x.SendEmail(It.IsAny<string>(), It.IsAny<string>()))

.Returns(false);

// Act

var result = \_notificationService.NotifyUser();

// Assert

Assert.IsFalse(result);

}

[Test]

public void NotifyUser\_ShouldCallSendEmail\_Once()

{

// Arrange

\_mockEmailService

.Setup(x => x.SendEmail(It.IsAny<string>(), It.IsAny<string>()))

.Returns(true);

// Act

\_notificationService.NotifyUser();

// Assert

\_mockEmailService.Verify(x => x.SendEmail(It.IsAny<string>(), It.IsAny<string>()), Times.Once);

}

[Test]

public void NotifyUser\_ShouldCallSendEmail\_WithCorrectArguments()

{

// Arrange

string expectedEmail = "user@example.com";

string expectedMessage = "Transaction successful!";

\_mockEmailService

.Setup(x => x.SendEmail(expectedEmail, expectedMessage))

.Returns(true);

// Act

\_notificationService.NotifyUser();

// Assert

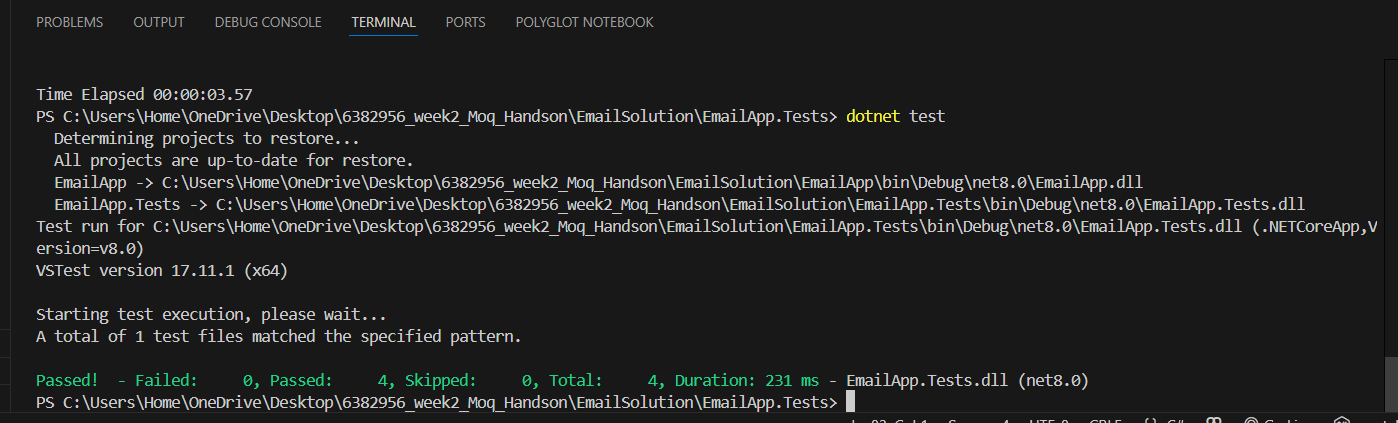
\_mockEmailService.Verify(x => x.SendEmail(expectedEmail, expectedMessage), Times.Once);

}

}

}

**Output:**

****