## Create Class Library (CustomerCommLib)

// 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)

{

// Actual SMTP implementation (not used in unit tests)

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");

}

}

}

## Unit Tests with Moq (NUnit Project)

// Install NuGet packages:

// Install-Package Moq

// Install-Package NUnit

using CustomerCommLib;

using Moq;

using NUnit.Framework;

namespace CustomerCommTests

{

[TestFixture]

public class MailServiceTests

{

[Test]

public void SendMailToCustomer\_ShouldReturnTrue()

{

// Arrange

var mockMailSender = new Mock<IMailSender>();

mockMailSender

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

.Returns(true);

var customerComm = new CustomerComm(mockMailSender.Object);

// Act

bool result = customerComm.SendMailToCustomer();

// Assert

Assert.IsTrue(result);

}

[Test]

public void Verify\_EmailParameters()

{

// Arrange

var mockMailSender = new Mock<IMailSender>();

var customerComm = new CustomerComm(mockMailSender.Object);

// Act

customerComm.SendMailToCustomer();

// Assert: Verify correct parameters were used

mockMailSender.Verify(

m => m.SendMail("cust123@abc.com", "Some Message"),

Times.Once

);

}

}

}

## Key Concepts Demonstrated:

**Mocking**:

Created Mock<IMailSender> to simulate mail sending

.Setup() defines mock behavior without real SMTP calls

**Dependency Injection**:

public CustomerComm(IMailSender mailSender) // Constructor injection

{

\_mailSender = mailSender;

}

**Testable Design**:

Separated concerns with interface (IMailSender)

Constructor injection enables mock substitution

**Verification**:

Verify() checks if method was called with correct parameters

Times.Once ensures method was called exactly once

## Test Execution:

**Passing Test Output:**