# Moq-Handson

1. **Write Testable Code with Moq**

using NUnit.Framework; using Moq;

using CustomerCommLib;

namespace CustomerCommLib.Tests

{

[TestFixture]

public class CustomerCommLibTests

{

private Mock<IMailSender> mockMailSender;

private CustomerCommLib.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);

}

[TestCase]

public void SendMailToCustomer\_ShouldReturnTrue()

{

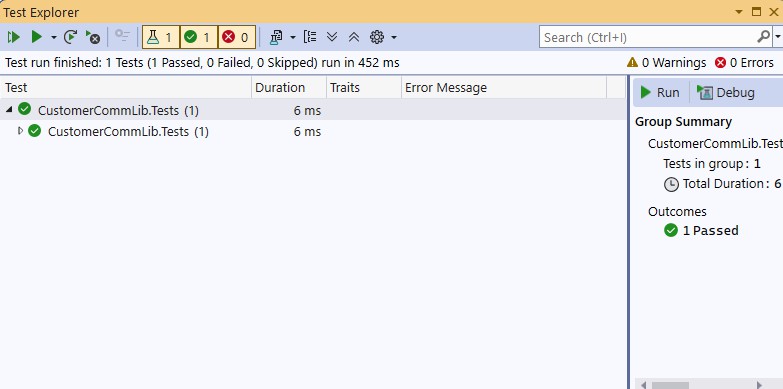
bool result = customerComm.SendMailToCustomer();

Assert.That(result);

}

}

}



1. **Mock file object for Unit Tests**

using NUnit.Framework; using Moq;

using MagicFilesLib;

using System.Collections.Generic; namespace DirectoryExplorer.Test

{

[TestFixture]

public class DirectoryExplorerTests

{

private Mock<IDirectoryExplorer> mockExplorer; private readonly string \_file1 = "file.txt"; private readonly string \_file2 = "file2.txt";

[OneTimeSetUp] public void Init()

{

mockExplorer = new Mock<IDirectoryExplorer>(); mockExplorer.Setup(m => m.GetFiles(It.IsAny<string>()))

.Returns((ICollection<string>)new List<string> { \_file1,

\_file2 });

;

}

[TestCase]

public void GetFiles\_ShouldReturnExpectedFiles()

{

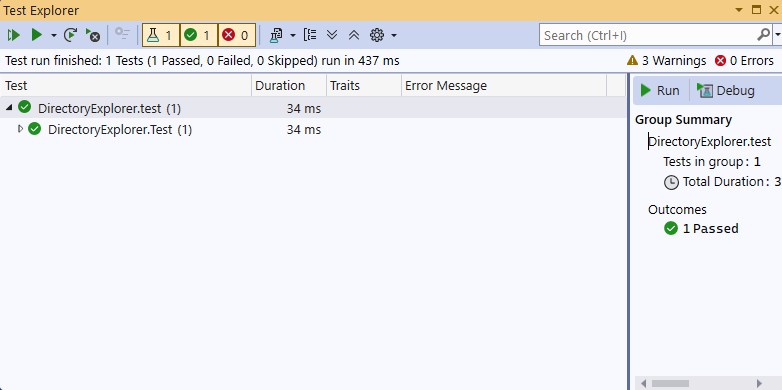
var result = mockExplorer.Object.GetFiles("dummy/path");

Assert.That(result, Is.Not.Null); Assert.That(result.Count, Is.EqualTo(2)); Assert.That(result, Contains.Item(\_file1));

}

}

}

****

1. **Mock database for Unit Tests**

using NUnit.Framework; using Moq;

using PlayersManagerLib; using System;

namespace PlayerManager.Tests

{

[TestFixture]

public class PlayerTests

{

private Mock<IPlayerMapper> mockMapper;

[OneTimeSetUp] public void Setup()

{

mockMapper = new Mock<IPlayerMapper>();

mockMapper.Setup(x => x.IsPlayerNameExistsInDb(It.IsAny<string>())).Returns(false);

}

[Test]

public void RegisterNewPlayer\_ShouldCreatePlayer\_WhenNameIsValidAndNotExists()

{

var player = Player.RegisterNewPlayer("Virat", mockMapper.Object);

Assert.That(player, Is.Not.Null); Assert.That(player.Name, Is.EqualTo("Virat")); 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\_ShouldThrowException\_WhenNameIsEmpty()

{

var ex = Assert.Throws<ArgumentException>(() => Player.RegisterNewPlayer("", mockMapper.Object));

Assert.That(ex.Message, Is.EqualTo("Player name can’t be empty."));

}

[Test] public void

RegisterNewPlayer\_ShouldThrowException\_WhenNameAlreadyExists()

{

mockMapper.Setup(x => x.IsPlayerNameExistsInDb("Sachin")).Returns(true);

var ex = Assert.Throws<ArgumentException>(() => Player.RegisterNewPlayer("Sachin",

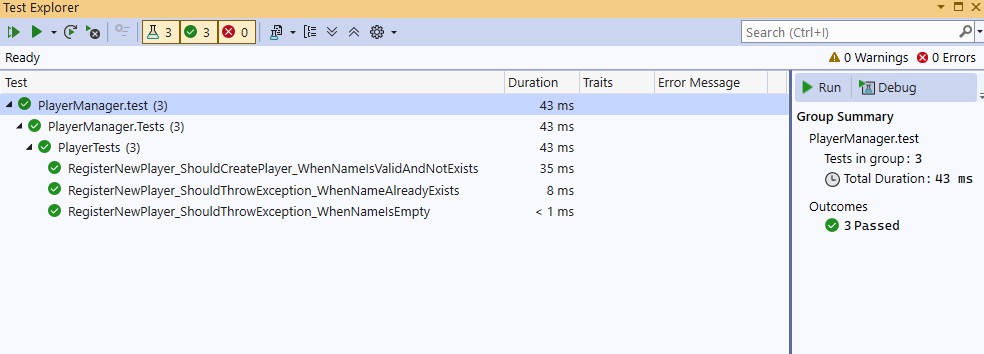
mockMapper.Object));

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

}

}

}



# NUnit-Handson

**TestFixture s Test**

using NUnit.Framework; using CalcLibrary; using System;

namespace CalcLibrary.Tests

{

[TestFixture]

public class SimpleCalculatorTests

{

private SimpleCalculator \_calculator;

[SetUp]

public void Setup()

{

\_calculator = new SimpleCalculator();

}

[TearDown]

public void TearDown()

{

\_calculator = null;

}

[TestCase(10, 5, 15)]

[TestCase(-10, -5, -15)]

[TestCase(10.5, 2.5, 13.0)]

public void Addition\_ShouldReturnCorrectSum(double num1, double num2, double expected)

{

double result = \_calculator.Addition(num1, num2); Assert.That(result, Is.EqualTo(expected));

}

[TestCase(10, 5, 5)]

[TestCase(-10, -5, -5)]

[TestCase(10.5, 2.5, 8.0)]

public void Subtraction\_ShouldReturnCorrectDifference(double num1, double num2, double expected)

{

double result = \_calculator.Subtraction(num1, num2); Assert.That(result, Is.EqualTo(expected));

}

[TestCase(10, 5, 50)]

[TestCase(-10, 5, -50)]

[TestCase(10, 0, 0)]

public void Multiplication\_ShouldReturnCorrectProduct(double num1, double num2, double expected)

{

double result = \_calculator.Multiplication(num1, num2); Assert.That(result, Is.EqualTo(expected));

}

[TestCase(10, 5, 2)]

[TestCase(-10, 5, -2)]

public void Division\_ShouldReturnCorrectQuotient(double num1, double num2, double expected)

{

double result = \_calculator.Division(num1, num2); Assert.That(result, Is.EqualTo(expected));

}

[Test]

public void Division\_ShouldThrowException\_WhenDividingByZero()

{

var ex = Assert.Throws<ArgumentException>(() =>

\_calculator.Division(10, 0));

Assert.That(ex.Message, Is.EqualTo("Second Parameter Can't be

Zero"));

}

[Test]

public void GetResult\_ShouldReturnLastOperationResult()

{

\_calculator.Addition(20, 22);

double result = \_calculator.GetResult;

Assert.That(result, Is.EqualTo(42));

}

[Test]

public void AllClear\_ShouldResetTheInternalResult()

{

\_calculator.Multiplication(10, 10); Assert.That(\_calculator.GetResult, Is.EqualTo(100));

\_calculator.AllClear(); Assert.That(\_calculator.GetResult, Is.EqualTo(0));

}

}

}

