# **4. Objective**

§ This lab will help you become skilled at writing automated unit tests using the NUnit framework.

§ Explain & demonstrate various NUnit custom attributes to identify tests.

Create a Unit Test Project using NUnit Framework for the following requirement. Click [here](https://cognizantonline.sharepoint.com/:u:/r/sites/GTP-Solutions/Gencsharepath/Shared%20Documents/Internship2020/FSE/DotNet/02%20-%20NUnit,%20C%23%204.5,%20ASP.Net%20Core/Handson/AccountsManagerLib.zip?csf=1&web=1&e=rtHrQD) to download the source project.

You are given a user account module called **AccountsManagerLib** using which an employee can login to the EMS portal using his/her credentials. If he/she provides valid login credentials, then the function should return the following message.

“Welcome <user\_id>!!!”

In case user provides invalid details, the function should return the following error message.

“Invalid user id/password”

If user doesn’t provide either userid or password, the function should throw an ArgumentException.

Valid Credentials are the following

| **User Id** | **Password** |
| --- | --- |
| user\_11 | secret@user11 |
| user\_22 | secret@user22 |

**Recommendations:**

Test Project Name:*<ClassLib\_Project>.Tests*

Test Class Name: *<SUT>Tests*

Test Method Name:  *UnitUnderTest\_Scenario\_ExpectedOutcome*

After writing the above test methods, run the tests and assert the results with that of the success/failure messages.

**Note:**

· *Enforce the Single Assertion Rule*

· *Use Assert.That()*

**Steps to perform:**

1) Create a Class Library project in the same solution which is provided and name it as suggested.

2) Rename the class file name (<SUT>Tests.cs).

3) Add the assembly reference of the UtilLib project to the test project.

4) Additionally add the reference of both NUnit and NUnit3TestAdapter in the test project using NuGet Package Manager (NPM).

5) Write the suggested test methods.

6) Run your tests.

7) Break the test by modifying the source project functionality.

8) Rerun the test.

9) Observe the test result.

**Solution:**

using NUnit.Framework;

using AccountsManagerLib;

using System;

namespace AccountsManagerLib.Tests

{

[TestFixture]

public class AccountsManagerTests

{

private AccountsManager \_manager;

[SetUp]

public void Setup()

{

\_manager = new AccountsManager();

}

[Test]

public void ValidateUser\_ValidCredentials\_ReturnsWelcomeMessage()

{

var result = \_manager.ValidateUser("user\_11", "secret@user11");

Assert.That(result, Is.EqualTo("Welcome user\_11!!!"));

}

[Test]

public void ValidateUser\_InvalidCredentials\_ReturnsInvalidMessage()

{

var result = \_manager.ValidateUser("user\_11", "wrongpassword");

Assert.That(result, Is.EqualTo("Invalid user id/password"));

}

[Test]

public void ValidateUser\_EmptyUserId\_ThrowsFormatException()

{

var ex = Assert.Throws<FormatException>(() => \_manager.ValidateUser("", "secret@user11"));

Assert.That(ex.Message, Is.EqualTo("Both user id and password are mandatory"));

}

[Test]

public void ValidateUser\_EmptyPassword\_ThrowsFormatException()

{

var ex = Assert.Throws<FormatException>(() => \_manager.ValidateUser("user\_11", ""));

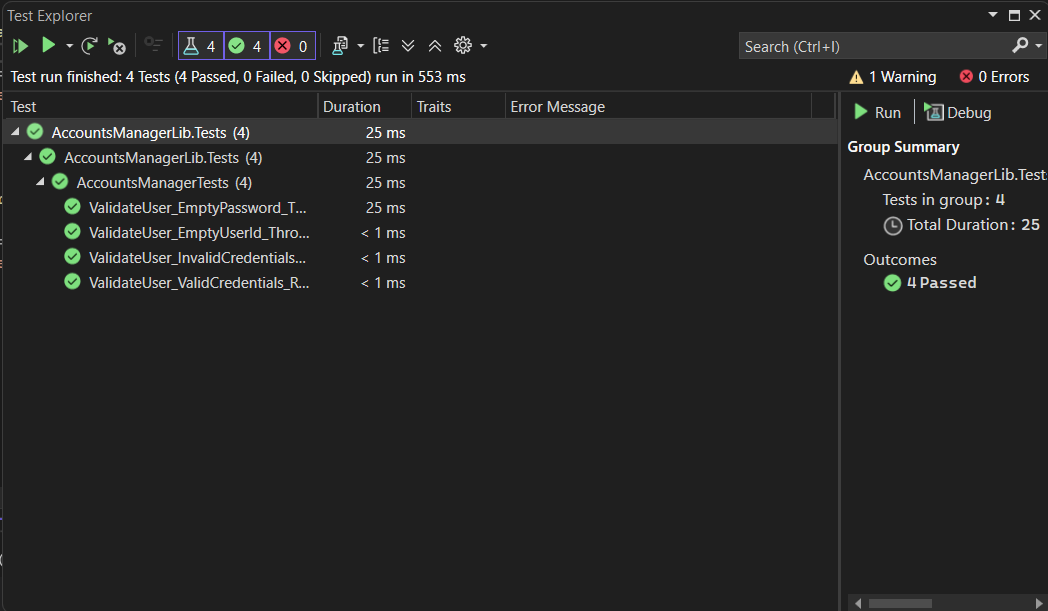
Assert.That(ex.Message, Is.EqualTo("Both user id and password are mandatory"));

}

}

}

**Result:**

****

**5. Objectives**

This lab will help you become skilled at writing automated unit tests using the NUnit framework.

Explain & demonstrate various NUnit custom attributes to identify tests

Explain & Demonstrate on CollectionAssert

Create a Unit Test Project using NUnit Framework for the following requirement. Click here to download the source project.

You have been given a source project called CollectionsLib that deals with set of collection objects. Write test methods for the below scenarios. Make sure that your tests pass. You may modify the collection values in the source project in order to make the test passed. Use the appropriate assert functionalities.

The method called GetEmployees returns a collection of Employee object.

Scenario 1

• Ensure that there is no null value in the collection

Scenario 2

• Verify whether the employee having his/her id 100 exists in the collection.

Scenario 3

• Check whether the GetEmployees function returns only unique employees. If employee id is different in every employee object, then the list is considered as unique list. You may modify the source project so that you can achieve the goal here.

Hint: Override Equals() & GetHashCode() methods in Empoyee class.

Scenario 3

• Both GetEmployees() and GetEmployeesWhoJoinedInPreviousYears() return a set of employee object. Verify whether all items in both the collections are same or not.

Try both Classic Model as well as Constraint Model of Assertions while writing test cases.

Recommendations:

Test Project Name: <ClassLib\_Project>.Tests

Test Class Name: <SUT>Tests

Test Method Name: UnitUnderTest\_Scenario\_ExpectedOutcome

Note:

• Enforce the Single Assertion Rule

• Use Assert.That()

Steps to perform

1) Create a Class Library project in the same solution which is provided and name it as suggested.

2) Rename the class file name (<SUT>Tests.cs).

3) Add the assembly reference of the UtilLib project to the test project.

4) Additionally add the reference of both NUnit and NUnit3TestAdapter in the test project using NuGet Package Manager (NPM).

5) Write the suggested test methods.

6) Run your tests.

7) Break the test by modifying the source project functionality.

8) Rerun the test.

9) Observe the test result.

**Solution:**

using CollectionsLib;

using NUnit.Framework;

using NUnit.Framework.Legacy;

using System;

using System.Collections.Generic;

using System.Linq;

namespace CollectionsLib.Tests

{

[TestFixture]

public class EmployeeManagerTests

{

private EmployeeManager manager;

[SetUp]

public void Init()

{

manager = new EmployeeManager();

}

[Test]

public void GetEmployees\_WhenCalled\_ShouldNotContainNullValues()

{

var result = manager.GetEmployees();

Assert.That(result, Has.None.Null);

}

[Test]

public void GetEmployees\_ShouldIncludeEmployeeWithId100()

{

var result = manager.GetEmployees();

var exists = result.Any(emp => emp.EmpId == 100);

Assert.That(exists, Is.True);

}

[Test]

public void GetEmployees\_ShouldContainOnlyUniqueEmployees()

{

var employees = manager.GetEmployees();

var uniqueCount = employees.Select(e => e.EmpId).Distinct().Count();

Assert.That(uniqueCount, Is.EqualTo(employees.Count));

}

[Test]

public void GetEmployeesAndPreviousYearEmployees\_ShouldBeEquivalent\_UsingConstraint()

{

var currentList = manager.GetEmployees();

var previousList = manager.GetEmployeesWhoJoinedInPreviousYears();

Assert.That(currentList, Is.EquivalentTo(previousList));

}

[Test]

public void GetEmployeesAndPreviousYearEmployees\_ShouldBeEquivalent\_UsingClassic()

{

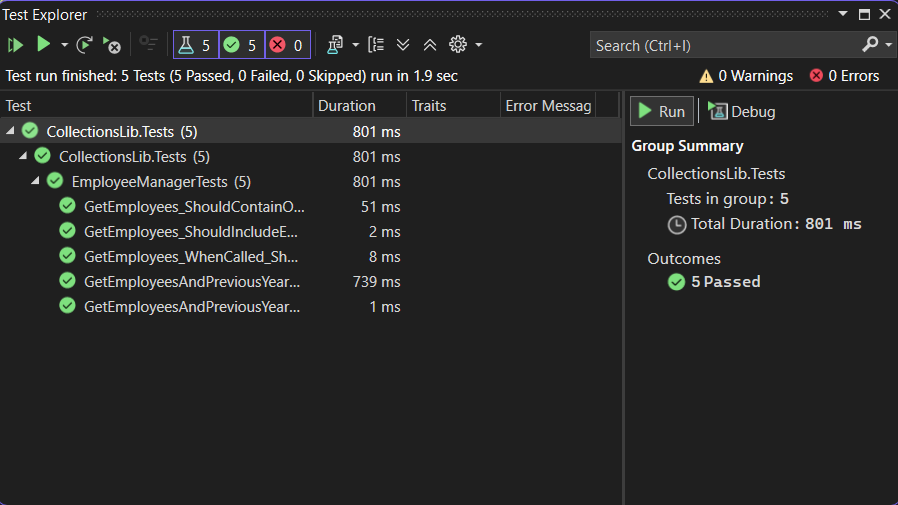
var currentList = manager.GetEmployees();

var previousList = manager.GetEmployeesWhoJoinedInPreviousYears();

CollectionAssert.AreEquivalent(currentList, previousList);

}}}

**Result:**

****

**6. Objectives**

This lab will help you become skilled at writing automated unit tests using the NUnit framework.

Explain & Demonstrate TestCaseSource to minimize amount of code used for testing.

Create a Unit Test Project using NUnit Framework for the following requirement. Click here to download the source project FourSeasonsLib.

Following is the business scenario implemented in the source project.

• The function accepts a month name and returns what season in that month (both inclusive) based on certain logic. The criteria is elaborated in the matrix below.

Seasons Month Climate

Spring February to March Sunny and pleasant

Summer April to June Hot

Monsoon July to September Wet, hot and humid

Autumn September to November Pleasant

Winter December to January Very Cool

Make sure that you are not writing multiple test methods that are equal to the number of execution paths. Your focus should be of writing minimum code while unit testing.

Try both straight forward and alternate ways of working with the TestCaseSource attribute for the above scenario.

Recommendations:

Test Project Name: <ClassLib\_Project>.Tests

Test Class Name: <SUT>Tests

Test Method Name: UnitUnderTest\_Scenario\_ExpectedOutcome

Note:

• Enforce the Single Assertion Rule

• Use Assert.That()

Steps to perform

1) Create a Class Library project in the same solution which is provided and name it as suggested.

2) Rename the class file name (<SUT>Tests.cs).

3) Add the assembly reference of the ConverterLib project to the test project.

4) Additionally add the reference of NUnit, NUnit3TestAdapter and Moq in the test project using NuGet Package Manager (NPM).

5) Write the suggested test methods.

6) Run your tests.

7) Break the test by modifying the source project functionality.

8) Rerun the test.

9) Observe the test result.

**SOLUTION:**

using NUnit.Framework;

using SeasonsLib;

using System.Collections.Generic;

namespace SeasonsLib.Tests

{

[TestFixture]

public class SeasonTellerTests

{

private SeasonTeller \_seasonTeller;

[SetUp]

public void Setup()

{

\_seasonTeller = new SeasonTeller();

}

public static IEnumerable<TestCaseData> ValidMonthCases()

{

yield return new TestCaseData("February", "Spring");

yield return new TestCaseData("March", "Spring");

yield return new TestCaseData("April", "Summer");

yield return new TestCaseData("May", "Summer");

yield return new TestCaseData("June", "Summer");

yield return new TestCaseData("July", "Monsoon");

yield return new TestCaseData("August", "Monsoon");

yield return new TestCaseData("September", "Monsoon");

yield return new TestCaseData("October", "Autumn");

yield return new TestCaseData("November", "Autumn");

yield return new TestCaseData("December", "Winter");

yield return new TestCaseData("January", "Winter");

}

[Test, TestCaseSource(nameof(ValidMonthCases))]

public void DisplaySeasonBy\_ValidMonth\_ReturnsExpectedSeason(string month, string expectedSeason)

{

var result = \_seasonTeller.DisplaySeasonBy(month);

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

}

[Test]

public void DisplaySeasonBy\_InvalidMonth\_ReturnsInvalidSeason()

{

var result = \_seasonTeller.DisplaySeasonBy("HelloMonth");

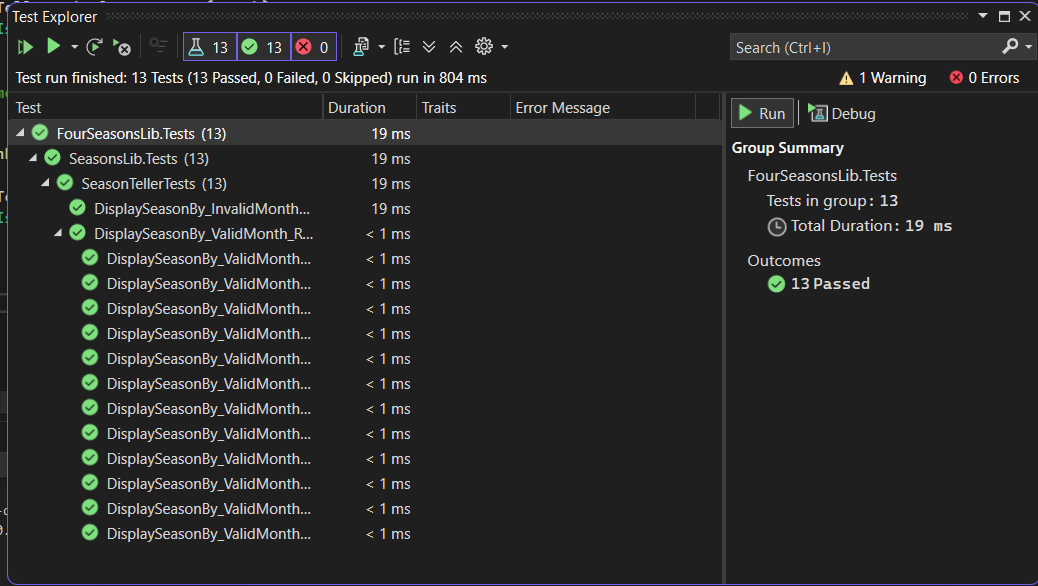
Assert.That(result, Is.EqualTo("Invalid Season"));

}

}

}

**Result:**

****

**7. Objectives**

This lab will help you become skilled at writing automated unit tests using the NUnit framework.

Explain the benefit of writing parameterized test cases.

Demonstrate on parameterized test cases using an example.

Create a Unit Test Project using NUnit Framework for the following requirement. Click here to download the source project.

The LeapYearCalculatorLib application tells the user whether the entered year is a leap year or not. Also it checks the input value to make sure that the data given is a valid year.

• If the given year is a Leap Year, the program will output 1. If it’s not, then the program will result 0.

• Any value between 1753 and 9999 (both inclusive) will be a valid year. Violation of this rule will result -1.

Recommendations:

Test Project Name: <ClassLib\_Project>.Tests

Test Class Name: <SUT>Tests

Test Method Name: UnitUnderTest\_Scenario\_ExpectedOutcome

After writing the above test methods, run the tests and assert the results with that of the success/failure messages.

Note:

• Enforce the Single Assertion Rule

• Use Assert.That()

Steps to perform:

1) Create a Class Library project in the same solution which is provided and name it as suggested.

2) Rename the class file name (<SUT>Tests.cs).

3) Add the assembly reference of the UtilLib project to the test project.

4) Additionally add the reference of both NUnit and NUnit3TestAdapter in the test project using NuGet Package Manager (NPM).

5) Write the suggested test methods.

6) Run your tests.

7) Break the test by modifying the source project functionality.

8) Rerun the test.

9) Observe the test result.

**Solution:**

using NUnit.Framework;

using LeapYearCalculatorLib;

namespace LeapYearCalculatorLib.Tests

{

[TestFixture]

public class LeapYearTests

{

private LeapYearCalculator calculator;

[SetUp]

public void Initialize()

{

calculator = new LeapYearCalculator();

}

[TestCase(2000, 1)]

[TestCase(1996, 1)]

[TestCase(2020, 1)]

public void IsLeapYear\_InputIsLeapYear\_ReturnsOne(int inputYear, int expectedResult)

{

var result = calculator.IsLeapYear(inputYear);

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

}

[TestCase(1900, 0)]

[TestCase(2001, 0)]

[TestCase(2100, 0)]

public void IsLeapYear\_InputIsNotLeapYear\_ReturnsZero(int inputYear, int expectedResult)

{

var result = calculator.IsLeapYear(inputYear);

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

}

[TestCase(1000, -1)]

[TestCase(12000, -1)]

[TestCase(1700, -1)]

public void IsLeapYear\_YearOutsideValidRange\_ReturnsNegativeOne(int inputYear, int expectedResult)

{

var result = calculator.IsLeapYear(inputYear);

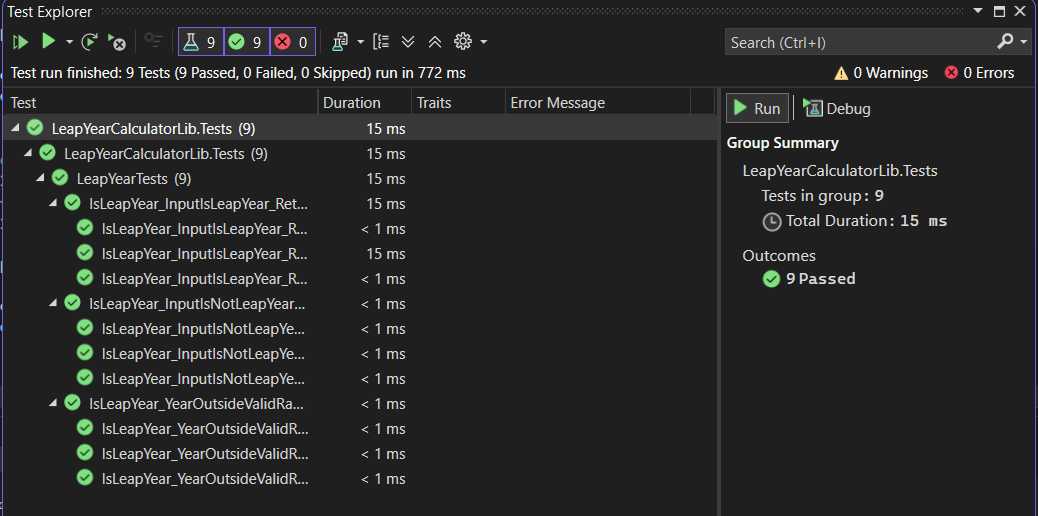
Assert.That(result, Is.EqualTo(expectedResult));

}

}

}

**Result:**

****

**8. Objectives**

This lab will help you become skilled at writing automated unit tests using the NUnit framework.

Demonstrate on testing methods that throw an exception.

Create a Unit Test Project using NUnit Framework for the following requirement. Click here to download the source project.

Following is the application logic of the given UserManagerLib source project.

User creation will be successful under the below given condition

• PANCardNo property reads only 10 characters length value from the user. It is a mandatory property while creating the user.

Following exceptions may occur while creating the user.

o NullReferenceException- If the input value is empty or null

o FormatException-If the input string does not meet the length criteria.

• While writing test cases for the above program, you need to ensure that you are handling all types of exception that may raise during the CreateUser method call. Also, write test method for happy path in the function.

Recommendations:

Test Project Name: <ClassLib\_Project>.Tests

Test Class Name: <SUT>Tests

Test Method Name: UnitUnderTest\_Scenario\_ExpectedOutcome

Note:

• Enforce the Single Assertion Rule

• Use Assert.That()

Steps to perform

1) Create a Class Library project in the same solution which is provided and name it as suggested.

2) Rename the class file name (<SUT>Tests.cs).

3) Add the assembly reference of the UtilLib project to the test project.

4) Additionally add the reference of both NUnit and NUnit3TestAdapter in the test project using NuGet Package Manager (NPM).

5) Write the suggested test methods.

6) Run your tests.

7) Break the test by modifying the source project functionality.

8) Rerun the test.

9) Observe the test result.

**Solution:**

using NUnit.Framework;

using System;

namespace UserManagerLib.Tests

{

[TestFixture]

public class UserTests

{

private User \_user;

[SetUp]

public void Init()

{

\_user = new User();

}

[TestCase(null)]

[TestCase("")]

public void ValidatePANCardNumber\_InputIsNullOrEmpty\_ThrowsNullReferenceException(string input)

{

var ex = Assert.Throws<NullReferenceException>(() => \_user.ValidatePANCardNumber(input));

Assert.That(ex.Message, Is.EqualTo("Invalid Pan Card Number"));

}

[TestCase("ABCDE", TestName = "TooShortPAN\_ThrowsFormatException")]

[TestCase("ABCDEFGHIJKLMNOP", TestName = "TooLongPAN\_ThrowsFormatException")]

public void ValidatePANCardNumber\_InvalidLength\_ThrowsFormatException(string input)

{

var ex = Assert.Throws<FormatException>(() => \_user.ValidatePANCardNumber(input));

Assert.That(ex.Message, Is.EqualTo("Pan Card Number Should contain only 10 characters"));

}

[Test]

public void ValidatePANCardNumber\_ValidPAN\_ReturnsValid()

{

string result = \_user.ValidatePANCardNumber("ABCDE1234Z");

Assert.That(result, Is.EqualTo("Valid"));

}

[Test]

public void CreateUser\_ValidUser\_DoesNotThrow()

{

var validUser = new User

{

PANCardNo = "ABCDE1234Z"

};

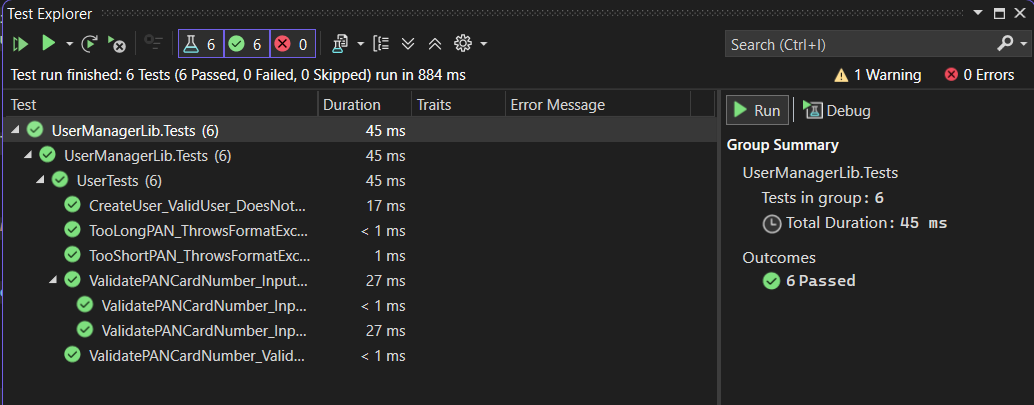
Assert.That(() => validUser.CreateUser(validUser), Throws.Nothing);

}

}

}

**Result:**

****

**9. Objectives**

This lab will help you become skilled at using one of the flexible mock frameworks called Moq and how it can be used to isolate units of code from their dependencies.

Demonstrate on how to create a testable code with Moq.

Create a unit test project using NUnit for the given ConverterLib project. Click here to download the source project.

One of the functionalities called USDToEuro which is defined in the Converter class should be your primary target while unit testing. It takes the US dollar as an input and convert it to Euro with the help of an external service, IDollarToEuroExchangeRateFeed. Since your application requires this functionality and the same can’t be tested while unit testing because you may not have a grip on the logic behind that service. On top of that, this particular functionality might have tested before it’s made available.

Use Moq framework in order to bypass the functionality which is defined in the IDollarToEuroExchangeRateFeed service.

Write test methods for the given functionalities to make sure that it returns the expected result under various circumstances.

Recommendations:

Test Project Name: <ClassLib\_Project>.Tests

Test Class Name: <SUT>Tests

Test Method Name: UnitUnderTest\_Scenario\_ExpectedOutcome

Note:

• Enforce the Single Assertion Rule

• Use Assert.That()

Steps to perform

1) Create a Class Library project in the same solution which is provided and name it as suggested.

2) Rename the class file name (<SUT>Tests.cs).

3) Add the assembly reference of the ConverterLib project to the test project.

4) Additionally add the reference of NUnit, NUnit3TestAdapter and Moq in the test project using NuGet Package Manager (NPM).

5) Write the suggested test methods.

6) Run your tests.

7) Break the test by modifying the source project functionality.

8) Rerun the test.

9) Observe the test result.

**Solution:**

using NUnit.Framework;

using Moq;

using ConverterLib;

using CurrencyConverterApp;

namespace ConverterLib.Tests

{

[TestFixture]

public class ConverterTests

{

private Mock<IDollarToEuroExchangeRateFeed> \_mockFeed;

private Converter \_converter;

[SetUp]

public void Setup()

{

\_mockFeed = new Mock<IDollarToEuroExchangeRateFeed>();

\_converter = new Converter(\_mockFeed.Object);

}

[Test]

public void USDToEuro\_ValidInput\_ReturnsExpectedConversion()

{

// Arrange

\_mockFeed.Setup(x => x.GetActualUSDollarValue()).Returns(0.9);

double inputDollars = 100;

// Act

var result = \_converter.USDToEuro(inputDollars);

// Assert

Assert.That(result, Is.EqualTo(90.0));

}

[Test]

public void CelsiusToKelvin\_InputCelsius\_ReturnsKelvin()

{

double result = \_converter.CelsiusToKelvin(0);

Assert.That(result, Is.EqualTo(273.15));

}

[Test]

public void KilogramToPound\_InputKilogram\_ReturnsPounds()

{

double result = \_converter.KilogramToPound(1);

Assert.That(result, Is.EqualTo(2.205));

}

[Test]

public void KilometerToMile\_InputKilometer\_ReturnsMiles()

{

double result = \_converter.KilometerToMile(1.609);

Assert.That(result, Is.EqualTo(1).Within(0.01)); // using Within for precision tolerance

}

[Test]

public void LiterToGallon\_InputLiter\_ReturnsGallon()

{

double result = \_converter.LiterToGallon(3.785);

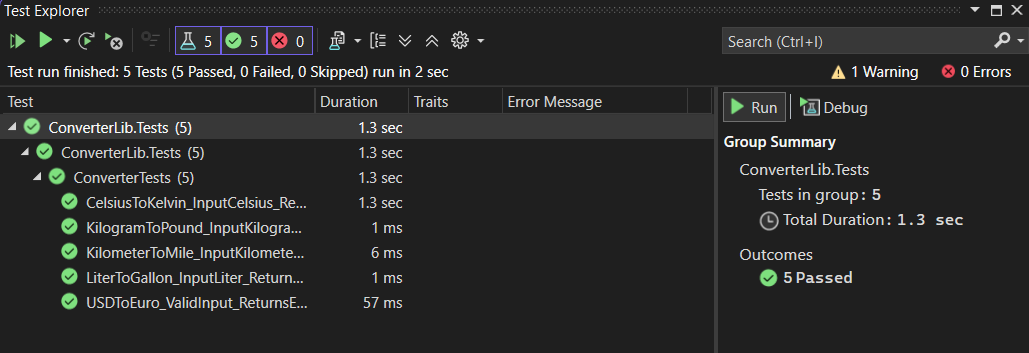
Assert.That(result, Is.EqualTo(1).Within(0.01));

}

}

}

**Result:**

****