NUnit Implementation Report with Mocking

**Moq and Test-Driven Development Exercises**

# Objectives

* + Understand how mocking supports Test-Driven Development (TDD).
  + Explain mocking in Unit Testing with real-world dependencies.
  + Explore Dependency Injection: constructor and method injection.
  + Demonstrate testable code using Moq (mail, file system, database).

# Task 1 – Mocking MailSender using Moq

**Step 1: Create** CustomerCommLib **class library IMailSender.cs**

public interface IMailSender

{

bool Send Mail( string to Address , string message );

}

**MailSender.cs**

using System . Net; using System . Net. Mail;

namespace CustomerComm Lib

{

public class MailSender : IMailSender

{

public bool Send Mail( string to Address , string message )

{

MailMessage mail = new MailMessage ();

Smtp Client smtp Server = new Smtp Client(" smtp . gmail. com ");

mail. From = new MailAddress (" your\_email@ gmail . com "); mail. To. Add ( to Address );

mail. Subject = " Test␣Mail"; mail. Body = message ;

smtp Server. Port = 587;

smtp Server. Credentials = new Network Credential (" username ", "

*‹→* password ");

smtp Server. Enable Ssl = true ;

smtp Server. Send ( mail); return true ;

}

}

}

**CustomerComm.cs**

namespace CustomerComm Lib

{

public class CustomerComm

{

private readonly IMailSender \_mailSender;

public CustomerComm ( IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool Send MailTo Customer ()

{

return \_mailSender. Send Mail(" cust123 @ abc . com ", " Some ␣Message ");

}

}

}

**Step 2: Create** CustomerComm.Tests **with Moq CustomerCommTests.cs**

using NUnit. Framework ; using Moq ;

using CustomerComm Lib ;

namespace CustomerComm . Tests

{

[ TestFixture ]

public class CustomerComm Tests

{

private Mock < IMailSender > \_mock MailSender ; private CustomerComm \_customerComm ;

[ One Time SetUp ] public void Init ()

{

\_mock MailSender = new Mock < IMailSender >();

\_mock MailSender . Setup ( m => m. Send Mail( It. IsAny < string >(), It. IsAny <

*‹→* string >()))

. Returns ( true );

\_customerComm = new CustomerComm ( \_mock MailSender . Object);

}

[ Test]

public void Send MailTo Customer\_Returns True ()

{

var result = \_customerComm . Send MailTo Customer (); Assert. Is True ( result);

}

}

}

**Expected Output:**

Test Run Summary:

SendMailToCustomer\_ReturnsTrue: Passed Result: Passed 1 test

# Task 2 – Mocking File System Access

**Step 1: Create** MagicFilesLib

**IDirectoryExplorer.cs**

public interface IDirectory Explorer

{

ICollection < string > GetFiles ( string path );

}

**DirectoryExplorer.cs**

using System . Collections . Generic ; using System . IO;

namespace Magic Files Lib

{

public class Directory Explorer : IDirectory Explorer

{

public ICollection < string > GetFiles ( string path )

{

return Directory . GetFiles ( path );

}

}

}

**Step 2: Create** DirectoryExplorer.Tests

**DirectoryExplorerTests.cs**

using NUnit. Framework ; using Moq ;

using Magic Files Lib ;

using System . Collections . Generic ;

namespace Directory Explorer . Tests

{

[ TestFixture ]

public class Directory ExplorerTests

{

private Mock < IDirectory Explorer > \_mock Explorer ; private readonly string \_file 1 = " file . txt"; private readonly string \_file 2 = " file2 . txt";

[ One Time SetUp ] public void Init ()

{

\_mock Explorer = new Mock < IDirectory Explorer >();

\_mock Explorer . Setup ( d => d. GetFiles ( It. IsAny < string >()))

. Returns ( new List < string > { \_file1 , \_file 2 });

}

[ Test]

public void GetFiles\_Returns Expected Files ()

{

var files = \_mock Explorer . Object. GetFiles (" dummy Path ");

Assert. Is NotNull( files );

Assert. Are Equal (2 , files . Count); Collection Assert . Contains ( files , \_file 1 );

}

}

}

**Expected Output:**

Test Run Summary:

GetFiles\_ReturnsExpectedFiles: Passed Result: Passed 1 test

# Task 3 – Mocking Database with Moq

**Step 1: Create** PlayersManagerLib

**IPlayerMapper.cs**

public interface IPlayerMapper

{

bool Is PlayerName Exists In Db ( string name ); void Add New PlayerInto Db ( string name );

}

**PlayerMapper.cs**

using System . Data . SqlClient;

namespace Players ManagerLib

{

public class PlayerMapper : IPlayerMapper

{

private readonly string \_connection String =

" Data ␣Source =( local); Initial␣Catalog = Game DB ; Integrated ␣Security =

*‹→* True ";

public bool Is PlayerName Exists In Db ( string name )

{

using var conn = new SqlConnection ( \_connection String ); conn . Open ();

using var cmd = conn . Create Command ();

cmd . Command Text = " SELECT ␣COUNT (\*) ␣FROM ␣Player␣WHERE ␣Name ␣=␣@ name "; cmd . Parameters . Add With Value ("@ name ", name );

return ( int) cmd . Execute Scalar () > 0;

}

public void Add New PlayerInto Db ( string name )

{

using var conn = new SqlConnection ( \_connection String ); conn . Open ();

using var cmd = conn . Create Command ();

cmd . Command Text = " INSERT ␣INTO ␣Player␣([ Name ]) ␣VALUES ␣(@ name )"; cmd . Parameters . Add With Value ("@ name ", name );

cmd . Execute Non Query ();

}

}

}

**Player.cs**

using System ;

namespace Players ManagerLib

{

public class Player

{

public string Name { get; } public int Age { get; }

public string Country { get; } public int No OfMatches { get; }

public Player( string name , int age , string country , int no OfMatches )

{

Name = name ; Age = age ;

Country = country ;

No OfMatches = no OfMatches ;

}

public static Player RegisterNew Player ( string name , IPlayerMapper

*‹→* playerMapper = null)

{

playerMapper ??= new PlayerMapper ();

if ( string . Is NullOrWhite Space ( name ))

throw new ArgumentException (" Player␣name ␣ c a n t ␣be␣empty .");

if ( playerMapper. Is PlayerName Exists In Db ( name ))

throw new ArgumentException (" Player␣name ␣already ␣exists .");

playerMapper. Add New PlayerInto Db ( name ); return new Player( name , 23 , " India ", 30);

}

}

}

**Step 2: Create** PlayerManager.Tests

**PlayerTests.cs**

using NUnit. Framework ; using Moq ;

using Players ManagerLib ; using System ;

namespace PlayerManager . Tests

{

[ TestFixture ]

public class PlayerTests

{

private Mock < IPlayerMapper > \_mock Mapper;

[ One Time SetUp ] public void Init ()

{

\_mock Mapper = new Mock < IPlayerMapper >();

}

[ Test]

public void RegisterNew Player\_Valid Name\_Returns Player ()

{

\_mock Mapper. Setup ( m => m. Is PlayerName Exists In Db ( It. IsAny < string >())

*‹→* ). Returns ( false );

\_mock Mapper. Setup ( m => m. Add New PlayerInto Db ( It. IsAny < string >()));

var player = Player. RegisterNew Player (" Rohit", \_mock Mapper. Object); Assert. Are Equal(" Rohit", player. Name );

Assert. Are Equal (23 , player. Age );

Assert. Are Equal(" India ", player. Country ); Assert. Are Equal (30 , player. No OfMatches );

}

[ Test]

public void RegisterNew Player\_Empty Name\_Throws Exception ()

{

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

Player. RegisterNew Player ("", \_mock Mapper. Object));

Assert. That( ex. Message , Is. EqualTo (" Player␣name ␣ c a n t ␣be␣empty .")

*‹→* );

}

[ Test]

public void Register New Player\_Duplicate Name\_Throws Exception ()

{

\_mock Mapper. Setup ( m => m. Is PlayerName Exists In Db ( It. IsAny < string >())

*‹→* ). Returns ( true );

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

Player. RegisterNew Player (" Rohit", \_mock Mapper. Object));

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

}

}

}

**Expected Output:**

Test Run Summary:

RegisterNewPlayer\_ValidName\_ReturnsPlayer: Passed RegisterNewPlayer\_EmptyName\_ThrowsException: Passed RegisterNewPlayer\_DuplicateName\_ThrowsException: Passed Result: Passed 3 tests