Superset ID-

Moq Hands on:-

// Task 1: CustomerCommLib

namespace CustomerCommLib

{

public interface IMailSender

{

bool SendMail(string toAddress, string message);

}

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);

return true;

}

}

public class CustomerComm

{

IMailSender \_mailSender;

public CustomerComm(IMailSender mailSender)

{

\_mailSender = mailSender;

}

public bool SendMailToCustomer()

{

\_mailSender.SendMail("cust123@abc.com", "Some Message");

return true;

}

}

}

// Task 2: CustomerComm.Tests

[TestFixture]

public class CustomerCommTests

{

private Mock<IMailSender> \_mockMailSender;

private CustomerComm \_customerComm;

[OneTimeSetUp]

public void Init()

{

\_mockMailSender = new Mock<IMailSender>();

\_mockMailSender.Setup(x => x.SendMail(It.IsAny<string>(), It.IsAny<string>())).Returns(true);

\_customerComm = new CustomerComm(\_mockMailSender.Object);

}

[TestCase]

public void SendMailToCustomer\_ReturnsTrue()

{

var result = \_customerComm.SendMailToCustomer();

Assert.IsTrue(result);

}

}

// Task 3: MagicFilesLib

namespace MagicFilesLib

{

public interface IDirectoryExplorer

{

ICollection<string> GetFiles(string path);

}

public class DirectoryExplorer : IDirectoryExplorer

{

public ICollection<string> GetFiles(string path)

{

return Directory.GetFiles(path);

}

}

}

// Task 4: DirectoryExplorer.Tests

[TestFixture]

public class DirectoryExplorerTests

{

private Mock<IDirectoryExplorer> \_mockDirectoryExplorer;

private readonly string \_file1 = "file.txt";

private readonly string \_file2 = "file2.txt";

[OneTimeSetUp]

public void Init()

{

\_mockDirectoryExplorer = new Mock<IDirectoryExplorer>();

\_mockDirectoryExplorer.Setup(x => x.GetFiles(It.IsAny<string>())).Returns(new List<string> { \_file1, \_file2 });

}

[TestCase]

public void GetFiles\_ReturnsExpectedFiles()

{

var files = \_mockDirectoryExplorer.Object.GetFiles("/some/path");

Assert.IsNotNull(files);

Assert.AreEqual(2, files.Count);

Assert.Contains(\_file1, files.ToList());

}

}

// Task 5: PlayersManagerLib

namespace PlayersManagerLib

{

public interface IPlayerMapper

{

bool IsPlayerNameExistsInDb(string name);

void AddNewPlayerIntoDb(string name);

}

public class PlayerMapper : IPlayerMapper

{

private readonly string \_connectionString = "Data Source=(local);Initial Catalog=GameDB;Integrated Security=True";

public bool IsPlayerNameExistsInDb(string name)

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

using (SqlCommand command = connection.CreateCommand())

{

command.CommandText = "SELECT count(\*) FROM Player WHERE Name = @name";

command.Parameters.AddWithValue("@name", name);

var count = (int)command.ExecuteScalar();

return count > 0;

}

}

}

public void AddNewPlayerIntoDb(string name)

{

using (SqlConnection connection = new SqlConnection(\_connectionString))

{

connection.Open();

using (SqlCommand command = connection.CreateCommand())

{

command.CommandText = "INSERT INTO Player ([Name]) VALUES (@name)";

command.Parameters.AddWithValue("@name", name);

command.ExecuteNonQuery();

}

}

}

}

public class Player

{

public string Name { get; private set; }

public int Age { get; private set; }

public string Country { get; private set; }

public int NoOfMatches { get; private set; }

public Player(string name, int age, string country, int noOfMatches)

{

Name = name;

Age = age;

Country = country;

NoOfMatches = noOfMatches;

}

public static Player RegisterNewPlayer(string name, IPlayerMapper playerMapper = null)

{

if (playerMapper == null)

{

playerMapper = new PlayerMapper();

}

if (string.IsNullOrWhiteSpace(name))

throw new ArgumentException("Player name can’t be empty.");

if (playerMapper.IsPlayerNameExistsInDb(name))

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

playerMapper.AddNewPlayerIntoDb(name);

return new Player(name, 23, "India", 30);

}

}

}

// Task 6: PlayerManager.Tests

[TestFixture]

public class PlayerTests

{

private Mock<IPlayerMapper> \_mockPlayerMapper;

[OneTimeSetUp]

public void Init()

{

\_mockPlayerMapper = new Mock<IPlayerMapper>();

}

[TestCase]

public void RegisterNewPlayer\_WithUniqueName\_ShouldReturnValidPlayer()

{

\_mockPlayerMapper.Setup(m => m.IsPlayerNameExistsInDb(It.IsAny<string>())).Returns(false);

var player = Player.RegisterNewPlayer("Rohit", \_mockPlayerMapper.Object);

Assert.AreEqual("Rohit", player.Name);

Assert.AreEqual(23, player.Age);

Assert.AreEqual("India", player.Country);

Assert.AreEqual(30, player.NoOfMatches);

}

[TestCase]

public void RegisterNewPlayer\_WithExistingName\_ShouldThrowException()

{

\_mockPlayerMapper.Setup(m => m.IsPlayerNameExistsInDb(It.IsAny<string>())).Returns(true);

Assert.Throws<ArgumentException>(() => Player.RegisterNewPlayer("Virat", \_mockPlayerMapper.Object));

}

}