WEEK 2 MANDATORY HANDS-ON QUESTIONS

* **Advanced SQL Exercises for Online Retail Store**

Exercise 1: Ranking and Window Functions

Goal: Use ROW\_NUMBER(), RANK(), DENSE\_RANK(), OVER(), and PARTITION BY.

Scenario:

Find the top 3 most expensive products in each category using different ranking functions.

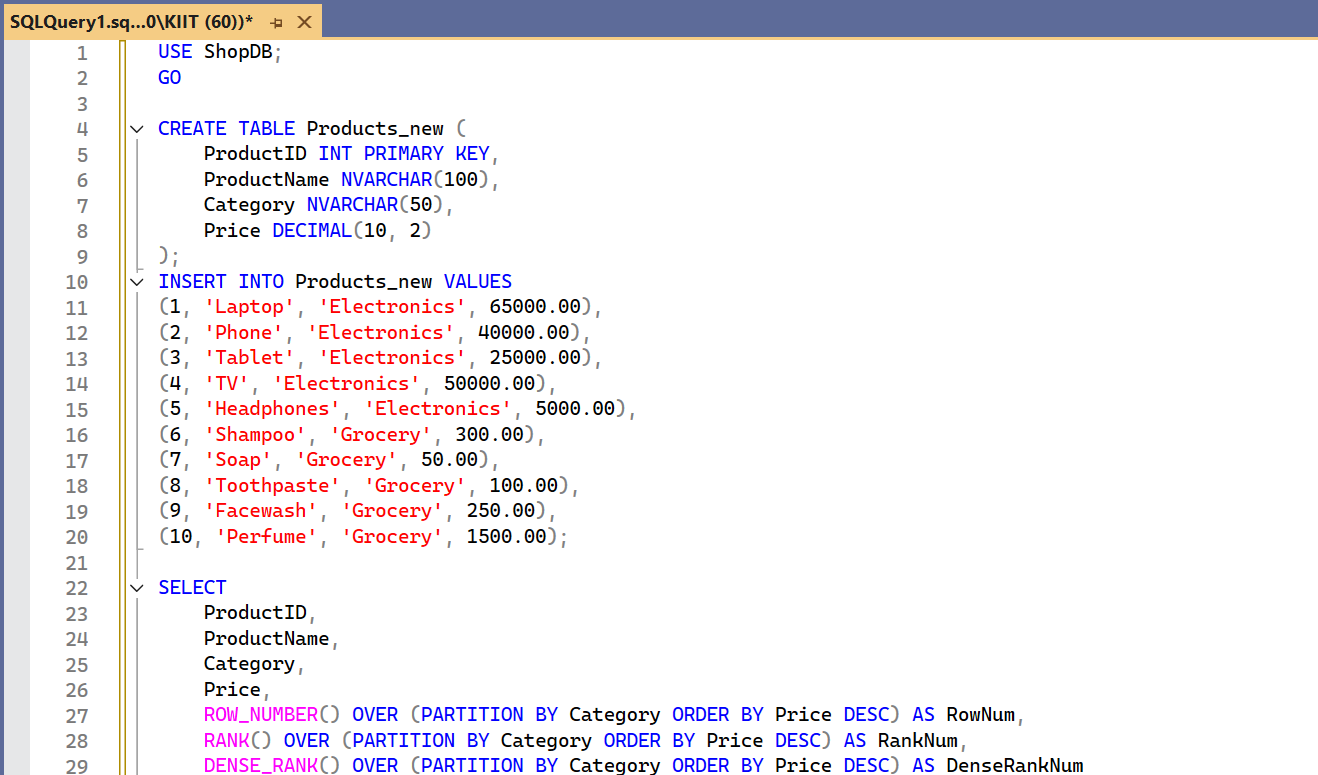
Steps:

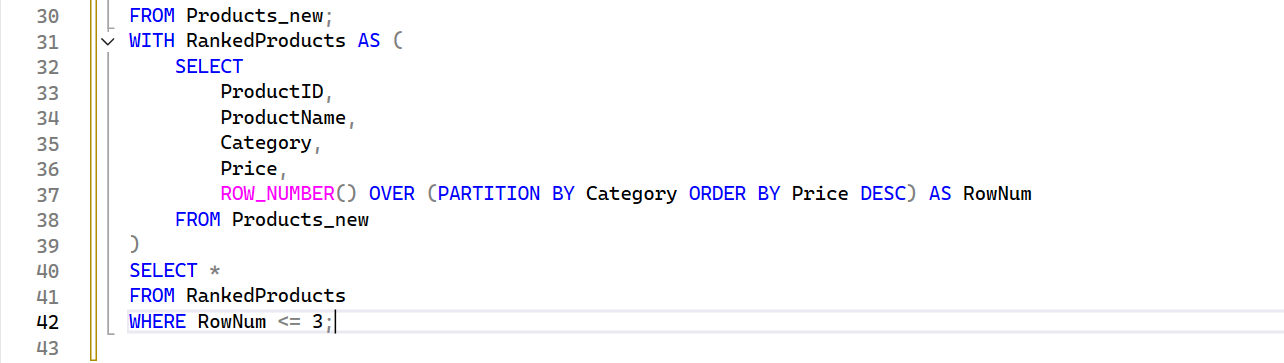
1. Use ROW\_NUMBER() to assign a unique rank within each category.

2. Use RANK() and DENSE\_RANK() to compare how ties are handled.

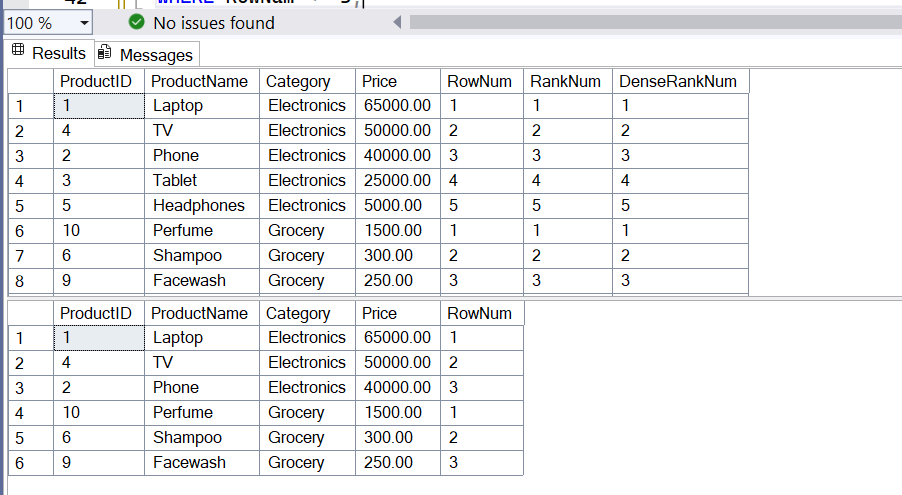
3. Use PARTITION BY Category and ORDER BY Price DESC

CODE:



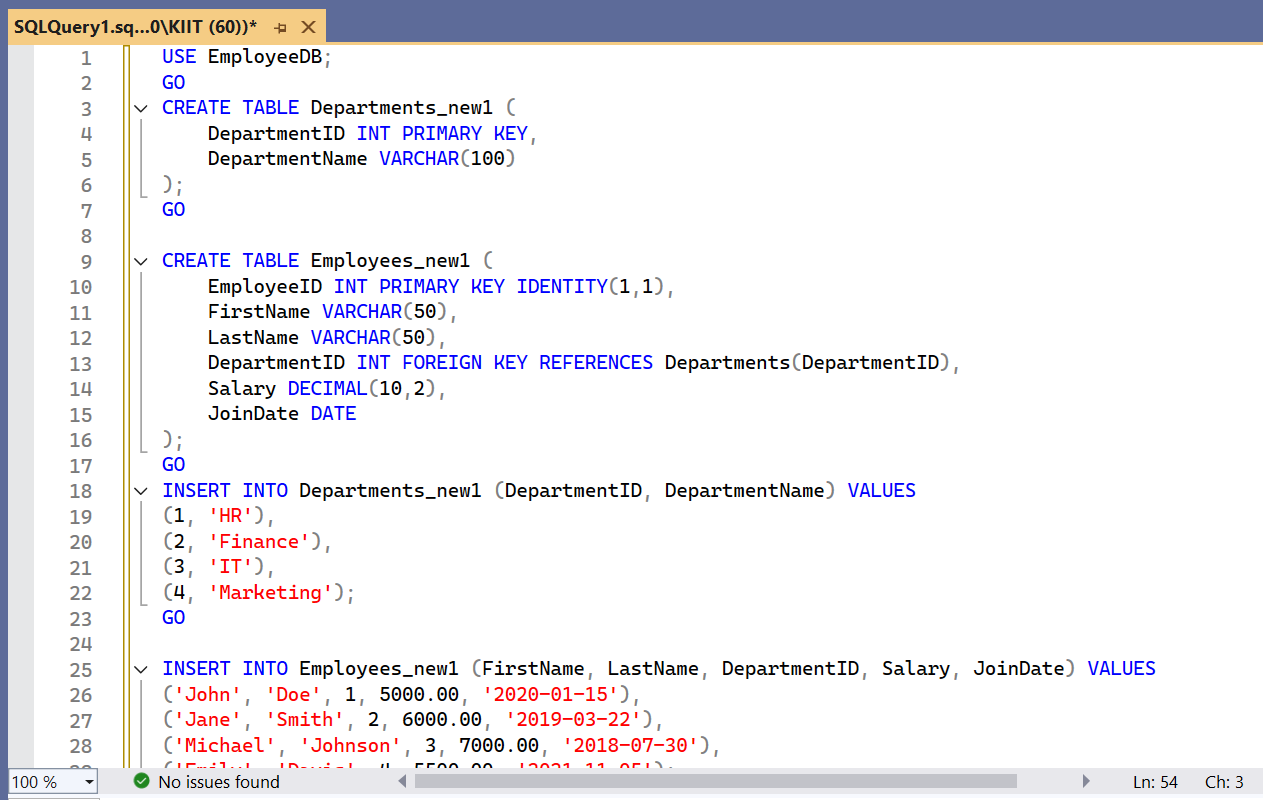


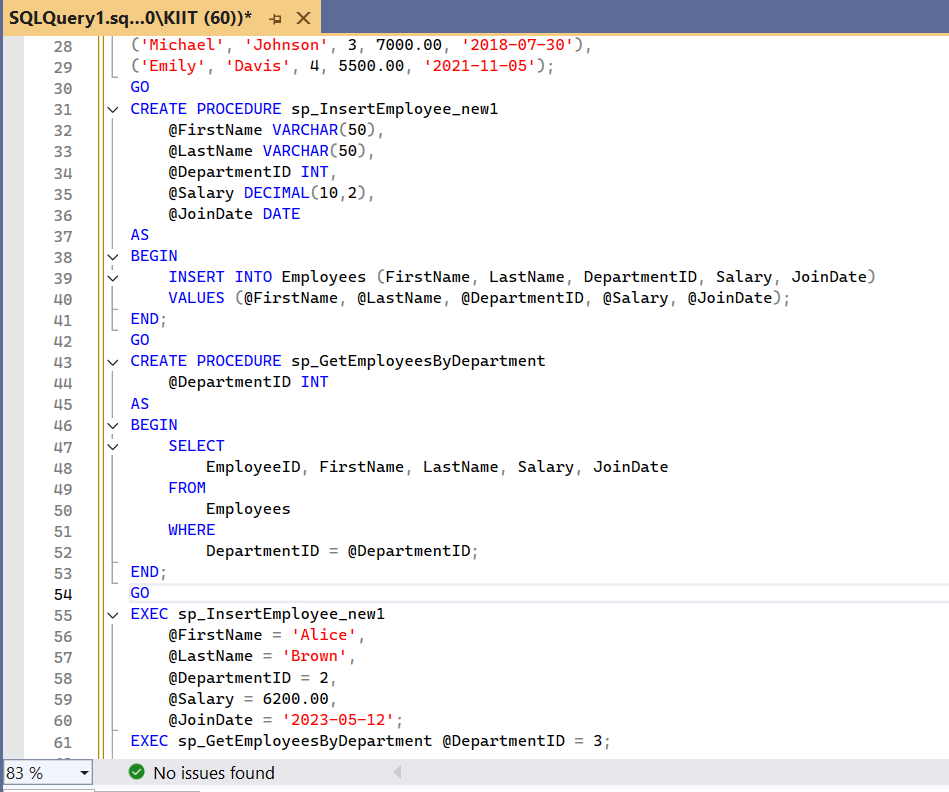
OUTPUT:



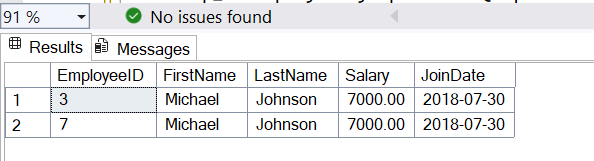
* Create a stored procedure using the Database Schema provided.

CODE:





OUTPUT:



* RETURN DATA FROM A STORED PROCEDURE

Goal: Create a stored procedure that returns the total number of employees in a

department.

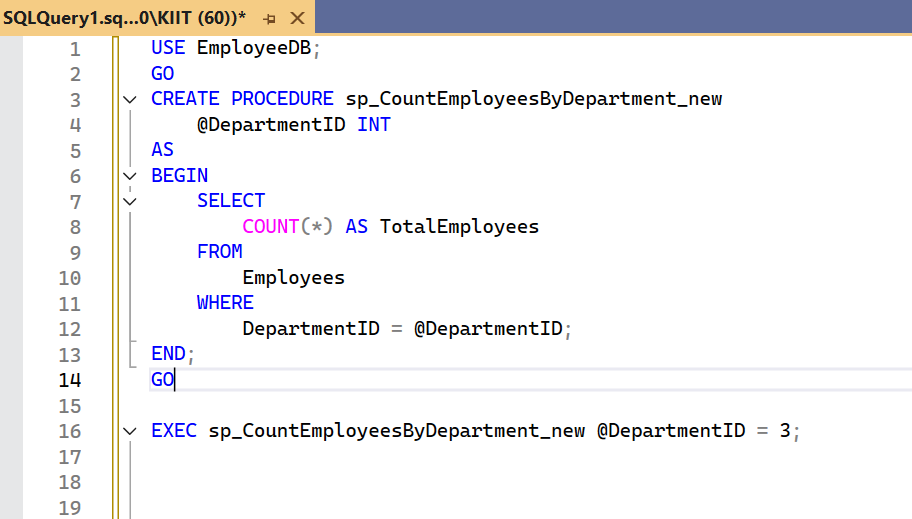
Steps:

1. Define the stored procedure with a parameter for DepartmentID.

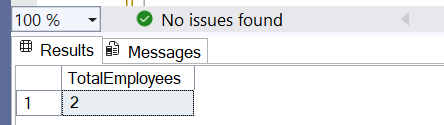
2. Write the SQL query to count the number of employees in the specified department.

3. Save the stored procedure by executing the Stored procedure content

CODE:



OUTPUT:

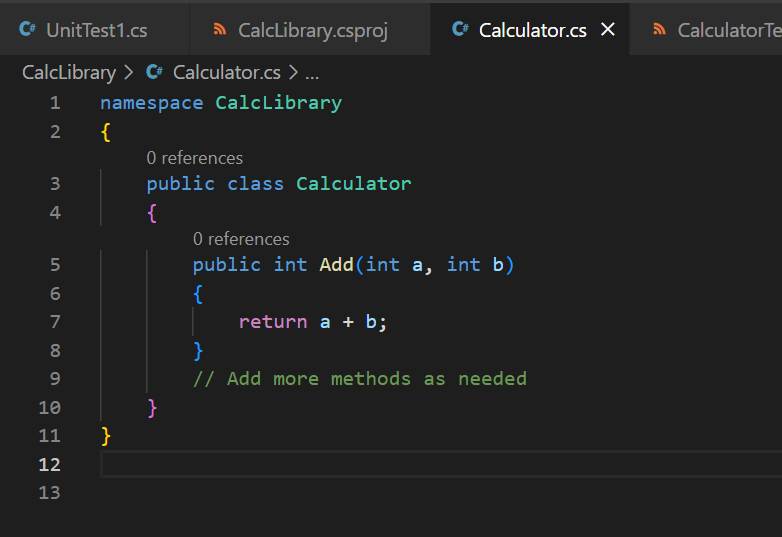


4. Write your first testing program to validate a calculator addition operation

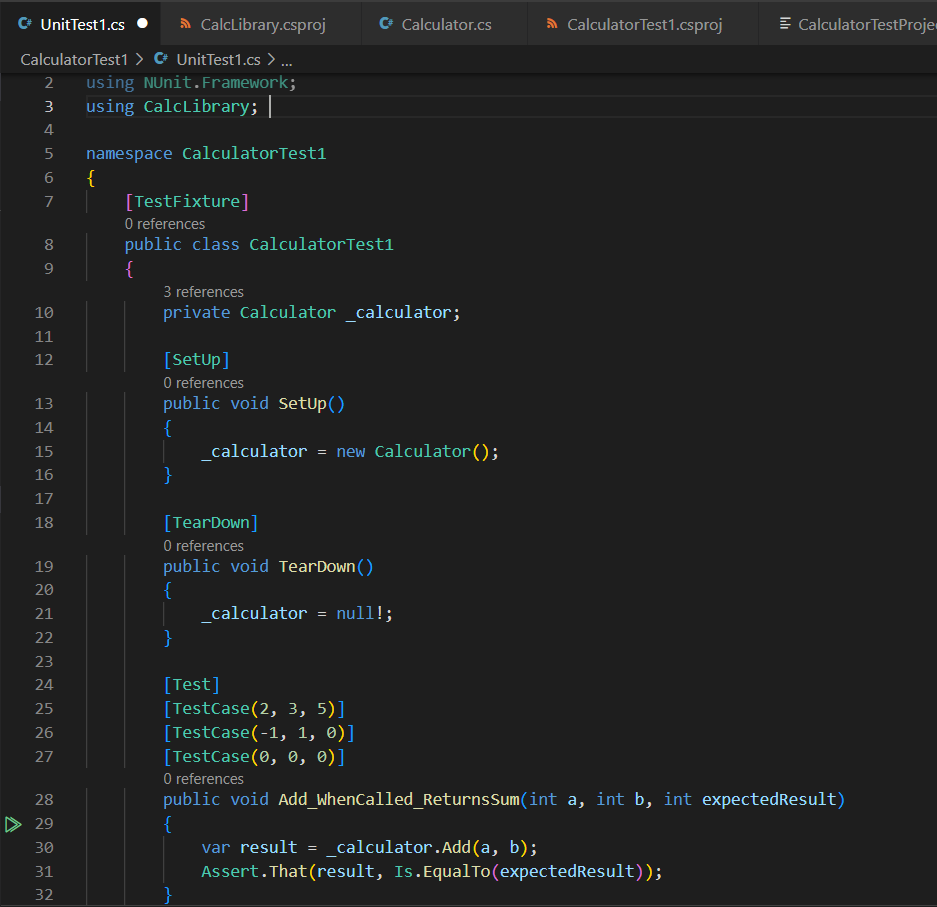
* + TestFixture, Test

Understand the need of [SetUp], [TearDown] & [Ignore] attributes.

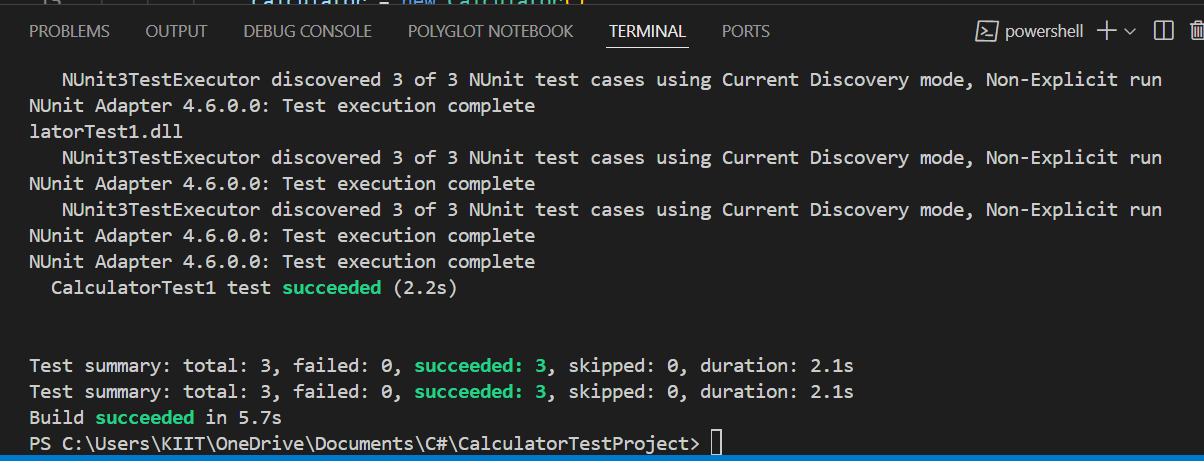
CODE: Implement Calculator Logic in CalcLibrary



CalculatorTests Class



OUTPUT:

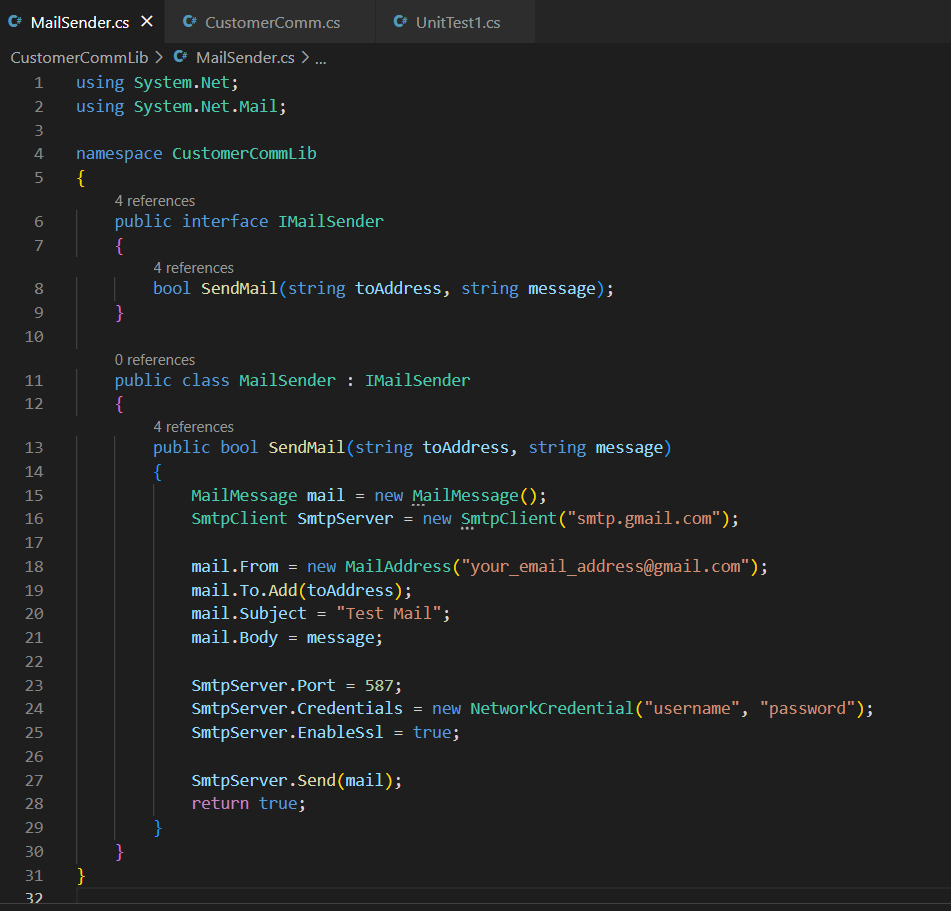


5. MOQ HANDS-ON QUESTION

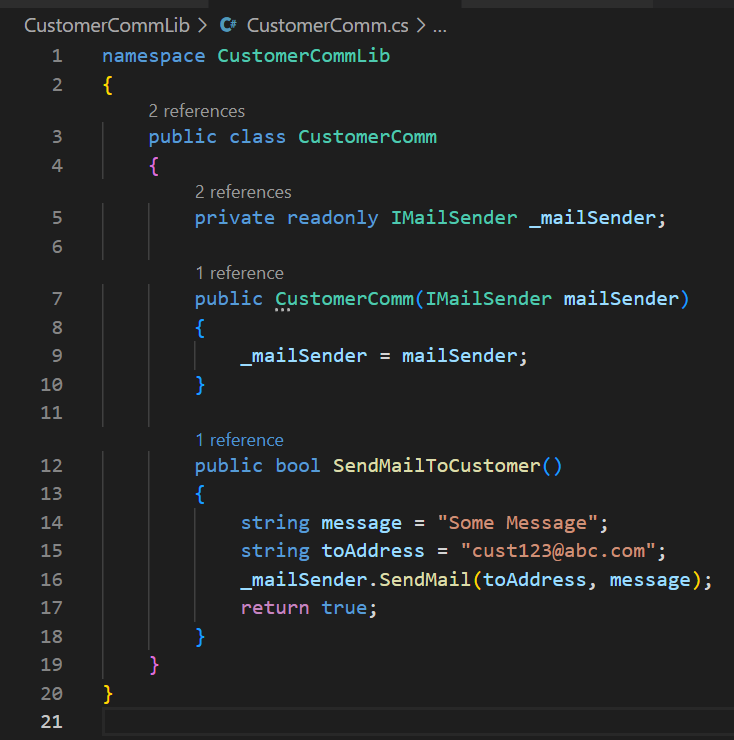
Write Testable Code with Moq for the mentioned scenario. The application in which you are teamed up with, deals with a mail server communication in which your application tries to send mail to its users upon every transaction. Your role is to write unit testing the module that contains send mail functionality. You wanted to perform testing the module without sending any email.

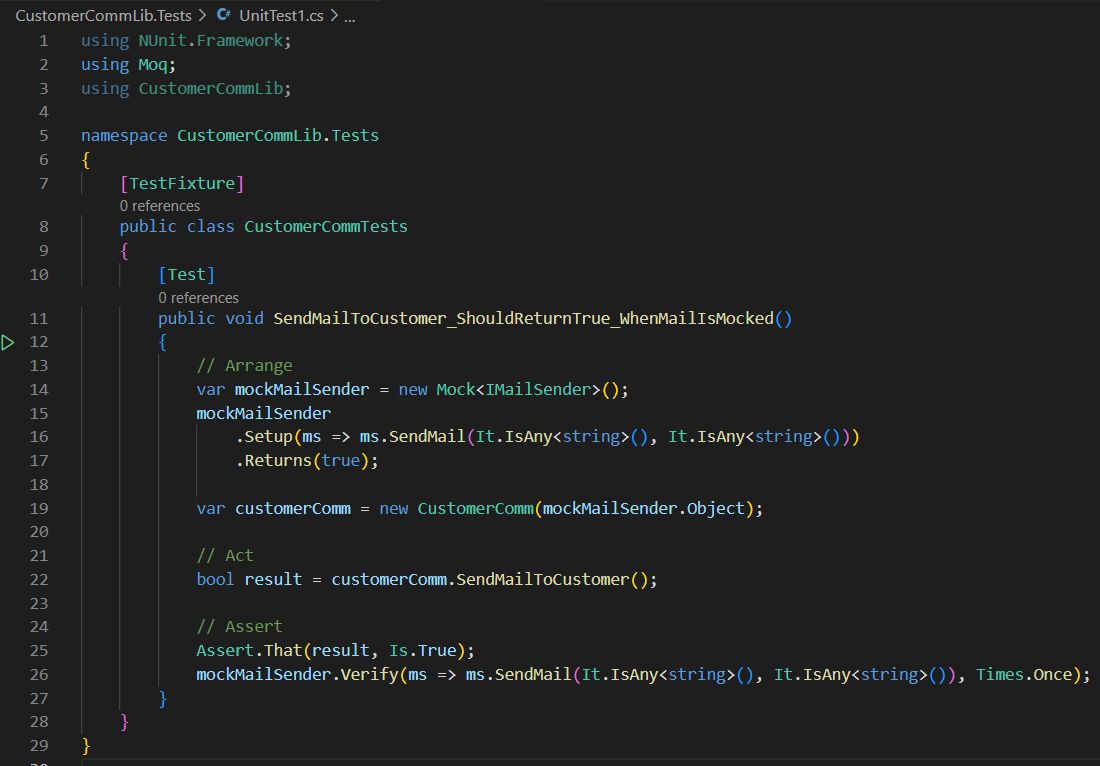
TASK 1

CODE: Definining the IMailSender Interface and MailSender Class

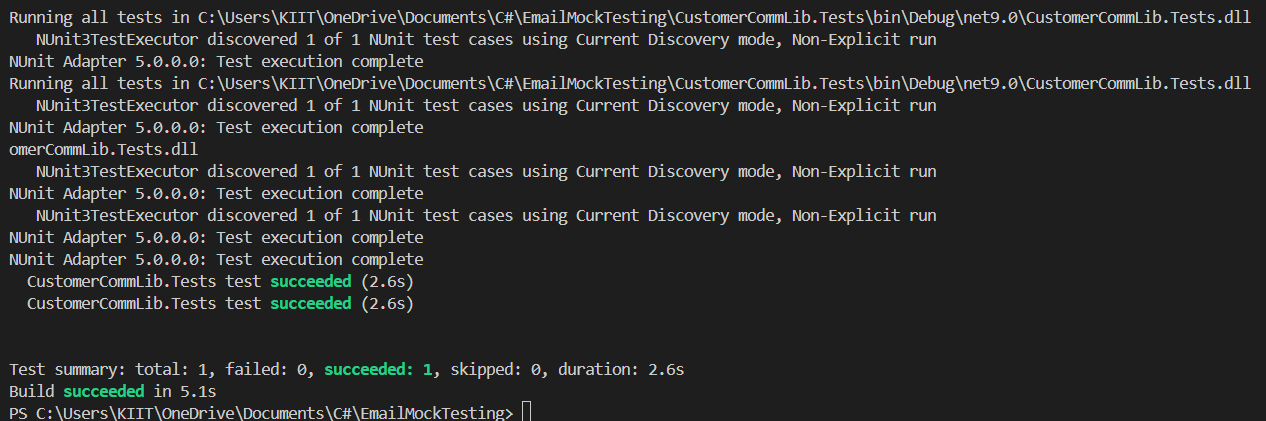


Adding the CustomerComm Class





OUTPUT:



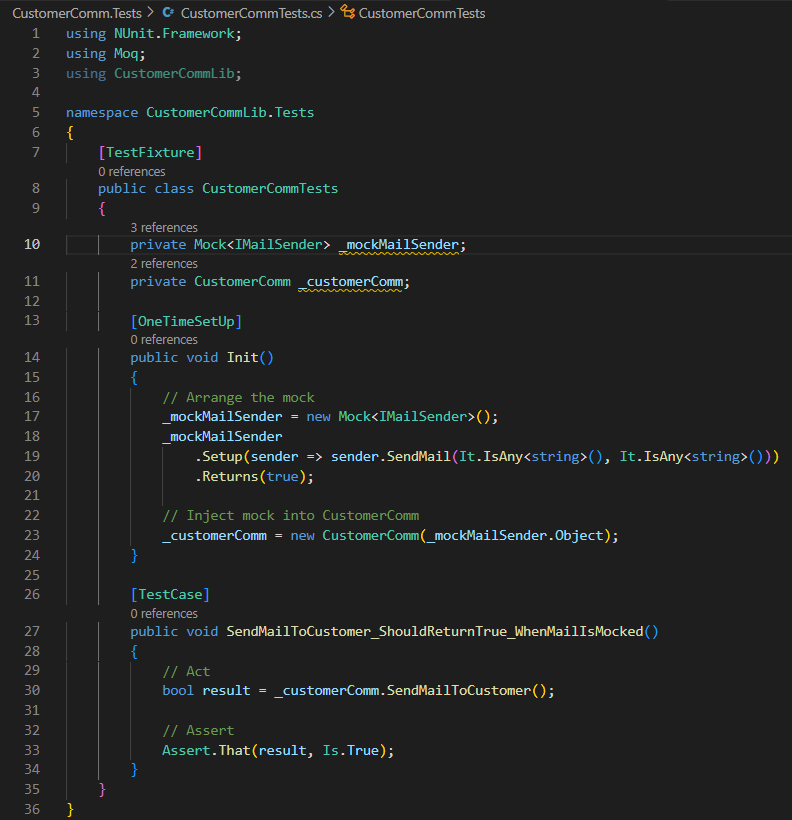
TASK 2

* Create a new class library project called **CustomerComm.Tests** and add the following external dependencies to it using **NuGet Package Manager.**Write unit test code and **mock** the **MailSender (IMailSender)** class. Use **TestFixture**, **OneTimeSetUp** and **TestCase** attribute classes on top of test class, init method and test method respectively.

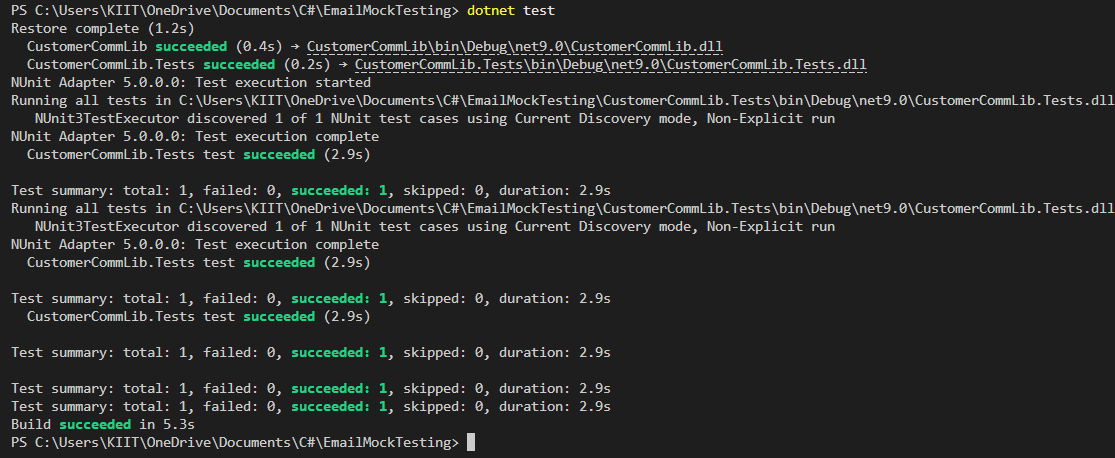
**Configure** the mock object in such away that **SendMail()** method will accept any two string arguments and always return true when **SendMailToCustomer()** gets invoked.

Finally **assert** the return value to “true”.

CODE:



OUTPUT:



Theoretical mandatory questions

1. Explain the meaning of Mocking in Unit Testing and why use mocks in Unit Testing Mocking and Isolation in Unit Testing, Isolating dependencies in Tests using Mocks and Stubs

Mocking is the practice of creating fake objects (called **mocks**) that simulate the behavior of real objects or dependencies in your code (like databases, APIs, or mail servers), without executing them. Mocking in Unit Testing is used to **t**est only the logic of the unit (e.g., a method or class) or avoid calling real services (e.g., no emails sent, no database used).

1. Explain the meaning of Unit testing and its difference on comparison with Functional testing

Unit testing is a software testing method where individual components or functions of a program (called "units") are tested in isolation to ensure they work as expected. It focuses on verifying the smallest testable parts of an application, such as methods or classes, without relying on external systems like databases or APIs.

In contrast, functional testing checks the overall functionality of the application based on the requirements. It tests whether the system behaves correctly from the user's perspective, often involving multiple components working together. While unit testing ensures internal correctness of code logic, functional testing ensures the system performs its intended functions end-to-end.

1. Explain what loosely coupled & testable design is.

A loosely coupled and testable design is one where different parts of the code are independent and interact through well-defined interfaces rather than being tightly linked. This allows each component to be developed, modified, or tested separately without affecting others. Such a design improves testability because dependencies can be easily replaced with mock objects during unit testing, enabling isolated and reliable tests. It promotes flexibility, maintainability, and better code quality.

1. Explain the benefit of writing parameterized test cases

The benefit of writing parameterized test cases is that they allow you to test the same method with multiple sets of input values and expected results using a single test method. This reduces code duplication, makes tests easier to manage, and improves test coverage by validating different scenarios in a compact and readable way. It also helps identify edge cases and ensures consistent behavior of the code across various inputs.

1. What is the benefit of automated testing.

The benefit of automated testing is that it allows tests to be run quickly, repeatedly, and consistently without manual effort. This improves development speed, ensures code reliability, and helps catch bugs early in the development cycle. Automated tests also support continuous integration and deployment (CI/CD), making it easier to maintain high-quality software as the codebase grows or changes over time.