Lesson 01 Demo 05

Maven Surefire Plugin

Objective: To explore Maven Surefire plugin by defining test cases and adding different configurations for test cases

Tools Required: Visual Studio Code and Maven

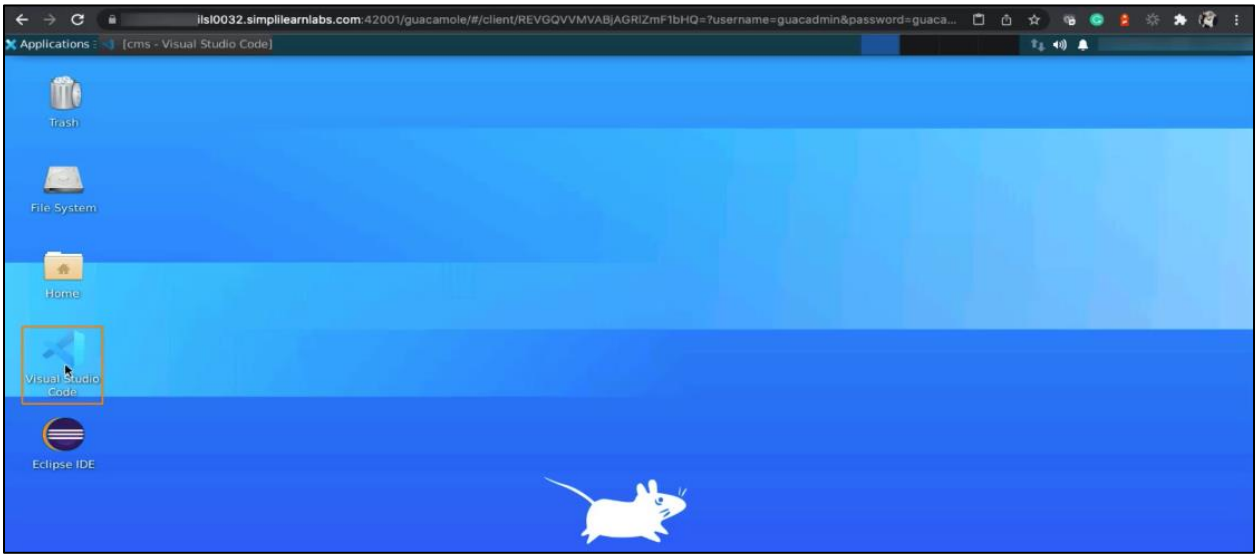
Prerequisites: None

Steps to be followed:

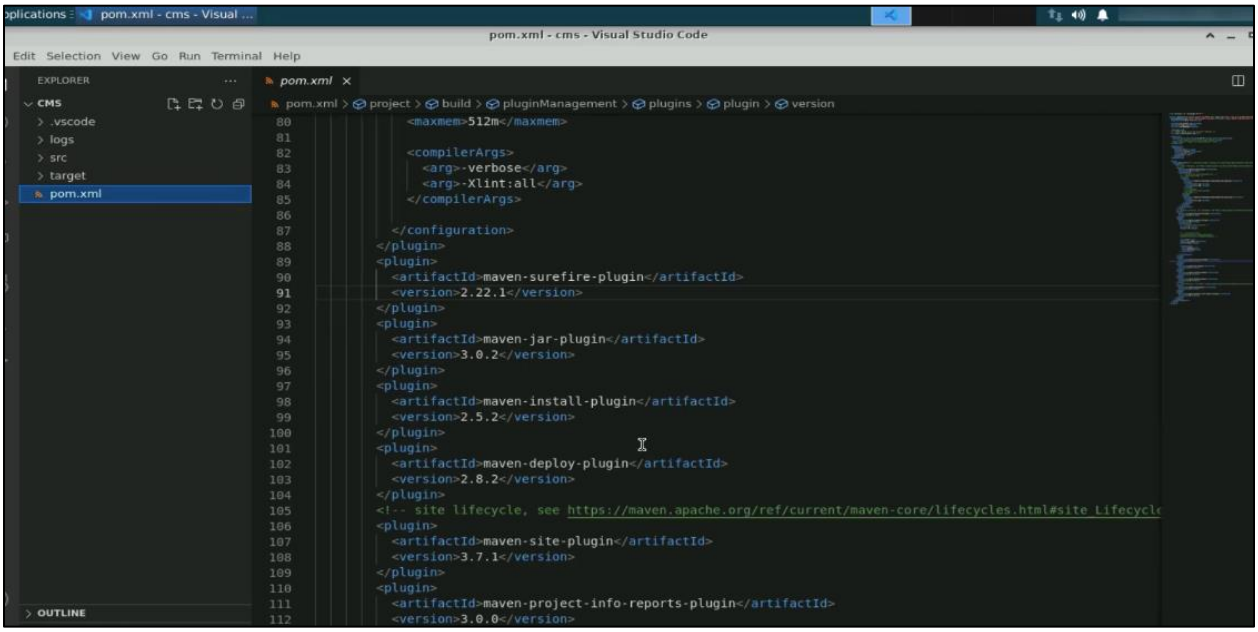
1. Check Maven Surefire plugin
2. Define test cases
3. Test defined test cases
4. Add different configurations for test cases
5. Work with the security manager

Step 1: Check Maven Surefire plugin

1.1 Open Visual Studio Code

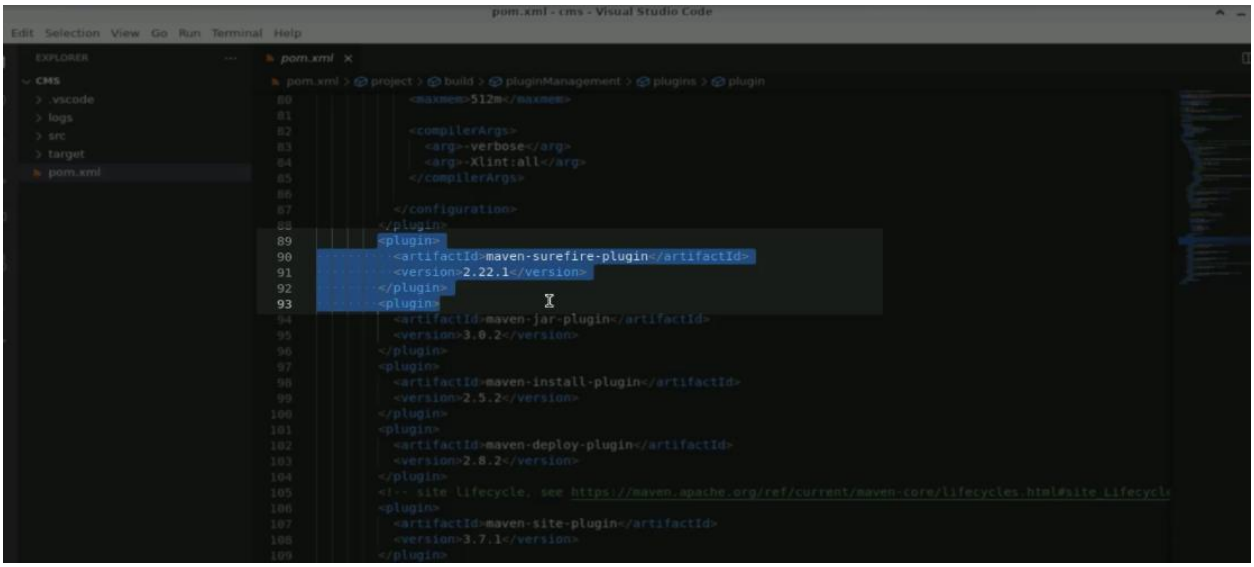


1.2 Under the CMS project, open the **pom.xml** file

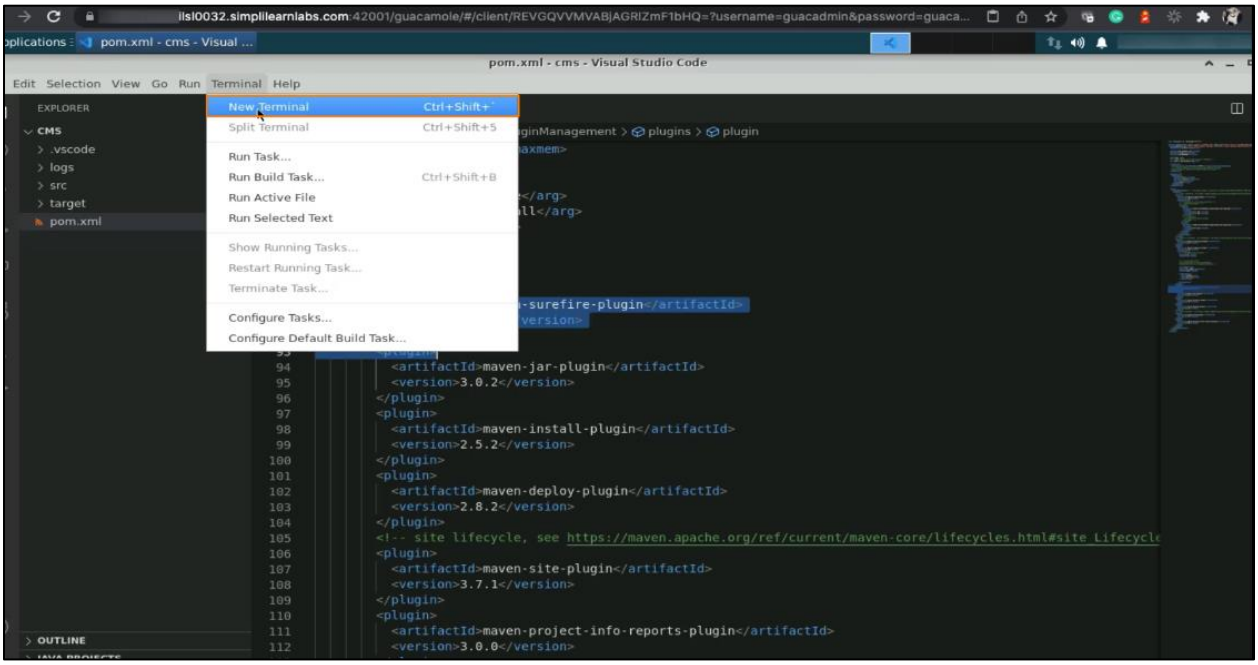


Note: Please refer to the previous demo on how to create the CMS project

You can see the plugin as **maven-surefire-plugin**:

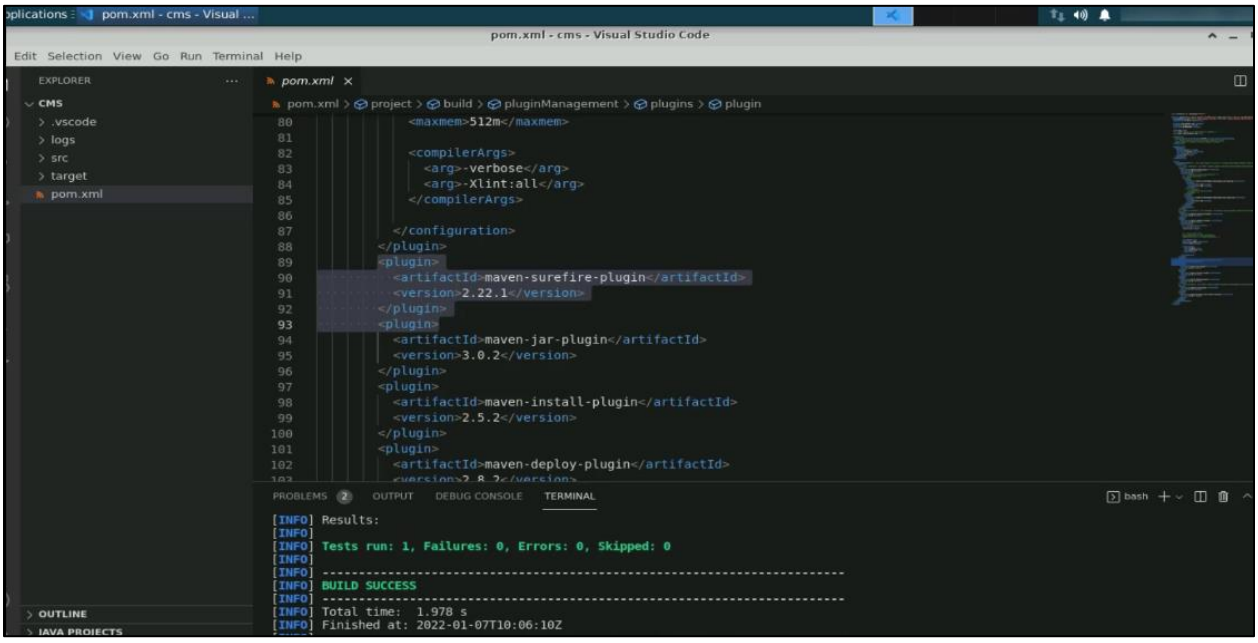


1.3 Go to the **Terminal** tab and click on **New Terminal**



1.4 Run the below command to check Surefire plugin usages and run the test cases:

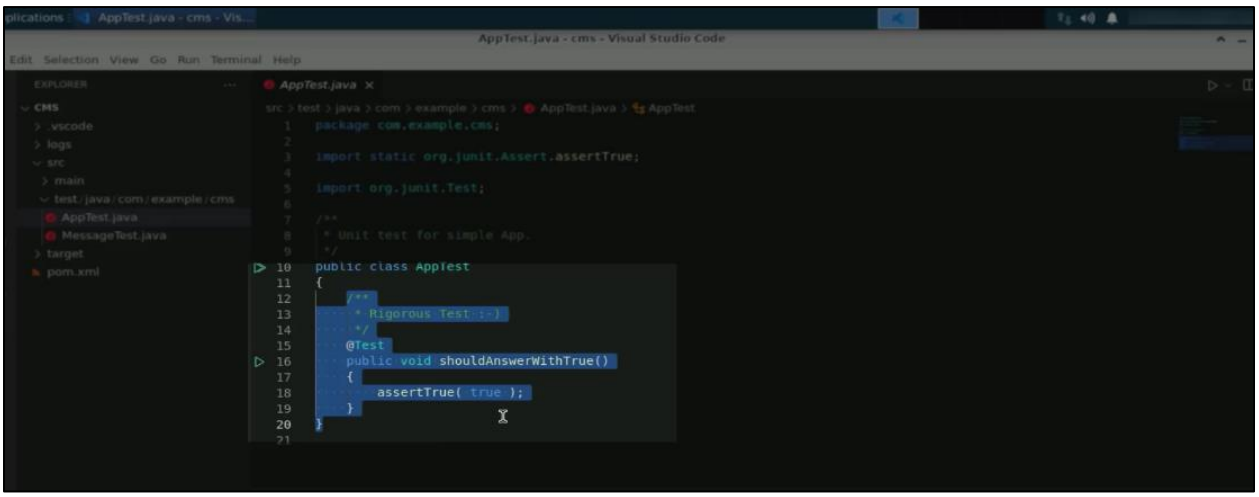
mvn test



The command **mvn test** is used in Maven to execute the unit tests for a Java project. It compiles the source code and runs all the test cases in the project.

Step 2: Define test cases

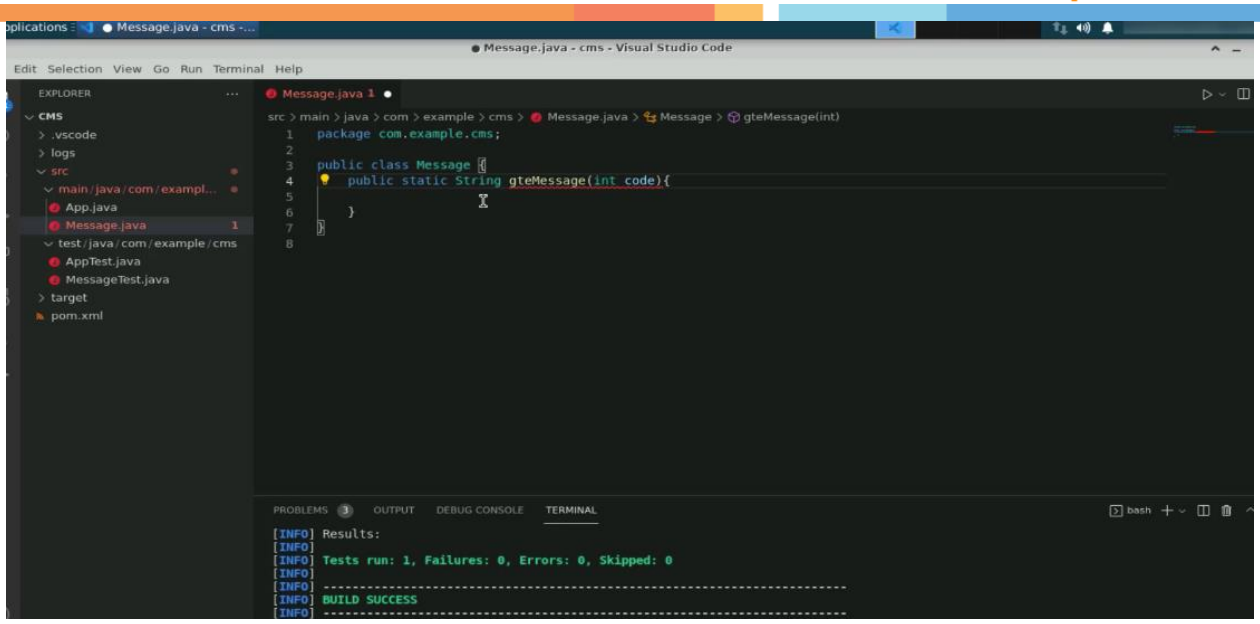
2.1 Under src, open the **AppTest.java** from the **test** directory



2.2 Go to **Message.java** and create a few methods in it

public static String getMessage(int code){

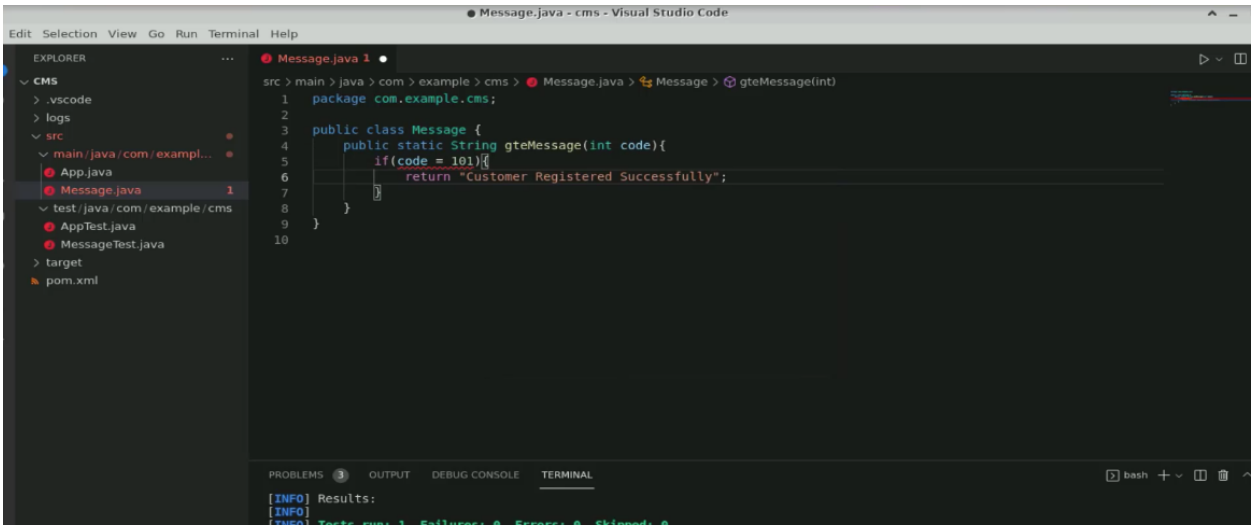
}



The code snippet declares a public static method named **getMessage** which takes an integer parameter named **code**. The method is expected to return a string value.

2.3 Add the if condition and the return statement

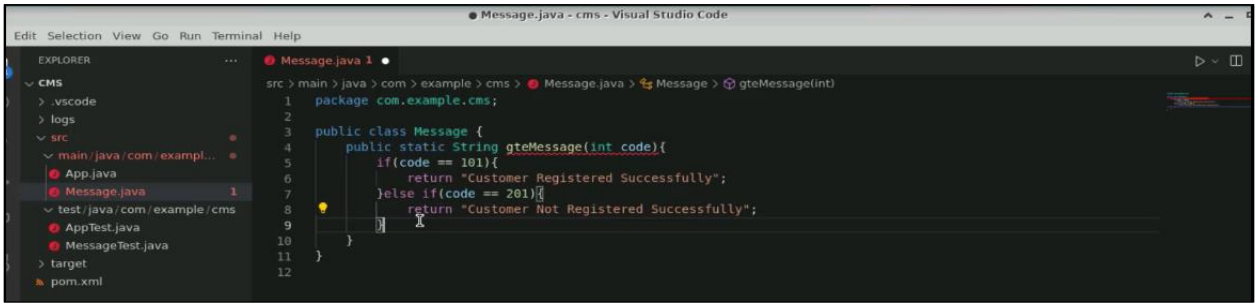
```
if(code 101){
        return "Customer Registered Successfully";
}
```



This code snippet checks if the value of the variable **code** is **101**, and then the function returns the string **Customer Registered Successfully**.

2.4 Add the else if case and return statement

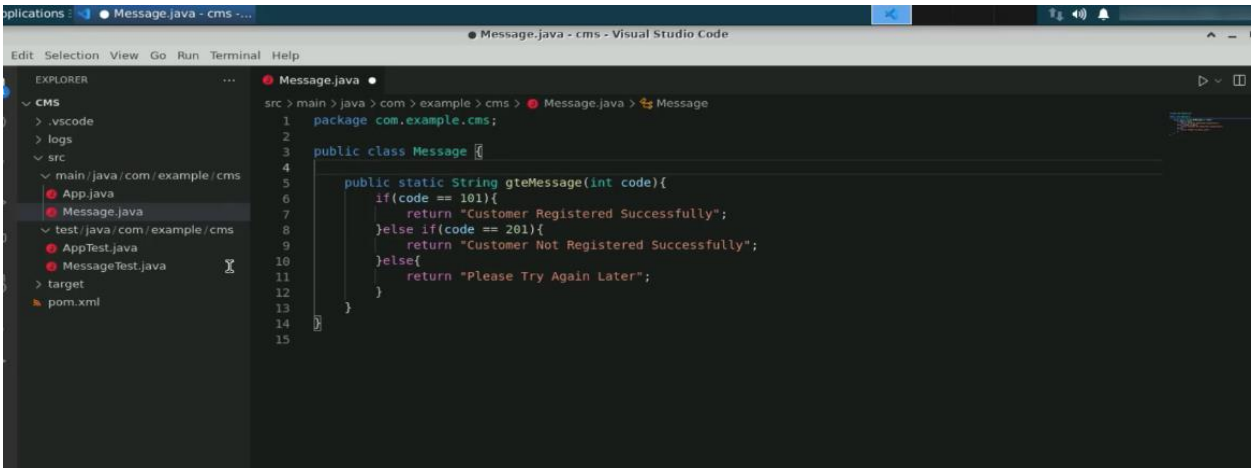
```
else if(code== 201){
        return "Customer Not Registered Successfully";
}
```



These two lines of code use the conditional statement **else if** to check if the variable **code** is **201**, and then the function returns a string indicating that the customer was not registered successfully.

2.5 Add else case and the return statement

```
else{
        return "Please Try Again Later";
}
```

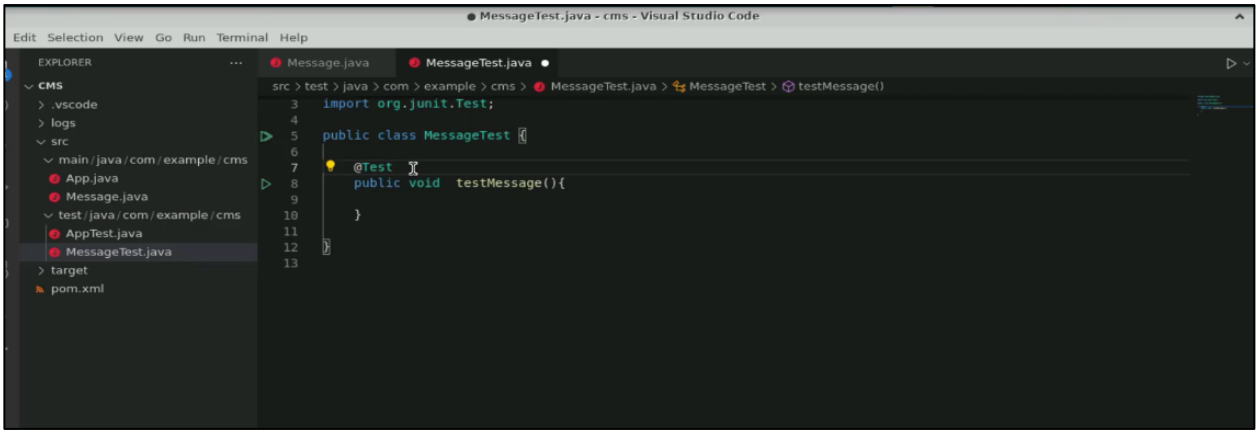


This code snippet uses the **else** keyword to define what should happen if a certain condition is not met in an if statement. In this case, if the condition is not met, the function will return the string **Please Try Again Later**.

Step 3: Test defined test cases

3.1 Create a **test** method in the **MessageTest.java** file to test the cases

```
@Test
public void testMessage(){
}
```



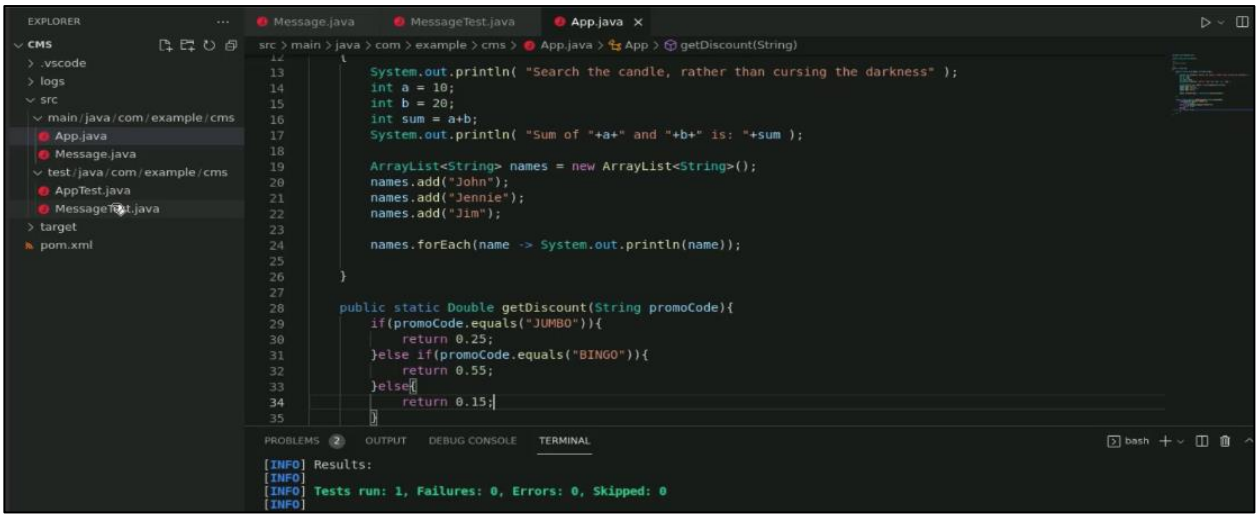
3.2 Define a scenario for the test and add the expected and the actual output for it

assertEquals("Customer Registered Successfully", Message.getMessage(101));

The given code is a test method that verifies if the message returned by the method **getMessage()** with the argument **101** is equal to the expected string **Customer Registered Successfully**. The **assertEquals()** method is used to compare the actual and expected values.

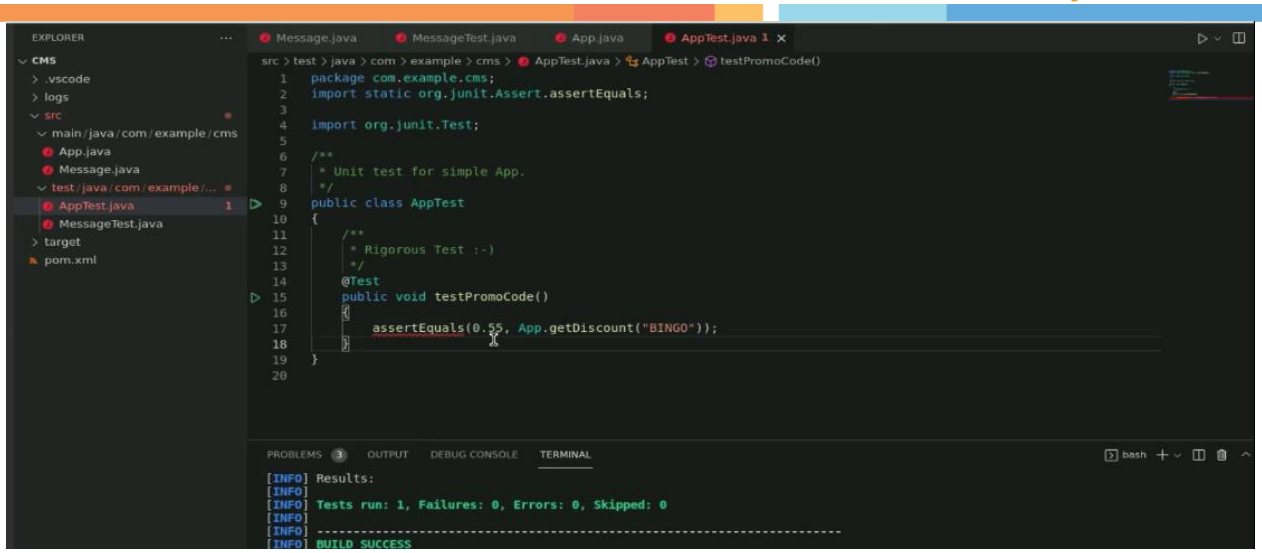
3.3 Add a method in **App.java** to define the discount

```
public static Double getDiscount (String promoCode) {
        if(promoCode.equals("JUMBO")){
                return 0.25;
        }else if (promoCode.equals("BINGO")){
                return 0.55;
        }else{
                return 0.15;
        }
}
```

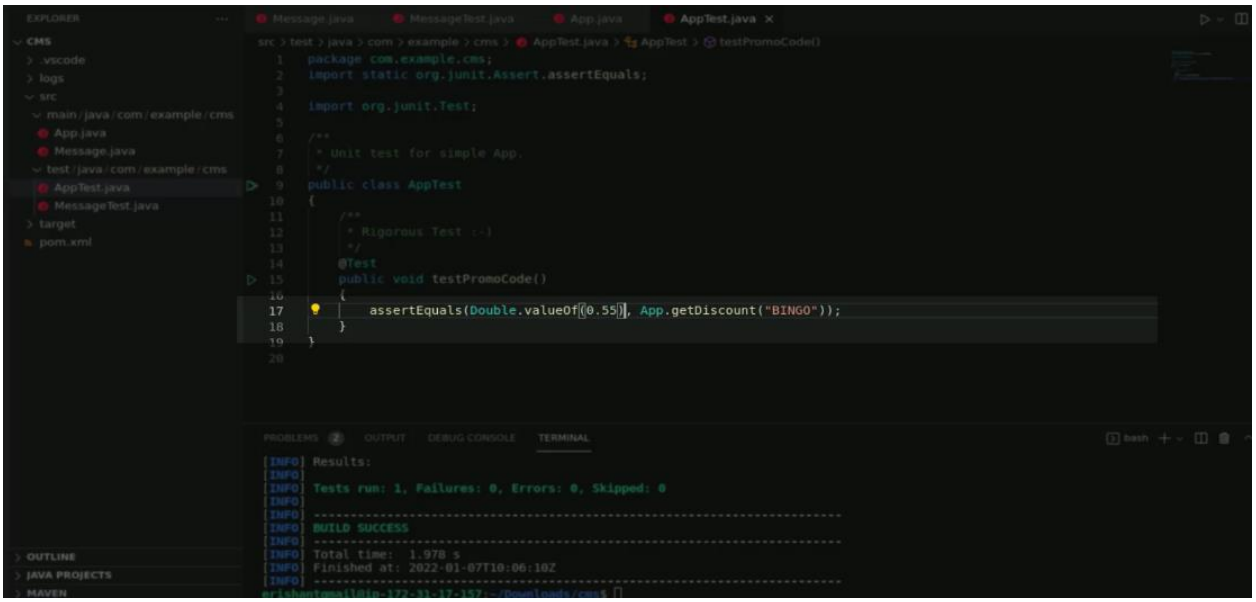


This code defines a static method named **getDiscount** that takes a String argument named **promoCode**. If the **promoCode** is equal to **JUMBO**, the method returns a discount of **25%**; if it is equal to **BINGO**, the method returns a discount of **55%**; and if it is anything else, the method returns a discount of **15%**.

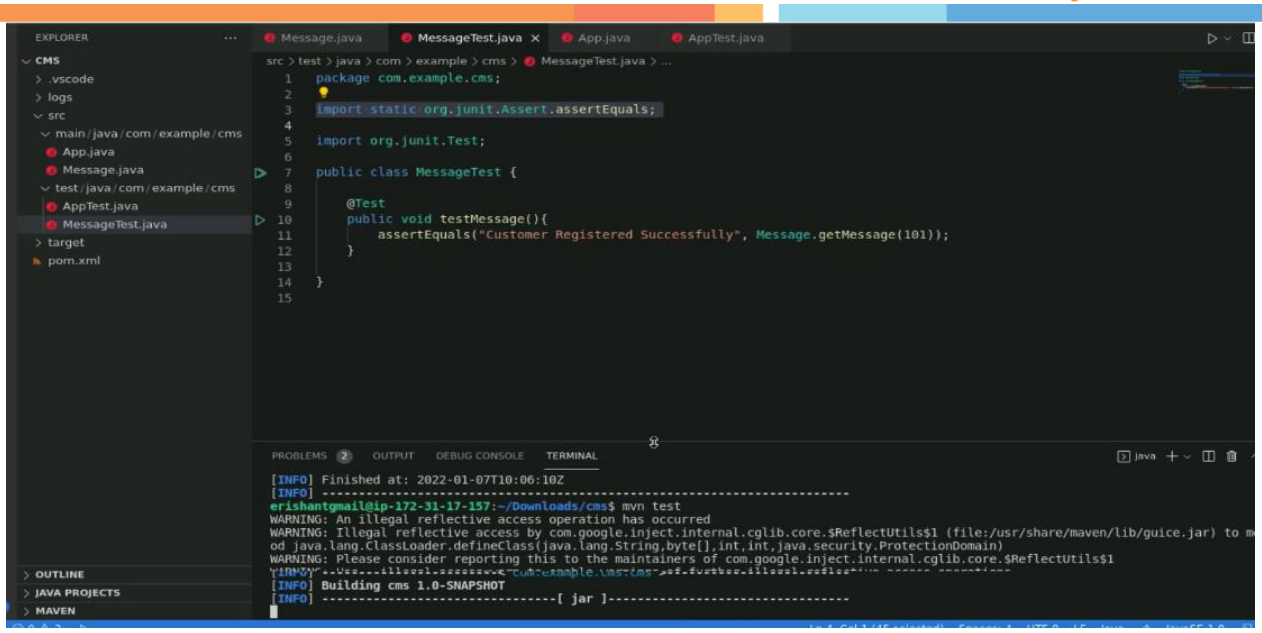
3.4 Write test promo code in the **AppTest.java** file and add the **assertEqual** statement, including the actual and expected output



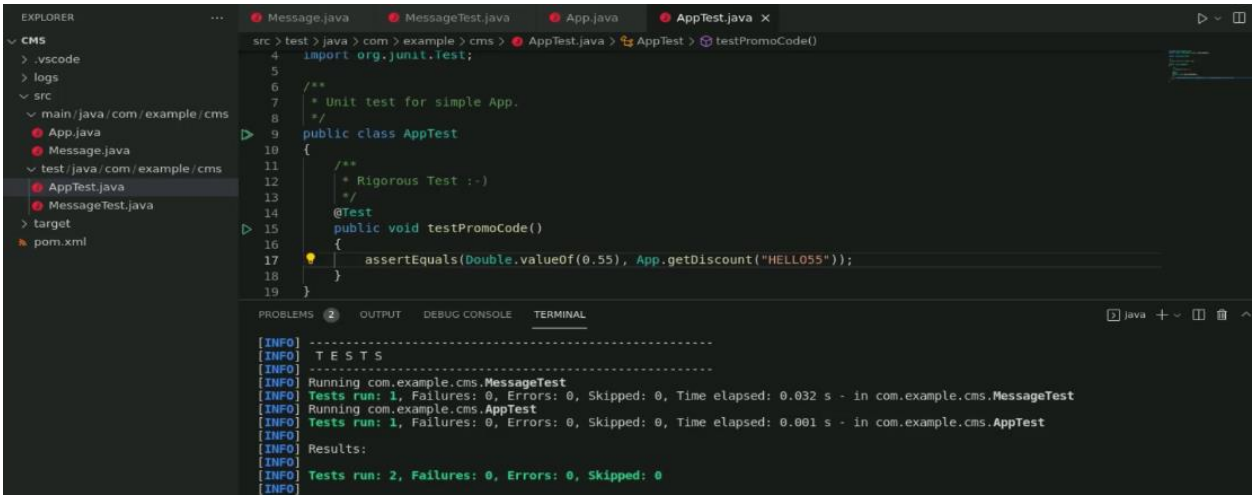
3.5 Change the expected output value to a double data type value



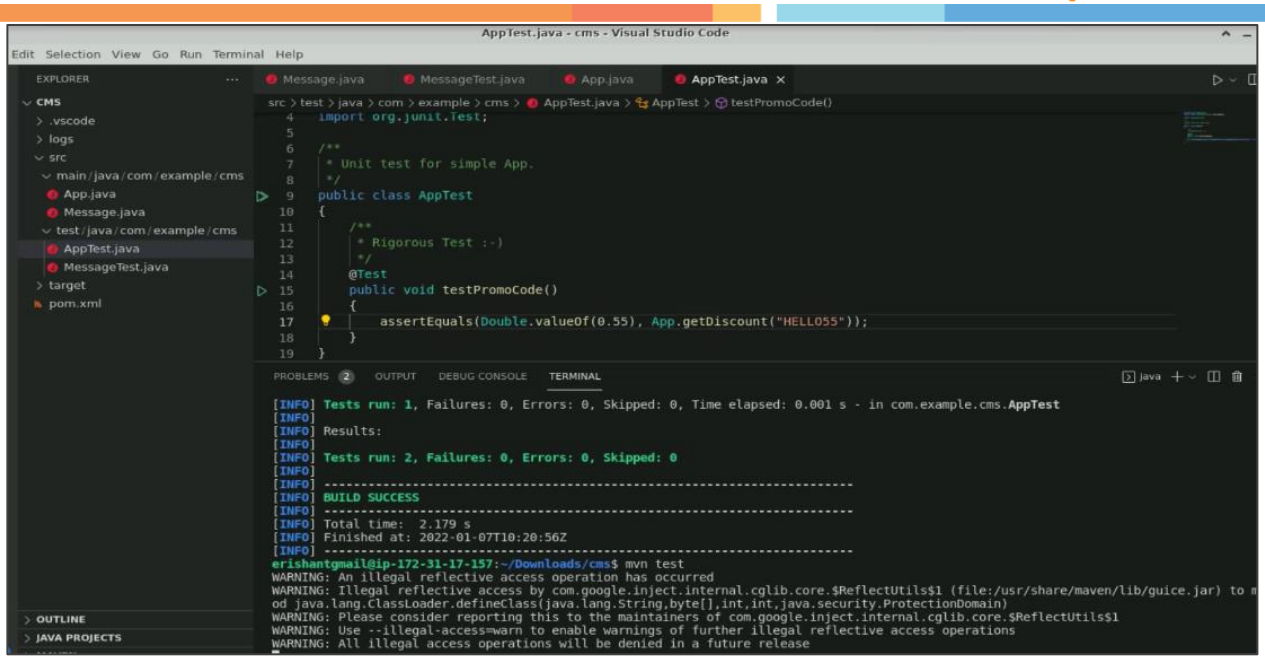
3.6 Use the **mvn test** again to run the test cases



3.7 Change the promo code in the **AppTest.java** file to **Hello55**



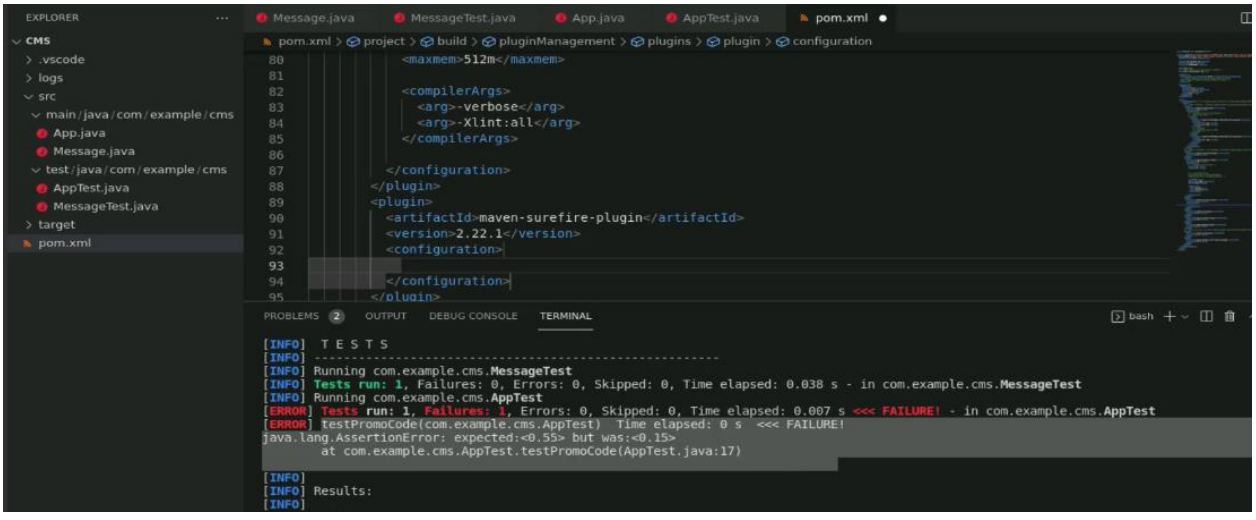
3.8 Run the **mvn test** again to run the test cases



Step 4: Add different configurations for test cases

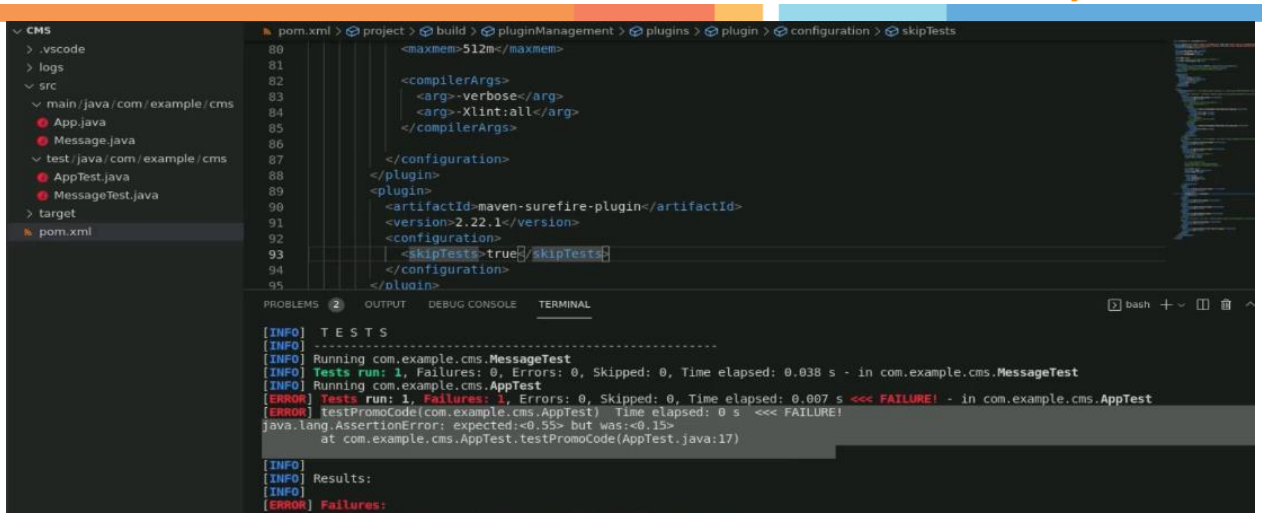
4.1 Add configuration in the **pom.xml** file

<configuration></ configuration >

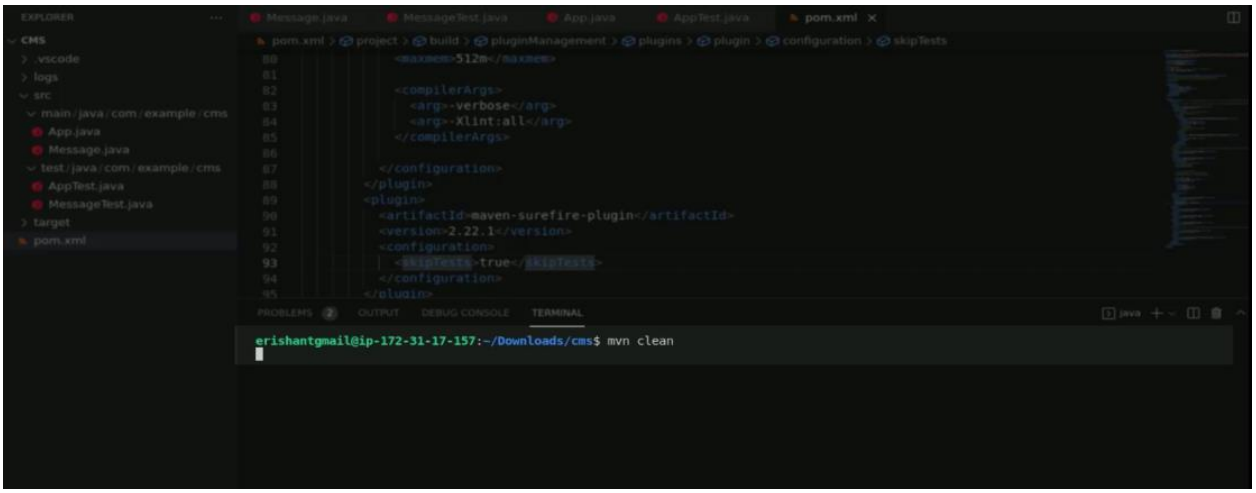


4.2 Under configurations, add **SkipTest** and pass the value as **true**

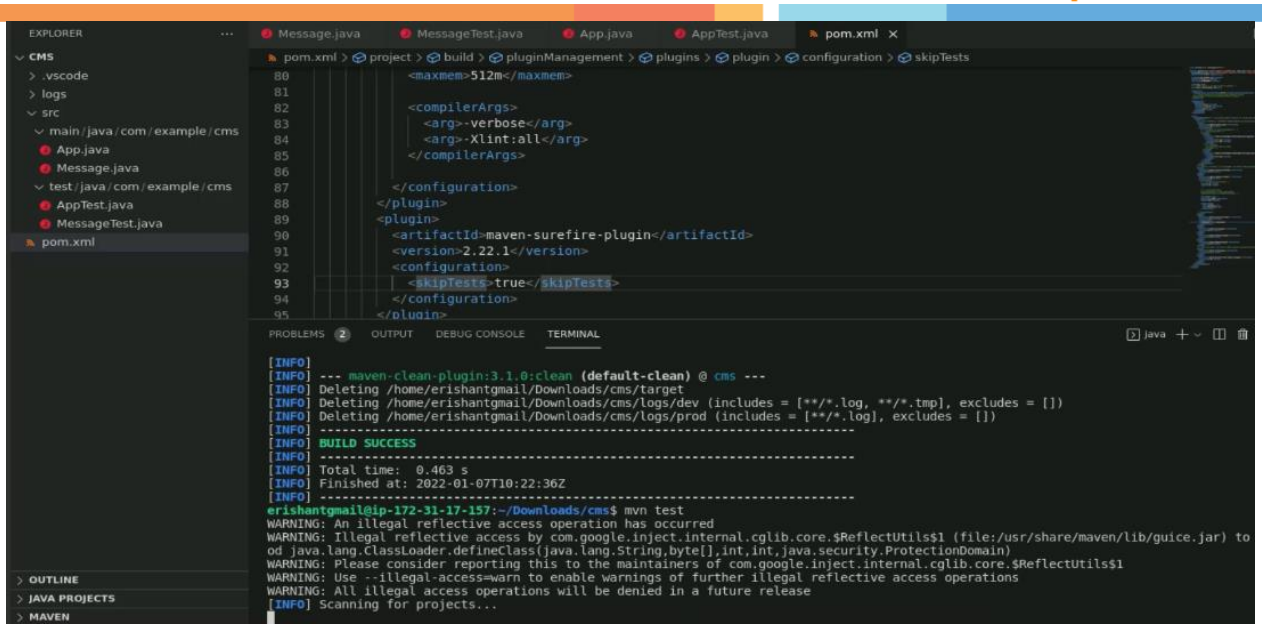
<skipTests>true</skipTests>



4.3 Run the **mvn clean** command to remove the **target** directory

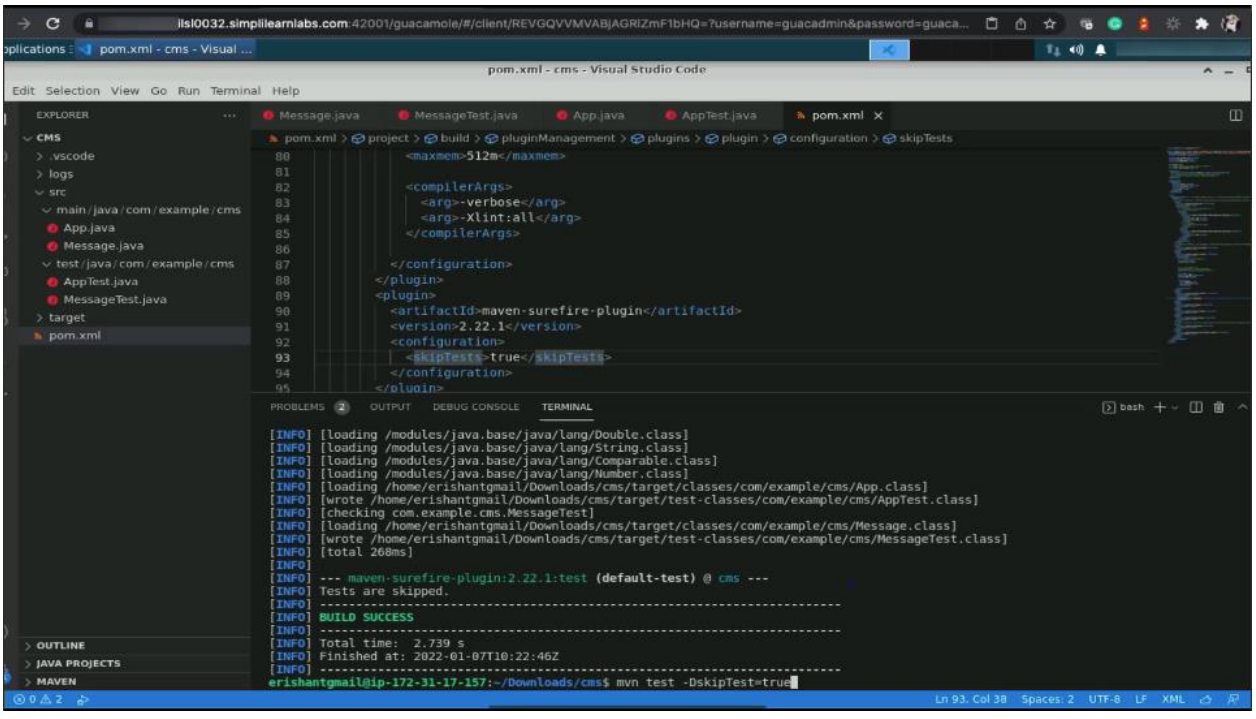


4.4 Type the **mvn test** command and run the test cases again to check if it will work or not



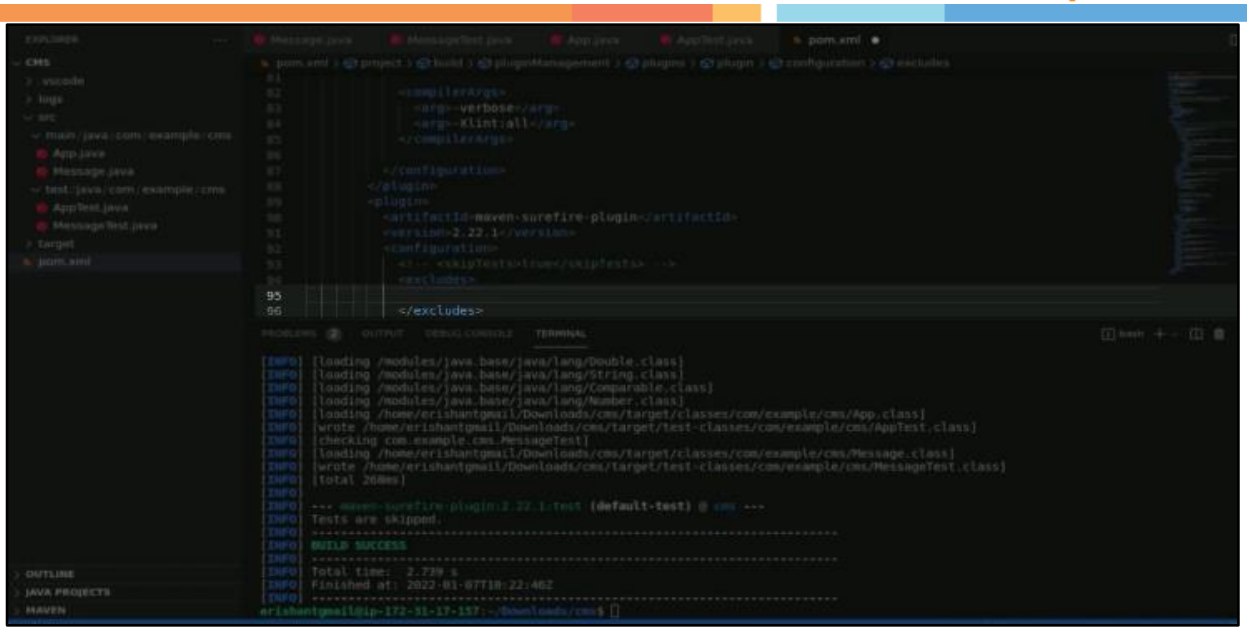
4.5 To directly use the skip test without writing in the **pom.xml** file, use the below command:

mvn test -DskipTest=true



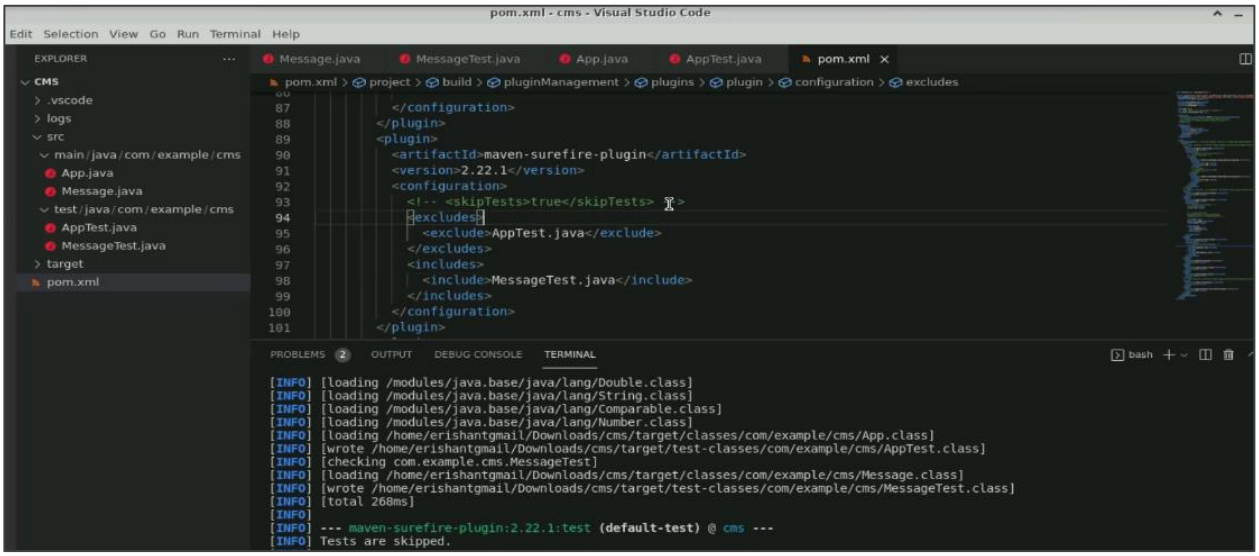
4.6 Add excludes in the configuration

<excludes></excludes>



4.7 Under the excludes tag, add the Java file in which you want to exclude

<exclude>AppTest.java</exclude>

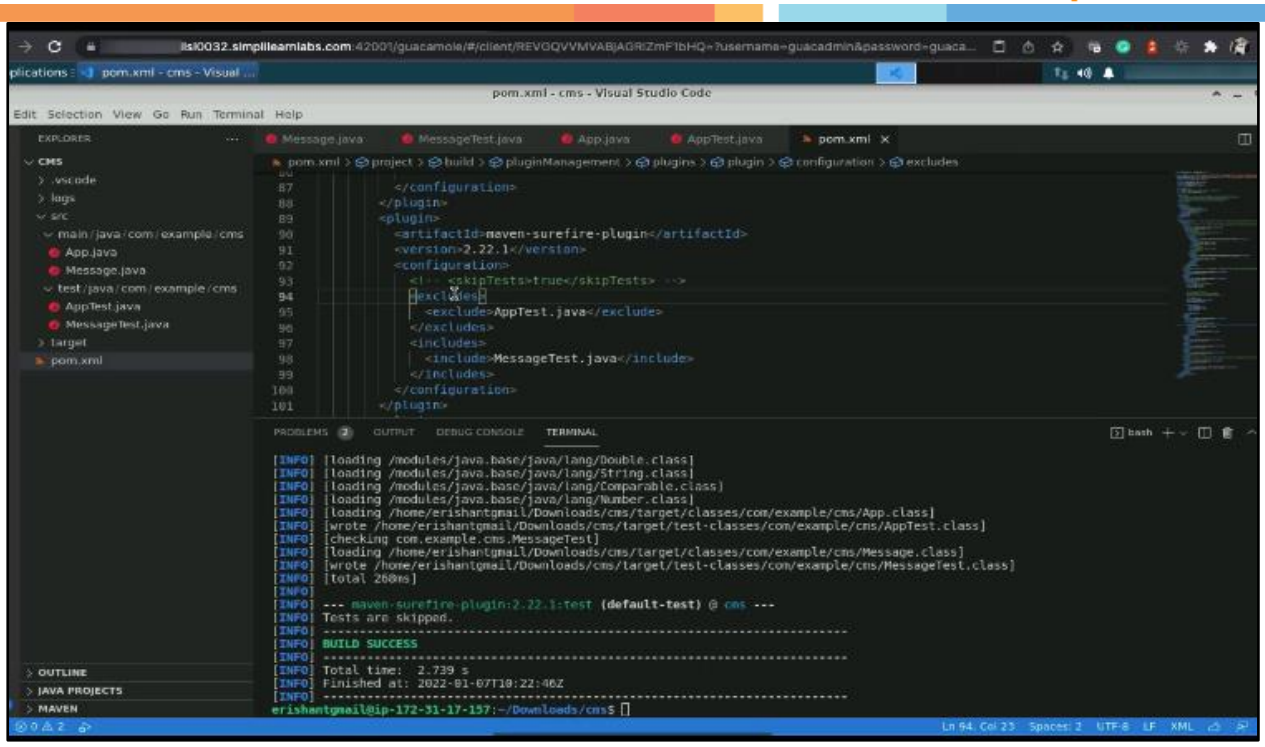


4.8 Create includes and add a file in it

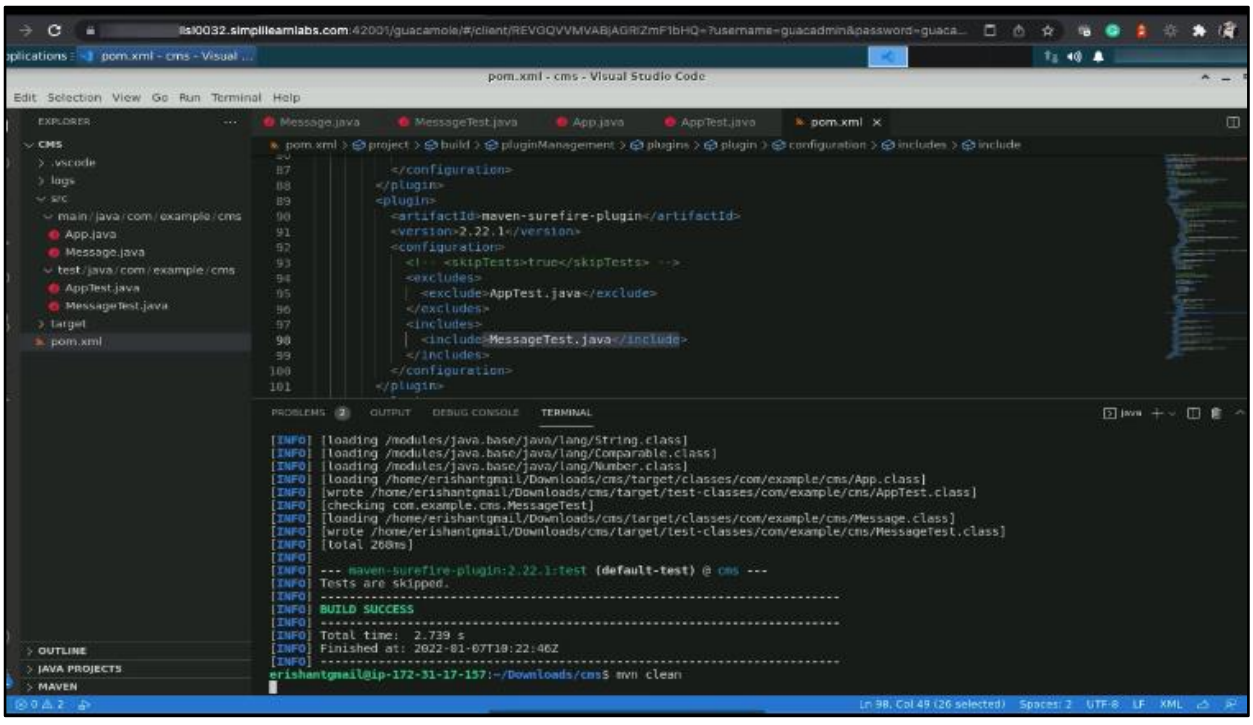
<includes>

<include>MessageTest.java</include>

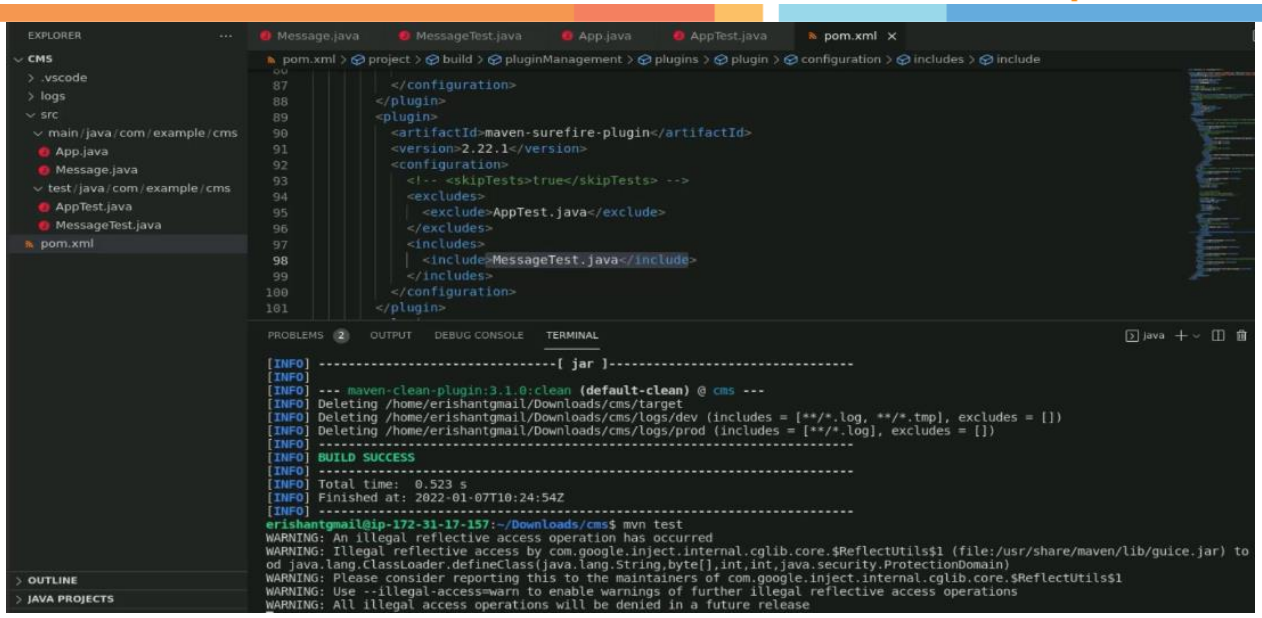
</includes>



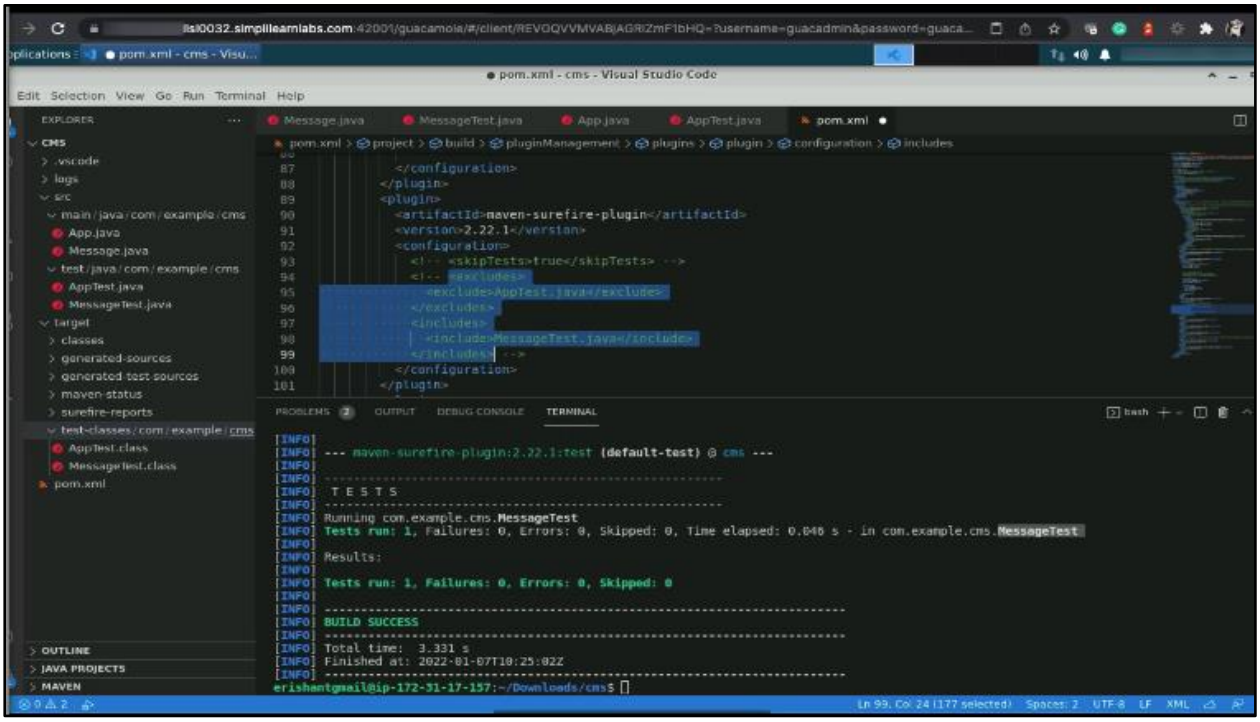
4.9 Write **mvn clean** in terminal



4.10 Run the test using the **mvn test** command

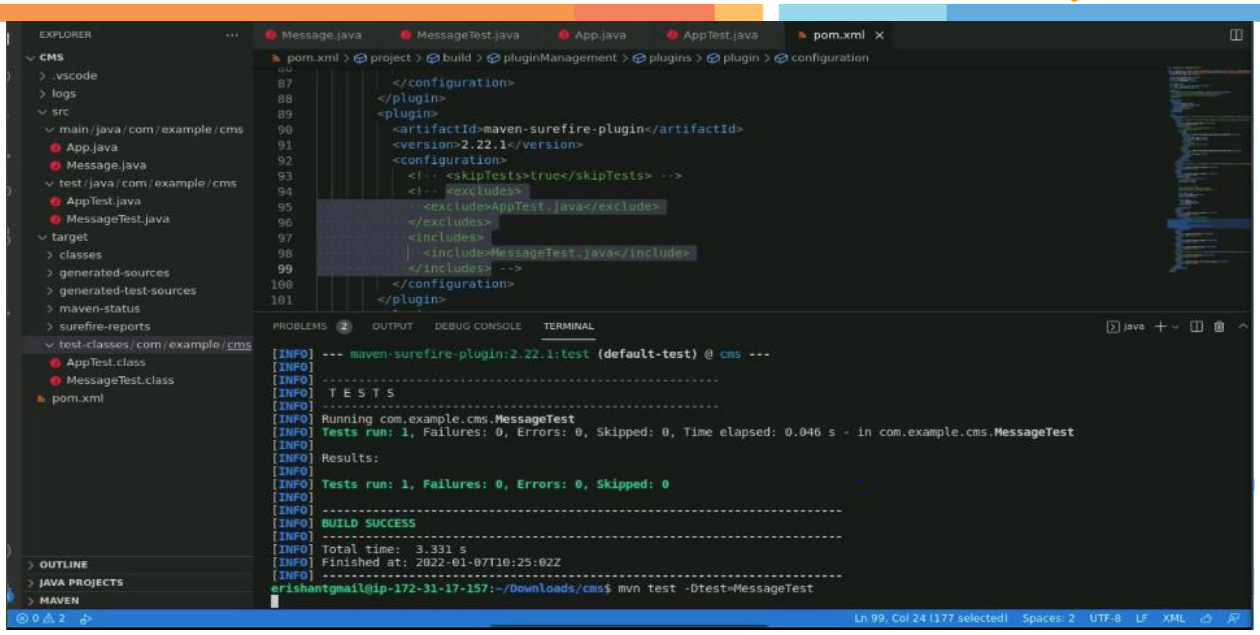


4.11 Comment the code inside the **<configuration>** tag



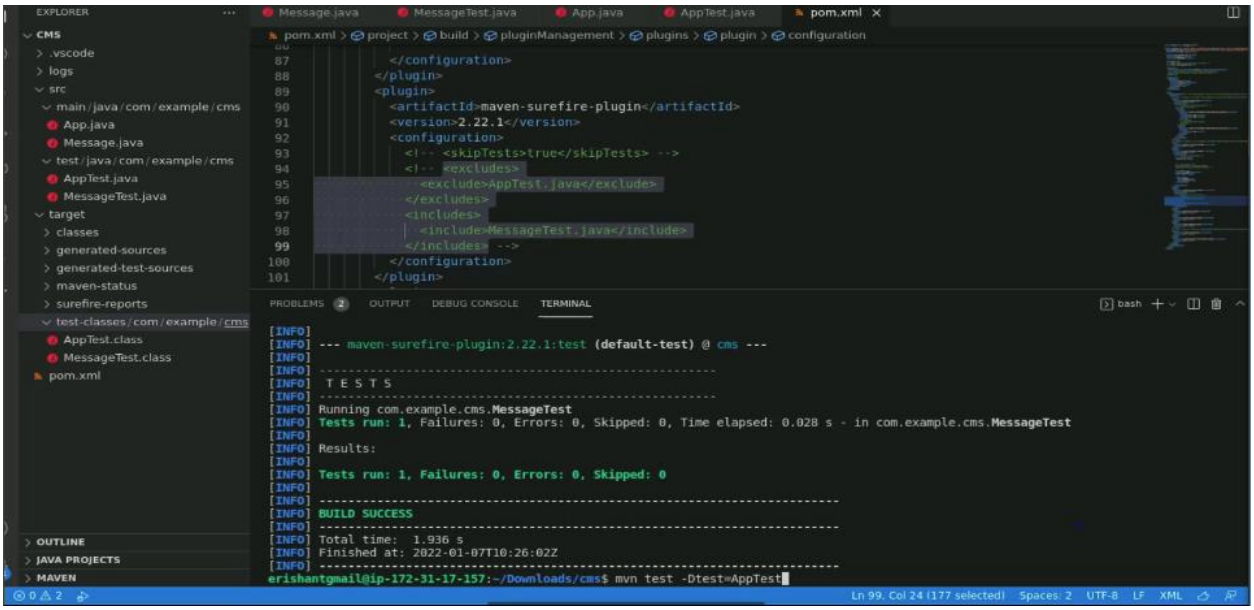
4.12 To run a single test class **MessageTest** through the command prompt, use the below command to execute the **MessageTest**:

mvn test -Dtest=MessageTest



4.13 To run a single test class **MessageTest** through the command prompt, use the below command:

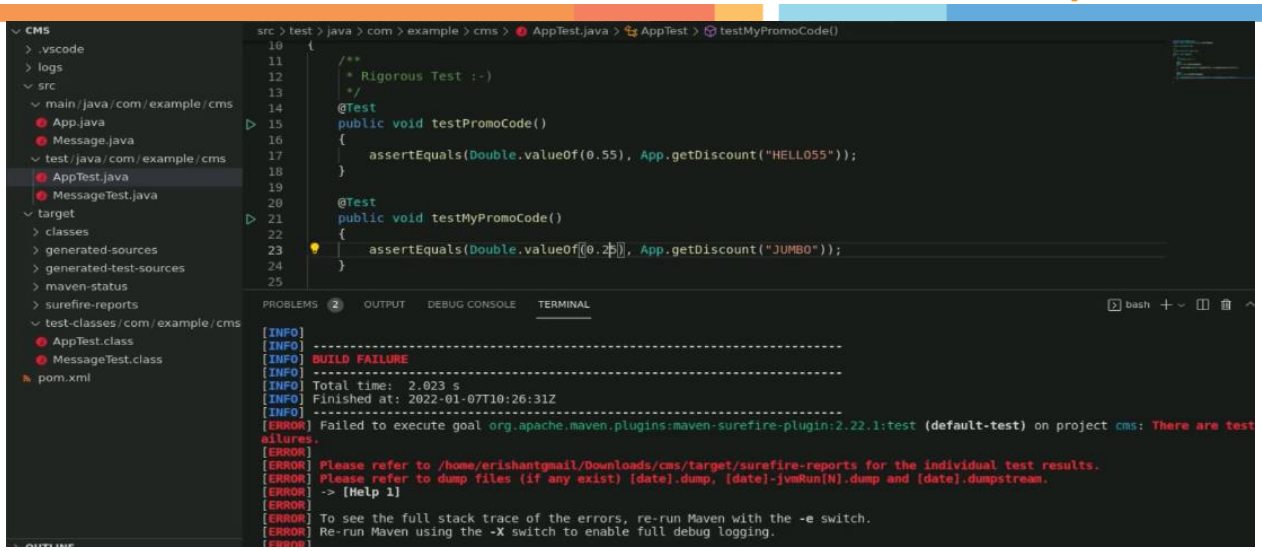
mvn test -Dtest=MessageTest



4.14 Go to the **AppTest.java** file and write one more test case. Copy-paste the above test case and make changes as shown below:

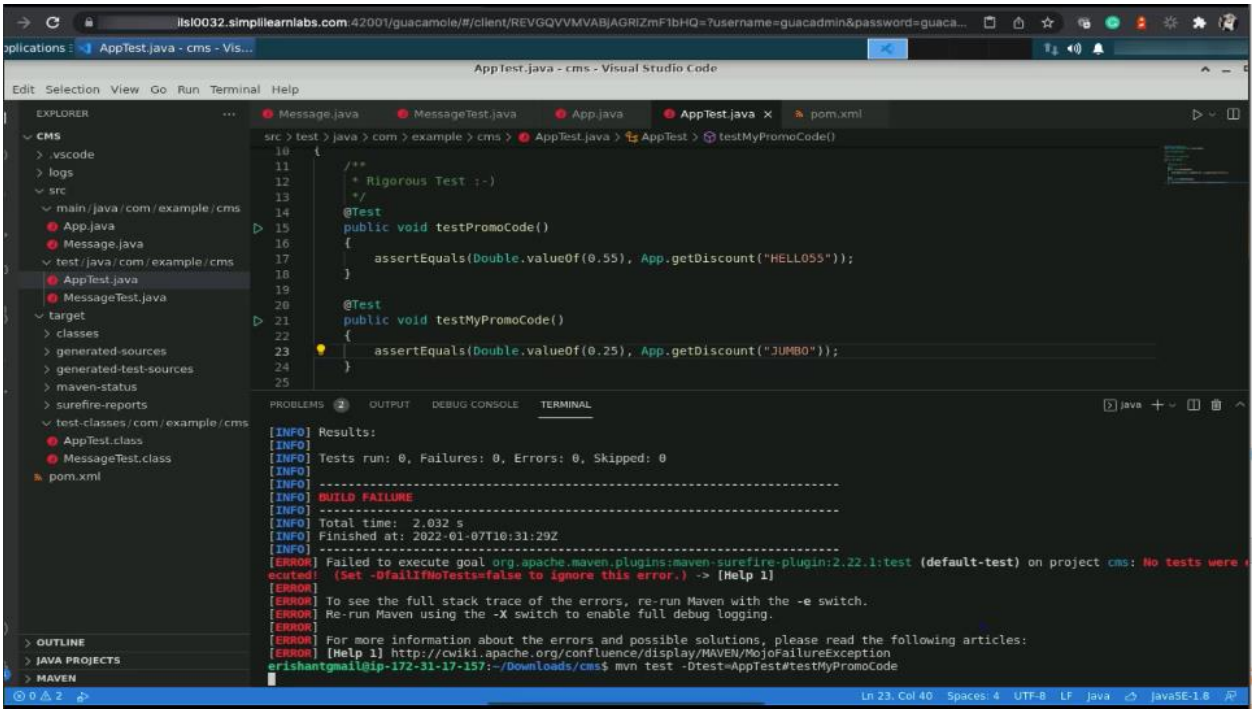
Change name to **testMyPromoCode**

Change value to **0.25** and discount coupon name to **JUMBO**



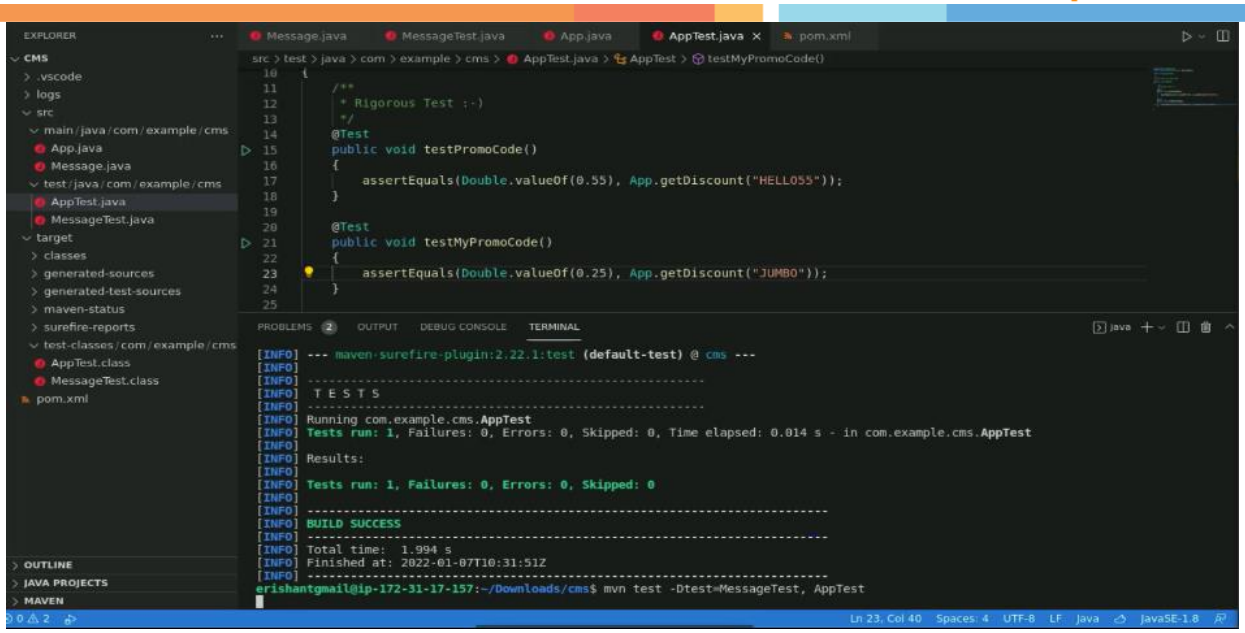
4.15 Use **mvn test** with method name in it, that is, **-DTest,** and specify the promo code

mvn test -Dtest=AppTest#testMyPromoCode



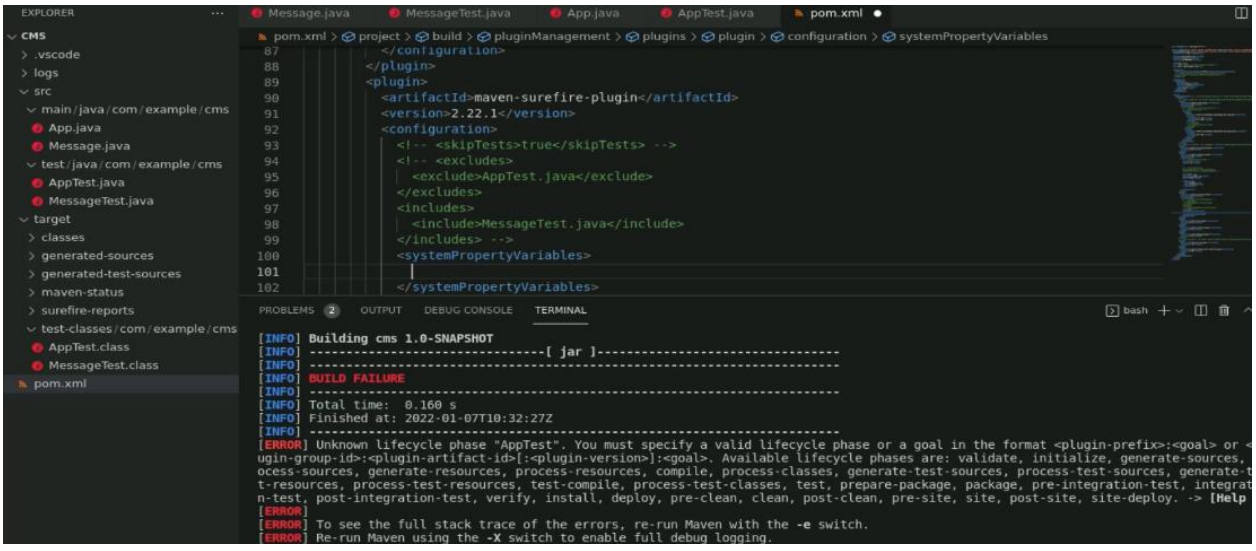
4.16 To test multiple classes at once, add multiple test names separated by a comma

mvn test -Dtest=MessageTest, AppTest

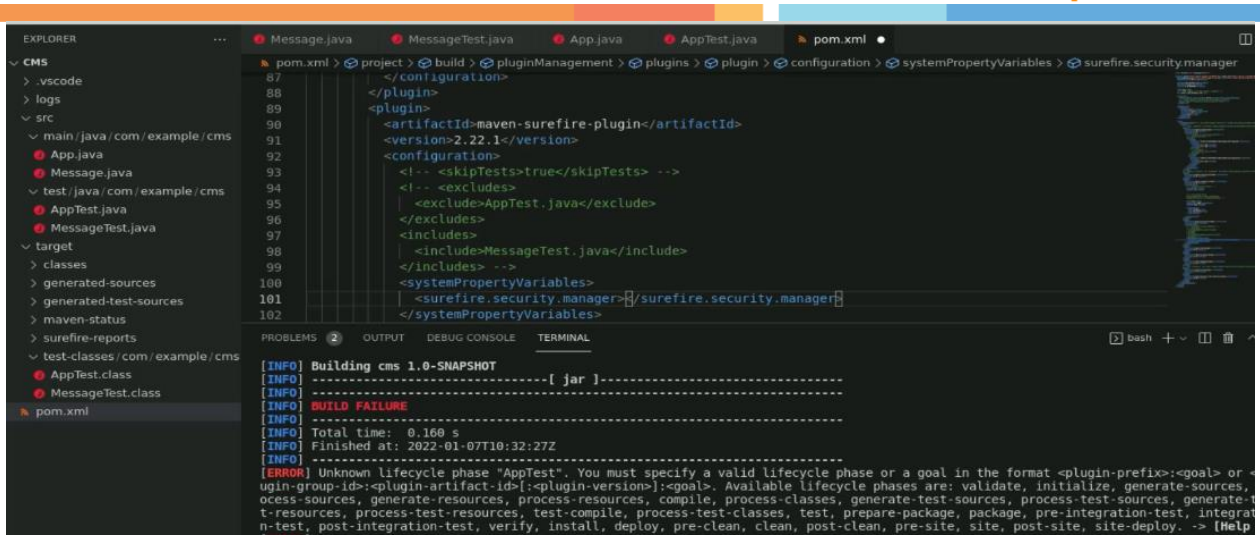


Step 5: Work with the security manager

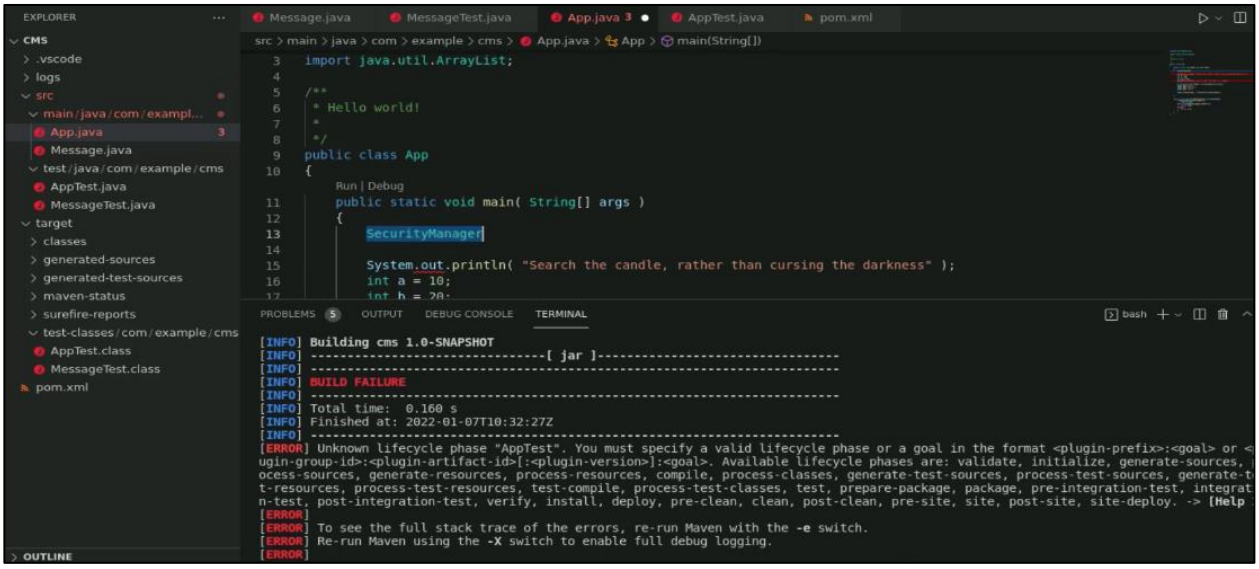
5.1 In the **pom.xml** file, add system property variables



5.2 Use the **surefire.security.manager** tag to work with older versions of unit tests:
<surefire.security.manager></surefire.security.manager>

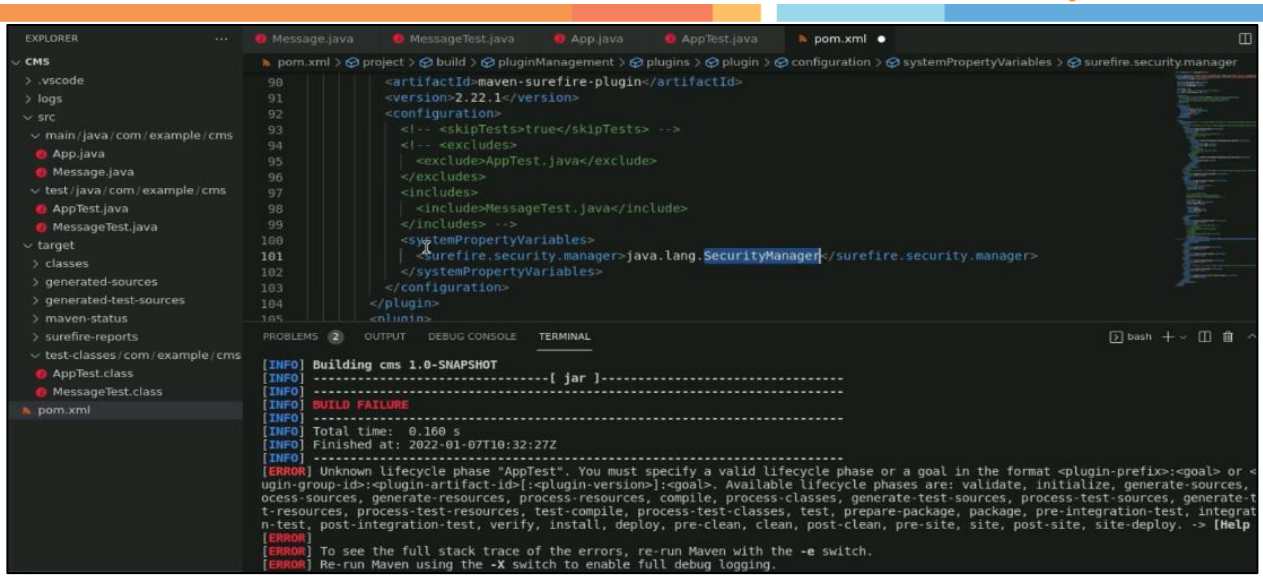


5.3 Add a security manager in the **App.java** file



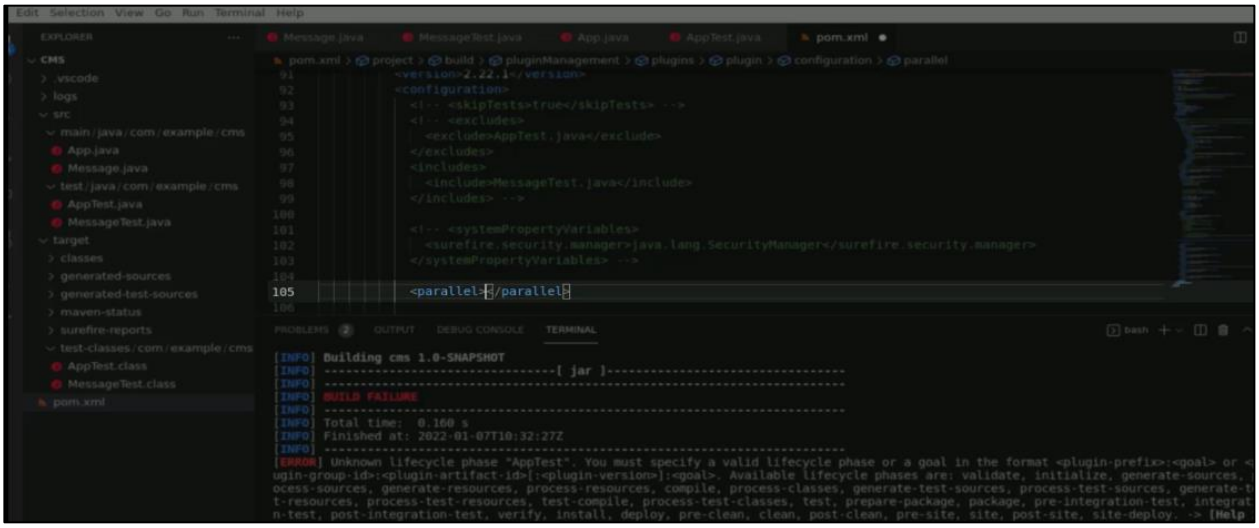
5.4 Go to the **pom.xml** file and configure the API for JUnit 3, which is:

java.lang.SecurityManager



5.5 Use the **parallel** tag in configuration to execute the tests in parallel and have unlimited threads to run the configuration

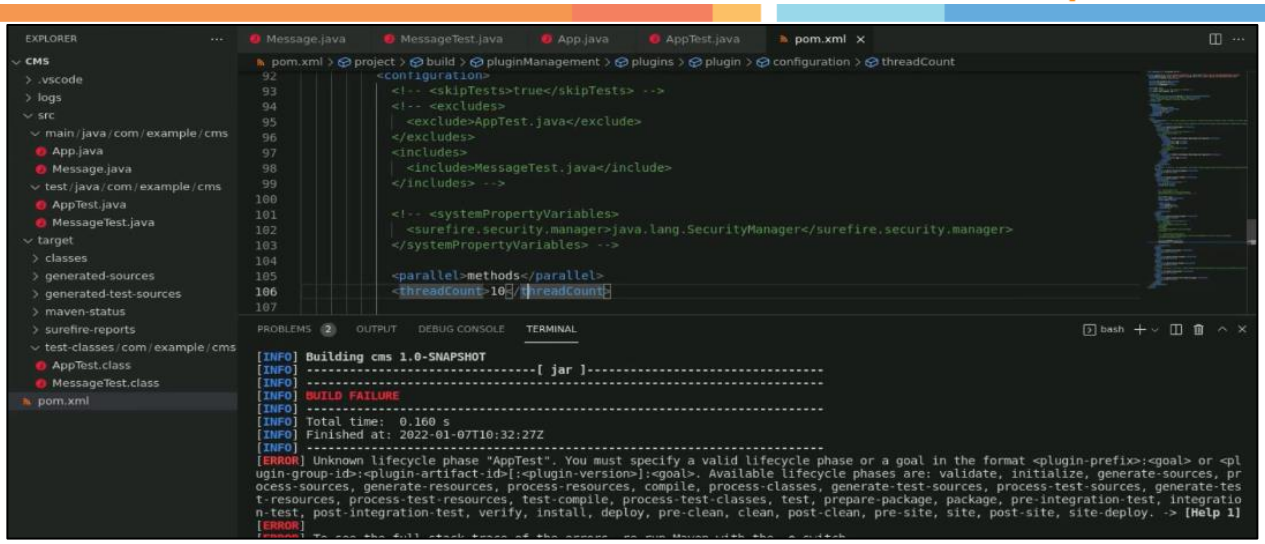
<parallel></parallel>



5.6 Add **methods** in the parallel tag and provide value for the **threadCount** as **10** to execute ten different tests

<parallel>methods</parallel>

<threadCount>10</threadCount>



By following these steps, you have successfully configured the Surefire plugin for various roles. You have achieved the goal of running unit tests for your application using the test goal.