**Shaikh Zaid Muddaris Husain 23dco06 batch-3**

**Experiment No 09**

**Aim:**

To write test cases for white box testing.

**Theory:**

White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security. In white box testing, code is visible to testers so it is also called Clear box testing, open box testing, transparent box testing, Code-based testing and Glass box testing. White box testing in software engineering is based on the inner workings of an application and revolves around internal testing. Advantages of White Box Testing:

* Code optimization by finding hidden errors.
* White box tests cases can be easily automated.
* Testing is more thorough as all code paths are usually covered.
* Testing can start early in SDLC even if GUI is not available.

Disadvantages of White Box Testing:

* White box testing can be quite complex and expensive.
* Developers who usually execute white box test cases detest it.
* The white box testing by developers is not detailed can lead to production errors.
* White box testing requires professional resources, with a detailed understanding of programming and implementation.
* White-box testing is time-consuming, bigger programming applications take the time to test fully.

In this technique, all the possible combinations of the possible outcomes of conditions are tested at least once.

Let‟s consider the following example:

READ X, Y

IF(X == 0 || Y == 0)

PRINT “0”

#TC1: X = 0, Y = 0

#TC2: X = 0, Y = 5

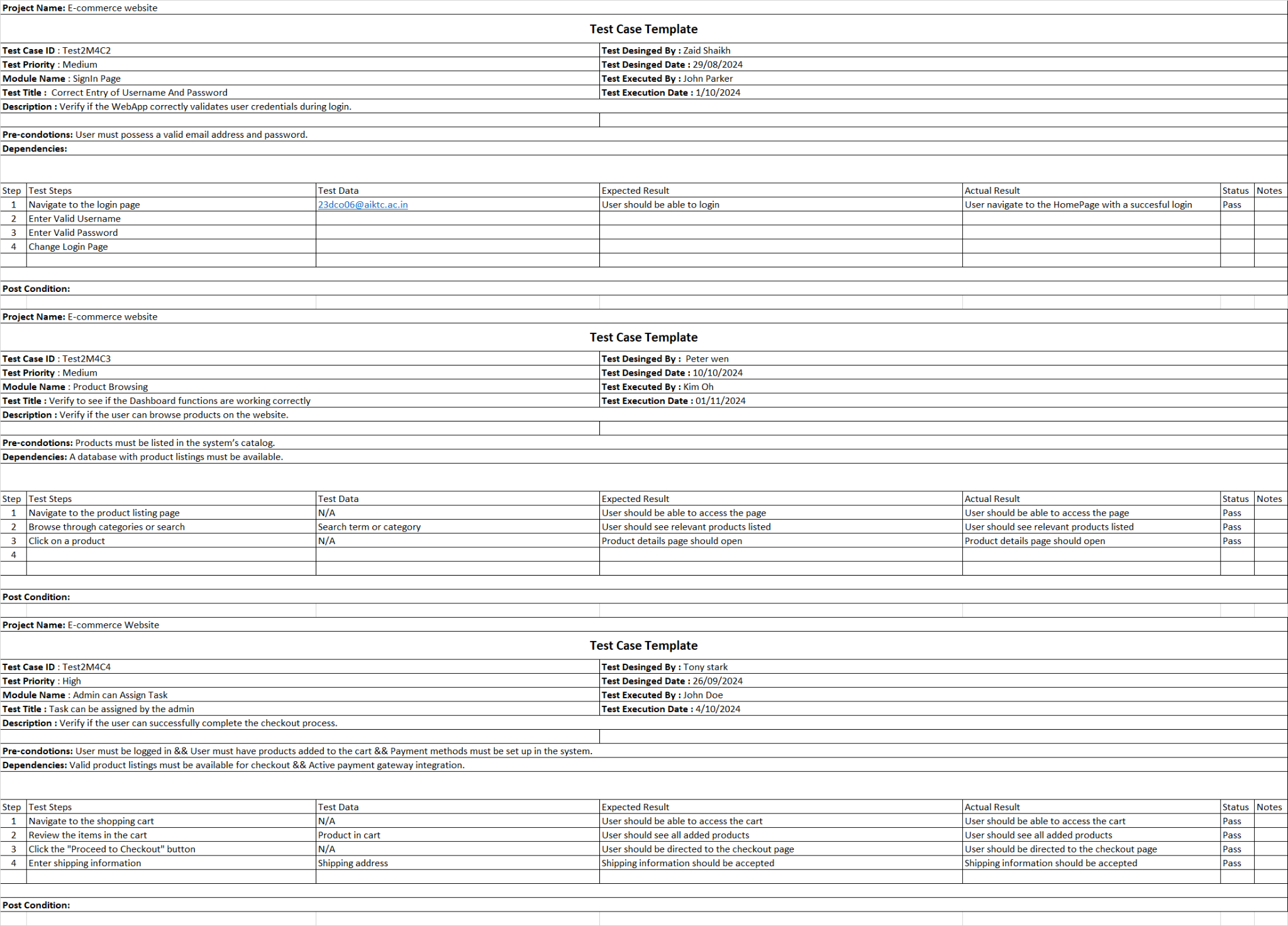
#TC3: X = 55, Y = 0

#TC4: X = 55, Y = 5

Hence, four test cases required for two individual conditions. Similarly, if there are n conditions then 2n test cases would be required.

**Procedure:**

* 1. Create at least 10 test cases for your selected project in the prescribed format:
* Test case ID: Unique ID is required for each test case. Follow some conventions to indicate the types of the test. For Example, „TC\_UI\_1‟ indicating „user interfaces test case #1‟.



* Test priority (Low/Medium/High): This is very useful during test execution. Test priorities for business rules and functional test cases can be medium or higher, whereas minor user interface cases can be of a low priority. Testing priorities should always be set by the reviewer.
* Module Name: Mention the name of the main module or the sub-module.
* Test Designed By Name of the Tester.
* Test Designed Date: Date when it was written.
* Test Executed By: Name of the Tester who executed this test.
* Test Execution Date: Date when the test was executed.
* Test Title/Name: Test case title. For example, verify the login page with a valid username and password.
* Test Summary/Description: Describe the test objective in brief.
* Pre-conditions: Any prerequisite that must be fulfilled before the execution of this test case. List all the pre-conditions in order to execute this test case successfully.
* Dependencies: Mention any dependencies on other test cases or test requirements. Test Steps: List all the test execution steps in detail. Write test steps in the order in which they should be executed.
* Expected Result: What should be the system output after test execution? Describe the expected result in detail including the message/error that should be displayed on the screen.
* Post-condition: What should be the state of the system after executing this test case?
* Actual result: The actual test result should be filled after test execution. Describe the system behavior after test execution.
* Status (Pass/Fail): If the actual result is not as per the expected result, then mark this test as failed. Otherwise, update it as passed.
* Notes/Comments/Questions: If there are any special conditions to support the above fields, which can‟t be described above or if there are any questions related to expected or actual results then mention them here.

**Conclusion:**

This technique ensures that all pathways, conditions, and loops within the program are executed and functioning as expected. By focusing on the internal workings, it helps identify issues like code inefficiencies, hidden errors, and security vulnerabilities early in the development process. Ultimately, White Box Testing contributes to delivering robust, high-quality software by offering a deep, systematic evaluation of the codebase.