

Vellore – 632014, Tamil Nadu, India

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**Winter Semester 2024-2025**

**Continuous Assessment -4**

**Exercise 9**

**SLOT :** L41+L42

**Programme Name : B.Tech.**

**Course Name & code : Software Engineering Lab & BCSE301P**

**Class Number :** VL2024250502644

**Faculty Name : Dr. Mehfooza M**

**04/03/2025 Due date: 10/03/2025**

9) Design and demonstration of test cases. Functional Testing and Non- Functional Testing (using any open source tools)

**1. Test Cases**

**Test cases help define the expected behavior of the system under various conditions.**

**Steps to Create Test Cases:**

1. **Identify Features to Test:**
   * **List the features of the Student Management System to test, such as:**
     + **User login**
     + **Student registration**
     + **Course enrollment**
     + **Faculty management**
     + **Payment processing**
     + **Attendance tracking**
2. **Define Test Case Template: A typical test case should include:**
   * **Test Case ID: Unique identifier for the test case.**
   * **Test Case Description: A short description of the test.**
   * **Test Steps: Steps to perform the test.**
   * **Test Data: Input data for the test case.**
   * **Expected Result: Expected output or behavior.**
   * **Actual Result: Actual output (filled during test execution).**
   * **Pass/Fail: The result of the test.**

**Example test case for user login:**

**Test Case ID: TC\_Login\_01**

**Test Case Description: Verify user login with valid credentials.**

**Test Steps:**

**1. Navigate to the login page.**

**2. Enter valid username and password.**

**3. Click on the "Login" button.**

**Test Data:**

**Username: student1**

**Password: password123**

**Expected Result:**

**User should be logged in successfully and redirected to the dashboard.**

**2. Functional Testing with PyTest**

**Functional testing ensures that the features of the system work as expected.**

**Steps to Perform Functional Testing:**

1. **Install PyTest:**
   * **Install PyTest by running:**

**pip install pytest**

1. **Create Test Directory:**
   * **Create a directory called tests to store all your test cases.**

**mkdir tests**

1. **Write Functional Test Cases: Create Python files in the tests directory to write functional test cases.**

**Example test for student login functionality (test\_login.py):**

**import pytest**

**from student\_management\_system import login # Import the function to test**

**def test\_login\_valid\_credentials():**

**username = "student1"**

**password = "password123"**

**result = login(username, password)**

**assert result == "Login successful"**

**def test\_login\_invalid\_credentials():**

**username = "invalid\_user"**

**password = "wrong\_password"**

**result = login(username, password)**

**assert result == "Login failed"**

1. **Create Additional Test Cases for Other Features:**
   * **Example for course registration functionality:**

**def test\_course\_registration():**

**student\_id = 1**

**course\_id = 101**

**result = register\_student\_for\_course(student\_id, course\_id)**

**assert result == "Registration successful"**

1. **Run Tests:**
   * **Run your test cases using the following command:**

**pytest tests/**

* + **PyTest will discover all the tests and execute them. The results will be shown in the terminal with "PASSED" or "FAILED" status for each test case.**

1. **Check for Errors:**
   * **If any test fails, review the error messages in the terminal and fix the issues in the code or the test.**

**3. Non-Functional Testing**

**Non-functional testing validates aspects of the system like performance, security, and scalability.**

**Types of Non-Functional Testing:**

1. **Performance Testing:**
   * **Objective: To evaluate how the system performs under various load conditions.**
   * **Tool: PyTest can be used with other libraries like Locust or JMeter for performance testing, but we'll focus on PyTest usage here.**
   * **Steps:**
     1. **Install Locust:**

**pip install locust**

* + 1. **Create a locustfile.py to simulate load testing. For example, testing the student registration functionality:**

**from locust import HttpUser, task, between**

**class StudentRegistrationTest(HttpUser):**

**wait\_time = between(1, 2)**

**@task**

**def register\_student(self):**

**self.client.post("/register", json={**

**"student\_name": "John Doe",**

**"student\_id": "12345",**

**"course": "Math 101"**

**})**

* + 1. **Run the load test:**

**locust -f locustfile.py**

* + 1. **Open a browser and go to http://localhost:8089 to start the load test and simulate multiple users registering at the same time.**
    2. **Analyze the results (response times, errors, etc.).**

1. **Security Testing:**
   * **Objective: Test the application for vulnerabilities like SQL injection, XSS, etc.**
   * **Tool: PyTest itself cannot perform security tests directly, but we can use it to automate basic security checks, like form input validation.**
   * **Example test for SQL injection vulnerability:**

**def test\_sql\_injection():**

**malicious\_input = "' OR 1=1 --"**

**result = login(malicious\_input, malicious\_input)**

**assert result == "Login failed"**

1. **Usability Testing:**
   * **Objective: Ensure the application is easy to use, intuitive, and meets user expectations.**
   * **Tool: While PyTest is primarily for functional and non-functional testing, usability testing typically involves manual testing with feedback from users.**
   * **Steps:**
     1. **Observe users interacting with the system (e.g., students registering, browsing courses).**
     2. **Collect feedback on ease of navigation and usability.**
     3. **Identify areas for improvement in the user interface (UI).**
2. **Compatibility Testing:**
   * **Objective: Verify the system works across various browsers, devices, and operating systems.**
   * **Tool: Selenium is commonly used for browser-based compatibility testing, and it can be combined with PyTest.**
   * **Steps:**
     1. **Install Selenium WebDriver and PyTest:**

**pip install selenium pytest**

* + 1. **Create test cases to ensure the Student Management System works on different browsers:**

**from selenium import webdriver**

**from selenium.webdriver.common.keys import Keys**

**def test\_login\_on\_chrome():**

**driver = webdriver.Chrome(executable\_path='/path/to/chromedriver')**

**driver.get("http://localhost:8000/login")**

**username\_field = driver.find\_element\_by\_name("username")**

**password\_field = driver.find\_element\_by\_name("password")**

**username\_field.send\_keys("student1")**

**password\_field.send\_keys("password123")**

**password\_field.send\_keys(Keys.RETURN)**

**assert "Dashboard" in driver.title**

**driver.quit()**

* + 1. **You can use similar tests for other browsers by changing the webdriver (e.g., webdriver.Firefox()).**

1. **Reliability Testing:**
   * **Objective: Verify the system's behavior under unexpected conditions, including server crashes or network issues.**
   * **Tool: Chaos Monkey (for distributed systems) or custom scripts.**
   * **Steps:**
     1. **Create tests that simulate failure conditions like server restarts or database unavailability.**
     2. **Ensure the system can recover without data loss.**
     3. **Use pytest-failures plugin to simulate and track failures.**

**This procedure outlines how to create Test Cases, perform Functional Testing, and Non-Functional Testing on a Student Management System using PyTest. PyTest is a powerful tool for writing and running tests, while additional tools like Locust and Selenium can be integrated for performance, security, and compatibility testing. By using this approach, you can ensure that your system is robust, scalable, and secure.**

**Report Template:**

**1. Title Page**

* **Title: PROJECT TITLE**
* **Subtitle: Ex. 9 Design and demonstration of test cases. Functional Testing and Non- Functional Testing using PyTest**
* **Your Name**
* **Course/Subject**
* **Instructor’s Name**
* **Date of Submission**
  1. **Test Table**