# Test Report for Unit Test Case Generation

## Test Summary:

The **Unit Test Case Generation** functionality was tested to ensure proper interaction between the **InputComponent** and **OutputComponent**. The key areas tested included:

- Code input handling
- Language selection
- Test case specification (total, positive, negative test cases)
- Code validation
- Error handling
- Test case generation
- Copy functionality for generated test cases

## Test Results:

1. **Code Input Handling:**
    a. **Result:** All test cases passed successfully.
    b. **Observation:**
        i. The **InputComponent** correctly captured code entered in the textarea and passed it for validation.
        ii. Invalid code inputs triggered the appropriate error message ("Code snippet cannot be empty.") when validation failed.
        iii. The **OutputComponent** displayed the error message for invalid code and prevented further test case generation.
2. **Language Selection:**
    a. **Result:** All test cases passed successfully.
    b. **Observation:**
        i. The user could select a programming language from the available radio buttons: C, Java, Python, JavaScript, and Ruby.
        ii. The selected language was correctly passed to the **OutputComponent** for generating language-specific test cases.
        iii. Language selection was validated alongside test case input and code validation.
3. **Test Case Specification:**
    a. **Result:** All test cases passed successfully.
    b. **Observation:**
        i. Users could input values for total test cases, positive test cases, and negative test cases.

        ii. Validation ensured that the sum of positive and negative test cases matched the total test cases, with appropriate error messages when mismatched values were entered.
        iii. If values for positive or negative test cases were negative, error messages appeared, such as "Positive test cases cannot be negative." and "Negative test cases cannot exceed total test cases."

4. **Code Validation:**
    a. **Result:** All test cases passed successfully.
    b. **Observation:**
        i. The **OutputComponent** validated whether the provided code was relevant.
        ii. If the code was invalid, the system displayed "IRRELEVANT CODE, CANNOT GENERATE TEST CASES".
        iii. The **InputComponent** correctly displayed an error message if the code input was empty.
5. **Error Handling:**
    a. **Result:** All test cases passed successfully.
    b. **Observation:**
        i. Error messages were displayed for invalid inputs, including when the sum of test cases didn't match the total or when negative test case numbers were provided.
        ii. If code was empty, the appropriate error message ("Code snippet cannot be empty.") was displayed, ensuring clear feedback to users.
6. **Test Case Generation:**
    a. **Result:** All test cases passed successfully.
    b. **Observation:**
        i. When valid inputs were provided, the **OutputComponent** sent a request to the backend to generate the unit test cases for the given code.
        ii. The generated test cases were displayed in the **OutputComponent** as expected.
        iii. If any error occurred in the backend communication (e.g., invalid response or server failure), an error message ("UNABLE TO LOAD SERVER") was shown.
7. **Copy Functionality:**
    a. **Result:** All test cases passed successfully.
    b. **Observation:**
        i. The **OutputComponent** successfully copied the generated test cases to the clipboard upon clicking the copy button.
        ii. The button text changed to "copied" for 2 seconds, confirming that the result was copied.

## Overall Observations:

- The integration between the **InputComponent** and **OutputComponent** worked seamlessly, allowing users to input code, select languages, and specify test cases.
- The **validateTestCases** function in the **InputComponent** effectively ensured that user inputs were valid before submission to the backend.
- Error handling was comprehensive, providing clear feedback to users for invalid inputs.
- The **OutputComponent** correctly displayed the results of the test case generation, or appropriate error messages, based on input validity.
- The copy functionality in the **OutputComponent** provided a convenient way for users to copy generated test cases to the clipboard.

## Conclusion:

The **Unit Test Case Generation** functionality performed well across all test scenarios. The system effectively managed user inputs, validated them, and communicated with the backend to generate appropriate test cases. Error handling and copy functionality worked as expected, enhancing the overall user experience.

The system is ready for deployment, providing a reliable tool for generating unit tests in various programming languages.