### **Test Case Document for Test Case Creator**

This document provides test cases to validate the functionality, robustness, and usability of the Test Case Creator. Each test verifies a core feature, such as prompt submission, validation, result display, and server communication.

#### **Test Environment:**

* **Operating System:** Windows 11
* **Programming Language:** JavaScript (React)
* **Backend Server:** Flask with OpenAI integration
* **API Library:** Axios for HTTP requests
* **Libraries/Frameworks:** React, Flask, OpenAI API

### **Test Cases**

#### **1. Prompt Submission (Functionality)**

* **Description:** Verify that a complete prompt can be successfully submitted and processed by the server.
* **Steps:**
  + Open the Test Case Creator application.
  + Enter valid values for all required fields (e.g., codeInput, totalTestCases, positiveTestCases, negativeTestCases, selectedLanguage, associationProcess).
  + Click the "Go" button.
* **Expected Result:** A loading spinner is displayed, and generated test cases are retrieved and shown in the result modal.

#### **2. Prompt Validation (Validation)**

* **Description:** Ensure the application prevents submission of incomplete prompt configurations.
* **Steps:**
  + Open the Test Case Creator application.
  + Leave one or more required fields blank (e.g., codeInput, totalTestCases).
  + Click the "Go" button.
* **Expected Result:** An error message appears, informing the user to complete all required fields.

#### **3. Result Display in Popup Modal (UI Functionality)**

* **Description:** Verify that the generated test cases are displayed in a popup modal upon completion.
* **Steps:**
  + Submit a valid prompt.
  + Wait for the response, and observe the modal popup with the result.
* **Expected Result:** Generated test cases appear in a popup modal that includes options to close or copy the result.

#### **4. Server Response Handling (Robustness)**

* **Description:** Verify that the application handles server errors or network issues gracefully.
* **Steps:**
  + Temporarily disconnect the network or stop the Flask server.
  + Attempt to submit a valid prompt.
* **Expected Result:** An error message is displayed, indicating a network or server issue, allowing the user to retry.

#### **5. CORS Configuration (Security)**

* **Description:** Verify that CORS is correctly configured for cross-origin requests between frontend (localhost:3000) and backend (localhost:5000).
* **Steps:**
  + Run the frontend and backend on different localhost ports.
  + Submit a valid prompt.
* **Expected Result:** The frontend successfully connects to the backend, retrieves data, and displays the result without CORS errors.

#### **6. Multiple Submissions (Performance)**

* **Description:** Test application performance by submitting prompts in quick succession.
* **Steps:**
  + Rapidly submit multiple prompts with different configurations.
* **Expected Result:** Each prompt is processed without significant delay or crashes, and each result is displayed correctly.

### **Test Report**

#### **Test Report for Test Case Creator**

* **Test Date:** 14-11-2024
* **Tester:** KAVIPRIYAA P

#### **Overall Test Results:**

* **Passed:** 5
* **Failed:** 0