Calculator Application

Functional Test Plan

September 13, 2023

Product Description

The Calculator app created by Eljin Whitehead-Stinson is a basic arithmetic calculator designed to perform four primary operations: Addition, Subtraction, Multiplication, and Division. It has a user-friendly interface with a numerical keypad for input and a display screen for showing results. The app has certain constraints on input values and handles errors gracefully.

Objectives

1. Ensure that the Calculator app performs accurate arithmetic calculations.
2. Validate that the app handles input constraints, error conditions, and user interactions correctly.
3. Verify that the user interface provides an intuitive experience for users.
4. Confirm that the app allows consecutive calculations without errors.

Testing Strategies

1. **Functional Testing:** Test the basic arithmetic operations (Addition, Subtraction, Multiplication, and Division) to ensure they produce accurate results.
2. **Boundary Value Testing:** Verify that the calculator correctly handles values at the lower and upper boundaries of the accepted range (-9999.99 to 9999.99).
3. **Error Handling Testing:** Test the calculator's behavior when it encounters errors, such as out-of-range results. Ensure that error messages are displayed appropriately.
4. **User Interface Testing:** Evaluate the user interface for ease of use, including number input, sign change, and clearing functionality.

5. **Consecutive Calculations Testing:** Verify that the app allows users to perform multiple calculations in succession without unexpected behavior.
6. **Decision Table Testing:** Verify the calculator's behavior based on input range and selected operation

Scope of Testing

Functional Testing:

1. Verify that the calculator can perform addition, subtraction, multiplication, and division with positive and negative numbers.
2. Verify that the calculator correctly handles decimal numbers and truncation of extra digits.
3. Verify that the calculator displays an error message for out-of-range results and invalid characters.

Boundary Testing:

1. Verify that the calculator works correctly with input values at the lower and upper boundaries.(-9999.99 - 9999.99)

Error Handling Testing:

1. .Verify that the calculator displays an error message for out-of-range results and invalid characters.
2. Verify that the calculator displays a proper error message for division by zero.

User Interface Testing:

1. Verify that the number input is responsive and accurate.
2. Verify that the sign change functionality works correctly for positive and negative numbers.
3. Verify that the clear functionality clears the display correctly.
4. Verify that the display updates correctly after each operation.

Consecutive Calculations Testing:

1. Verify that the calculator can handle a series of consecutive calculations without issues.

Decision Table Testing:

1. Verify the calculator's behavior based on input range and selected operation
2. Input Range (Valid, Below Lower Bound, Above Upper Bound)
3. Arithmetic Operation (Addition, Subtraction, Multiplication, Division)
4. Based on the combination of conditions, determine whether the calculator displays a valid result or an error message

Test Automation Approach

(Update sections 1-3 to match descrip)

The product under test is a calculator application developed using TypeScript, Selenium and Node.JS. Automation test scripts will validate that the calculator performs basic arithmetic operations, including Addition, Subtraction, Multiplication, and Division. It has specific requirements and functionality as outlined below:

1. Testing Approach Breakdown:

1. Test Scenarios: The test scenarios are the high-level descriptions of the tests that will be automated. The scenarios should be independent of each other, and they should be written in a way that makes them easy to understand and maintain.

2. Test Cases: The test cases are the detailed descriptions of the steps that will be executed for each test scenario. The test cases should be specific and unambiguous, and they should be written in a way that makes them easy to automate.

3. Test Steps: The test steps are the individual actions that will be executed for each test case. The test steps should be clear and concise, and they should be written in a way that makes them easy to understand and execute.

2. Tools Used for Automation:

1. **Programming Language:** TypeScript.
2. **TypeScript IDE:** Eclipse (Add Plugins)..
3. **UI Automation Framework:** Selenium WebDriver, WebdriverIO , Mocha, Jest
4. **Compiler:** TypeScript Compiling (TSC).
5. **Version Control:** Git.
6. **Server:** http-server public

3. Test Scenarios:

1. **Scenario 1: Valid Arithmetic Operations**
    - Test Case 1.1: Addition of two valid positive numbers.
    - Test Case 1.2: Subtraction of a valid negative number from a valid positive number.
    - Test Case 1.3: Multiplication of two valid numbers.
    - Test Case 1.4: Division of a valid number by another valid number.

2. **Scenario 2: Input Validation**
- Test Case 2.1: Verify that the calculator accepts numbers within the valid range.
- Test Case 2.2: Ensure that numbers outside the valid range are rejected.
- Test Case 2.3: Test that extra decimal digits are truncated.
3. **Scenario 3: Error Handling**
- Test Case 3.1: Verify that the calculator displays an error message for out-of-range calculations.
4. **Scenario 4: Sign Change**
- Test Case 4.1: Test the ability to change the sign of a number.
5. **Scenario 5: Clear Display**
- Test Case 5.1: Verify that the display can be cleared.
6. **Scenario 6: Consecutive Calculations**
- Test Case 6.1: Perform multiple consecutive calculations without clearing the display.