Calculator

Test Documentation

SHEETAL KASHID

Contents

Description

This document contains the Test plan, Test Scope, Test Strategy, Test Cases, it's results and bug report for the calculator application. First the App was implemented and then this document was created.

The main requirements that are to be tested are the following:

1. All the buttons on the app
2. All the functions/ operations are correctly performed
3. The calculator only accepts numbers between -9999.99 and 9999.99.
4. Internal calculations should truncate extra digits after the second decimal digit.
5. Internal calculations should throw an error when the calculated number is out of range, and display it on the screen.
6. The sign can be changed at any time when a number is entered.
7. The display can be cleared at any time.
8. Consecutive calculations are allowed before clearing the display.

Test Scope

The test plan is to conclude a final assessment on the Calculator application. The tests will ensure that the data inputs are correctly taken care of and allows numbers or operations accordingly. It ensures that the results are calculated and displayed correctly. The test would also guarantee that the application is compliant with all the requirements. It starts with test strategy followed by test plan, test cases with its result considering various scenarios are presented. It also informs any areas or functionality that did not work and also provides a bug report at the end.

Test Strategy

User Interface Testing

The goal of this test is to ensure that the user interface provides the user with an appropriate access and navigation through the entire process of the application. It would check that the user interface meets all the requirements and objective. It would also test the design, layout, links, buttons and all the other application behaviors. The test will perform a set of user operations to the app and verify that is it behaving as expected. This will ensure that the interface is behaving as expected whenever the user enters their information and returns the correct output response. Finally testing will be conducted on multiple environments to ensure compliance with multiple platforms and browsers.

Function Testing

The function testing of this application will focus on all the requirements that can be traced directly to use cases. It will check if all the functions are

appropriately implemented and are working correctly. It will ensure proper entering of data, then processing and finally displaying the data. It will also check if an error occurs when invalid information is used. The function test will help execute and identify all the defects and help address them.

During function testing the tester will choose between valid inputs to check whether application is process correctly. As well as invalid input to determine if the application can detect them. The tester will test the functionality of the application and compare the actual output with the expected output. The test will be conducted in and viewed in all possible angles. This will ensure that the system is working as per the functionalities of the application in all possible and impossible criteria. The main objective of this testing will be to check whether the system is functionally probably and improbably.

Negative Testing and Positive Testing

All scenarios negative as well as positive are tested. Positive scenarios include all the valid typical inputs and the negative scenarios include all the incorrect input that can be given to the application.

Test Cases/Scenarios

Boundary Value

Index	1
Title	Inner Boundary Values
Scenario	User enters -9999.99 as one of the inputs

Line of Code	`if (parseFloat(this.curr) > 9999.99`
	`||  parseFloat(this.curr) < -9999.99){`
	`this.curr = "NA"`
	`}`

Test case Pass	It computes the value and if the result is > 9999.99 or <-9999.99 then should output NA
Test Case Fail	If 9999.99 < result < - 9999.99 and result is displayed

Result	PASS

Number 1	Operation	Number 2	Result
9999.99	ADD	0.01	Pass
-9999.99	ADD	-0.01	Pass
10000	SUBTRACT	0.01	Pass
10000	SUBTRACT	0.001	Pass
0	ADD	9999.99	Pass
4999.999	ADD	4999.999	Pass

Boundary Value with Consecutive Calculations

Index	2
Title	Boundary Values with Consecutive calculations
Scenario	User enters numbers that leads to out of boundary consecutively

Line of Code	`//Compute() is called after the operator is hit`
	`if (parseFloat(this.curr) > 9999.99`
	`||  parseFloat(this.curr) < -9999.99){`
	`this.curr = "NA"`
	`}`

Test case Pass	It computes the value and if the result is > 9999.99 or <-9999.99 then should output NA
Test Case Fail	If 9999.99 < result < - 9999.99 and result is displayed
Result	PASS

Number 1/ Result of previous operation	Operation	Number 2	Result
5000	ADD	5000	Pass
NA	ADD	40	Pass
40	SUBTRACT	10040	Pass
NA	SUBTRACT	0.001	Fail
	ADD	9999.99	Pass
4999.999	ADD	4999.999	Pass

Bug: 3rd step failed as NA cannot be subtracted by any number and hence before entering an AC press is required

Truncate values after two floating point decimal places

Index	3
Title	Truncate values after two floating point decimal places
Scenario	User enters numbers such that they have decimal places > 2 truncate while calculating
Line of Code	`computation = computation.toFixed(2)`
Test case Pass	Result has two floating point decimal places
Test Case Fail	Result has greater than two floating point decimal places
Result	PASS

Number 1/ Result of previous operation	Operation	Number 2	Result
4567.589	ADD	3961.852	Pass
4999.445	ADD	4999.555	Pass
9999.99	ADD	1	Pass

Sign can be changed any time a number is entered

Index	4
Title	Sign can be changed any time a number is entered
Scenario	User enters numbers and press NEG which negates the current sign of the number

Line of Code	`//If already negative then make it positive by removing the "-"`
	`if (this.curr.toString().charAt(0) == "-"){`
	`this.curr = this.curr.toString().slice(-(this.curr.toString().length - 1))`
	`}`
	`//else add - at the start`
	`else{`
	`this.curr = "-" + this.curr.toString()`
	`}`

Test case Pass	Result calculates the value accordingly
Test Case Fail	Negate not correctly applied(gives any other output)
Result	PASS

Series of Inputs	Result
45 + 35 NEG	Pass
NEG 39 NEG – 28 NEG	Pass

Screen can be cleared/ Reset any time

Index	5
Title	Screen can be cleared/ Reset any time
Scenario	User enters inputs and clears the screen in multiple scenarios which resets everything

Line of Code	`clear(){`
	`this.curr = ""`
	`this.prev = ""`
	`this.operation = undefined`
	`}`

Test case Pass	Screen should be clear
Test Case Fail	Anything visible on the screen
Result	PASS

Series of Inputs	Result
45 + 53 = AC	Pass
29 - AC	Pass

Consecutive calculations allowed

Index	6
Title	Consecutive calculations allowed
Scenario	User enters inputs and does multiple operations consecutively
Line of Code	`//If previous number is not none ie user is doing consecutive operations eg 2 + 3 - (after encountering the minus here 2 + 3 is computed and the result becomes the previous)` `if (this.prev !== ''){`

	this.compute() }
Test case Pass	Should give the desired result after consecutive operations
Test Case Fail	Incorrect result
Result	PASS

Series of Inputs	Result
24 + 59 – 29 * 2 / 4	Pass
27356 / 7 – 234 * 3	Pass

Bug Report

The requirements are all met. Each of the functions work well and the UI correctly handles the input and output data. Negative and positive tests were conducted. The positive and negative scenarios tests successfully gave expected results.

Some of the design issues that I found were the following:

1. When NA is shown due to either out of boundary or invalid calculation it has to be erased by clear all (AC) before starting a new calculation.
2. One cannot enter negative numbers just by adding – you will have to press NEG in order to do so.