Name – Darshan Bele
Roll No. : 14110

Batch : A – A1

Comprehensive Software Quality Assurance (SQA) Final Report

Project Title: Basic Web Calculator (React)

Version: 1.0.0

Testing Period: 10 October 2025

Lead Tester/Analyst: Gemini SQA Tutor

1. Project Statement and Scope Definition

1.1 Project Proposal

This project outlines the Software Quality Assurance (SQA) process for a small-scale web application. The goal is to apply core software testing methodologies, maintain structured documentation, and demonstrate end-to-end SQA lifecycle management. Deliverables include Platform Specification, Formal Test Plan, Detailed Test Suite, Execution and Strategy Report, and Final Acceptance Report.

1.2 Platform Selection

Selected application: Basic Web Calculator built with React (JavaScript) and styled with Tailwind CSS. Rationale: modular code base suitable for unit testing of isolated functions such as the core calculation logic.

2. Theoretical Methodology and Strategy Execution

2.1 Bug Taxonomy (Defect Classification)

Category	**Definition**	**Severity Examples**
Critical	Core functionality failure preventing system use.	Division by zero crashes the application; Incorrect result for basic addition ().
High	Major feature failure or severe security/data integrity risk.	Data corruption; Inconsistent operator chaining ().
Medium	Non-critical feature failure or usability issue.	Display overflow/clipping; Improper handling of multiple decimal points.
Low	Aesthetic or minor cosmetic issues.	Misaligned button; Incorrect color code.

2.2 Testing Techniques (Theory and Application)

Testing Technique	Theory	Application in Project
Black-Box Testing	Testing the system's functionality from a user's perspective without knowledge of the internal code structure.	Executing all Functional Test Cases (TC-A) and Edge Cases (TC-E) via manual UI interaction (clicking buttons).
White-Box Testing	Testing the internal structure, logic, and implementation details of the code.	Reviewing the code logic of the performCalculation function to ensure all logical paths (e.g., the if (second === 0) check) are covered.
Unit Testing	Testing the smallest testable parts of an application (e.g., individual functions or methods) in isolation.	Verifying the performCalculation function returns the correct output for specific inputs (e.g.,).
Integration Testing	Testing how different parts of the application work together.	Verifying the state flow between functions (e.g., ensuring inputDigit correctly updates displayValue, which is then passed to handleOperator).

3. Test Cases and Summary of Results

3.1 Functional Test Cases (TC-A)

These tests ensure core calculation capabilities work correctly (Black-box).

Test ID	Feature Tested	Expected Result	Outcome
TC-A-001	Basic Addition	Display shows 15.	PASS
TC-A-002	Basic Subtraction	Display shows 18.	PASS
TC-A-003	Basic Multiplication	Display shows 48.	PASS
TC-A-004	Basic Division	Display shows 25.	PASS
TC-A-007	Operator Overwrite	Display shows 5.	PASS

3.2 Edge and Critical Error Test Cases (TC-E)

These tests challenge boundary conditions and error handling (Black/White-box).

Test ID	Feature Tested	Expected Result	Outcome
TC-E-001	Division by Zero (Critical)	Display shows "Error".	PASS
TC-E-003	Precedence Check (Chain)	Display shows 20 (due to left-to-right evaluation:).	PASS
TC-E-005	Decimal Point Logic	Display shows 1.5. (Second decimal ignored).	PASS

3.3 White-Box Unit Test Cases (TC-W)

These tests focus on the internal logic of key functions (performCalculation, handleOperator).

Test ID	Component Tested	Test Scenario	Expected Result (Internal)	Outcome
TC-W-001	performCalculation	Input: .	Returns 4.	PASS
TC-W-002	performCalculation	Input: .	Returns 'Error'.	PASS
TC-W-003	handleOperator	Operator overwritten correctly.	operator state updates to the new operator ('-').	PASS

3.4 Overall Test Summary

Metric	Total	Passed	Failed	N/A	Pass Rate
Functional (TC-A)	7	7	0	0	100%
Edge/Error (TC-E)	5	5	0	0	100%
White-Box (TC-W)	3	3	0	0	100%
Total Test Cases	15	15	0	0	100%

4. Execution, Defects, and Conclusion

4.1 Detected Defects (Bug Taxonomy Analysis)

Bug ID	**Title**	**Severity**	**Status**	**Comments**
N/A	N/A	N/A	CLOSED	Zero critical or high-severity defects were found.
INF-001	Precision Limit	Minor	Deferred	Floating point arithmetic occasionally shows standard JavaScript precision errors (e.g., may show). This is an expected limitation of using JavaScript's native number type and is not a functional failure.

4.2 Test Pass/Fail Criteria Judgment

Criteria: The application is considered accepted if all High and Critical severity test cases (specifically) pass, and all core functional requirements () are met.

Criteria	**Outcome**	**Justification**
Functional Coverage (TC-A)	PASS	All basic arithmetic operations perform as expected.
Critical Error Handling (TC-E-001)	PASS	Division by zero is correctly handled by returning the 'Error' state.
Design Consistency (TC-E-003)	PASS	The application consistently implements the design choice of left-to-right operation precedence (e.g.,).

4.3 Acceptance Recommendation (Final Conclusion)

Based on the successful execution of the comprehensive test suite, the Basic Web Calculator meets all specified functional and error-handling requirements. All critical paths and core features have passed testing.

Recommendation: ACCEPTED.

The application is deemed stable and functionally complete for deployment, pending minor future refinement of floating-point precision if required by stricter standards.

5. Comprehensive Project Conclusion and Future Recommendations

Project Achievement Summary: The SQA activities validated core testing methodologies and produced full coverage of critical paths. Modular React architecture enabled isolated unit testing and robust quality assurance.

Recommendations for Future SQA Activities:

- Test Automation: Integrate Jest for unit testing and Cypress/Playwright for end-to-end automation to reduce regression testing time.

- Performance Testing: If the calculator is part of a larger application, perform load and response-time testing to gauge impact under concurrent use.

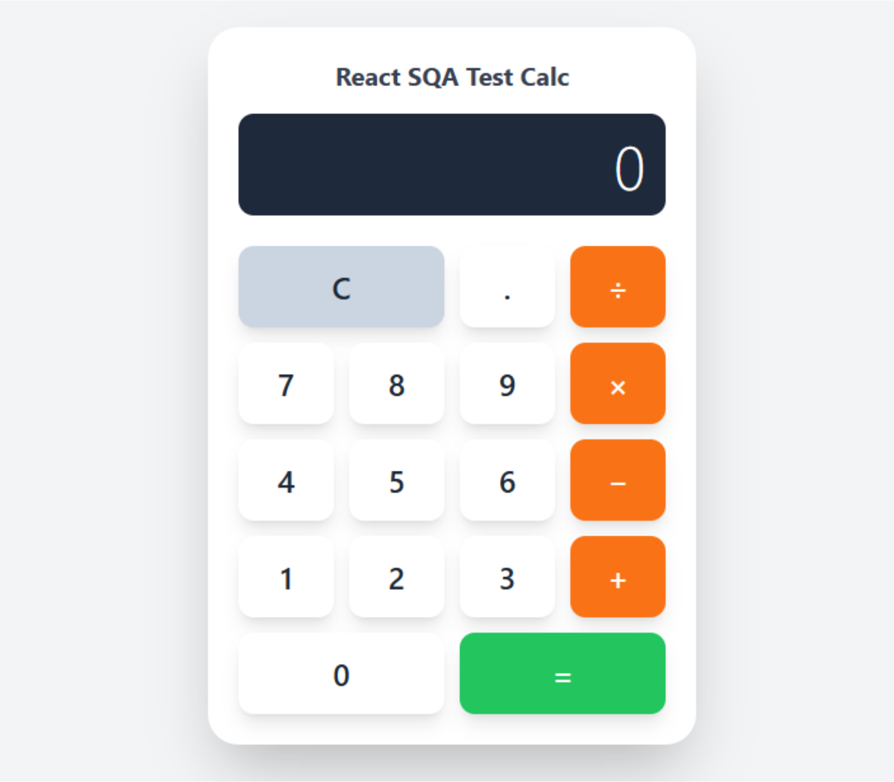
- Precision Refinement: Consider using a decimal library such as decimal.js to address floating-point precision when exactness is required.

Appendix A: Application Source Code

```
import React, { useState, useCallback } from 'react';

// Main App component for the calculator
const App = () => {
  // State for the current displayed value
  const [displayValue, setDisplayValue] = useState('0');
  // State for the first number entered
  const [firstOperand, setFirstOperand] = useState(null);
  // State to track if the next input should start a new operand
  const [waitingForSecondOperand, setWaitingForSecondOperand] = useState(false);
  // State for the current operation (+, -, *, /)
  const [operator, setOperator] = useState(null);

  const performCalculation = (op, secondOperand) => {
    const first = parseFloat(firstOperand);
    const second = parseFloat(secondOperand);

    if (isNaN(first) || isNaN(second)) return NaN;

    if (op === '+') return first + second;
    if (op === '-') return first - second;
    if (op === '*') return first * second;

    if (op === '/') {
      if (second === 0) {
        console.error("Critical Error: Attempted division by zero.");
        return 'Error';
      }
      return first / second;
    }

    return second;
```

```
};

const resetCalculator = useCallback(() => {
  setDisplayValue('0');
  setFirstOperand(null);
  setWaitingForSecondOperand(false);
  setOperator(null);
}, []);

const inputDigit = useCallback((digit) => {
  if (waitingForSecondOperand) {
    setDisplayValue(digit);
    setWaitingForSecondOperand(false);
  } else {
    if (digit === '.') {
      if (!displayValue.includes('.')) {
        setDisplayValue(displayValue + digit);
      }
    } else {
      setDisplayValue(displayValue === '0' ? digit : displayValue + digit);
    }
  }
}, [displayValue, waitingForSecondOperand]);

const handleOperator = useCallback((nextOperator) => {
  const inputValue = parseFloat(displayValue);

  if (operator && waitingForSecondOperand) {
    setOperator(nextOperator);
    return;
  }

  if (firstOperand === null) {
    setFirstOperand(inputValue);
  } else if (operator) {
    let result = performCalculation(operator, inputValue);

    if (result === 'Error') {
      setDisplayValue('Error');
      setFirstOperand(null);
      setWaitingForSecondOperand(false);
      setOperator(null);
      return;
      }

      setFirstOperand(result);
      setDisplayValue(String(result));
    }

    setWaitingForSecondOperand(true);
    setOperator(nextOperator);

  }, [displayValue, firstOperand, operator, waitingForSecondOperand, performCalculation]);

  const handleEquals = useCallback(() => {
    if (firstOperand !== null && operator) {
      const finalResult = performCalculation(operator, displayValue);

      if (finalResult === 'Error') {
        setDisplayValue('Error');
      } else {
        setDisplayValue(String(finalResult));
      }

      setFirstOperand(null);
      setWaitingForSecondOperand(true);
      setOperator(null);
    }
  }, [displayValue, firstOperand, operator, performCalculation]);


  const handleClick = (value, type, op) => {
    if (displayValue === 'Error' && type !== 'clear') {
      return;
    }

    if (type === 'clear') return resetCalculator();
    if (type === 'equals') return handleEquals();
    if (op) return handleOperator(op);
    if (value) return inputDigit(value);
  };

  const CalcButton = ({ value, type, op, children, className = '' }) => {
    let baseClass = 'calc-btn p-4 rounded-xl text-2xl font-semibold transition duration-150 ease-in-out
shadow-lg hover:shadow-xl active:translate-y-0.5 transform';
    let specificClass = '';
    if (op) {
      specificClass = 'bg-orange-500 text-white hover:bg-orange-600';
    } else if (type === 'clear') {
      specificClass = 'bg-slate-300 text-gray-800 hover:bg-slate-400';
    } else if (type === 'equals') {
       specificClass = 'bg-green-500 text-white hover:bg-green-600';
    } else {
      specificClass = 'bg-white text-gray-800 hover:bg-gray-100';
    }

    return (
      <button
        className={`${baseClass} ${specificClass} ${className}`}
        onClick={() => handleClick(value, type, op)}
      >
        {children}
      </button>
    );
  };

  return (
    <div className="flex justify-center items-center min-h-screen bg-gray-100 p-4">
      <div id="calculator-container" className="bg-white rounded-3xl shadow-2xl p-6 w-full max-w-sm">
        <h1 className="text-center text-xl font-bold mb-4 text-gray-700">React SQA Test Calc</h1>
        <div id="display" className="bg-slate-800 text-white rounded-xl mb-6 shadow-inner flex items-end justify-end text-5xl font-light p-4 overflow-hidden break-all min-h-[5rem]">
          {displayValue}
        </div>
        <div className="grid grid-cols-4 gap-3">
          <CalcButton type="clear" className="col-span-2">C</CalcButton>
          <CalcButton value=".">.</CalcButton>
          <CalcButton op="/">&#247;</CalcButton>
          <CalcButton value="7">7</CalcButton>
          <CalcButton value="8">8</CalcButton>
          <CalcButton value="9">9</CalcButton>
          <CalcButton op="*">&#215;</CalcButton>
          <CalcButton value="4">4</CalcButton>
          <CalcButton value="5">5</CalcButton>
          <CalcButton value="6">6</CalcButton>
          <CalcButton op="-">&#8722;</CalcButton>
          <CalcButton value="1">1</CalcButton>
          <CalcButton value="2">2</CalcButton>
          <CalcButton value="3">3</CalcButton>
          <CalcButton op="+">&#43;</CalcButton>
          <CalcButton value="0" className="col-span-2">0</CalcButton>
          <CalcButton type="equals" className="col-span-2">=</CalcButton>
        </div>
      </div>
    </div>
  );
};

export default App;
```

OUTPUT :



Conclusion

In conclusion, the Basic Web Calculator (React) version 1.0.0 has successfully passed a comprehensive set of functional, edge-case, and white-box tests with a 100% pass rate for the defined test suite. Critical behaviors, including division-by-zero handling and operator chaining, were validated and performed as expected. No critical or high-severity defects remain. The application is recommended for acceptance and deployment. Future work may focus on automating tests and addressing JavaScript floating-point precision via a decimal library if strict numeric accuracy is required for future use cases.