Unit Testing for Addition Function - Test Scenarios

App.jsx (Component code):

```
import { useRef, useState } from "react";

import "./App.css";

function App() {

 const num1 = useRef();

 const num2 = useRef();

 const [result, setResult] = useState(0);

 const handleSubmit = (e) => {

  e.preventDefault();

  let num = Number(num1.current.value) + Number(num2.current.value);

  if (num1.current.value === "" || num2.current.value === "" || isNaN(num)) {

   document.getElementById("result").className = "red-text";

   setResult("Error invalue input!!!");

   return;

  }

  if (num > Number.MAX_SAFE_INTEGER) {

   document.getElementById("result").className = "red-text";

   setResult("overflow");

  } else if (num < Number.MIN_SAFE_INTEGER) {

   document.getElementById("result").className = "red-text";

   setResult("underflow");

  } else {

   document.getElementById("result").className = "green-text";

   setResult(num);

  }

 };

 return (

  <div className="App">

   <h2>

    <span>Add two number</span>
```

```
      </h2>
      <form>
        <label>Enter number 1:</label>
        <input type="text" ref={num1} data-testid="num-1" id="num-1"/>
        <br />
        <label>Enter number 2:</label>
        <input type="text" ref={num2} data-testid="num-2" />
        <br />
        <input
          type="submit"
          value="Calculate"
          onClick={handleSubmit}
          data-testid="calculate"
        />
      </form>
      <br />
      <p>
        Result:{" "}
        <span id="result" data-testid="result" className="green-text">
          {result}
        </span>
      </p>
    </div>
  );
}

export default App;
```

App.css code:

```
@import url("https://fonts.googleapis.com/css2?family=Montserrat&display=swap");
body {
  height: 100vh;
  margin: 0;
  display: flex;
  justify-content: center;
  align-items: center;

  background: rgb(96, 216, 221);
  background: linear-gradient(
    63deg,
    rgba(96, 216, 221, 1) 0%,
    rgba(121, 116, 229, 1) 30%,
    rgba(222, 57, 245, 1) 100%
  );
}
.App {
  padding: 3rem;
  font-family: "Montserrat", sans-serif;
  background-color: white;
  border-radius: 5px;
  width: 20rem;
}
.App form {
  display: flex;
  flex-direction: column;
}
.App form label {
  margin-bottom: 0.3rem;
}
.App form input {
  padding: 10px;
  font-size: 16px;
}
.App form input[type="submit"]{
  cursor: pointer;
```

```
}
.App h2 span {
  font-family: "Montserrat", sans-serif;
}
.red-text{
  color: red;
}
.green-text{
  color: green;
}
```

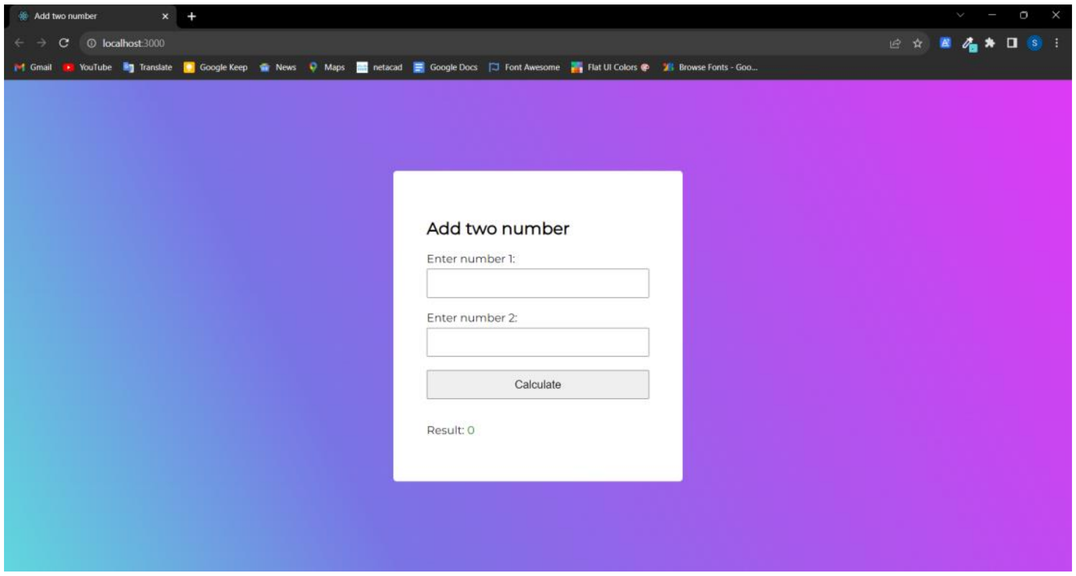
App.test.js

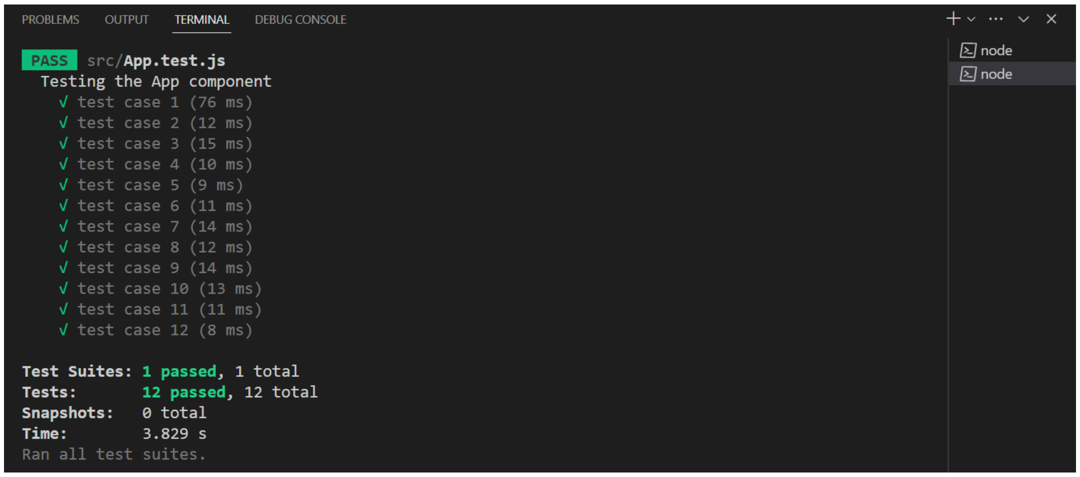
```
import { render, screen, fireEvent } from "@testing-library/react";
import App from "./App";
describe("Testing the App component", () => {
  test("test case 1", () => {
    render(<App />);
    const num1 = screen.getByTestId("num-1");
    const num2 = screen.getByTestId("num-2");
    const button = screen.getByTestId("calculate");
    fireEvent.change(num1, { target: { value: 5 } });
    fireEvent.change(num2, { target: { value: 7 } });
    fireEvent.click(button);
    expect(screen.getByTestId("result").innerHTML).toBe("12");
  });
  test("test case 2", () => {
    render(<App />);
    const num1 = screen.getByTestId("num-1");
    const num2 = screen.getByTestId("num-2");
    const button = screen.getByTestId("calculate");
    fireEvent.change(num1, { target: { value: -5 } });
    fireEvent.change(num2, { target: { value: -3 } });
    fireEvent.click(button);
    expect(screen.getByTestId("result").innerHTML).toBe("-8");
  });
  test("test case 3", () => {
    render(<App />);
    const num1 = screen.getByTestId("num-1");
    const num2 = screen.getByTestId("num-2");
    const button = screen.getByTestId("calculate");
    fireEvent.change(num1, { target: { value: 10 } });
    fireEvent.change(num2, { target: { value: -4 } });
    fireEvent.click(button);
    expect(screen.getByTestId("result").innerHTML).toBe("6");
  });
  test("test case 4", () => {
    render(<App />);
    const num1 = screen.getByTestId("num-1");
    const num2 = screen.getByTestId("num-2");
    const button = screen.getByTestId("calculate");
    fireEvent.change(num1, { target: { value: 0 } });
    fireEvent.change(num2, { target: { value: 9 } });
    fireEvent.click(button);
    expect(screen.getByTestId("result").innerHTML).toBe("9");
  });
  test("test case 5", () => {
    render(<App />);
    const num1 = screen.getByTestId("num-1");
    const num2 = screen.getByTestId("num-2");
    const button = screen.getByTestId("calculate");
    fireEvent.change(num1, { target: { value: 3 } });
    fireEvent.change(num2, { target: { value: 2 } });
    fireEvent.click(button);
    expect(screen.getByTestId("result").innerHTML).toBe("5");
  });
```

```
test("test case 6", () => {
  render(<App />);
  const num1 = screen.getByTestId("num-1");
  const num2 = screen.getByTestId("num-2");
  const button = screen.getByTestId("calculate");
  fireEvent.change(num1, { target: { value: 1000000 } });
  fireEvent.change(num2, { target: { value: 500000 } });
  fireEvent.click(button);
  expect(screen.getByTestId("result").innerHTML).toBe("1500000");
});
test("test case 7", () => {
  render(<App />);
  const num1 = screen.getByTestId("num-1");
  const num2 = screen.getByTestId("num-2");
  const button = screen.getByTestId("calculate");
  fireEvent.change(num1, { target: { value: 2.5 } });
  fireEvent.change(num2, { target: { value: 1.75 } });
  fireEvent.click(button);
  expect(screen.getByTestId("result").innerHTML).toBe("4.25");
});
test("test case 8", () => {
  render(<App />);
  const num1 = screen.getByTestId("num-1");
  const num2 = screen.getByTestId("num-2");
  const button = screen.getByTestId("calculate");
  fireEvent.change(num1, { target: { value: -7 } });
  fireEvent.change(num2, { target: { value: 7 } });
  fireEvent.click(button);
  expect(screen.getByTestId("result").innerHTML).toBe("0");
});
test("test case 9", () => {
  render(<App />);
  const num1 = screen.getByTestId("num-1");
  const num2 = screen.getByTestId("num-2");
  const button = screen.getByTestId("calculate");
  fireEvent.change(num1, { target: { value: Number.MAX_SAFE_INTEGER } });
  fireEvent.change(num2, { target: { value: 1 } });
  fireEvent.click(button);
  expect(screen.getByTestId("result").innerHTML).toBe("overflow");
});
test("test case 10", () => {
  render(<App />);
  const num1 = screen.getByTestId("num-1");
  const num2 = screen.getByTestId("num-2");
  const button = screen.getByTestId("calculate");
  fireEvent.change(num1, { target: { value: Number.MIN_SAFE_INTEGER } });
  fireEvent.change(num2, { target: { value: -1 } });
  fireEvent.click(button);
  expect(screen.getByTestId("result").innerHTML).toBe("underflow");
});
test("test case 11", () => {
  render(<App />);
  const num1 = screen.getByTestId("num-1");
  const num2 = screen.getByTestId("num-2");
  const button = screen.getByTestId("calculate");
  fireEvent.change(num1, { target: { value: null } });
  fireEvent.change(num2, { target: { value: 5 } });
  fireEvent.click(button);
  expect(screen.getByTestId("result").innerHTML).toBe(
    "Error invalue input!!!"
  );
});
test("test case 12", () => {
  render(<App />);
```

```
    const num1 = screen.getByTestId("num-1");
    const num2 = screen.getByTestId("num-2");
    const button = screen.getByTestId("calculate");
    fireEvent.change(num1, { target: { value: "hello" } });
    fireEvent.change(num2, { target: { value: 3 } });
    fireEvent.click(button);
    expect(screen.getByTestId("result").innerHTML).toBe(
      "Error invalue input!!!"
    );
  });
});
```

Output:





```
PASS src/App.test.js
  Testing the App component
    √ test case 1 (76 ms)
    √ test case 2 (12 ms)
    √ test case 3 (15 ms)
    √ test case 4 (10 ms)
    √ test case 5 (9 ms)
    √ test case 6 (11 ms)
    √ test case 7 (14 ms)
    √ test case 8 (12 ms)
    √ test case 9 (14 ms)
    √ test case 10 (13 ms)
    √ test case 11 (11 ms)
    √ test case 12 (8 ms)

Test Suites: 1 passed, 1 total
Tests:       12 passed, 12 total
Snapshots:   0 total
Time:        3.829 s
Ran all test suites.
```