CALCULADORA.

Requerimientos Técnicos:

1. **Backend:**
   - Implementado en Java 8 con Spring Framework.
   - Uso de Spring Boot para simplificar la configuración y despliegue de la aplicación.
   - Exposición de una API RESTful para que el frontend pueda comunicarse con el backend.
   - Manejo de excepciones y validación de datos en el servidor.
2. **Frontend:**
   - Desarrollado con React.
   - Implementación de componentes para cada elemento de la calculadora (botones, pantalla de resultados, etc.).
   - Uso de fetch o Axios para realizar llamadas HTTP al backend y obtener los resultados de las operaciones.
   - Manejo de estado utilizando useState o useReducer para almacenar el historial de operaciones.
3. **Despliegue:**
   - El proyecto debe ser desplegado localmente y demostrar su funcionamiento en un navegador web.

Comandos Node.js y React:

npm install: Instala todas las dependencias listadas en el archivo package.json.

npx create-react-app: Crea una nueva aplicación React con una configuración predeterminada y lista para usar. (calculadora-frontend)

npm start: Inicia el servidor de desarrollo para una aplicación React (o Node.js). localhost:3000

Comandos de Terminal para Spring Boot

./mvnw spring-boot:run: Ejecuta una aplicación Spring Boot usando Maven Wrapper.

CONTROLLER:

```
package com.example.calculadora.controllers;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

// Anotación que define esta clase como un controlador REST.
// Permite el acceso a los métodos de la clase a través de HTTP.
@RestController
// Permite solicitudes CORS desde el frontend en http://localhost:3000
@CrossOrigin(origins = "http://localhost:3000")
public class CalculadoraController {

    // Mapea solicitudes GET a la URL /sumar
    // Recibe dos parámetros num1 y num2 y devuelve la suma de ambos.
    @GetMapping("/sumar")
    public double sumar(@RequestParam double num1, @RequestParam double num2) {
        return num1 + num2;
    }

    // Mapea solicitudes GET a la URL /restar
    // Recibe dos parámetros num1 y num2 y devuelve la resta de ambos.
    @GetMapping("/restar")
    public double restar(@RequestParam double num1, @RequestParam double num2) {
        return num1 - num2;
    }

    // Mapea solicitudes GET a la URL /multiplicar
    // Recibe dos parámetros num1 y num2 y devuelve el producto de ambos.
    @GetMapping("/multiplicar")
    public double multiplicar(@RequestParam double num1, @RequestParam double num2) {
        return num1 * num2;
    }

    // Mapea solicitudes GET a la URL /dividir
    // Recibe dos parámetros num1 y num2.
    // Si num2 es cero, devuelve un error 400 (Bad Request) con un mensaje de error.
    // Si no, devuelve el resultado de la división.
    @GetMapping("/dividir")
```

```
    public ResponseEntity<?> dividir(@RequestParam double num1, @RequestParam double num2)
{
        // Verifica si el divisor es cero para evitar una división por cero
        if (num2 == 0) {
            // Devuelve una respuesta con estado 400 y un mensaje de error
            return ResponseEntity.badRequest().body("No se puede dividir por cero");
        }
        // Calcula el resultado de la división
        double resultado = num1 / num2;
        // Devuelve una respuesta con estado 200 (OK) y el resultado de la división
        return ResponseEntity.ok(resultado);
    }
}
```

App.js

```
import React, { useState } from "react";
import './App.css';

function App() {
  // Estado para almacenar el valor del primer número
  const [num1, setNum1] = useState("");
  // Estado para almacenar el valor del segundo número
  const [num2, setNum2] = useState("");
  // Estado para almacenar el resultado de la operación
  const [resultado, setResultado] = useState(null);
  // Estado para almacenar la operación seleccionada
  const [operacion, setOperacion] = useState("");
  // Estado para almacenar el historial de operaciones
  const [historial, setHistorial] = useState([]);

  // Maneja el cambio en el valor del primer número
  const handleNum1Change = (e) => setNum1(e.target.value);
  // Maneja el cambio en el valor del segundo número
  const handleNum2Change = (e) => setNum2(e.target.value);
  // Maneja el cambio en la operación seleccionada
  const handleOperacionChange = (e) => setOperacion(e.target.value);

  // Función para calcular el resultado de la operación seleccionada
  const calcularResultado = async () => {
```

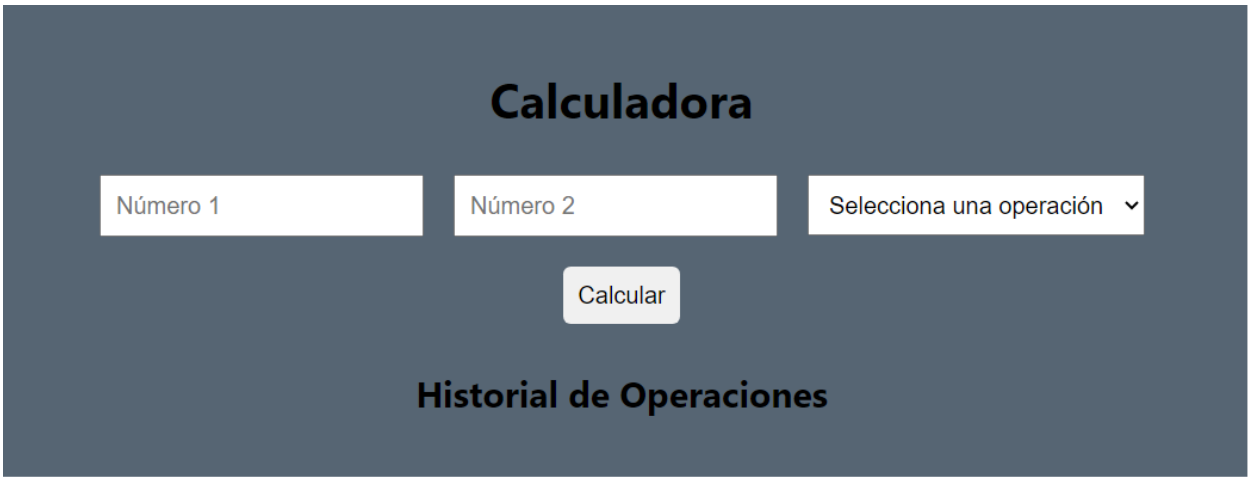
```
  // Verifica si los números y la operación están definidos
  if (!num1 || !num2 || !operacion) {
    alert("Por favor, ingresa ambos números y selecciona una operación.");
    return;
  }

  // Construye la URL para la solicitud a la API
  const url = `http://localhost:8080/${operacion}?num1=${num1}&num2=${num2}`;
  try {
    // Realiza la solicitud a la API
    const response = await fetch(url);
    const data = await response.json();
    // Verifica si la respuesta fue exitosa
    if (response.ok) {
      // Establece el resultado de la operación
      setResultado(data);
      // Crea un nuevo objeto para el historial
      const nuevaOperacion = {
        operacion: operacion,
        num1: num1,
        num2: num2,
        resultado: data,
      };
      // Actualiza el historial, manteniendo solo las últimas 10 operaciones
      setHistorial(prevHistorial => [nuevaOperacion, ...prevHistorial.slice(0, 9)]);
    } else {
      // Muestra un mensaje de error si la respuesta no fue exitosa
      alert("Error: " + data.message);
    }
  } catch (error) {
    // Maneja errores en la solicitud
    console.error("Error al calcular", error);
  }
 };
//Visual
 return (
  <div className="App">
   <h1>Calculadora</h1>
   <div>
    <input
     type="number"
     value={num1}
     onChange={handleNum1Change}
     placeholder="Número 1"
```

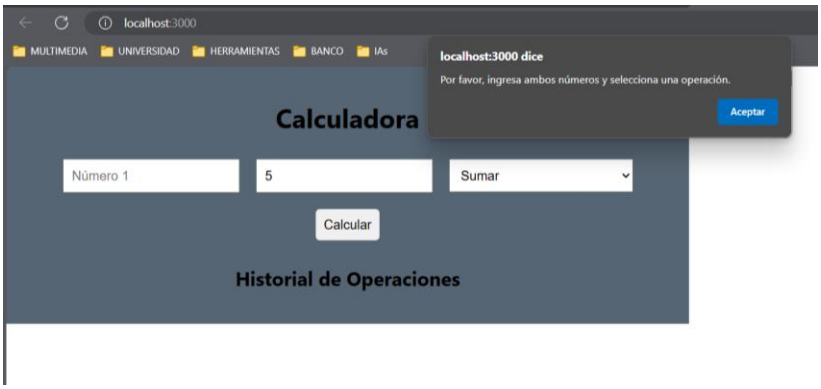
```
      />
      <input
        type="number"
        value={num2}
        onChange={handleNum2Change}
        placeholder="Número 2"
      />
      <select value={operacion} onChange={handleOperacionChange}>
        <option value="">Selecciona una operación</option>
        <option value="sumar">Sumar</option>
        <option value="restar">Restar</option>
        <option value="multiplicar">Multiplicar</option>
        <option value="dividir">Dividir</option>
      </select>
      <button
        style={{ backgroundColor: 'green', color: 'white' }}
        onClick={calcularResultado}
      >
        Calcular
      </button>
    </div>
    {resultado !== null && (
      <div>
        <h2>Resultado: {resultado}</h2>
      </div>
    )}
    {historial.length > 0 && (
      <div>
        <h2>Historial de Operaciones</h2>
        <ul>
          {historial.map((item, index) => (
            <li key={index}>
              {index + 1}. {item.num1} {item.operacion} {item.num2} = {item.resultado}
            </li>
          ))}
        </ul>
      </div>
    )}
  </div>
 );
}

export default App;
```

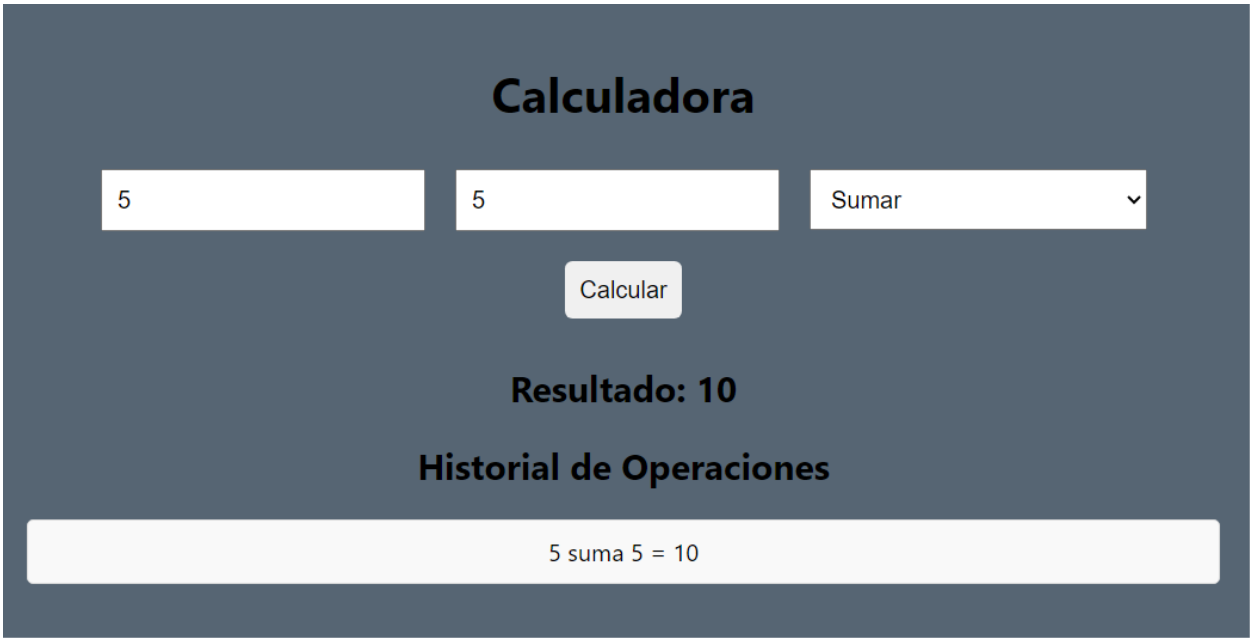
Calculadora.



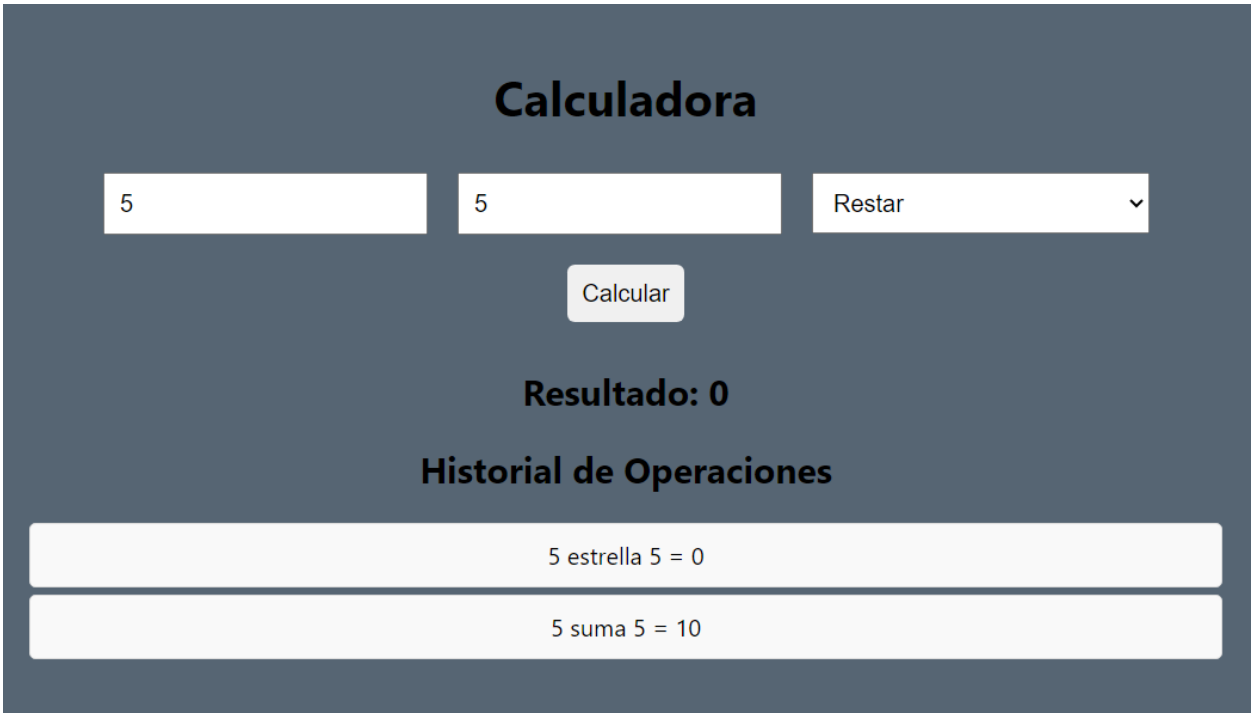
SI NO SE INGRESA ALGÚN NUM



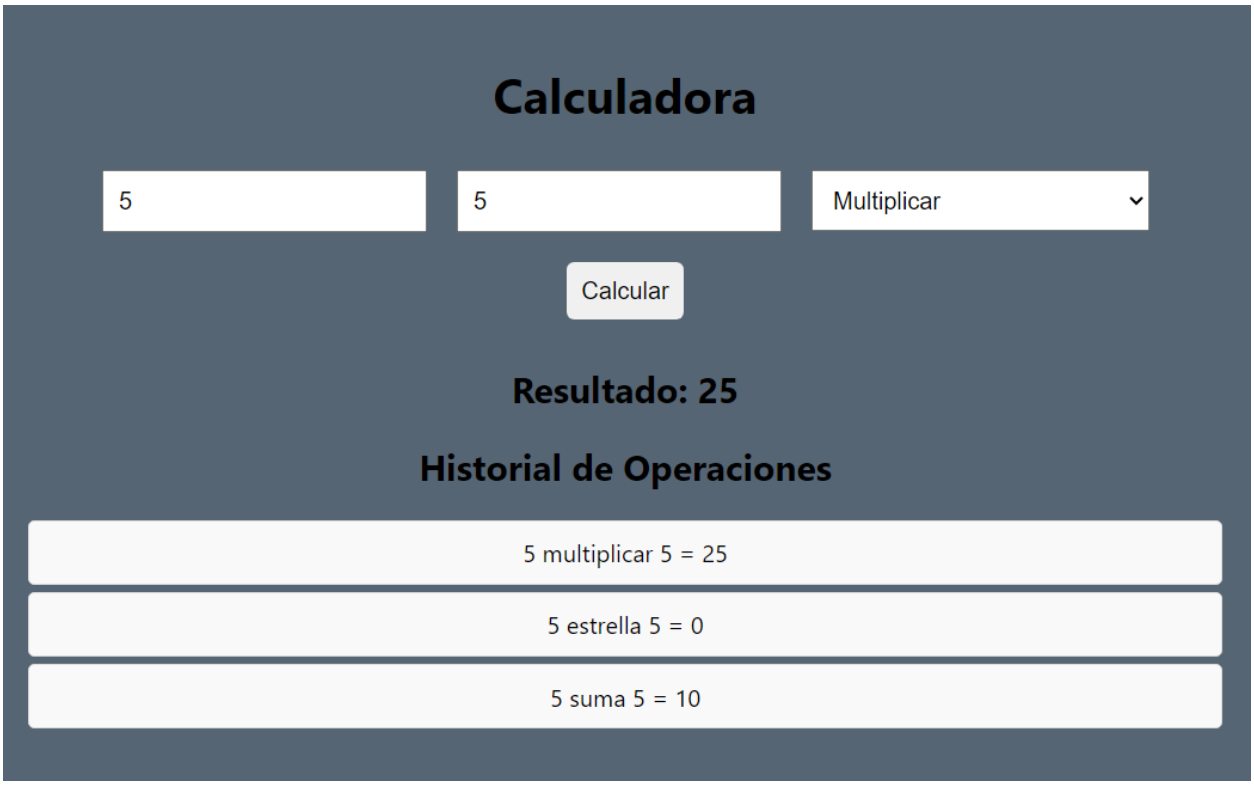
SUMA


Calculadora
5
5
Sumar
Calcular
Resultado: 10
Historial de Operaciones
5 suma 5 = 10

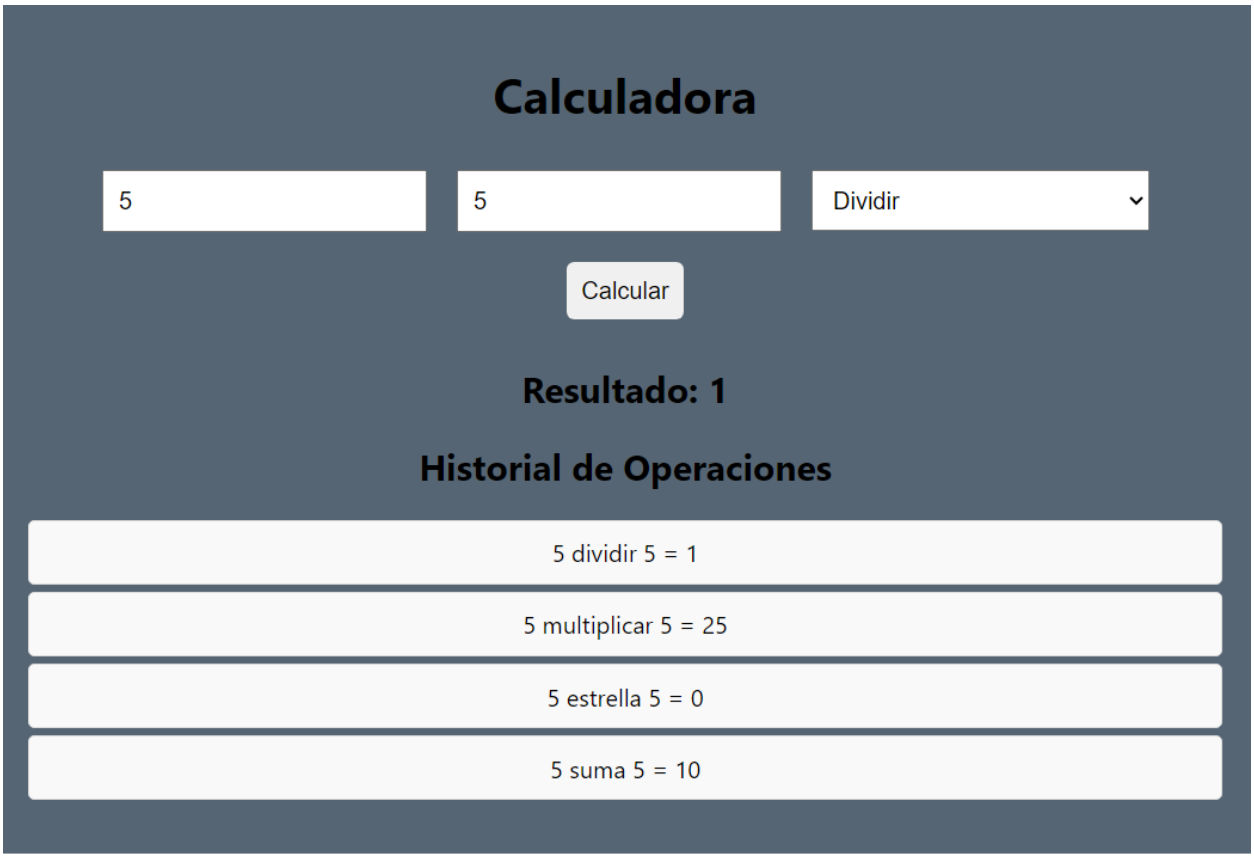
RESTA


Calculadora
5
5
Restar
Calcular
Resultado: 0
Historial de Operaciones
5 estrella 5 = 0
5 suma 5 = 10

MULTIPLICACION

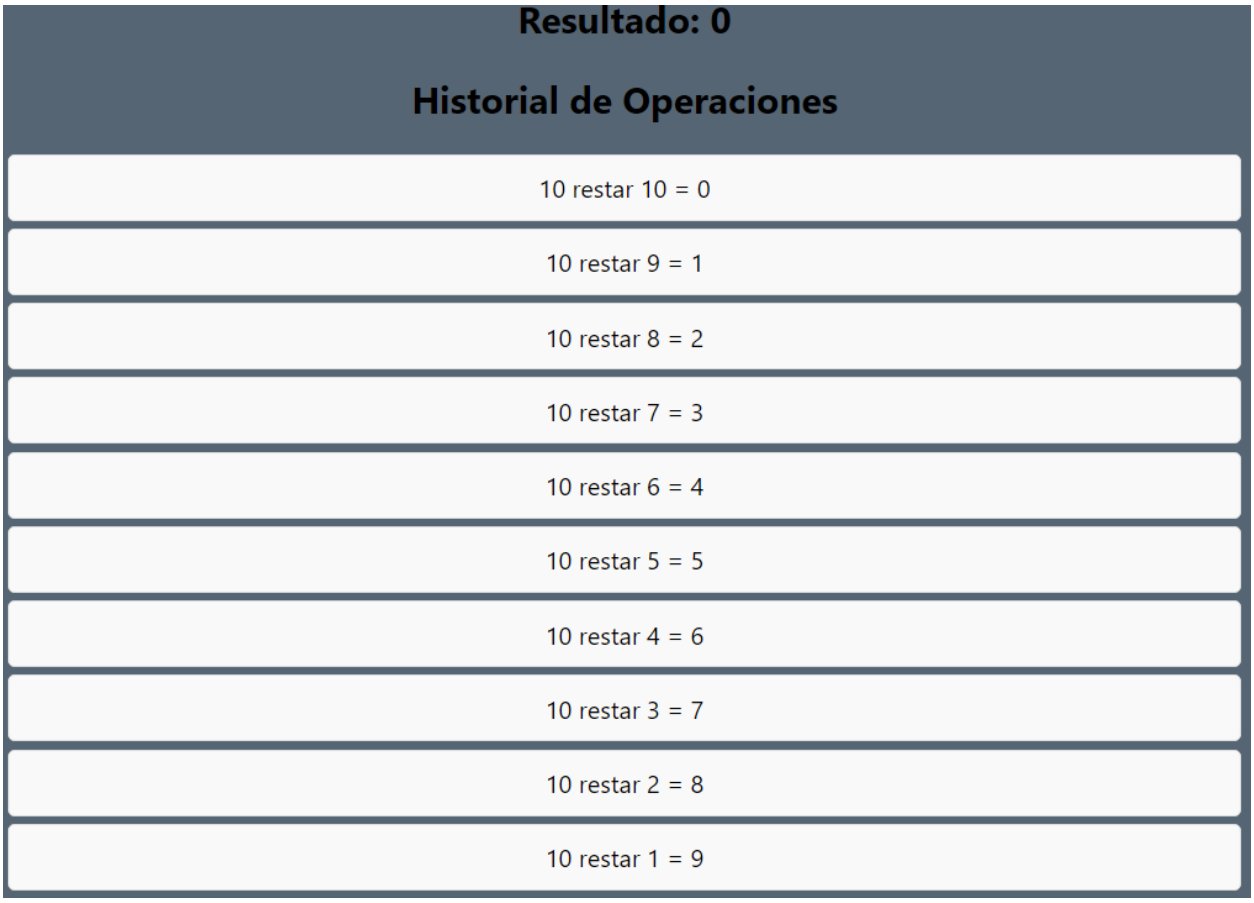


DIVISION



DIVIDIR ENTRE 0

HISTORIAL:



HITHUB: https://github.com/Egos808/Calculadora.git