Proyecto individual 1: Desarrollo de una Aplicación Full Stack de Calculadora con Java 8, Spring Framework y React.

Descripción: El objetivo de este proyecto es desarrollar una aplicación de calculadora utilizando tecnologías modernas en el desarrollo de software. El backend será implementado en **Java 8** con el **Spring Framework**, mientras que el frontend será desarrollado utilizando **React** para garantizar una experiencia de usuario fluida en aplicaciones web.

La calculadora deberá soportar operaciones aritméticas básicas (suma, resta, multiplicación y división) y mostrar los resultados en tiempo real. Además, la aplicación deberá contar con una interfaz de usuario intuitiva y fácil de usar.

Requerimientos Funcionales:

1. **Operaciones Aritméticas Básicas:** La calculadora debe permitir al usuario realizar operaciones de suma, resta, multiplicación y división.
2. **Validación de Entradas:** El sistema debe validar las entradas del usuario para evitar operaciones inválidas, como divisiones por cero.
3. **Historia de Operaciones:** La aplicación debe permitir visualizar el historial de las últimas 10 operaciones realizadas.
4. **Interfaz de Usuario:** El frontend debe ser responsivo, permitiendo una experiencia de usuario atractiva y funcional en navegadores web utilizando React.

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.

Rubrica de Evaluación (Máximo 8 Puntos):

1. **Funcionalidad de la Calculadora (3 puntos)** ○ [1 punto] Implementación correcta de las operaciones básicas (suma, resta, multiplicación, división).
    - [1 punto] Manejo adecuado de entradas inválidas y excepciones (como la división por cero).
    - [1 punto] Funcionalidad del historial de operaciones.
2. **Diseño e Implementación del Backend (2 puntos)** ○ [1 punto] Correcta implementación de la API RESTful utilizando Spring Framework.
    - [1 punto] Estructura clara y organizada del código en Java, siguiendo buenas prácticas de desarrollo.
3. **Diseño e Implementación del Frontend (2 puntos)** ○ [1 punto] Interfaz de usuario intuitiva y fácil de usar, desarrollada con React.
    - [1 punto] Integración efectiva con el backend, mostrando los resultados en tiempo real.
4. **Documentación y Despliegue (1 punto)** ○ [0.5 puntos] Documentación clara y detallada del proyecto, incluyendo cómo desplegar y probar la aplicación. ○ [0.5 puntos] Funcionamiento correcto del proyecto al ser desplegado y probado en un entorno local.

Entrega:

Los estudiantes deberán entregar el código fuente del proyecto en un repositorio de GitHub, junto con un documento que explique la arquitectura del proyecto e imágenes del funcionamiento, las tecnologías utilizadas, y cualquier instrucción necesaria para desplegar y probar la aplicación

DOCUMENTACION DE PROGRAMA

El presente proyecto tiene como objetivo el desarrollo de una calculadora web que permita realizar operaciones aritméticas básicas, brindando una interfaz atractiva y funcional para los usuarios.

La aplicación está diseñada para ofrecer una experiencia fluida y responsiva, asegurando la correcta validación de las entradas y evitando errores comunes como divisiones por cero.

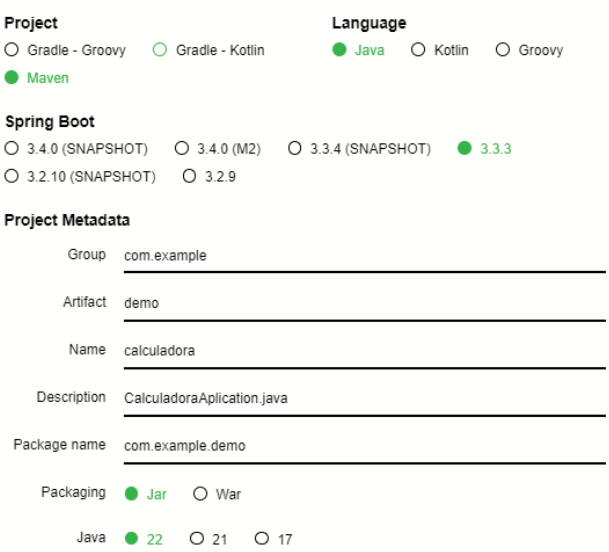
Además, cuenta con una funcionalidad de historial que permite visualizar las últimas 10 operaciones realizadas, mejorando la interacción del usuario con la herramienta.

El proyecto está construido utilizando tecnologías modernas. En el frontend, se ha utilizado **React** para desarrollar una interfaz dinámica y adaptable a diferentes dispositivos, mientras que en el backend se emplea **Java 8** junto con el **Spring Framework**, lo que facilita la creación de una API RESTful robusta y la gestión eficiente del flujo de datos entre el cliente y el servidor. La integración entre el frontend y el backend se realiza a través de solicitudes HTTP utilizando **fetch** o **Axios**, permitiendo la actualización en tiempo real de los resultados.

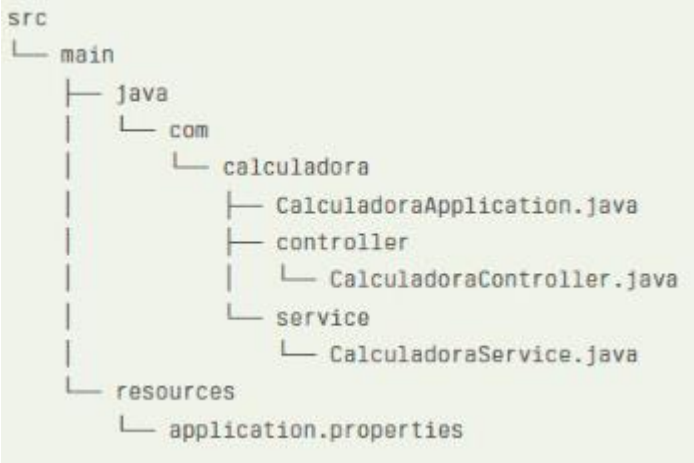
El despliegue del proyecto se realiza en un entorno local, asegurando que todos los componentes funcionen correctamente en conjunto.

A través de este proyecto, se busca demostrar la capacidad de implementar una solución completa que integre tanto aspectos de frontend como de backend, cumpliendo con los requerimientos funcionales y técnicos establecidos.

1) Ingresamos a SPRING INITIALIZR para crear nuestra dependencia esto con el fin de simplificar nuestra etapa inicial del proyecto.



2) Nos quedara una carpeta con nuestras dependencias:



3) A continuación, tendremos brevemente una explicación de los archivos que aparecen en la estructura proyecto:

CalculadoraApplication.java:

- Este es el archivo principal de la aplicación, comúnmente conocido como la clase **Spring Boot Application**. Es donde se encuentra el método main, que es el punto de entrada de la aplicación. Aquí es donde se configura y se inicia todo el proyecto Spring Boot.

CalculadoraController.java:

- Este archivo pertenece al paquete controller y es responsable de manejar las solicitudes HTTP que recibe el backend. Define los endpoints o rutas que permiten a los usuarios realizar operaciones a través de la API RESTful, como sumar, restar, multiplicar o dividir. El controlador toma las solicitudes del cliente y las delega al servicio adecuado para procesarlas.

CalculadoraService.java:

- Ubicado en el paquete service, este archivo contiene la lógica del negocio, es decir, las funciones que realizan las operaciones aritméticas (suma, resta, multiplicación, división). El servicio recibe las solicitudes del controlador, ejecuta la operación solicitada y devuelve los resultados.

application.properties:

- Este archivo se encuentra en la carpeta resources y contiene la configuración de la aplicación Spring Boot, como el puerto en el que se ejecutará el servidor, configuraciones de bases de datos, o cualquier otro parámetro relacionado con el entorno.

4) Realizamos la configuración de nuestro **Backend** para ser funcional el proyecto en el programa de **Intellij.**

- Modificación del código de nuestra Aplicación Java:

```java
package com.calculadora.carlossandoval;
// Define el paquete en el que se encuentra esta clase. Es una buena práctica organizar las clases en paquetes.

import org.springframework.boot.SpringApplication;
// Importa la clase SpringApplication, que se utiliza para lanzar la aplicación Spring Boot.
import org.springframework.boot.autoconfigure.SpringBootApplication;
// Importa la anotación SpringBootApplication, que habilita la configuración automática de Spring Boot.

@SpringBootApplication
// Esta anotación combina tres anotaciones:
// 1. @Configuration: Indica que la clase puede contener definiciones de beans.
// 2. @EnableAutoConfiguration: Permite que Spring Boot configure automáticamente la aplicación basada en las dependencias presentes.
// 3. @ComponentScan: Permite que Spring busque componentes, configuraciones y servicios en el paquete actual y sus subpaquetes.
public class CarlossandovalApplication {

    public static void main(String[] args) {
        // Método principal que se ejecuta al iniciar la aplicación.
        SpringApplication.run(CarlossandovalApplication.class, args);
        // Llama al método run de la clase SpringApplication, que inicia la aplicación Spring Boot.
        // El primer argumento es la clase principal y el segundo son los argumentos de línea de comandos.
    }

}
```

- Modificación del código de nuestra Calculadora Controller:

```java
package com.calculadora.carlossandoval.controlador;
import com.calculadora.carlossandoval.modelos.Operacion;
import com.calculadora.carlossandoval.servicios.CalculadoraService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
// Indica que esta clase es un controlador REST, lo que significa que maneja solicitudes HTTP
@RequestMapping("/api/calculadora")
// Define la ruta base para todas las solicitudes que se manejan en este controlador
public class CalculadoraController {

    @Autowired
    // Indica que Spring debe inyectar automáticamente una instancia de CalculadoraService en esta variable
    CalculadoraService calculadoraService;

    @PostMapping("/operacion")
    // Mapea las solicitudes HTTP POST a la ruta "/api/calculadora/operacion"
    public float realizarOperacion(@RequestBody Operacion operacion) {
        // Método que recibe un objeto Operacion en el cuerpo de la solicitud y devuelve un resultado como float
        }
        if (operacion.tipo.equals("multiplicacion")) {
            // Si el tipo de operación es "multiplicacion", llama al método multiplicacion del servicio de calculadora
            return calculadoraService.multiplicacion(operacion);
        }
        if (operacion.tipo.equals("division")) {
            // Si el tipo de operación es "division", llama al método division del servicio de calculadora
            return calculadoraService.division(operacion);
        }

        // Si el tipo de operación no coincide con ninguno de los anteriores, devuelve 0 como valor predeterminado
        return 0;
    }
}
```

- Modificación del código de nuestra Calculadora Service:

```java
package com.calculadora.carlossandoval.servicios;
// Define el paquete en el que se encuentra esta clase. Es una buena práctica organizar las clases en paquetes relacionados.

import org.springframework.stereotype.Service;
// Importa la anotación Service, que indica que esta clase es un componente de servicio en el contexto de Spring.
import com.calculadora.carlossandoval.modelos.Operacion;
// Importa la clase Operacion, que se espera que contenga los números y el tipo de operación a realizar.
import java.util.ArrayList;
import java.util.List;
// Importa las clases ArrayList y List, que se utilizan para manejar colecciones de elementos.

@Service  2 usages
// Indica que esta clase es un servicio de Spring, lo que significa que puede ser inyectada en otras clases como un bean.
public class CalculadoraService {
    // Declara una lista para almacenar el historial de operaciones realizadas.
    private List<String> historial = new ArrayList<>();  no usages

    // Método que realiza la suma de dos números.
    public float suma(Operacion operacion) {  1 usage
        // Devuelve la suma de los dos números contenidos en el objeto Operacion.
        return operacion.numero1 + operacion.numero2;
    }
```

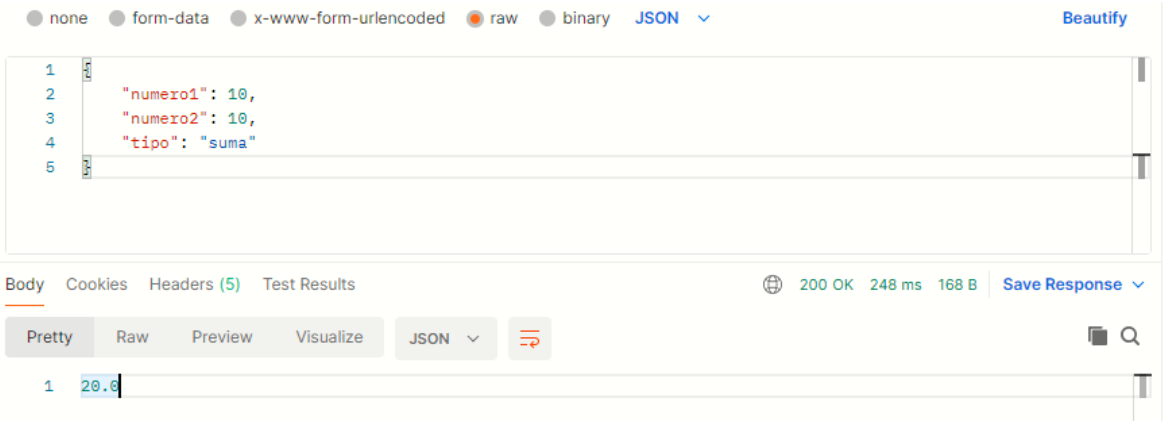
```java
    // Método que realiza la resta de dos números.
    public float resta(Operacion operacion) {  1 usage
        // Devuelve la resta del primer número menos el segundo número contenidos en el objeto Operacion.
        return operacion.numero1 - operacion.numero2;
    }

    // Método que realiza la multiplicación de dos números.
    public float multiplicacion(Operacion operacion) {  1 usage
        // Devuelve el producto de los dos números contenidos en el objeto Operacion.
        return operacion.numero1 * operacion.numero2;
    }

    // Método que realiza la división de dos números.
    public float division(Operacion operacion) {  1 usage
        // Devuelve el cociente del primer número dividido por el segundo número contenidos en el objeto Operacion.
        // Es recomendable agregar manejo de errores para evitar la división por cero.
        return operacion.numero1 / operacion.numero2;
    }
}
```

- Modificamos nuestras propiedades de aplicación

```
ing.application.name=carlossandoval
// Define el nombre de la aplicación. Este valor puede ser utilizado en diferentes partes de la aplicación, como en logs,
// mensajes de error, o para identificar la aplicación en un entorno de microservicios.

server.port=3000
// Especifica el puerto en el que la aplicación Spring Boot se ejecutará. En este caso, la aplicación escuchará en el puerto 3000.
// Esto es útil para evitar conflictos con otras aplicaciones que puedan estar ejecutándose en el mismo entorno,
// y permite a los desarrolladores y usuarios acceder a la aplicación a través de http://localhost:3000.
```

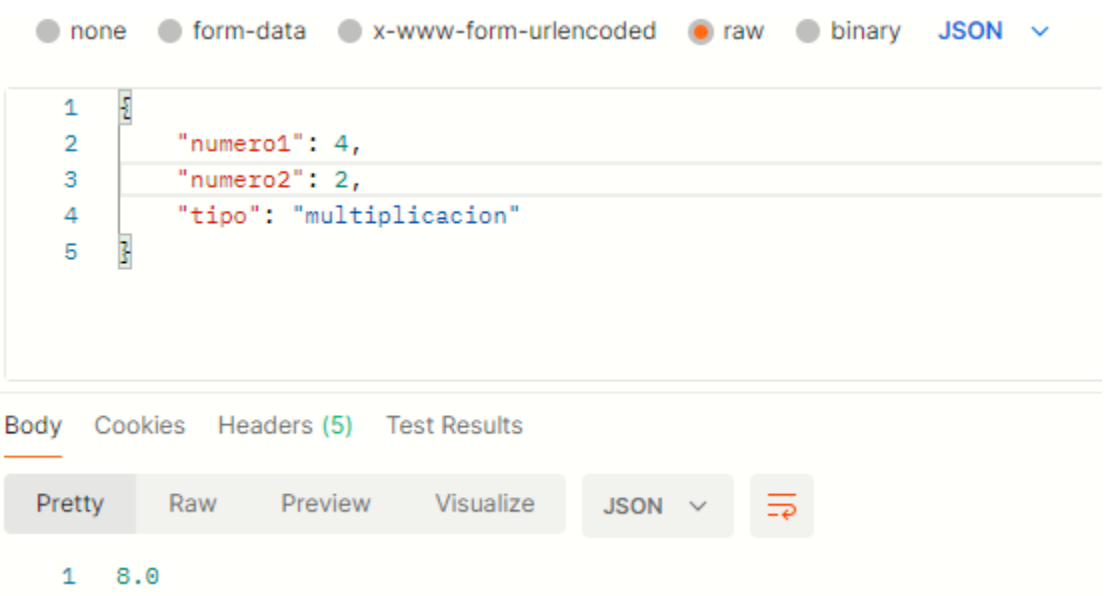
5) Ejecutamos nuestro programa Postman:

Realizamos las pruebas en la parte del **Backend** para verificar que nuestras operaciones aritméticas funcionan. **SUMA**

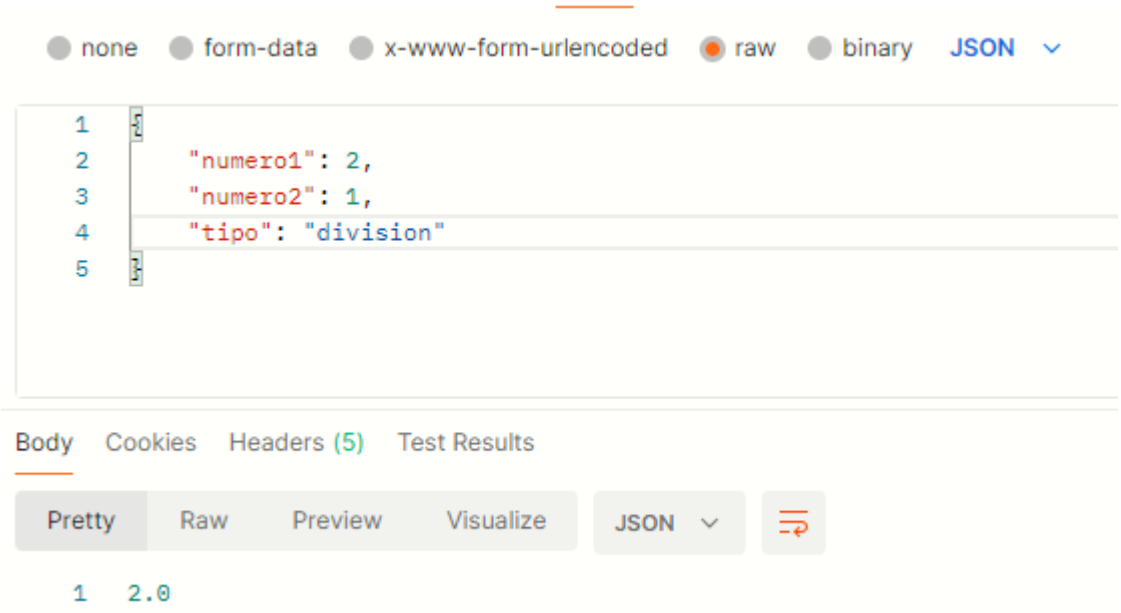


- **RESTA**

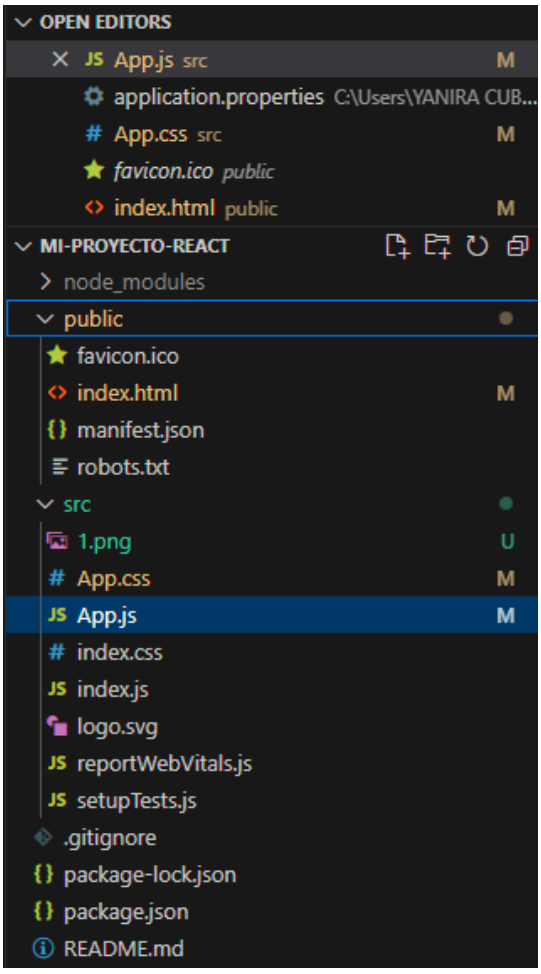
- **MULTIPLICACION**



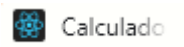
- **DIVISION**



6) Empezamos a realizar la configuración de nuestro **Frontend.** Para esto realizamos nuestro programa de VSCODE y creamos nuestras subcarpetas de nuestro proyecto:



7) Escogemos nuestro logo favicon:



8) Creamos nuestro archivo HTML este archivo va a ser nuestra estructura del proyecto **Frontend**:

```
<!DOCTYPE html>
<!-- Declara el tipo de documento como HTML5. Esto ayuda a los navegadores a interpretar el contenido correctamente. -->
<html lang="en">
<!-- Abre la etiqueta <html> y establece el atributo "lang" en "en", indicando que el contenido está en inglés. -->
  <head>
    <!-- La sección <head> contiene metadatos sobre el documento. -->
    <meta charset="utf-8" />
    <!-- Establece la codificación de caracteres del documento como UTF-8, que es compatible con la mayoría de los caracteres y símbolos. -->

    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <!-- Configura la vista para dispositivos móviles, asegurando que la aplicación sea responsiva.
         "width=device-width" establece el ancho del viewport al ancho del dispositivo,
         y "initial-scale=1" establece el nivel de zoom inicial. -->

    <meta name="theme-color" content="#000000" />
    <!-- Define el color del tema de la aplicación, que puede afectar la barra de herramientas del navegador en dispositivos móviles.
         En este caso, se establece en negro (#000000). -->

    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <!-- Proporciona una descripción de la página, que puede ser utilizada por motores de búsqueda y redes sociales para mostrar información
```

```
    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
    <!-- Define un icono para la aplicación que se mostrará en dispositivos Apple cuando se guarde en la pantalla de inicio.
         El valor "%PUBLIC_URL%" es un marcador de posición que se reemplazará con la URL pública de la aplicación. -->

    <link rel="stylesheet" type="text/css" href="styles.css">
    <!-- Enlaza un archivo CSS externo llamado "styles.css" para aplicar estilos a la aplicación. -->

    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
    <!-- Enlaza un archivo de manifiesto que proporciona información sobre la aplicación web, como su nombre, iconos y colores de tema.
         Esto es importante para las aplicaciones web progresivas (PWA). -->

    <title>Calculadora Carlos Sandoval</title>
    <!-- Establece el título de la página, que se mostrará en la pestaña del navegador. -->
  </head>
  <body>
    <!-- La sección <body> contiene el contenido visible de la página. -->
    <div id="root"></div>
    <!-- Crea un contenedor <div> con el id "root", que es donde se montará la aplicación React.
         React utiliza este contenedor para renderizar sus componentes. -->
  </body>
</html>
```

9) Creamos nuestro archivo CSS este archivo va a ser nuestro diseño del proyecto **Frontend**:

```css
body {
  background-image: url('/src/1.png');
  /* Establece una imagen de fondo para el cuerpo del documento. La imagen se carga desde la ruta especificada. */

  background-size: cover;
  /* Asegura que la imagen de fondo cubra todo el área del cuerpo, manteniendo su relación de aspecto. */

  background-repeat: no-repeat;
  /* Evita que la imagen de fondo se repita si es más pequeña que el área del cuerpo. */

  background-attachment: fixed;
  /* Establece que la imagen de fondo permanezca fija en su posición al desplazarse por la página. */
}

.error-message {
  color: red;
  /* Establece el color del texto de los mensajes de error en rojo, para que sean fácilmente visibles. */

  margin-top: 10px;
  /* Agrega un margen superior de 10 píxeles para separar el mensaje de error de otros elementos. */

  font-size: 1rem;
  /* Establece el tamaño de la fuente en 1 rem, que es relativo al tamaño de fuente del elemento raíz. */
}
```

```css
.App {
  display: flex;
  /* Utiliza el modelo de caja flexible (flexbox) para organizar los elementos hijos. */

  justify-content: center;
  /* Centra los elementos hijos horizontalmente en el contenedor. */

  align-items: center;
  /* Centra los elementos hijos verticalmente en el contenedor. */

  height: 100vh;
  /* Establece la altura del contenedor a 100% de la altura de la ventana del navegador. */
}
```

```css
.calculator {
  background-color: #a0a0a0;
  /* Establece un color de fondo gris claro para la calculadora. */

  border-radius: 10px;
  /* Aplica un radio de borde de 10 píxeles para esquinas redondeadas. */

  padding: 20px;
  /* Agrega un relleno interno de 20 píxeles alrededor del contenido de la calculadora. */

  width: 320px;
  /* Establece un ancho fijo de 320 píxeles para la calculadora. */

  border: 3px solid rgb(255, 252, 50);
  /* Agrega un borde de 3 píxeles de grosor y color amarillo claro alrededor de la calculadora. */
}
```

```css
.calculator-screen {
  width: 90%;
  /* Establece el ancho de la pantalla de la calculadora al 90% del ancho del contenedor padre. */

  height: 80px;
  /* Establece una altura fija de 80 píxeles para la pantalla de la calculadora. */

  background-color: #252525;
  /* Establece un color de fondo oscuro para la pantalla de la calculadora. */

  color: rgb(255, 252, 50);
  /* Establece el color del texto en la pantalla a amarillo claro. */

  font-size: 3rem;
  /* Establece un tamaño de fuente grande para el texto en la pantalla. */

  border: none;
  /* Elimina el borde predeterminado de la pantalla. */

  border-radius: 5px;
  /* Aplica un radio de borde de 5 píxeles para esquinas redondeadas. */

  padding: 15px;
  /* Agrega un relleno interno de 15 píxeles alrededor del contenido de la pantalla. */

  margin: 0 auto 10px auto;
  /* Establece márgenes automáticos a los lados (centrando horizontalmente) y un margen inferior de 10 píxeles. */

  text-align: right;
  /* Alinea el texto a la derecha, que es común en las pantallas de calculadoras. */
}
```

```css
.calculator-keys {
  display: grid;
  /* Utiliza un diseño de cuadrícula para organizar las teclas de la calculadora. */

  grid-template-columns: repeat(4, 1fr);
  /* Define 4 columnas de igual tamaño en la cuadrícula. */

  grid-gap: 10px;
  /* Establece un espacio de 10 píxeles entre las filas y columnas de la cuadrícula. */
}
```

```css
button {
  /* Establece una altura fija de 60 píxeles para todos los botones. */

  background-color: #e5e6e0;
  /* Establece un color de fondo gris claro para los botones. */

  border: none;
  /* Elimina el borde predeterminado de los botones. */

  border-radius: 5px;
  /* Aplica un radio de borde de 5 píxeles para esquinas redondeadas. */

  font-size: 1.5rem;
  /* Establece un tamaño de fuente grande para el texto en los botones. */

  box-shadow: 0 5px rgba(0, 0, 0, 0.1);
  /* Agrega una sombra sutil para dar un efecto de elevación a los botones. */

  cursor: pointer;
  /* Cambia el cursor a una mano al pasar sobre el botón, indicando que es interactivo. */

  transition: background-color 0.3s ease;
  /* Añade una transición suave para cambios en el color de fondo. */
}
```

```css
button:active {
  background-color: #d1d2d7;
  /* Cambia el color de fondo cuando el botón es presionado (estado activo). */

  box-shadow: none;
  /* Elimina la sombra al presionar el botón, para un efecto de "hundimiento". */
}

button.key-operator {
  background-color: #f08a5d;
  /* Establece un color de fondo diferente (naranja) para los botones de operación. */

  color: white;
  /* Cambia el color del texto a blanco para mejorar la visibilidad sobre el fondo naranja. */
}
```

```css
button.key-equal {
  background-color: #f08a5d;
  /* Establece el mismo color de fondo (naranja) para el botón de igual. */

  color: black;
  /* Cambia el color del texto a negro para un buen contraste. */
}

button.key-clear {
  background-color: #f08a5d;
  /* Establece el mismo color de fondo (naranja) para el botón de limpiar. */

  color: white;
  /* Cambia el color del texto a blanco para mejorar la visibilidad. */
}

button.span-2 {
  grid-column: span 2;
  /* Hace que el botón ocupe el espacio de 2 columnas en la cuadrícula, útil para botones como "0" o "igual". */
}
```

```css
button:hover {
  background-color: #d1d2d7;
  /* Cambia el color de fondo al pasar el mouse sobre el botón, para indicar que es interactivo. */
}
```

- En este mismo archivo css se agregó la siguiente imagen para fondo de pantalla de nuestro programa:

10) Creamos nuestro archivo JS este archivo va a ser nuestra operación del proyecto **Frontend**:

```
JS App.js > App > handleButtonClick
import React, { useState } from 'react';
// Importa React y el hook useState desde la biblioteca de React. useState se utiliza para manejar el estado en componentes funcionales.
import './App.css';
// Importa el archivo de estilos CSS para aplicar estilos a este componente.

function App() {
  // Declara el estado del componente utilizando useState.
  const [input, setInput] = useState('');
  // Estado para almacenar la entrada actual del usuario en la calculadora.

  const [history, setHistory] = useState([]);
  // Estado para almacenar el historial de operaciones realizadas.
```

```
const [errorMessage, setErrorMessage] = useState('');
// Estado para almacenar mensajes de error que puedan ocurrir durante los cálculos.

const handleButtonClick = (value) => {
  // Función que se llama cuando se hace clic en un botón de número u operador.
  setInput(input + value); // Actualiza el estado de entrada concatenando el valor del botón.
  setErrorMessage(''); // Limpia el mensaje de error al ingresar un nuevo valor.
};

const handleCalculate = () => {
  // Función que se llama al hacer clic en el botón de igual para calcular el resultado.
  try {
    // Intenta ejecutar el bloque de código a continuación.
    if (input.includes('/0')) {
      // Verifica si la entrada contiene una división por cero.
      setErrorMessage("No se puede realizar su operación: división por cero");
      return; // Sale de la función si se detecta una división por cero.
    }
    const resultado = eval(input);
    // Evalúa la expresión matemática en la entrada. (Nota: eval puede ser peligroso si no se controla adecuadamente la entrada del usuario).
    setInput(resultado.toString());
    // Actualiza la entrada con el resultado convertido a cadena.

    const newHistory = [...history, `${input} = ${resultado}`];
    // Crea un nuevo historial que incluye la operación actual y su resultado.

    if (newHistory.length > 10) {
      newHistory.shift();
      // Si el historial supera las 10 entradas, elimina la más antigua.
    }
    setHistory(newHistory); // Actualiza el estado del historial.
  } catch (error) {
    // Captura cualquier error que ocurra durante la evaluación.
    console.error("Error en la expresión:", error);
    setInput("Error"); // Muestra "Error" en la entrada si ocurre un error.
  }
};

const handleClear = () => {
  // Función que se llama al hacer clic en el botón de limpiar.
  setInput(''); // Limpia la entrada.
  setErrorMessage(''); // Limpia el mensaje de error.
};

const showHistory = () => {
  // Función que muestra el historial de operaciones en un cuadro de alerta.
  alert(history.join('\n')); // Une las operaciones en el historial y las muestra en un cuadro de alerta.
};

return (
  <div className="App">
    {/* Contenedor principal de la aplicación */}
    <div className="calculator">
      {/* Contenedor de la calculadora */}
      <input type="text" className="calculator-screen" value={input} readOnly />
      {/* Campo de entrada que muestra el valor actual. Es de solo lectura para evitar ediciones manuales. */}
      <div className="calculator-keys">
        {/* Contenedor de las teclas de la calculadora */}
        <button onClick={handleClear} className="key-clear span-2">C</button>
        {/* Botón para limpiar la entrada */}
        <button onClick={() => handleButtonClick('/')} className="key-operator">/</button>
        {/* Botón para la división */}
        <button onClick={() => handleButtonClick('*')} className="key-operator">*</button>
        {/* Botón para la multiplicación */}
```

```jsx
<button onClick={() => handleButtonClick('7')}>7</button>
<button onClick={() => handleButtonClick('8')}>8</button>
<button onClick={() => handleButtonClick('9')}>9</button>
<button onClick={() => handleButtonClick('-')} className="key-operator">-</button>
{/* Botón para la resta */}

<button onClick={() => handleButtonClick('4')}>4</button>
<button onClick={() => handleButtonClick('5')}>5</button>
<button onClick={() => handleButtonClick('6')}>6</button>
<button onClick={() => handleButtonClick('+')} className="key-operator">+</button>
{/* Botón para la suma */}

<button onClick={() => handleButtonClick('1')}>1</button>
<button onClick={() => handleButtonClick('2')}>2</button>
<button onClick={() => handleButtonClick('3')}>3</button>
```

```jsx
            <button onClick={() => handleButtonClick('0')} className="span-2">0</button>
            <button onClick={() => handleButtonClick('.')}>.</button>
            {/* Botón para el punto decimal */}
            <button onClick={handleCalculate} className="key-equal">=</button>
            {/* Botón para calcular el resultado */}
            <button onClick={showHistory} className="key-history span-2">Historial</button>
            {/* Botón para mostrar el historial de operaciones */}
          </div>
          {errorMessage && <div className="error-message">{errorMessage}</div>}
          {/* Muestra el mensaje de error si existe */}
        </div>
      </div>
    );
  }

  export default App;
  // Exporta el componente App para que pueda ser utilizado en otras partes de la aplicación.
```

11) Modificamos nuestro **INDEX.CSS**

```css
body {
  margin: 0;
  /* Elimina el margen predeterminado del navegador alrededor del cuerpo del documento.
  Esto asegura que el contenido ocupe todo el espacio disponible sin espacios en blanco no deseados. */

  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  /* Establece la fuente del texto del cuerpo. La lista incluye varias fuentes del sistema y fuentes sa
     -apple-system y BlinkMacSystemFont son fuentes del sistema para dispositivos Apple,
     mientras que las otras son fuentes comunes en diferentes sistemas operativos.
     Si la primera fuente no está disponible, el navegador intentará usar la siguiente en la lista. */

  -webkit-font-smoothing: antialiased;
  /* Mejora la suavidad de las fuentes en navegadores WebKit (como Chrome y Safari),
     haciendo que las fuentes se vean más lisas y menos pixeladas. */

  -moz-osx-font-smoothing: grayscale;
  /* Mejora la suavidad de las fuentes en navegadores Firefox en macOS,
     utilizando un suavizado en escala de grises para hacer que las fuentes se vean más agradables. */
}
```

```
code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
  /* Establece la fuente para el texto dentro de elementos <code>.
     Estas fuentes son típicamente utilizadas para mostrar código fuente y texto monoespaciado.
     La lista incluye fuentes populares para desarrollo y programación,
     comenzando con source-code-pro, que es una fuente diseñada específicamente para código. */
}
```

12) Modificamos nuestro **INDEX.JS.**

```
import React from 'react';
// Importa la biblioteca React, que es necesaria para crear componentes de React y utilizar JSX.

import ReactDOM from 'react-dom/client';
// Importa el módulo ReactDOM, que proporciona métodos para interactuar con el DOM y renderizar componentes de React en la página.

import './index.css';
// Importa un archivo CSS para aplicar estilos globales a la aplicación. Este archivo puede contener estilos que afectan a toda la aplicación.

import App from './App';
// Importa el componente principal de la aplicación, que normalmente contiene la estructura y la lógica de la aplicación.

import reportWebVitals from './reportWebVitals';
// Importa una función para medir el rendimiento de la aplicación. Esto es útil para analizar y optimizar el rendimiento de la aplicación.

const root = ReactDOM.createRoot(document.getElementById('root'));
// Crea un nodo raíz para la aplicación React. Utiliza el método createRoot para inicializar un contenedor React en el elemento del DOM con el id
```

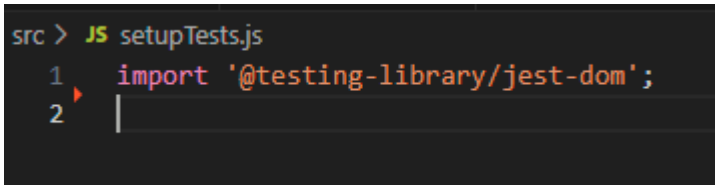
```
root.render(
  <React.StrictMode>
    {/* Envuelve la aplicación en <React.StrictMode>, que es una herramienta para identificar problemas potenciales en la aplicación.
        Ayuda a resaltar advertencias y errores durante el desarrollo. */}
    <App />
    {/* Renderiza el componente <App>, que es el componente principal de la aplicación. */}
  </React.StrictMode>
);

reportWebVitals();
// Llama a la función reportWebVitals para medir y registrar el rendimiento de la aplicación. Esto puede ayudar a identificar áreas de mejora.
```

13) **REPORTWEBVITAL.JS** es una aplicación que React tiene como propósito principal medir y reportar métricas de rendimiento de la aplicación.

```
const reportWebVitals = onPerfEntry => {
  // Define una función llamada reportWebVitals que toma un parámetro onPerfEntry.

  if (onPerfEntry && onPerfEntry instanceof Function) {
    // Verifica si onPerfEntry está definido y si es una función.

    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB }) => {
      // Importa dinámicamente el módulo 'web-vitals' y desestructura las funciones de medición de métricas.

      getCLS(onPerfEntry);
      // Llama a la función getCLS (Cumulative Layout Shift) pasando onPerfEntry como argumento.

      getFID(onPerfEntry);
      // Llama a la función getFID (First Input Delay) pasando onPerfEntry como argumento.

      getFCP(onPerfEntry);
      // Llama a la función getFCP (First Contentful Paint) pasando onPerfEntry como argumento.

      getLCP(onPerfEntry);
      // Llama a la función getLCP (Largest Contentful Paint) pasando onPerfEntry como argumento.

      getTTFB(onPerfEntry);
      // Llama a la función getTTFB (Time to First Byte) pasando onPerfEntry como argumento.
    });
  }
};
```

```
export default reportWebVitals
// Exporta la función reportWebVitals como el valor predeterminado, para que pueda ser importada en otros módulos.
```

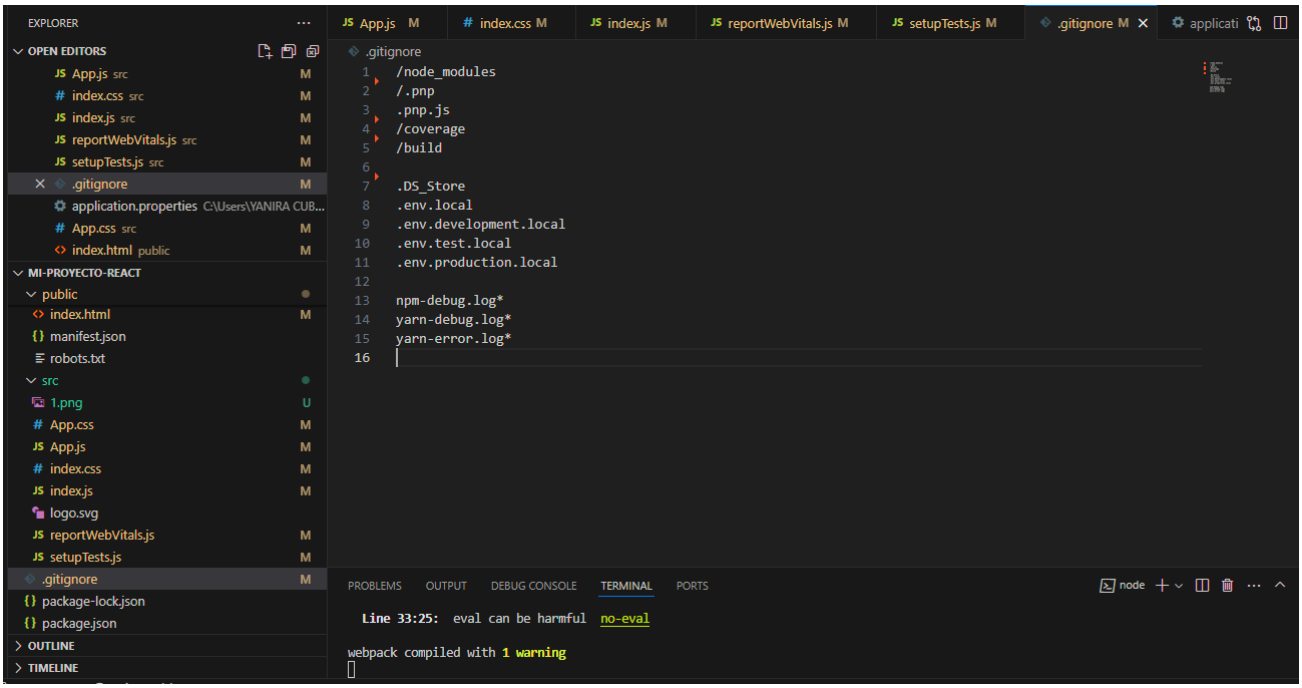
14) Agregamos nuestro archivo **SETUPTEST.JS** El archivo setupTests.js en una aplicación React, especialmente aquellas creadas con Create React App (CRA), se utiliza para configurar el entorno de pruebas antes de que se ejecuten las pruebas.



MANUAL DE USO DEL PROGRAMA

15) Realizando esto procederemos con el procedimiento para la ejecución de nuestro programa utilizando esta tecnología al igual que los programas:

- Abrimos nuestro programa VSCODE



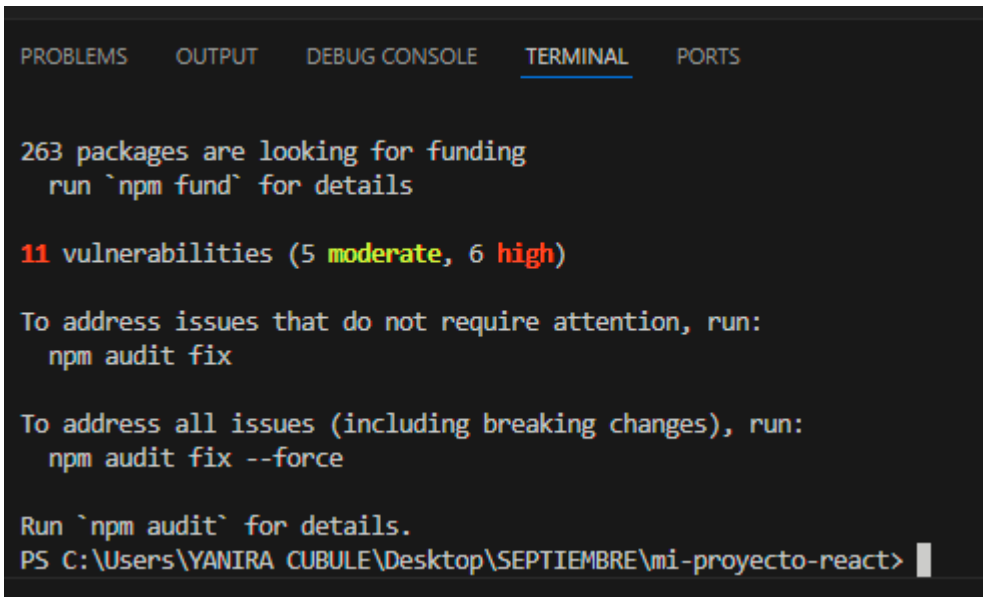
- Al momento de estar en el programa debemos de presionar la tecla **CONTROL + Ñ** para que se pueda abrir nuestra terminal dentro del programa:

- Al momento que se abra nuestra terminal debemos de correr el siguiente comando: **Npm install**

El comando npm install se utiliza para instalar paquetes (también conocidos como módulos) que se encuentran en el registro de npm. Esto incluye bibliotecas y herramientas.

El comando npm install es una herramienta esencial para cualquier desarrollador que trabaje con Node.js y JavaScript. Permite la instalación y gestión eficiente de paquetes, facilitando el desarrollo de aplicaciones y la colaboración en proyectos.

Nos desplegara el siguiente mensaje:



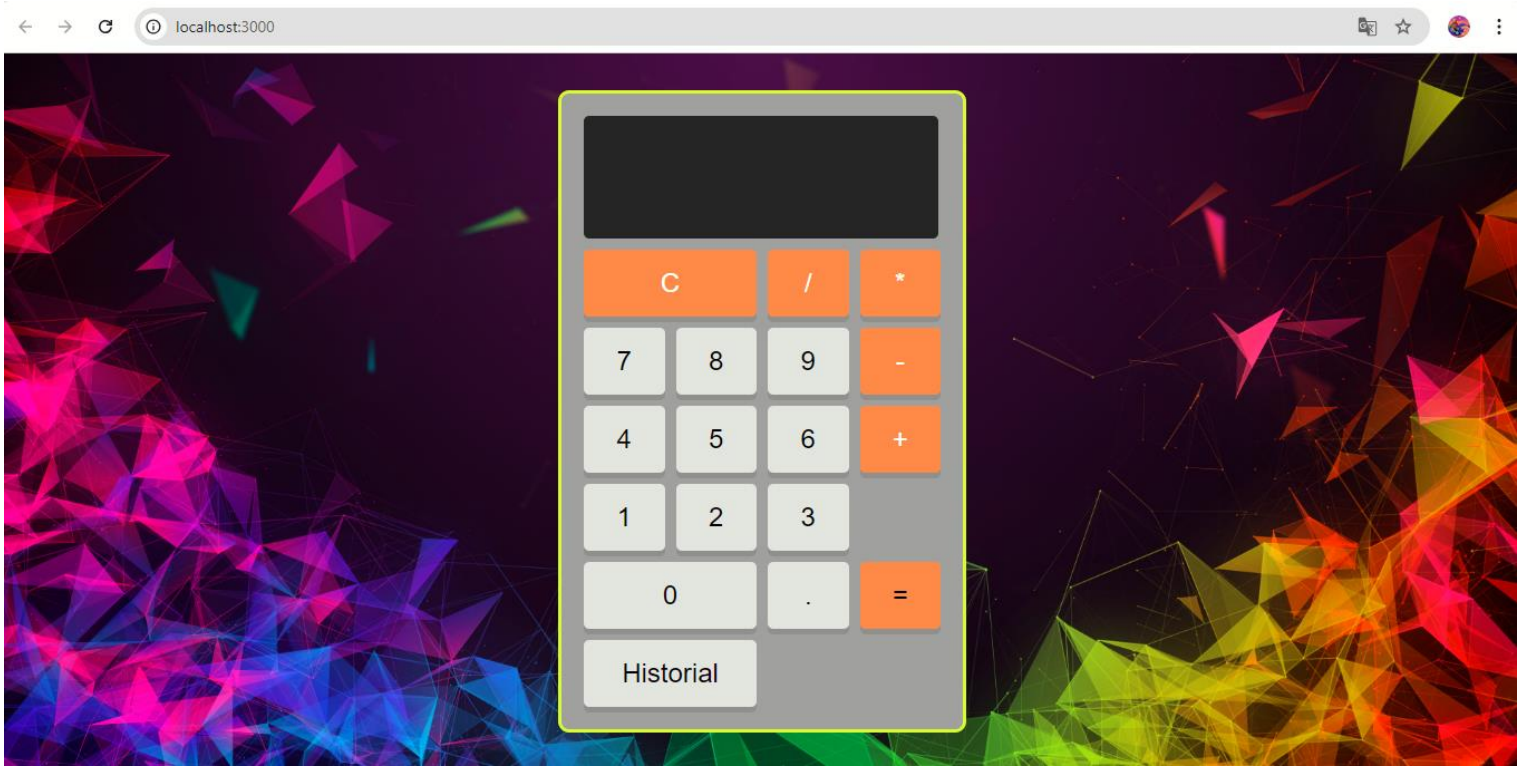
- Luego procederemos a correr el siguiente código: **npm star**

El comando npm start se utiliza para iniciar la aplicación definida en el archivo package.json del proyecto. Este comando ejecuta el script que está asociado a la propiedad start en la sección de scripts de package.json.

En nuestra terminal nos aparecerá el siguiente mensaje:

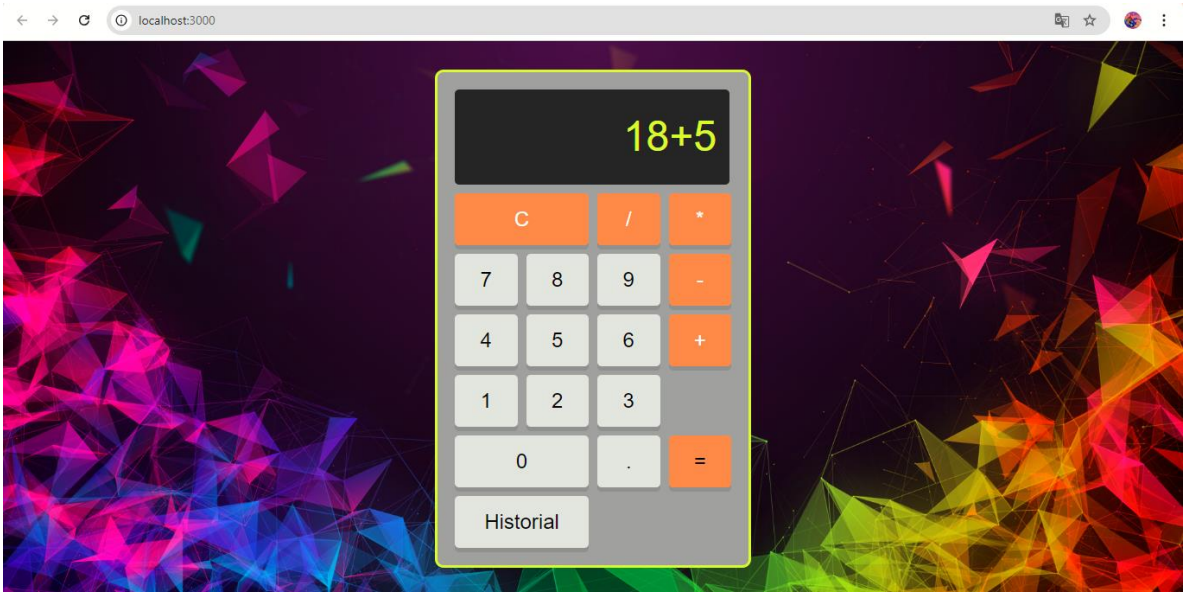
```
Compiled successfully!

You can now view mi-proyecto-react in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

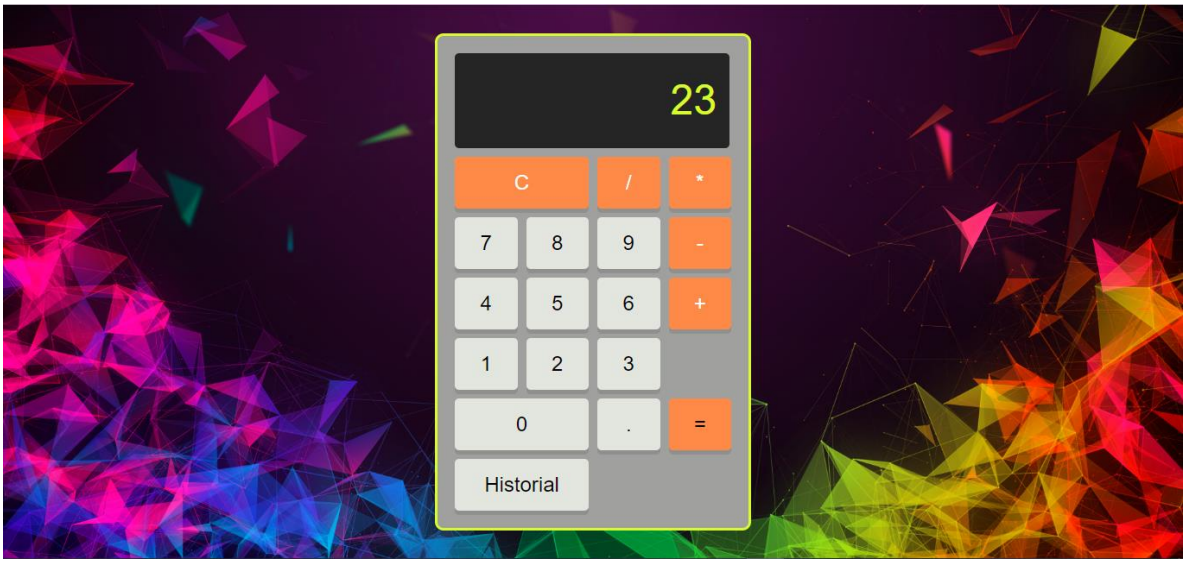
- Al momento que nos aparezca este mensaje automáticamente se abrirá el navegador con nuestro programa listo para utilizar:



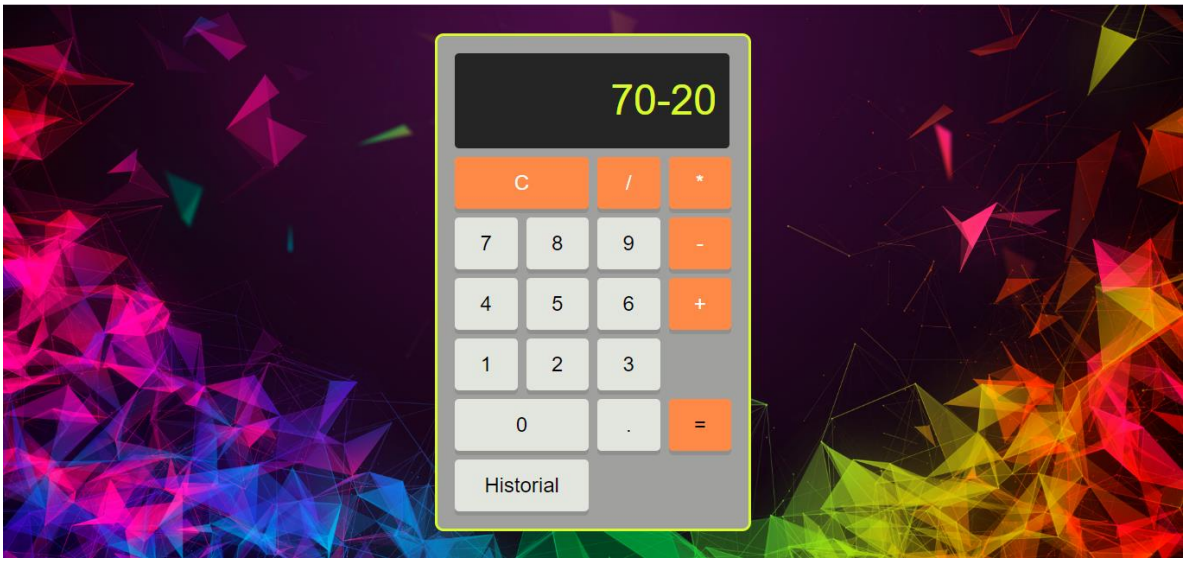
- Estando aquí podemos utilizar la calculadora con operaciones aritméticas como suma, resta, multiplicación, división o bien también podemos utilizar operaciones mixtas entre ellas.

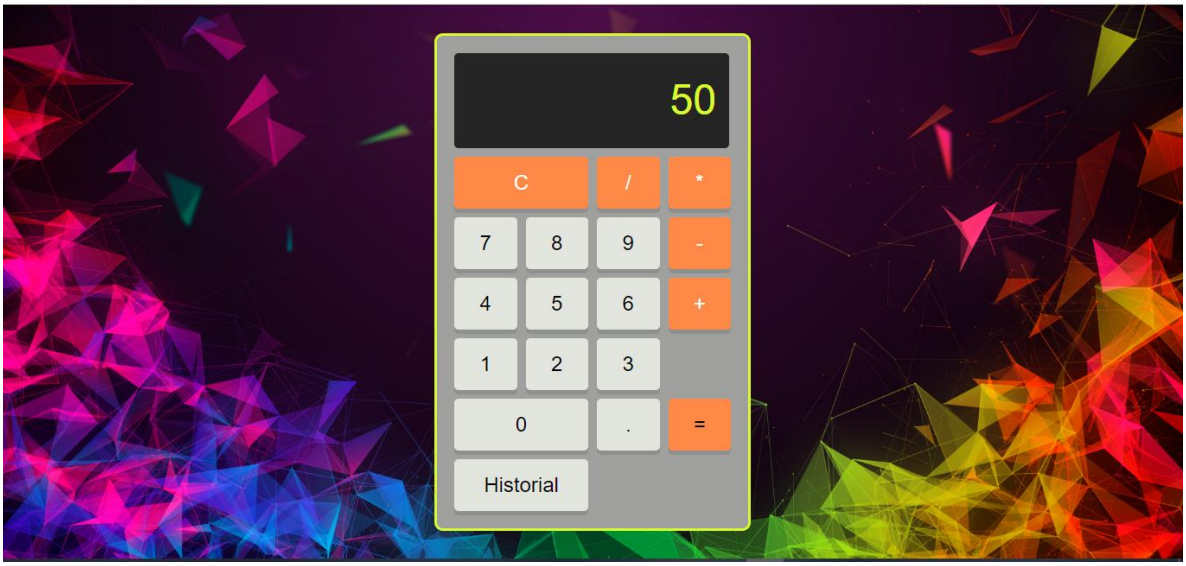
- **SUMA**



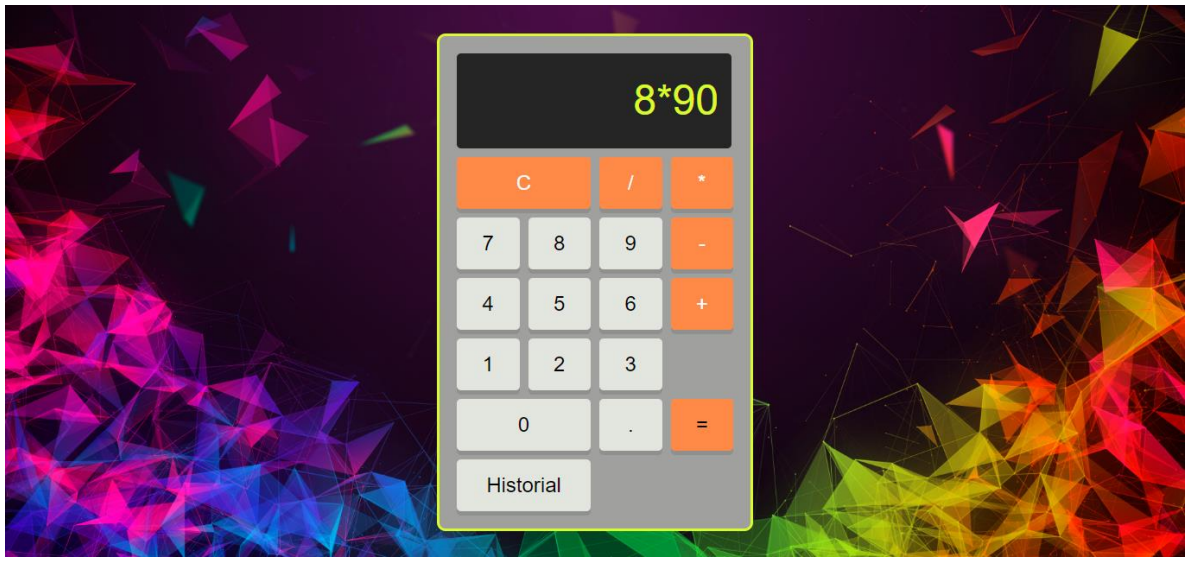


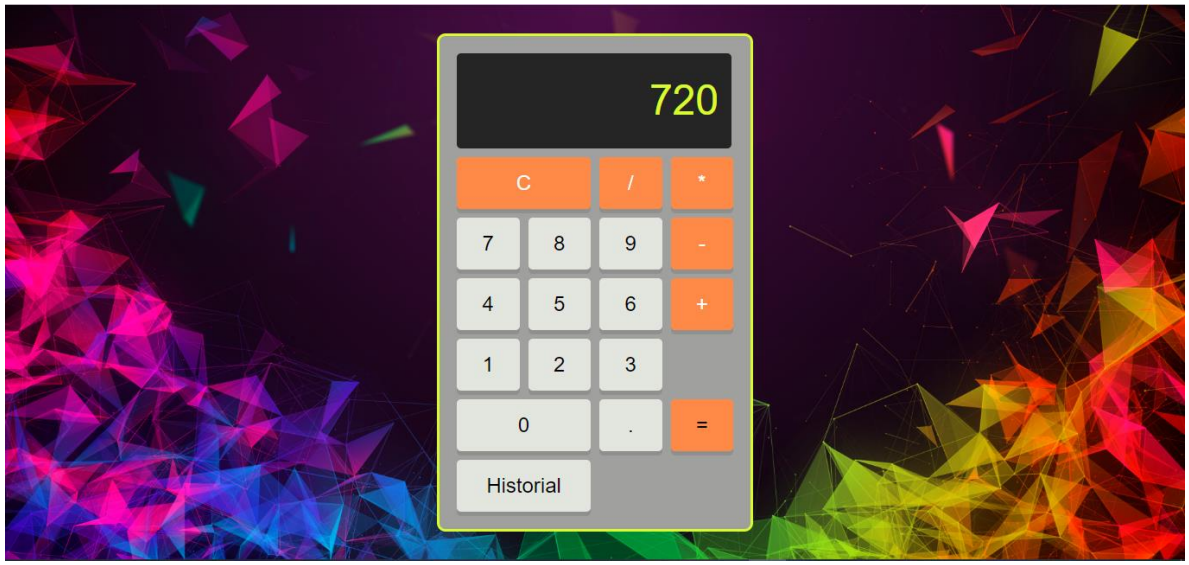
- **RESTA**



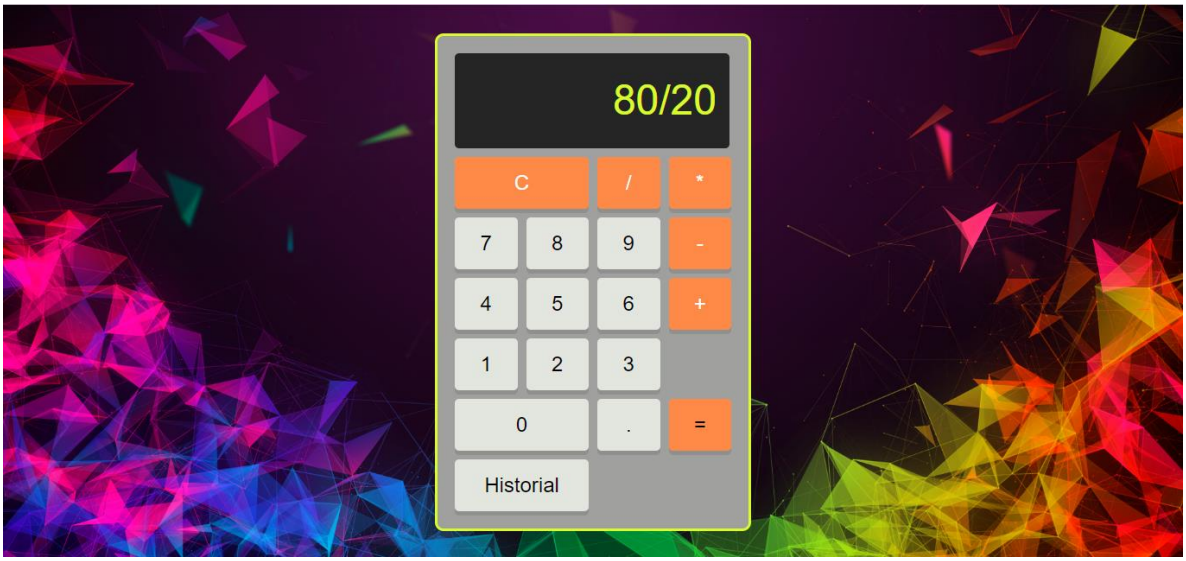


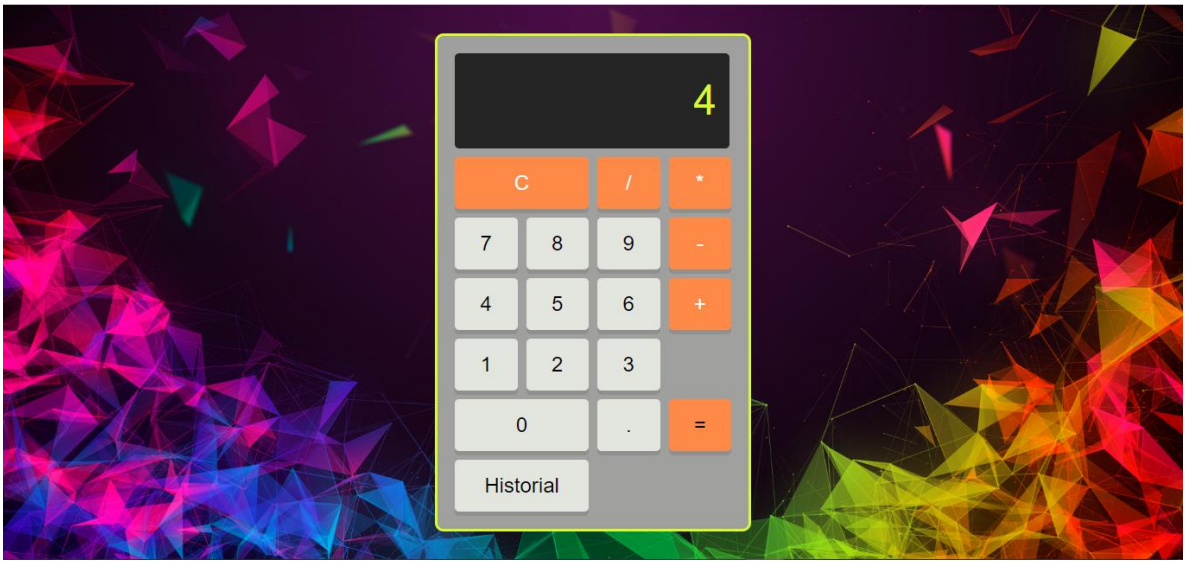
- **MULTIPLICACION**



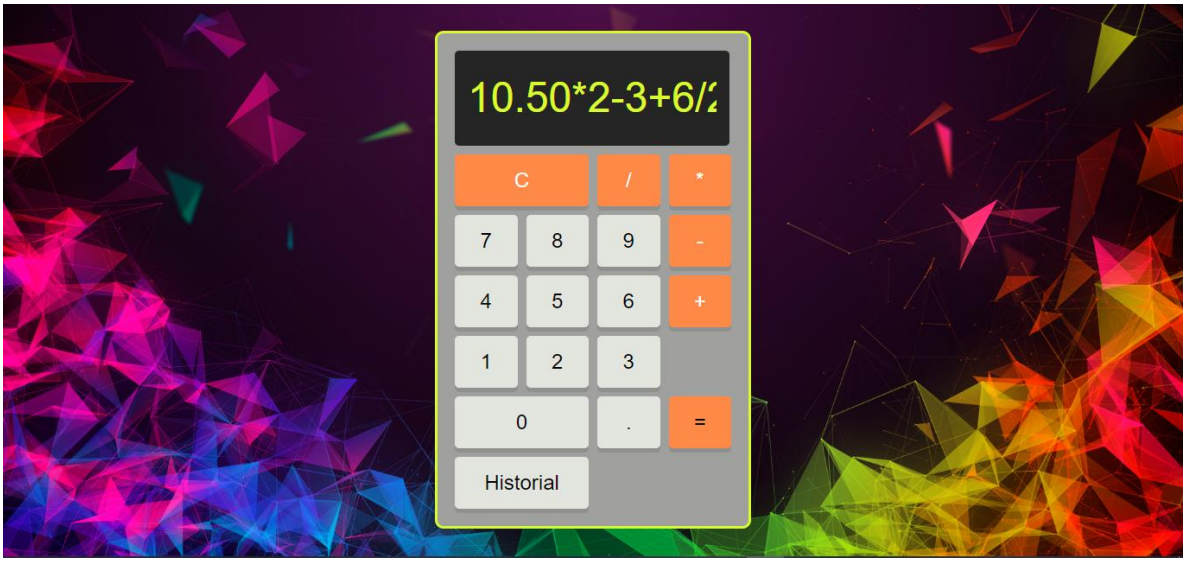


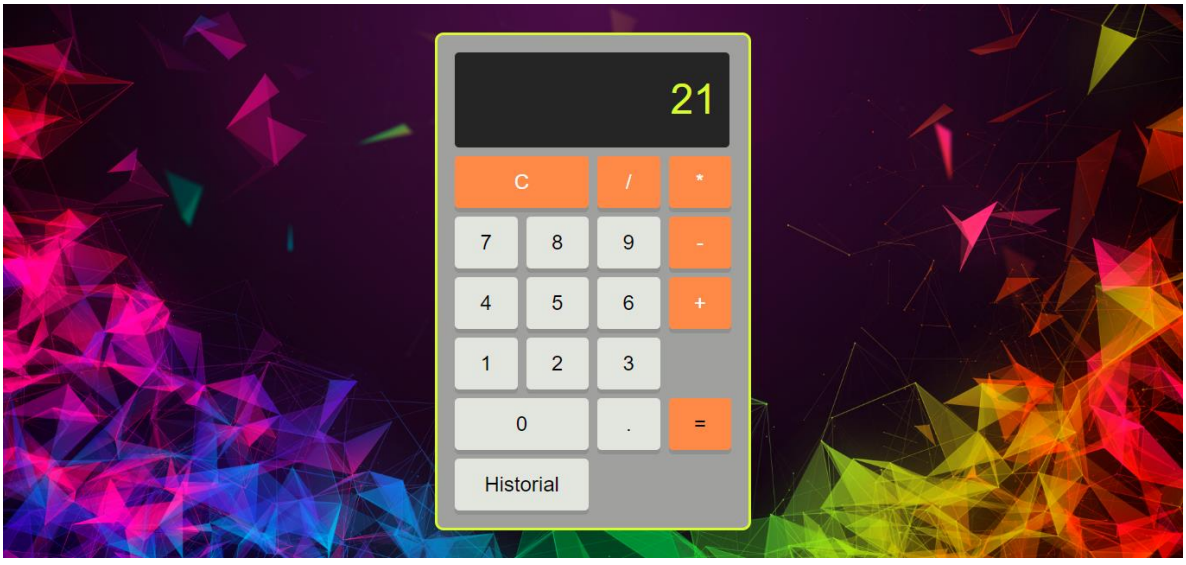
- **DIVISION**



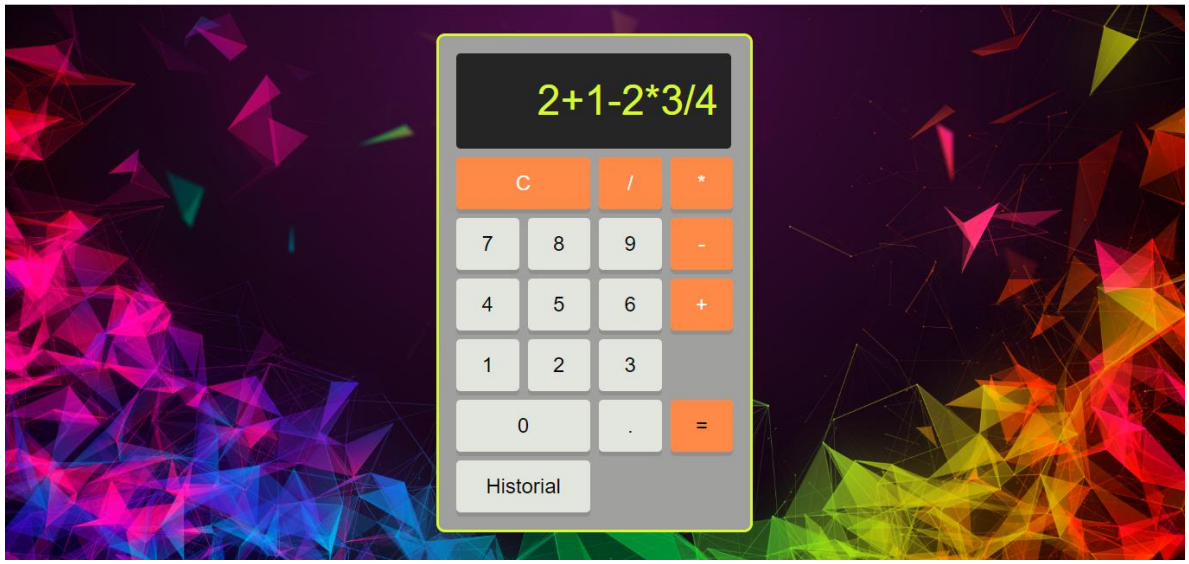


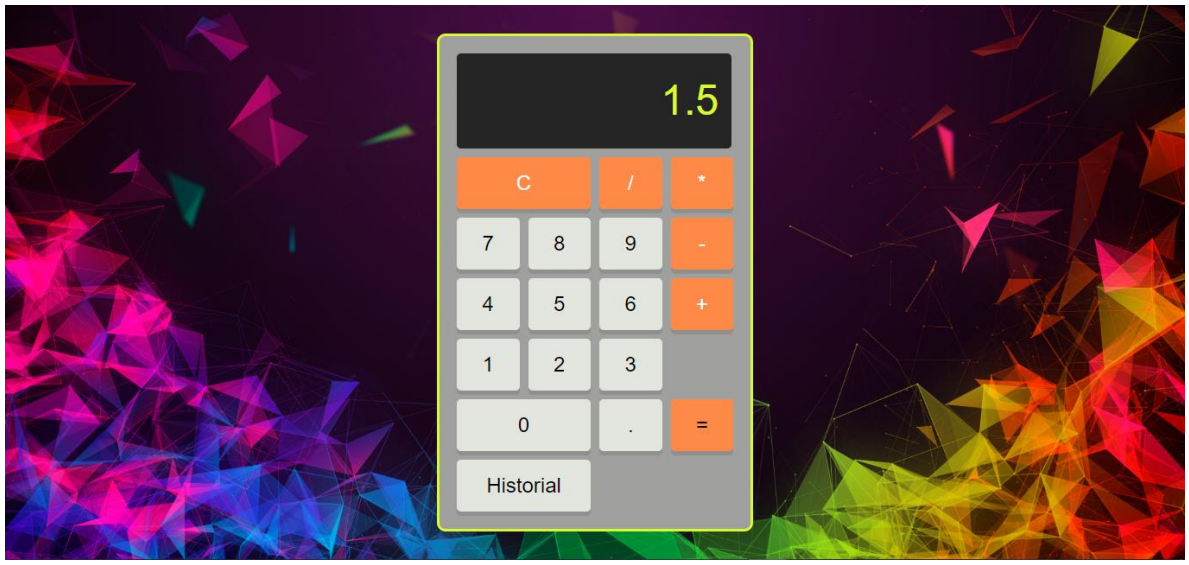
- **OPERACIÓN MIXTA CON DECIMAL**



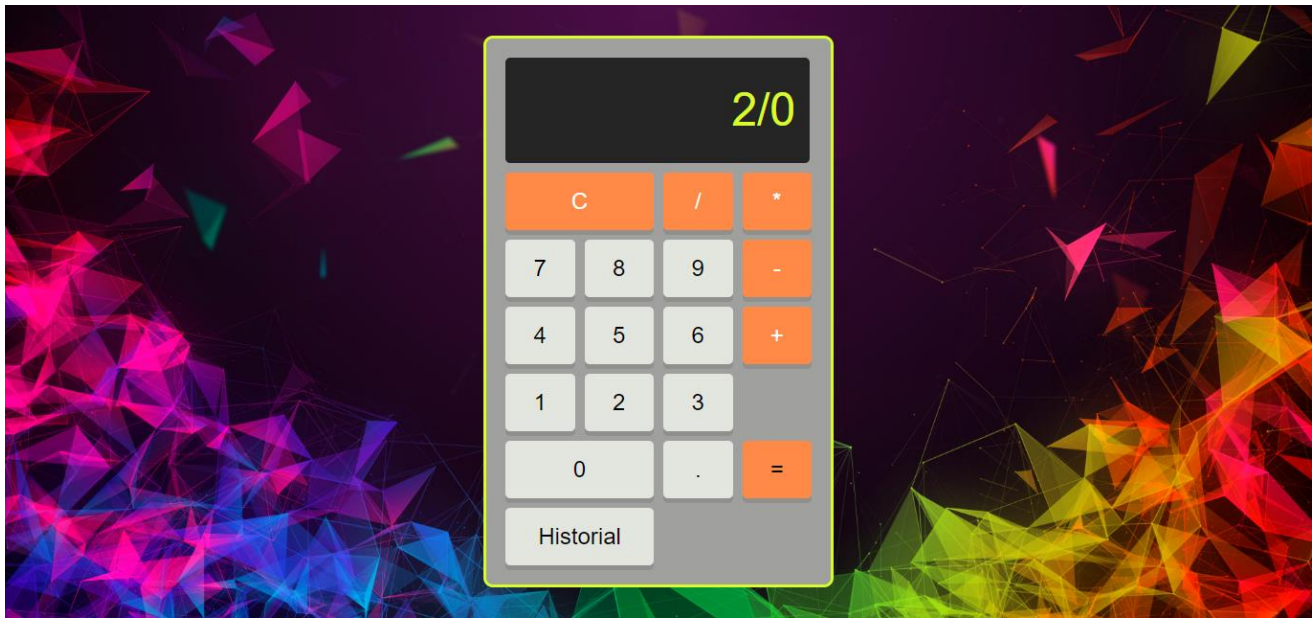


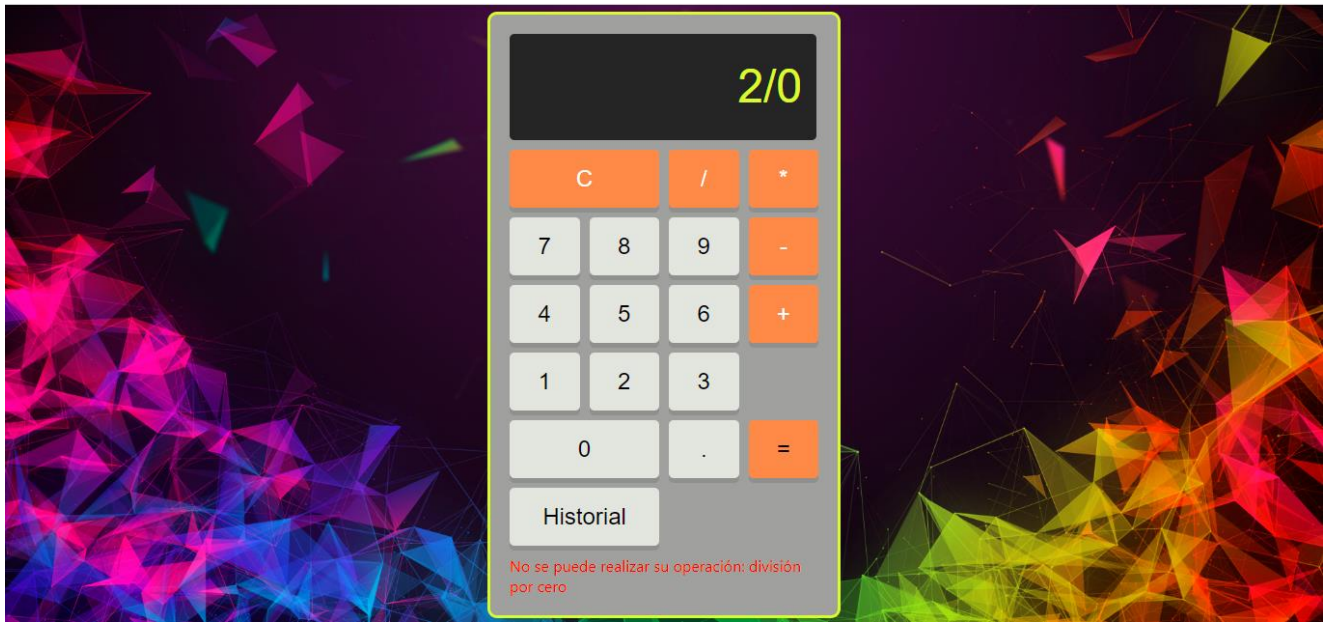
- **OPRECION MIXTA.**



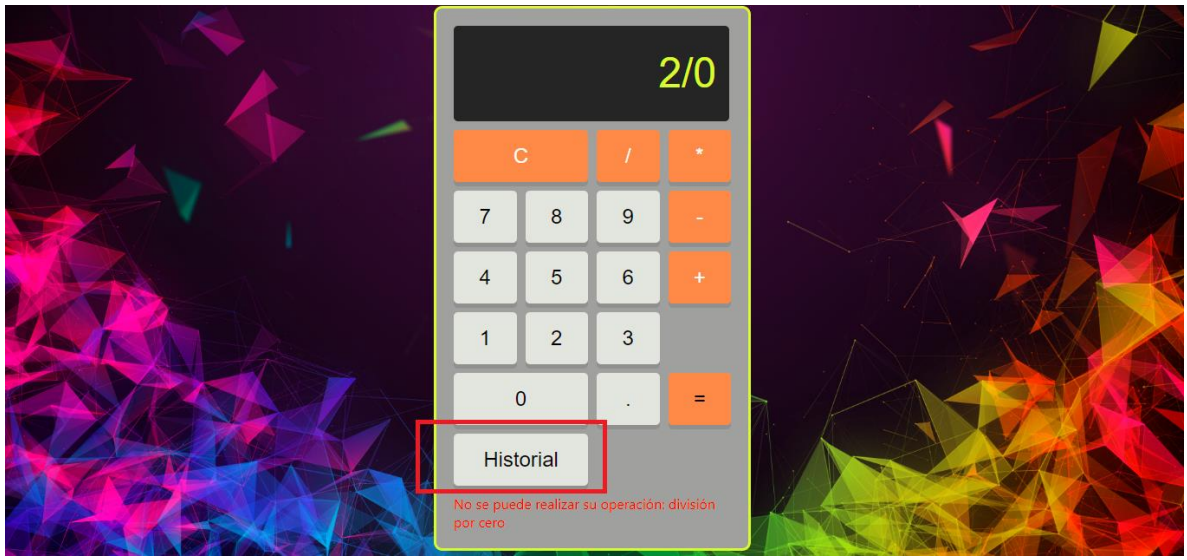


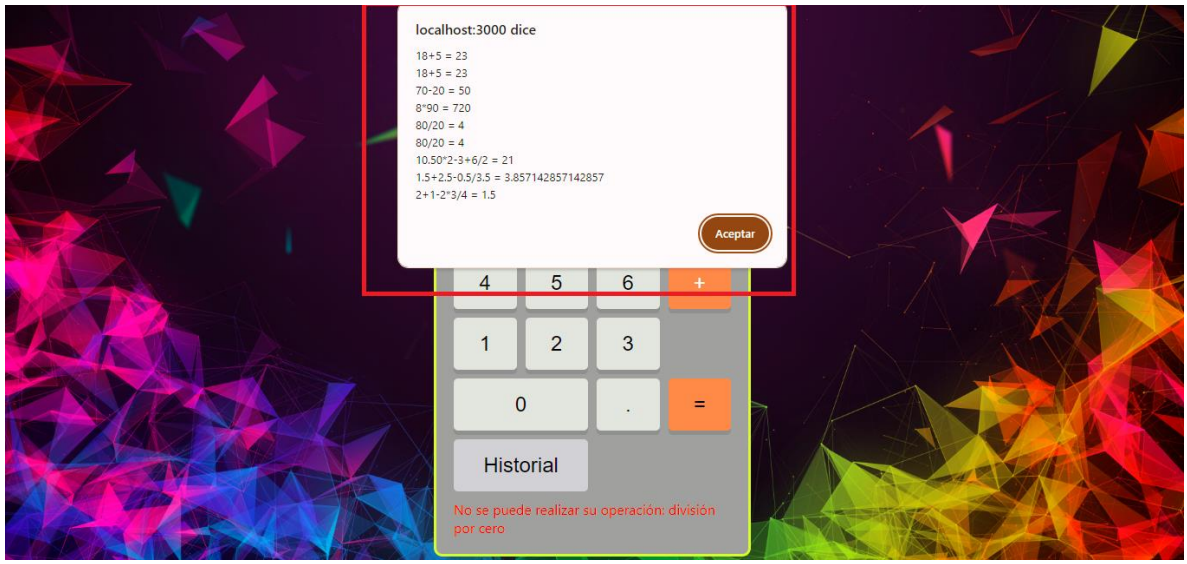
- **Mensaje de Excepción cuando el usuario divide dentro de 0**





- **Historial de operaciones realizadas**





Conclusión del Proyecto de Calculadora

En este proyecto se desarrolló una calculadora funcional que cumple con los requerimientos funcionales y técnicos especificados, utilizando tecnologías modernas y buenas prácticas de desarrollo. La calculadora permite realizar operaciones aritméticas básicas como suma, resta, multiplicación y división, con una validación de entradas eficiente para evitar errores comunes como divisiones por cero.

1. Funcionalidad: Las operaciones básicas de la calculadora fueron implementadas con éxito, garantizando un comportamiento correcto. El sistema también cuenta con un historial de las últimas 10 operaciones, lo que mejora la experiencia de usuario al permitirle revisar sus cálculos recientes.

2. Validación y Manejo de Excepciones: Se implementaron validaciones en el frontend y backend para evitar operaciones inválidas. Esto asegura que la aplicación maneje adecuadamente errores como la división por cero, mostrando mensajes de error claros al usuario.

3. Backend y Frontend: El backend fue desarrollado en Java 8 utilizando Spring Boot para la exposición de una API RESTful. Esta arquitectura permitió una comunicación eficiente con el frontend desarrollado en React. Los componentes de la interfaz fueron creados para ser intuitivos, responsivos y funcionales. El manejo de estado con useState o useReducer permitió la gestión eficiente del historial de operaciones. Se utilizaron fetch y Axios para las llamadas HTTP, garantizando una respuesta en tiempo real desde el servidor.

4. Despliegue y Herramientas: El proyecto fue probado y desplegado en el entorno local, utilizando Visual Studio Code para el desarrollo. Se ejecutó correctamente en el puerto http://localhost:3000/.

5. La documentación proporcionada fue clara, permitiendo a cualquier usuario desplegar y probar la aplicación sin dificultades.

En resumen, este proyecto ha sido exitoso, logrando una integración efectiva de tecnologías como Java, Spring Boot, React, JavaScript, CSS y HTML, brindando una aplicación funcional, responsiva y con un diseño acorde a los estándares actuales.