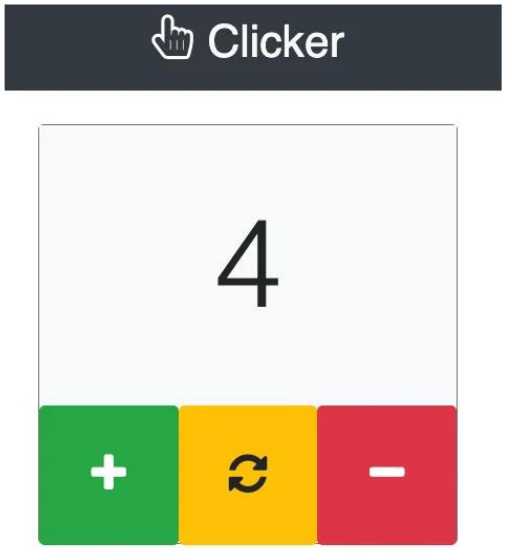
C5.1_

051ED51_ clicker App

En este tutorial, crearemos una aplicación de clic simple usando React.js. Este proyecto nos ayudará a comprender los conceptos básicos del renderizado JSX y nos presentará el gancho useState, una característica esencial en React.



Hook useState

Imagina que tienes una caja de juguetes mágica y dentro de ella hay un compartimento especial donde puedes realizar un seguimiento de algo importante. Llamémoslo "memoria mágica". Ahora, cada vez que quieras recordar algo diferente sobre tus juguetes, utilizas la “memoria mágica” para almacenar esa información.

En React, cuando hablamos de useState, es como tener este compartimento mágico para recordar cosas sobre una parte de tu aplicación. Entonces, si tiene un botón en su aplicación y desea recordar si se hizo clic en él o no, puede usar useState para crear un compartimento mágico (estado) solo para ese botón. Cuando presionas el botón, el compartimento mágico recuerda que se hizo clic y puedes usar esa información cuando la necesites.

Entonces, useState es como una caja de memoria mágica en React que ayuda a tu aplicación a recordar y realizar un seguimiento de diferentes cosas.

Técnicamente useState es un gancho de React que permite que los componentes funcionales tengan estado.

Requisitos previos

Asegúrese de tener Node.js y npm (Node Package Manager) instalados en su sistema

Paso 1: configurar una nueva aplicación React

Abra su terminal y ejecute el siguiente comando para crear una nueva aplicación React:

```
npx create-react-app clicker-app
```

Paso 2: crea la aplicación Clicker

Navegue a la carpeta src en el directorio de su proyecto y abra el archivo **App.js**. Reemplace su contenido con el siguiente código:

```
import React, { useState } from "react";
import Header from "./components/Header";
import Clicker from "./components/Clicker";
import "./App.css";

export default function App() {
  const [title] = useState("Clicker");
  return (
    <div className="App">
      <Header title={title} />
      <Clicker />
    </div>
  );
}
```

Explicación

JSX Rendering

JSX es una extensión de sintaxis para JavaScript, que a menudo se usa con React para describir cómo debería verse la interfaz de usuario.

En nuestra app component, utilizamos JSX para definir la estructura de nuestra aplicación, incluidos encabezados, párrafos, botones, marcadores de posición para contenido dinámico e incluso otros componentes como se muestra arriba.

Creemos una carpeta `**components**` dentro de la carpeta **src** y creemos un componente llamado **Header** como se muestra a continuación:

```
import React from "react";
export default function Header(props) {
 return (
  <nav className="navbar navbar-dark bg-dark">
   <div className="container">
    <div className="row m-auto">
     <i className="fa fa-hand-pointer-o fa-2x text-white mr-2" />
     <div className="text-light h2">{props.title}</div>
    </div>
   </div>
  </nav>
 );
}
```

A continuación, centrémonos en la creación del componente principal *de Clicker* : –

```
import React, { useState } from "react";
export default function Clicker() {
  const [clickerValue, updateClickerValue] = useState(0);
  return (
    <div className="container">
      <div className="clickerParent border border-secondary rounded mt-4">
        <div className="clickerDisplay d-flex align-items-center justify-content-center bg-light display-2">
          {clickerValue}
        </div>
        <div className="clickerButtonContainer d-flex flex-row">
          <button
            className="btn btn-success w-100"
            onClick={() => updateClickerValue(clickerValue + 1)}
          >
            <i className="fa fa-2x fa-plus" />
          </button>
          <button
            className="btn btn-warning w-100"
            onClick={() => updateClickerValue(0)}
          >
            <i className="fa fa-2x fa-refresh" />
          </button>
          <button
            className="btn btn-danger w-100"
            onClick={() => updateClickerValue(Math.max(0,clickerValue - 1))}
          >
            <i className="fa fa-2x fa-minus" />
          </button>
        </div>
      </div>
    </div>
  );
}
```

Explicación

En nuestro componente Clicker, declaramos una variable de estado *clickerValue* y una función *updateClickerValue* para actualizarla. El valor inicial de *clickerValue* se establece en 0.
Cuando el usuario hace clic en el botón "+", se llama a *updateClickerValue* e incrementamos *clickerValue* en 1.
Cuando el usuario hace clic en el botón "Restablecer", se llama a *updateClickerValue* y configuramos el *clickerValue* vuelve a 0.

Cuando el usuario hace clic en el botón "-", se llama a updateClickerValue y disminuimos clickerValue *en* 1. Además, utilizamos Math.max para asegurarnos de que *clickerValue* no baje de 0.

Paso 3: crear estilo CSS

Actualice la sección principal en el archivo `public/index.html` para agregar enlaces a las CDN Bootstrap y Font Awesome:

```
<head>
  <meta charset="utf-8" />
  <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <meta name="theme-color" content="#000000" />
  <meta
   name="description"
   content="Clicker app using reactjs"
  />
  <!--
   manifest.json provides metadata used when your web app is installed on a
   user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
  -->
  <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
  <link
   rel="stylesheet"
   href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
   integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
   crossorigin="anonymous"
  />
  <link
   href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
   rel="stylesheet"
  />
  <!--
   Notice the use of %PUBLIC_URL% in the tags above.
   It will be replaced with the URL of the `public` folder during the build.
   Only files inside the `public` folder can be referenced from the HTML.

   Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
   work correctly both with client-side routing and a non-root public URL.
```

```
    Learn how to configure a non-root public URL by running `npm run build`.
  -->
  <title>Clicker App</title>
 </head>
```

Cree o actualice un archivo llamado App.css en la carpeta src:

```
.App {
  font-family: sans-serif;
  text-align: center;
}

.clickerParent {
  width: 300px;
  height: 300px;
  margin: 0 auto;
}

.clickerDisplay {
  height: 200px;
}

.clickerButtonContainer {
  height: 100px;
}
```