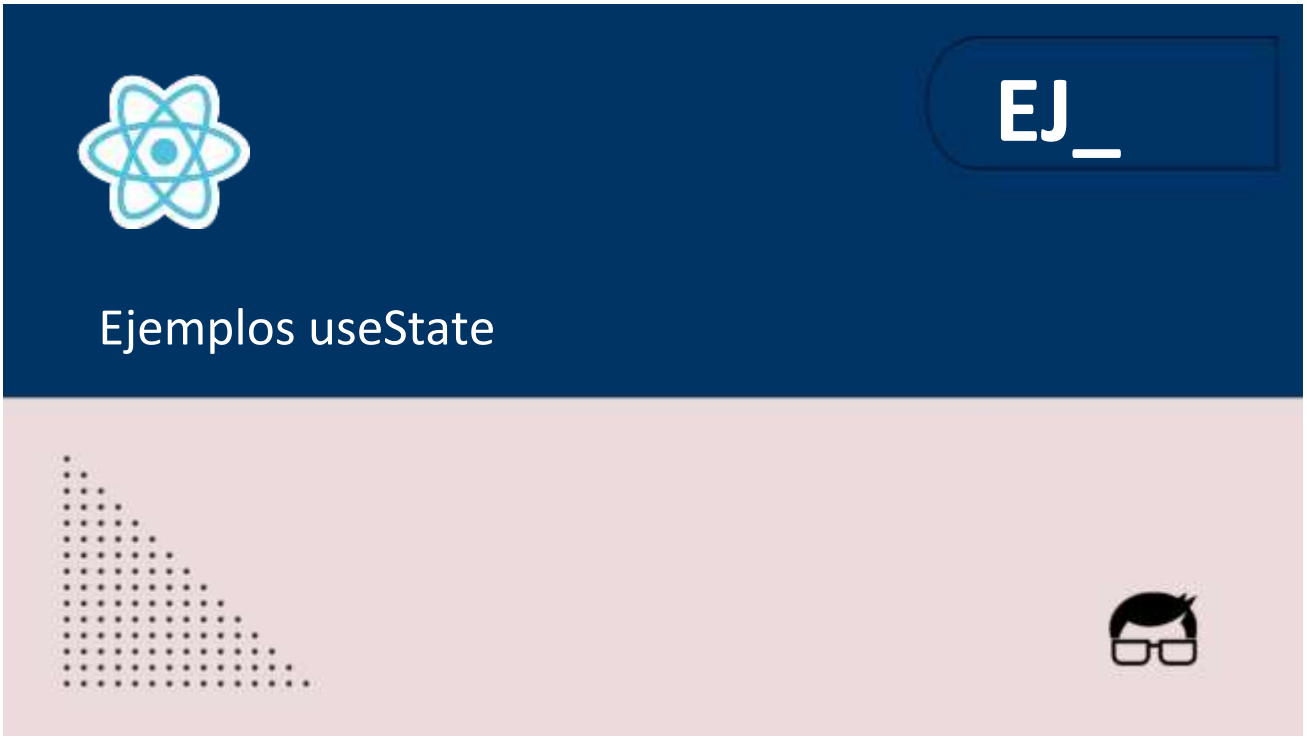

Ejemplo básico de un archivo HTML que incluye los tres ejemplos de uso de useEffect que te mostré anteriormente. Este archivo HTML incorporará los scripts de React y ReactDOM desde una CDN y usará Babel para interpretar el código JSX.

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo de React con useState en HTML</title>
  <script src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@17.0.2/umd/react-
dom.development.js"></script>
</head>
<body>
  <div id="root"></div>

  <script>
    // Definir el componente usando useState
    const Example = () => {
      const [count, setCount] = React.useState(660);

      const increment = () => {
        setCount(count + 1);
      };

      const decrement = () => {
        setCount(count - 1);
      };
```

```
      return (
        React.createElement('div', null,
          React.createElement('p', null, 'Contador: ' + count),
          React.createElement('button', { onClick: increment }, 'Incrementar'),
          React.createElement('button', { onClick: decrement }, 'Decrementar')
        )
      );
    };

    // Renderizar el componente en el elemento 'root'
    ReactDOM.render(
      React.createElement(Example),
      document.getElementById('root')
    );
  </script>
</body>
</html>
```

Este código HTML integra un componente de React que utiliza el hook **useState** para crear un contador simple. El componente se llama **Example** y se renderiza dentro del elemento **div** con **id="root"**. Aquí te explico cómo funciona:

1. **Carga de React y ReactDOM**: Al principio del archivo HTML, se cargan las bibliotecas React y ReactDOM mediante etiquetas **<script>**. Estas bibliotecas son esenciales para crear y manejar componentes de React.

2. **Contenedor del Componente**: El ** ** es el lugar en el DOM donde se montará el componente React.

3. **Definición del Componente Example**:

    - **useState para el Estado del Contador**: Se utiliza el hook **useState** de React para definir una variable de estado **count**, que se inicializa en 660. La función **setCount** asociada se usa para actualizar este estado.

    - **Funciones de Incremento y Decremento**: Se definen dos funciones, **increment** y **decrement**, que llaman a **setCount** para actualizar el valor de **count**, incrementando o decrementando su valor en 1, respectivamente.

    - **Renderizado del Componente**: El componente devuelve un **div** que contiene un párrafo mostrando el valor actual de **count** y dos botones para incrementar y decrementar el contador. Esta estructura se crea usando **React.createElement**.

4. **Renderizado con ReactDOM**: Finalmente, **ReactDOM.render** se utiliza para montar el componente **Example** en el elemento **div** con **id="root"** en el documento HTML.

```html
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo de React con Estado en HTML</title>
</head>
<body>
  <div id="root"></div>

  <script src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@17.0.2/umd/react-
dom.development.js"></script>
  <script>
    // Componente de React
    const Example = () => {
      const [name, setName] = React.useState('');

      const handleChange = (event) => {
        setName(event.target.value);
      };

      return (
        React.createElement('div', null,
          React.createElement('p', null, 'Nombre: ' + name),
          React.createElement('input', { type: 'text', value: name, onChange:
handleChange })
        )
      );
    };

    // Renderizar el componente en el elemento 'root'
    ReactDOM.render(
      React.createElement(Example),
      document.getElementById('root')
    );
  </script>
</body>
</html>
```

Este código HTML es un ejemplo de cómo se puede usar React directamente en un documento HTML para crear un componente simple con estado. Aquí está el análisis del código:

Estructura del Documento HTML

- **Inclusión de React y ReactDOM**: Las bibliotecas de React y ReactDOM se incluyen a través de etiquetas **<script>**. Estas son necesarias para utilizar React y renderizar componentes en el DOM.
- **Contenedor del Componente**: El ** ** es el lugar en el DOM donde se montará el componente de React.

Componente de React Example

- **Uso de useState para el Estado**: Se utiliza el hook **useState** para crear una variable de estado llamada **name**, la cual se inicializa con una cadena vacía. **setName** es la función que se utilizará para actualizar este estado.
- **Manejo de Cambios de Entrada**: Se define una función **handleChange** que se activa cada vez que cambia el valor del campo de entrada (**input**). Esta función actualiza el estado **name** con el valor actual del campo de entrada.
- **Renderizado del Componente**: El componente **Example** devuelve un **div** que contiene:
    - Un párrafo **<p>** que muestra el valor actual del estado **name**.
    - Un campo de entrada (**input**) de texto que permite al usuario escribir su nombre. El valor del campo de entrada está vinculado al estado **name**, y el evento **onChange** está configurado para llamar a **handleChange** cada vez que el usuario escribe en el campo.

Renderizado con ReactDOM

- **ReactDOM.render**: Esta función monta el componente **Example** en el elemento **div** con **id="root"** en el documento HTML.

Funcionamiento General

Cuando un usuario escribe en el campo de entrada, el evento **onChange** activa la función **handleChange**, que actualiza el estado **name** con el nuevo valor. Esto a su vez actualiza el párrafo que muestra el nombre, reflejando lo que el usuario ha escrito en tiempo real.

```html
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo de Renderizado Condicional en HTML</title>
</head>
<body>
  <div id="root"></div>

  <script src="https://unpkg.com/react@17.0.2/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@17.0.2/umd/react-
dom.development.js"></script>
  <script>
    // Componente de React
    const Example = () => {
  const [isLoggedIn, setIsLoggedIn] = React.useState(false);

  const handleLogin = () => {
    setIsLoggedIn(true);
    document.body.style.backgroundColor = 'blue';
    document.body.style.color = 'white';
  };

  const handleLogout = () => {
    setIsLoggedIn(false);
    document.body.style.backgroundColor = 'white';
    document.body.style.color = 'black';
  };

  // El contenido del componente
  return (
    React.createElement('div', null,
      isLoggedIn ? (
        React.createElement('p', null, 'Bienvenido, usuario. ',
          React.createElement('button', { onClick: handleLogout }, 'Cerrar sesión')
        )
      ) : (
        React.createElement('p', null, 'Inicia sesión para continuar. ',
          React.createElement('button', { onClick: handleLogin }, 'Iniciar sesión')
        )
      )
    )
  );
};

// Renderizar el componente en el elemento 'root'
ReactDOM.render(
  React.createElement(Example),
  document.getElementById('root')
```


```
);


  </script>
</body>
</html>
```

Este código HTML implementa un ejemplo simple de renderizado condicional en React, junto con la manipulación del estilo del cuerpo del documento (**body**) basado en el estado del componente. El componente **Example** utiliza el hook **useState** para manejar el estado de inicio de sesión y cambia el fondo y el color del texto del cuerpo de la página según este estado. Aquí está el detalle del funcionamiento:

Estructura del Documento HTML

- **Inclusión de React y ReactDOM**: Las bibliotecas de React y ReactDOM se cargan mediante etiquetas **<script>**. Estas son esenciales para crear componentes React y renderizarlos en el DOM.
- **Contenedor del Componente React**: El elemento ** ** es donde se montará el componente React.

Componente React Example

- **useState para Manejo de Estado**: Se utiliza **useState** para crear un estado **isLoggedIn** que rastrea si el usuario ha iniciado sesión (**true**) o no (**false**).
- **Funciones handleLogin y handleLogout**:
    - **handleLogin** establece **isLoggedIn** en **true** y cambia el estilo del cuerpo del documento a un fondo azul con texto blanco.
    - **handleLogout** hace lo contrario, estableciendo **isLoggedIn** en **false** y cambiando el estilo del cuerpo a un fondo blanco con texto negro.
- **Renderizado Condicional**: El componente renderiza diferentes elementos JSX dependiendo del valor de **isLoggedIn**. Si el usuario está logueado, muestra un mensaje de bienvenida y un botón para cerrar sesión. Si no está logueado, muestra un mensaje y un botón para iniciar sesión.

Renderizado con ReactDOM

- **ReactDOM.render**: Esta función monta el componente **Example** en el elemento **div** con **id="root"** en el documento HTML.

Funcionamiento General

Cuando un usuario hace clic en el botón "Iniciar sesión", se invoca **handleLogin**, que cambia el estado a logueado y actualiza el estilo del cuerpo del documento. De manera similar, al hacer clic en "Cerrar sesión", se invoca **handleLogout**, que revierte estos cambios. El renderizado condicional dentro del componente muestra diferentes interfaces dependiendo de si el usuario está logueado o no.