02 React: desde un CDN

Es posible usar React desde un CDN en una web. Esto es útil si no deseas configurar un entorno de desarrollo complejo con herramientas como Webpack o Vite, y solo quieres incluir React directamente en tu archivo HTML mediante etiquetas `<script>`.

A continuación te muestro un ejemplo básico de cómo integrar React usando un CDN:

Ejemplo: Usar React desde un CDN

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>React desde CDN</title>
  <!-- Cargar React y ReactDOM desde un CDN -->
  <script src="<https://unpkg.com/react@18/umd/react.development.js>" crossorigin></script
>
  <script src="<https://unpkg.com/react-dom@18/umd/react-dom.development.js>" crossorigi
n></script>
  <!-- Cargar Babel para compilar JSX en el navegador -->
  <script src="<https://unpkg.com/@babel/standalone/babel.min.js>"></script>
</head>
<body>
  <!-- Contenedor donde se renderizará la aplicación React -->
  <div id="root"></div>

  <!-- Código de React escrito en JSX -->
  <script type="text/babel">
    // Definir un componente funcional simple
    function App() {
      return (
        <div>
          <h1>Hola, mundo!</h1>
          <p>Este es un ejemplo de React cargado desde un CDN.</p>
        </div>
      );
    }

    // Renderizar el componente en el DOM
    const root = ReactDOM.createRoot(document.getElementById('root'));
    root.render(<App />);
  </script>
</body>
</html>
```

Explicación del código:

1. **CDN de React y ReactDOM**:
   - Se cargan las bibliotecas de React y ReactDOM desde `unpkg`, que es un CDN popular. Estas son las versiones de desarrollo (`development.js`), que son útiles para depuración. Para producción, puedes usar las versiones minificadas (`react.production.min.js` y `react-dom.production.min.js`).
2. **Babel**:
   - Como React utiliza JSX (una sintaxis especial que mezcla HTML con JavaScript), necesitamos un transpilador como Babel para convertir JSX en JavaScript puro que el navegador pueda entender.
   - En este caso, Babel se carga también desde un CDN y se utiliza especificando `type="text/babel"` en la etiqueta `<script>`.
3. **Contenedor** `#root`:
   - El elemento `<div id="root"></div>` actúa como el punto de montaje para la aplicación React. Aquí es donde se renderizará el contenido de tu aplicación.
4. **Componente** `App`:
   - Se define un componente funcional simple llamado `App` que devuelve un título y un párrafo.
5. **Renderizado**:
   - Usamos `ReactDOM.createRoot()` (disponible en React 18+) para crear un "root" y luego llamamos a `.render()` para renderizar el componente `App` dentro del contenedor `#root`.

Notas importantes:

- **Producción**: Este enfoque es adecuado para prototipos rápidos o pequeños proyectos. Sin embargo, para aplicaciones más grandes o en producción, es recomendable usar herramientas como Webpack, Vite o Create React App para optimizar el rendimiento y el tamaño del código.
- **Desempeño**: Usar Babel en el navegador puede ser lento, ya que el código JSX se transpila en tiempo real. Esto no es ideal para entornos de producción.
- **Versiones**: Asegúrate de usar versiones compatibles de React, ReactDOM y Babel. En este ejemplo, se usa React 18, que introduce `ReactDOM.createRoot()`.