FullStack Developer

React JS

Introducción a React

¿Qué es React?

React es una biblioteca de JavaScript de código abierto para construir interfaces de usuario, especialmente para aplicaciones web de una sola página. Fue desarrollada por Facebook y es mantenida por Facebook y una comunidad de desarrolladores individuales y empresas.

Principales características de React:

1.Componentes:

1. React se basa en componentes reutilizables. Cada componente puede mantener su propio estado interno y ser compuesto para crear interfaces de usuario complejas.

2.JSX:

1. JSX es una extensión de la sintaxis de JavaScript que permite escribir HTML dentro de JavaScript. Facilita la creación de componentes al hacer el código más legible y fácil de escribir.

3. **Virtual DOM**:
   - React utiliza un DOM virtual, que es una representación en memoria del DOM real. Cuando el estado de un componente cambia, React actualiza el DOM virtual y compara las diferencias con el DOM real (proceso conocido como "reconciliación"), actualizando solo las partes necesarias.

4. **Unidirectional Data Flow**:
   - En React, los datos fluyen en una sola dirección, de los componentes padres a los componentes hijos. Esto hace que el seguimiento de cómo los datos cambian con el tiempo sea más fácil y predecible.

5. **Declarativo**:
   - React permite describir cómo se verá la UI en cualquier estado dado, y maneja la actualización del DOM de manera eficiente cuando el estado cambia.

Ejemplo básico:

Aquí hay un pequeño ejemplo de cómo podría verse un componente básico en React:

```
import React from 'react';
import ReactDOM from 'react-dom';

function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

const element = <Welcome name="Sara" />;
ReactDOM.render(element, document.getElementById('root'));
```

En este ejemplo:

•Welcome es un componente de función que recibe props y devuelve un elemento JSX.
•ReactDOM.render monta el componente Welcome en un elemento del DOM con el id root.

Beneficios de usar React:

•**Reutilización de componentes**: Los componentes pueden ser reutilizados en diferentes partes de una aplicación, lo que aumenta la eficiencia del desarrollo.

•**Performance**: Gracias al virtual DOM, React puede actualizar solo las partes del DOM que han cambiado, lo que mejora el rendimiento.

•**Comunidad y Ecosistema**: React tiene una comunidad grande y activa, lo que significa que hay muchos recursos, librerías y herramientas disponibles.

ReactDOM.render

ReactDOM.render es un método proporcionado por la biblioteca react-dom que permite renderizar un elemento React (o un componente) en el DOM del navegador. Este método es crucial para iniciar una aplicación React, ya que conecta el árbol de componentes de React con un elemento del DOM real en la página web.

```
ReactDOM.render(
    element,
    container,
    [callback]
  );
```

•**element**: Es el elemento React que queremos renderizar. Puede ser un elemento JSX, un componente React, o cualquier otro nodo React.

•**container**: Es el contenedor DOM en el cual queremos montar nuestro componente. Generalmente, se trata de un elemento div con un id específico.

•**callback** (opcional): Es una función que se ejecuta después de que el componente ha sido renderizado o actualizado.

```jsx
import React from 'react';
import ReactDOM from 'react-dom';

// Un componente simple
function Hello(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// Crear un elemento del componente
const element = <Hello name="World" />;

// Seleccionar el contenedor en el DOM
const container = document.getElementById('root');

// Renderizar el elemento en el contenedor
ReactDOM.render(element, container);
```

En este ejemplo:

1.Definimos un componente Hello que recibe props y retorna un elemento JSX.

2.Creamos un elemento del componente Hello con el prop name establecido a "World".

3.Seleccionamos el contenedor en el DOM, en este caso, un elemento div con el id "root".

4.Usamos ReactDOM.render para renderizar el elemento en el contenedor.

Detalles importantes

•**Montaje inicial**: Cuando ReactDOM.render se llama por primera vez, monta el componente (o árbol de componentes) en el contenedor especificado.

•**Actualización**: Si ReactDOM.render se llama de nuevo con un elemento diferente (o con diferentes props o estado), React actualizará eficientemente el DOM para que coincida con el nuevo elemento.

•**Desmontaje**: React no proporciona directamente un método para desmontar un componente. Sin embargo, si se llama a ReactDOM.render con null en el contenedor, React limpiará cualquier componente previamente renderizado.

Ejemplo con callback

El tercer parámetro de ReactDOM.render es un callback que se ejecuta después de que el componente ha sido renderizado o actualizado. Esto puede ser útil para realizar acciones posteriores a la renderización.

```
import React from 'react';
import ReactDOM from 'react-dom';

function Hello(props) {
  return <h1>Hello, {props.name}!</h1>;
}

const element = <Hello name="React" />;
const container = document.getElementById('root');

ReactDOM.render(element, container, () => {
  console.log('Component has been rendered');
});
```

En este caso, la función de callback se ejecutará después de que el componente Hello haya sido renderizado en el DOM.

Importancia de ReactDOM.render

ReactDOM.render es fundamental para cualquier aplicación React, ya que es el punto de entrada donde se inicia la renderización de los componentes de React en el DOM del navegador. Sin este método, no podríamos conectar nuestros componentes de React con el árbol DOM del navegador, lo que es esencial para que la aplicación funcione correctamente en la web.

Creación de un Componente en React

Los componentes son la base de las aplicaciones React. Un componente en React puede ser una función o una clase que opcionalmente acepta entradas (conocidas como "props") y devuelve un elemento React que describe cómo debería aparecer una sección de la interfaz.

Tipos de Componentes en React

1.Componentes Funcionales:

- Son funciones de JavaScript que aceptan props como argumento y devuelven elementos React.
- Son más sencillos y se recomiendan para la mayoría de los casos.
- Pueden usar hooks para manejar estado y efectos.

2.Componentes de Clase:

- Son clases de ES6 que extienden de React.Component.
- Tienen un método render que devuelve elementos React.
- Pueden tener estado y métodos de ciclo de vida.

- (link para saber más sobre componentes de clase)

Ejemplo de un Componente Funcional

Vamos a crear un componente funcional simple llamado Greeting.

```jsx
import React from 'react';

// Definición del componente funcional
function Greeting(props) {
  return <h1>Hello, {props.name}!</h1>;
}

// Exportar el componente para usarlo en otros archivos
export default Greeting;
```

Este componente acepta un prop llamado name y muestra un saludo. Para usar este componente, podemos importarlo y renderizarlo en otro componente o en el archivo principal de nuestra aplicación.

Uso del Componente en el Archivo Principal

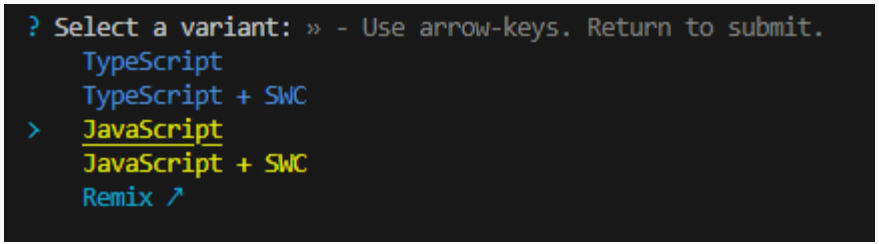
```
import React from 'react';
import ReactDOM from 'react-dom';
import Greeting from './Greeting';

// Crear un elemento del componente Greeting
const element = <Greeting name="World" />;

// Seleccionar el contenedor en el DOM
const container = document.getElementById('root');

// Renderizar el componente en el contenedor
ReactDOM.render(element, container);
```

Empecemos a crear un proyecto con Vite de React

```
npm create vite@latest .
```

Seleccionamos React

Seleccionamos JavaScript



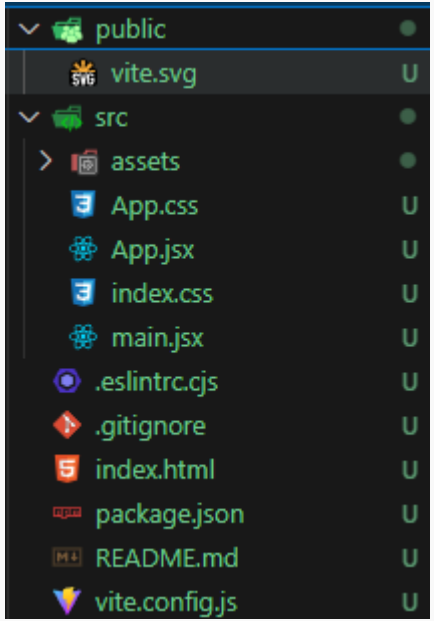
Ahora hacemos un npm install

```
Done. Now run:

  npm install
  npm run dev
```

Y luego un npm run dev

Con esto tenemos todas las dependencia básicas instaladas y ahora veamos que tenemos creado.

Por lo pronto nos vamos a concentrar en los archivos jsx



En el archivo main.jsx

```jsx
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

Cuando se inicia un proyecto React usando Vite, el archivo main.jsx es el punto de entrada principal de la aplicación. Este archivo es responsable de montar el componente raíz de React en el DOM del navegador.

Explicación Detallada

1.Importaciones:

- •import React from 'react';: Importa la biblioteca React, que es necesaria para usar JSX y crear componentes.
- •import ReactDOM from 'react-dom/client';: Importa la biblioteca ReactDOM, que contiene métodos específicos para renderizar componentes React en el DOM.
- •import App from './App';: Importa el componente principal de la aplicación, que normalmente se encuentra en App.jsx o App.js.
- •import './index.css';: Importa los estilos CSS globales de la aplicación. Este archivo suele contener estilos base para la aplicación.

2.Crear el contenedor raíz:

```
ReactDOM.createRoot(document.getElementById('root'))
```

Aquí, ReactDOM.createRoot se utiliza para crear un contenedor raíz en el DOM donde se renderizarán los componentes de React. El contenedor raíz es un elemento HTML con el id "root".

Renderizar el componente principal:

```
ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

• .render: Este método monta el componente App en el contenedor raíz creado en el paso anterior.

•<React.StrictMode>: Es un componente envolvente que ayuda a identificar problemas potenciales en la aplicación. Solo se usa en desarrollo y no afecta la producción. Asegura que el código cumple con las mejores prácticas y advertencias adicionales.

Función App():

Para entender bien esta función vamos a borrar todo el contenido y redefiniremos la misma

```jsx
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        <a href="https://vitejs.dev" target="_blank">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.jsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </>
  )
}

export default App
```

Función App():

```
import React from 'react';

function App() {

  return (
    <div>
      <header>
        <h1>Bienvenido a React con Vite</h1>
      </header>
    </div>
  );
}

export default App;
```

En la línea que importa la biblioteca React. Aunque en versiones más recientes de React, el import de React no es estrictamente necesario para usar JSX (React 17+), sigue siendo una práctica común importarlo en cada archivo de componente.

```
"react": "^18.3.1",
```

En esta oportunidad usarmos la versión 18.3.1

function App():

•Declara una función de JavaScript llamada App. En React, una función que retorna elementos JSX se convierte en un componente funcional.

return (...):

•La función App retorna un fragmento de JSX. JSX es una extensión de la sintaxis de JavaScript que permite escribir HTML directamente dentro de JavaScript.

JSX devuelto:

```
<div>
  <header>
    <h1>Bienvenido a React con Vite</h1>
  </header>
</div>
```

El JSX que se retorna describe la estructura HTML que se renderizará en el DOM. En este caso, se está creando un div contenedor que incluye un header, y dentro del header, un h1 con el texto "Bienvenido a React con Vite".

export default App;:

•Esta línea exporta el componente App como la exportación predeterminada del módulo. Esto permite que otros archivos importen y utilicen el componente App fácilmente.

El componente App es un componente funcional sencillo que sirve como punto de partida para construir una aplicación React. Renderiza un encabezado básico con un mensaje de bienvenida. Este es el primer paso para construir componentes más complejos y dinámicos en una aplicación React.

Convenciones de Nomenclatura en React

En React, hay ciertas convenciones de nomenclatura que se siguen para mantener el código organizado y fácilmente comprensible. Estas convenciones incluyen capitalizar los nombres de los archivos de componentes y las funciones de componentes. A continuación, se explican estas prácticas y sus razones.

Capitalización de Archivos de Componentes

1.Claridad y Consistencia:

- Nombrar los archivos de componentes con mayúscula inicial (por ejemplo, App.jsx, Header.jsx, Footer.jsx) hace que sea fácil identificar qué archivos contienen componentes React. Esta práctica es consistente con cómo se nombran los componentes mismos.

2.Distinguir Componentes de Otros Archivos:

- La capitalización ayuda a distinguir rápidamente los archivos que contienen componentes React de otros archivos que podrían ser funciones auxiliares, archivos de configuración, hojas de estilo, etc.

3.Convención Comunitaria:

- La comunidad de desarrolladores de React ha adoptado esta convención, por lo que seguirla facilita la colaboración y comprensión entre diferentes desarrolladores y equipos.

Capitalización de Funciones de Componentes

1.Compatibilidad con JSX:

- React trata cualquier función que empieza con una letra mayúscula como un componente React. Las funciones con nombres que empiezan con minúscula se consideran elementos del DOM nativo, como o .

Código JS en componentes funcionales React

En los componentes funcionales de React, puedes usar código JavaScript estándar para realizar diversas tareas. Esto incluye manipulación de datos, lógica condicional, y cualquier otra operación que necesites realizar

Variables JavaScript en JSX

En JSX, puedes insertar variables y expresiones de JavaScript dentro de tus elementos HTML utilizando llaves {}.

En un componente funcional React, podes usar cualquier lógica JavaScript que necesitemos para procesar datos antes de renderizar el JSX. Esto hace que los componentes de React sean muy flexibles y potentes.

```jsx
function App() {

  // Obtener la hora actual
  const horaActual = new Date().getHours();

  // Determinar el saludo basado en la hora
  let saludo;
  if (horaActual < 12) {
    saludo = 'Buenos días';
  } else if (horaActual < 18) {
    saludo = 'Buenas tardes';
  } else {
    saludo = 'Buenas noches';
  }

  // Devolver el JSX con el saludo adecuado
  return (
    <div>
      <h1>{saludo}, usuario!</h1>
    </div>
  );
}

export default App;
```

Una vez que tengamos todo listo verificamos que el proyecto esté corriendo en el puerto 5173.

Con http://localhost:5173

En caso de que no, ejecutamos en la consola:

npm run dev

Ejercicio

Crear un proyecto React con Vite y subirlo a GitHub

Empezar a crear tu proyecto y asegurarte de nunca subir la carpeta node_modules que se va a crear a la hora de instalar todas las dependencias. Lo bueno es que “Vite” ya nos crea un archivo “.gitignore” donde se excluyen todos los archivos y carpetas que no deberían subirse a un reposotorio