Fecha: 06/05/2025

Instituto tecnológico de Tepic

Ingeniería en Sistemas computacionales

Interfaces Web

Actividad:

Investigación General de React

Integrantes del equipo:

Cabral Machuca Alejandro Aaron

Díaz Canales José Lino

Hernández Rodriguez César Uriel 20400755

Valdez Poce Jose Alberto

¿Qué es React?

React (también conocido como **ReactJS**) es una **biblioteca de JavaScript** de código abierto diseñada para construir interfaces de usuario interactivas, especialmente en aplicaciones de una sola página (SPA).

Fue creada en Facebook (ahora Meta) por el ingeniero Jordan Walke y se lanzó como proyecto de código abierto en 2013.Su propósito principal es facilitar el desarrollo de la capa de **vista** (UI) en la arquitectura de una aplicación (es comúnmente descrita como la “V” en MVC).
React se centra sólo en el renderizado de componentes de interfaz; no es un framework completo, sino una capa que puede combinarse con otras librerías para el resto de la aplicación.
Actualmente React es mantenido por Meta (Facebook) y una amplia comunidad de desarrolladores de software libre.

Características técnicas principales

DOM virtual

React utiliza un DOM virtual para optimizar las actualizaciones de la interfaz. En lugar de modificar el DOM real directamente, React mantiene en memoria una representación virtual del árbol de elementos. Cuando cambia el estado de la aplicación, React genera un nuevo árbol virtual y lo compara (proceso llamado diffing) con el anterior.

De esta forma determina eficientemente qué partes del DOM real necesitan actualizarse, evitando renderizados completos innecesarios.
Esto mejora el rendimiento general, ya que “renderizar la versión actualizada tarda menos tiempo y utiliza menos recursos” que rehacer todo el árbol DOM.

Componentes y props

La UI en React se divide en componentes, que son bloques de código aislados y reutilizables que representan una parte de la interfaz.
Cada componente es esencialmente una función (o clase) de JavaScript que retorna elementos React (JSX) describiendo lo que debe mostrarse.
Los componentes pueden recibir props (propiedades) para parametrizar su contenido: los props son entradas arbitrarias que el componente usa para renderizar su UI. Por ejemplo, un componente sencillo de bienvenida podría ser:

```
function Welcome(props) {
  return <h1>Hola, {props.name}!</h1>;
}
// Uso del componente:
<Welcome name="María" />
```

Este componente acepta la propiedad `name` y muestra “Hola, María!” en pantalla. Los componentes pueden anidar unos dentro de otros, lo que permite construir interfaces complejas de forma estructurada y modular. Gracias a ello, el código es más fácil de mantener y reutilizar.

Gestión de Estado

Un estado es un objeto JavaScript que representa una parte de un componente. Cambia cada vez que un usuario interactúa con la aplicación, renderizando una nueva interfaz de cliente para reflejar las modificaciones.

La gestión de estados se refiere a la práctica de gestionar los estados de la aplicación React. Incluye el almacenamiento de datos en librerías de gestión de estados de terceros y la activación del proceso de re-renderización cada vez que los datos cambian.

Actualmente existen varias librerías de gestión de estados de terceros, siendo **Redux** y **Recoil** las dos más populares.

Redux

La librería de gestión de estado Redux tiene un almacén centralizado que mantiene el árbol de estado de una aplicación predecible. La librería también reduce la inconsistencia de los datos al evitar que dos componentes actualicen el estado de la aplicación simultáneamente.

La arquitectura de Redux admite el registro de errores para facilitar la depuración y tiene un método de organización de código estricto, lo que simplifica el mantenimiento. Además, cuenta con un gran número de complementos y es compatible con todas las capas de las interfaces de usuario.

Dicho esto, Redux es bastante complejo y, por lo tanto, no es óptimo para aplicaciones pequeñas con una sola fuente de datos.

Flujo unidireccional de datos

React implementa un flujo de datos en un solo sentido (de padres a hijos) para mejorar la previsibilidad. Los componentes padres pasan datos a los hijos mediante props, y los hijos no pueden modificar directamente esas props. De este modo, los datos fluyen hacia abajo en la jerarquía de componentes. Este enfoque simplifica la depuración y reduce errores, ya que es más claro de dónde provienen los cambios de estado.

JSX

React introduce JSX, una extensión sintáctica de JavaScript que permite escribir código HTML dentro de JavaScript. Con JSX se pueden crear elementos React mezclando sintaxis HTML y expresiones JavaScript, lo cual hace más legible la descripción de la UI. Por ejemplo, <h1>{titulo}</h1> en JSX generará un elemento <h1> cuyo contenido será el valor de la variable titulo JSX simplifica la escritura de componentes porque se parece al HTML

tradicional pero con todo el poder de JavaScript (puede incluir expresiones, condicionales, etc.). Internamente, JSX se transpira a llamadas a React.createElement para construir la estructura de componentes.

Rendimiento

Además del DOM virtual, React incorpora otras optimizaciones de rendimiento. Por ejemplo, en React 16+ se introdujo React Fiber, un nuevo algoritmo de reconciliación que permite renderizar de forma incremental y pausar/redireccionar la renderización si es necesario. React 18 añadió un modo de concurrencia (Concurrent Mode) y la capacidad de suspender componentes (Suspense) para mejorar la interactividad en actualizaciones pesadas. En general, el modelo declarativo de React ayuda a que la UI se mantenga consistente con el estado de la aplicación, minimizando las actualizaciones superfluas.

Guia para para usar React

Para comenzar a usar React de forma práctica, se siguen estos pasos básicos:

1. **Instalar Node.js y crear un proyecto:**

Primero, asegúrate de tener instalado Node.js (versión 14 o superior) y npm (versión ≥5.6) en tu sistema. Luego, crea un nuevo proyecto con Create React App (una herramienta oficial que configura todo lo necesario):

```
npx create-react-app mi-app
cd mi-app
npm start
```

El primer comando genera la estructura inicial del proyecto React, y npm start ejecuta el servidor de desarrollo. (Alternativamente se puede usar Yarn: npx yarn create react-app mi-app). Otro enfoque moderno es usar Vite: ejecuta npm create vite@latest mi-app --- --template react para generar un proyecto React ligero. En cualquier caso, esto configura ya las dependencias de React y Webpack/Babel para desarrollo.

2. **Explorar la estructura básica:**

El proyecto generado tiene esta estructura típica

```
mi-app/
 public/
  index.html    (plantilla HTML principal)
 src/
  index.js     (punto de entrada JavaScript)
  App.js       (componente principal)
  App.css      (estilos del App)
  index.css    (estilos globales)
  ...otros archivos...
```

El archivo public/index.html es la plantilla HTML que React usará como contenedor. El directorio src contiene el código fuente: el archivo src/index.js es el punto de entrada que renderiza el componente raíz, y src/App.js define el componente principal de la aplicación. Estos archivos son necesarios para que la aplicación compile correctamente, pero puedes crear más subdirectorios en src para organizar componentes adicionales.

3. **Crear el primer componente: Abre src/App.js y modifica su contenido para probar tu componente. Por ejemplo:**

```
import React from 'react';
function App() {
  return <h1>¡Hola, React!</h1>;
}
export default App;
```

Luego en src/index.js debes importar y renderizar este componente en el DOM:

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

ReactDOM.createRoot(document.getElementById('root')).render(<App />);
```

Esto mostrará “¡Hola, React!” en la página web. Así se puede comprobar que React está funcionando correctamente.

4. Uso de props: Los componentes pueden recibir datos mediante props. Por ejemplo, crea un componente Saludo.js:

```
import React from 'react';
function Saludo(props) {
  return <h2>¡Hola, {props.nombre}!</h2>;
}
export default Saludo;
```

Y úsalo en App.js:

```
import Saludo from './Saludo';
function App() {
  return (
    <div>
      <Saludo nombre="Carlos" />
      <Saludo nombre="Ana" />
    </div>
  );
}
```

Cada instancia de <Saludo /> muestra un mensaje diferente según la prop nombre. Las props permiten parametrizar componentes y utilizarlos con distintos valores.

5. **Manejo de estado:**

Para componentes que necesitan datos cambiantes (por ejemplo, contadores, formularios, peticiones asíncronas), React usa el concepto de estado. Con hooks como useState, puedes definir variables de estado locales en un componente. Por ejemplo, un simple contador:

```
import React, { useState } from 'react';
function Contador() {
 const [cuenta, setCuenta] = useState(0);
 return (
  <div>
   <p>Has hecho clic {cuenta} veces</p>
   <button onClick={() => setCuenta(cuenta + 1)}>Haz clic</button>
  </div>
 );
}
```

En este ejemplo, cuenta es el estado (un entero), inicializado en 0. Al presionar el botón se invoca setCuenta para actualizar el estado. Cada cambio de estado provoca que React “reanude” el render del componente, mostrando siempre la UI sincronizada con el estado actual. En React, el estado de un componente es simplemente un objeto o variable JavaScript que refleja parte de la interfaz y cambia con la interacción del usuario.

Recursos recomendados: Para profundizar, consulta la **documentación oficial** de React: la versión en español react.dev (ES) o en inglés reactjs.org contienen guías, tutoriales (por ejemplo, el tutorial de “Tres en línea”) y ejemplos prácticos. También puedes seguir tutoriales reconocidos como los de MDN o freeCodeCamp.

Otras herramientas útiles incluyen el **React Developer Tools** (extensión de navegador para depurar componentes) y editores de código como Visual Studio Code con plugins para React. Si trabajas con TypeScript, React ofrece plantillas y documentación específica. Para probar rápidamente componentes React sin instalar nada, servicios online como CodeSandbox también son muy prácticos.

.