

React

2º DAW IES Al-Mudeyne

Índice

Módulo 4: Conceptos Avanzados de Componentes

Módulo 5: Navegación en React

Módulo 6: Manejo de Estado Global

Módulo 1: Introducción a React

1.1. ¿Qué es React?

1.1.1. Principios fundamentales

1.1.2. Virtual DOM

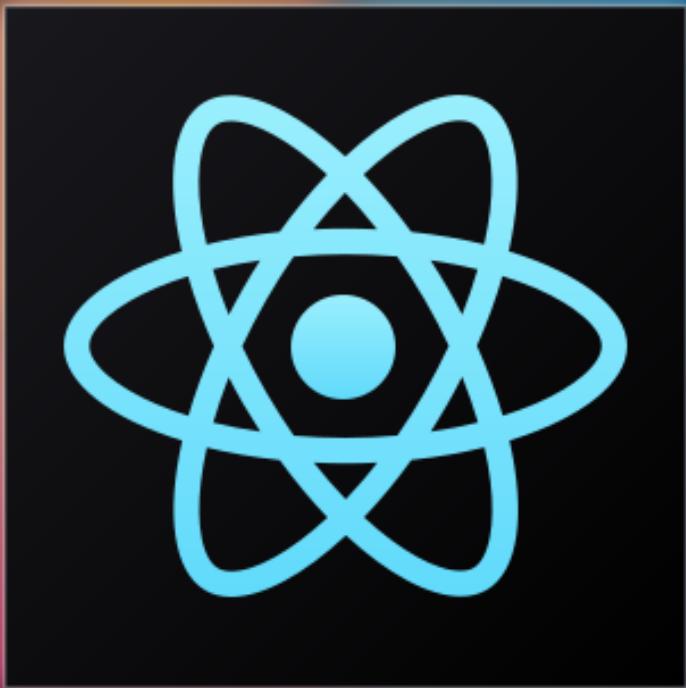
1.1. ¿Qué es React?

Biblioteca de Js creada por Facebook para construir interfaces de usuarios en interfaces web.

Se basa en componentes reutilizables, facilitando el desarrollo al evitar la repetición de código.

Funciona como una aplicación de una sola página. Carga el contenido desde los componentes para una renderización más rápida.

La sintaxis principal es JSX, una extensión de JS que integra la lógica con la interfaz. Su uso no es obligatorio pero mejora la eficiencia y legibilidad del código.



1.1.1 Principios Fundamentales

·Componentes Reutilizables

Cada componente representa una parte específica de la interfaz y puede ser reutilizado en diferentes partes de la aplicación.

·Unidireccionalidad de Datos

Los datos fluyen en una sola dirección. Los datos fluyen de los componentes padres a los componentes hijos. Universal: React se puede ejecutar tanto en el cliente como en el servidor.

·JSX (JavaScript XML)

Es una forma de escribir código que combina JavaScript y HTML de manera más legible. Hace que la creación de interfaces de usuario sea más sencilla al permitir escribir código similar al HTML

1.1.2 Virtual DOM

El DOM virtual (VDOM) es un concepto de programación donde una representación ideal o “virtual” de la IU se mantiene en memoria y en sincronía con el DOM “real”, mediante una biblioteca como ReactDOM.

El Virtual DOM minimiza las manipulaciones directas del DOM, lo que resulta en una renderización más eficiente y una mejor experiencia del usuario.

Resumen de cómo funciona:

Cambio de Datos: Cuando los datos cambian, React crea una nueva representación virtual del DOM.

Comparación: Compara la nueva representación virtual con la anterior para identificar los cambios.

Actualización eficiente: Solo los cambios identificados se aplican al DOM real, evitando actualizaciones innecesarias y mejorando el rendimiento.

Módulo 2: Componentes y Props

2.1. Componentes en React

2.1.1. Creación de componentes funcionales y de clase

2.1.2. Propiedades (Props) en React

Módulo 3: Manejo de eventos

3.1. Eventos en React

3.1.1. Eventos de usuario

3.1.2. Enlace de eventos y métodos

3.2. Formularios en React

3.1. Eventos en React

¿Qué son?

Acciones desencadenadas por el usuario, como hacer clic en un botón o escribir en un campo de entrada. Estos eventos permiten que las aplicaciones respondan dinámicamente a las acciones del usuario

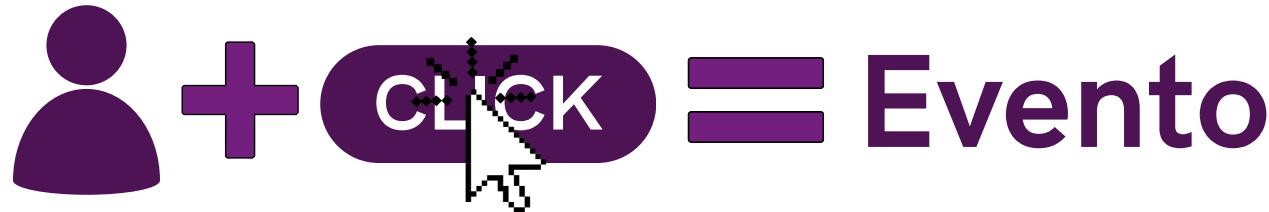
¿Qué ofrecen?

Mejorar la experiencia de usuario y la interactividad de la aplicación

¿Cómo se integran

los eventos se manejan utilizando sintaxis JSX similar a HTML, lo que facilita su integración en la lógica de los componentes.

3.1.1. Eventos de Usuario



React utiliza su propio sistema de manejo de eventos.

1. Se define el tipo de evento (onClick, onChange, ...)
2. Se crea la función que se ejecutará cuando ocurra el evento (Manejador de Eventos)
3. Se vincula el manejador al elemento de la interfaz de usuario.
4. Cuando el usuario realiza la acción, React ejecuta automáticamente el manejador especificado
5. El manejador de evento recibe un objeto de evento como argumento. Éste contiene información sobre el evento, la posición del ratón, el valor de un campo de entrada , ...

3.1.2. Enlace de eventos y métodos

Los enlaces de eventos se utilizan para manejar interacciones del usuario.

Se declaran usando una sintaxis similar a HTML, pero con **camelCase**.

Pueden definirse en el componente y luego referenciarlos en los enlaces de eventos. También es posible pasarles argumentos.

Los enlaces de eventos y métodos son herramientas clave en React para gestionar la interactividad y la lógica en los componentes de la interfaz de usuario.

Módulo 4: Conceptos Avanzados de Componentes

4.1. Composición de componentes

4.1.1. Reutilización de componentes

4.1.2. Patrones de composición

4.1. Composición de componentes

En el contexto de **desarrollo de software**, se refiere a la forma en que se pueden combinar y organizar diferentes partes de un sistema para construir aplicaciones más grandes y complejas. Esta práctica facilita la reutilización de código, mejora la mantenibilidad y promueve un diseño modular.

En **React**, es un principio esencial. Ya que la reutilización de componentes y la aplicación de patrones de composición son prácticas fundamentales.

4.1.1. Reutilización de componentes

Es un principio fundamental en el desarrollo de software.

Sus beneficios, **eficiencia, consistencia y mantenibilidad**.

La reutilización de componentes se puede lograr a través de **bibliotecas, frameworks y patrones de diseño** que facilitan la integración de componentes existentes en nuevos proyectos.

4.1.2. Patrones de composición

Son enfoques probados y prácticos para combinar componentes de manera efectiva.

En **React**, algunos patrones de composición son fundamentales para estructurar y organizar aplicaciones de manera eficiente.

Composición de Decorador

Se puede lograr mediante el uso de **HOCs** (Higher Order Components).

Estos son componentes que toman un componente y devuelven otro componente con funcionalidades adicionales.

Esto es útil cuando se desea extender o modificar el comportamiento de un componente sin cambiar su código fuente.

Otros patrones de composición:

- **Composición de Componentes:**

Construir componentes a partir de otros componentes, combinando partes más pequeñas para formar un todo.

- **Render Props:**

Pasar una función como prop a un componente para permitirle renderizar algo dentro del componente padre.

- **Hooks y Composición:**

Utilizar React Hooks para gestionar el estado y el ciclo de vida en componentes funcionales.

4.2. Contexto en React

4.2.1. Uso del contexto

4.2.2. Proveedores y consumidores de contexto

4.2. Contexto en React

“Imagina una aplicación con múltiples componentes anidados, y necesitas compartir ciertos datos entre ellos sin pasar las propiedades a través de cada nivel. Aquí es donde el contexto se vuelve útil.”

Es una característica que permite pasar datos a través del árbol de componentes sin tener que pasar explícitamente las propiedades a cada nivel.

4.2.1 Uso del contexto

Context está diseñado para compartir datos que pueden considerarse “globales” para un árbol de componentes en React, como el usuario autenticado actual, el tema o el idioma preferido.

El contexto se define utilizando **React.createContext()** y se comparte utilizando un componente **Provider** que envuelve a los componentes que necesitan acceder a ese contexto. Para consumir el contexto, se utiliza el componente **Consumer** o el hook **useContext** en componentes funcionales

4.2.2 Proveedores y consumidores de contexto

- **Proveedores de Contexto:** Un proveedor de contexto es un componente en React que proporciona el contexto a los componentes secundarios. Este componente utiliza la prop value para pasar los datos del contexto.
- Los componentes que desean acceder al valor proporcionado por el proveedor pueden hacerlo utilizando el **Consumer** o el hook **useContext**.

Ejemplos:

- **Proveedores de Contexto:**

```
const UserContext = React.createContext();
function App() {
  const nombreUsuario = "John";

  return (
    <UserContext.Provider value={nombreUsuario}>
      {/* Otros componentes que pueden acceder al nombreUsuario */}
    </UserContext.Provider>
  );
}
```

- **Usando Consumer:**

```
class ComponenteConsumidor extends React.Component {
  render() {
    return (
      <UserContext.Consumer>
        {nombreUsuario => <p>Hola, {nombreUsuario}</p>}
      </UserContext.Consumer>
    );
  }
}
```

Módulo 5: Navegación en React

5.1. Uso de React Router

5.1.1. Configuración básica

5.1.2. Rutas anidadas y parámetros

5.1. Uso de React Router

Partiendo desde el principio, React Router es la librería más popular para gestionar la navegación en aplicaciones internas hechas con dicha herramienta.

Su principal funcionalidad es proporcionar una forma declarativa y basada en componentes para manejar la navegación en aplicaciones React, facilitando la creación de interfaces de usuario dinámicas y amigables con el usuario. Y logrando a su vez reutilizar código de unas páginas en otras.

5.1.1. Configuración básica

Al ser React Router una librería solo nos tendremos que invocar un terminal en la raíz de nuestro proyecto y escribir el comando:

```
npm install react-router-dom
```

5.1.2. Rutas anidadas y parámetros

Definición de Rutas Anidadas:

- Agrupación de rutas React bajo la misma URL.
- Simplificación del código al organizar rutas relacionadas.
- Posibilidad de desplegar rutas desde la misma página de la aplicación.

Anidamiento de rutas

- Utilización de componentes <Route> para crear rutas simples y anidadas.
- Especificación de rutas "hijas" dentro del componente "padre".
- Componente <Outlet>: la Ruta padre utiliza el componente Outlet para renderizar la Ruta hijo coincidente.

```
function Public() {  
  return (  
    <div>  
      <Router>  
        <Menu />  
        <Routes>  
          <Route path="/rutas/" element={<RutasAnidadas />} >  
            <Route path="frontend" element={<p>FrontEnd ✨</p>}/>  
            <Route path="backend" element={<p>BackEnd 🤖</p>}/>  
          </Route>  
        </Routes>  
      </Router>  
    </div>  
  );  
}
```

```
function RutasAnidadas() {  
  return (  
    <div>  
      <ul style={{display: "flex"}}>  
        <li><Link to={"frontend"}>Frontend</Link></li>  
        <li><Link to={"backend"}>Backend</Link></li>  
      </ul>  
      <Outlet/>  
    </div>  
  );  
}
```

localhost:3000/rutas/backend

Ejemplo:

Parámetros en las rutas

¿Qué son los Parámetros en las Rutas?

- Herramienta para capturar valores dinámicos en las URL.
- Facilita la construcción de componentes reutilizables.
- Manipulación de datos basada en la ruta.

Ejemplo Práctico:

- Uso del componente principal App con <Router>.
- Creación de enlaces con diferentes valores para "username".
- Configuración de ruta con parámetro ":username" y componente UserProfile.

Funcionamiento:

- Enlaces dinámicos con valores de "username".
- React Router captura dinámicamente el valor en la ruta.
- Acceso al valor a través del objeto match.params en el componente UserProfile.

// Componente principal de la aplicación

```
const App = () => (
  <Router>
    <div>
      <ul>
        {/* Creación de enlaces con diferentes valores para "username" */}
        <li><Link to="/user/john">Usuario John</Link></li>
        <li><Link to="/user/jane">Usuario Jane</Link></li>
      </ul>

      {/* Ruta con parámetro ":username" y componente asociado UserProfile */}
      <Route path="/user/:username" component={UserProfile} />
    </div>
  </Router>
);
```

// Componente funcional UserProfile que recibe match como prop

```
const UserProfile = ({ match }) => (
  <div>
    <h2>Perfil de Usuario</h2>
    {/* Accediendo al parámetro de la ruta llamado "username" */}
    <p>Usuario: {match.params.username}</p>
  </div>
);
```

Ejemplo:

Módulo 6: Manejo de Estado Global

6.1. Introducción a la gestión de estado global

6.1.1. Context API

6.1.2. Redux

6.1. Introducción a la gestión de estado global

Tipos de estado:

- Local: específico del componente (useState)
- Global: accesible por todos los componentes (Context API y Redux)

El manejo del estado global permite:

- Compartir datos entre componentes sin necesidad de utilizar props.
- Solucionar el Prop Drilling: pasar datos a través de múltiples niveles.

6.1.2 Context API

Qué es:

- Característica que proporciona una forma de compartir datos mediante el árbol de componentes sin pasar **props** manualmente a todos los niveles.

Cuando usar:

- Principalmente cuando algunos datos tienen que ser accesibles por muchos componentes en diferentes niveles de anidamiento.

Características principales:

- **Provider:** Envuelve la jerarquía de componentes y provee un valor que será accesible por los componentes hijos. El valor puede ser cualquier tipo de dato, como un estado, funciones o cualquier otro objeto.
- **Consumer:** Se usa para consumir el valor proporcionado por el Provider. Se coloca dentro de los componentes hijos y permite acceder al valor del contexto y usarlo en el componente sin necesidad de pasar props manualmente

6.1.2 Context API

Como usarlo:

- 1. Crear el context:** Definir el context usando **React.createContext()**.
- 2. Proveer el valor:** Utilizar el componente '**Provider**' para envolver la parte superior de la aplicación y proveer el valor.
- 3. Consumir el valor:** Dentro de cualquier componente descendiente que necesite acceder al valor del contexto, utiliza el componente '**Consumer**' o el hook '**useContext**'.

6.1.2 Redux

Qué es:

- Biblioteca JavaScript

Principios:

- Única fuente de verdad: **store** contiene todo el estado de la aplicación.
- Estado de solo lectura: solo se puede modificar emitiendo una **acción**.
- Cambio mediante funciones puras: se usan **reducers**, que reciben el estado actual y la acción a realizar y devuelven un nuevo estado.

Instalación de Redux Toolkit y React Redux

- `npm i @reduxjs/Toolkit react-redux`

6.1.2 Redux

¿Cómo usarlo?

1. Definir un slice (segmento de estado global) usando **createSlice** de Redux Toolkit donde damos nombre, estado inicial y reducers. Se exporta.
2. Crear un store usando **configureStore** de React Toolkit. Se configura con un objeto reducer donde especificamos el slice al que nos referimos y el reductor que contiene las acciones de ese estado.
3. Integrar el store en la aplicación mediante **Provider** de react-redux, que va a contener todos los componentes con acceso al estado que se especifica en la propiedad store.
4. Crear componentes que interactúen con el Store usando **useSelector** de react-redux, que tiene acceso a los estados globales.

Módulo 7: Consumo de APIs

7.1. Realización de solicitudes HTTP

7.1.1. Uso de Fetch y Axios

7.1.2. Manejo de datos asincrónicos

7.1. Realización de solicitudes HTTP

Que es una API :

- Una API (Interfaz de Programación de Aplicaciones) permite la comunicación entre diferentes software.
- En el contexto web, se utiliza para solicitar y enviar datos entre el frontend y el backend.

El manejo del estado global permite:

- Compartir datos entre componentes sin necesidad de utilizar props.
- Solucionar el Prop Drilling: pasar datos a través de múltiples niveles.

7.2. Integración de datos en componentes

7.2.1. Actualización de componentes con datos externos

7.2.2. Manipulación de respuestas JSON

7.2. Integración de datos en componentes

La integración de datos en componentes se refiere a como los datos obtenidos de una API se incorporan en los componentes de React para su visualización y manipulación

7.2.1. Actualización de componentes con datos externos

La actualización de componentes con datos externos en React implica solicitar datos de una API, almacenarlos en el estado del componente y volver a renderizarlo para reflejar los nuevos datos.

Si se obtienen datos diferentes en futuras solicitudes, se actualiza el estado y se vuelve a renderizar el componente para mantener la interfaz de usuario actualizada.

Sin embargo, React solo renderiza el componente si los nuevos datos son diferentes a los antiguos, evitando renderizaciones innecesarias.

7.2.2. Manipulación de respuestas JSON

La manipulación de respuestas JSON en desarrollo web implica procesar los datos devueltos por una API en formato JSON para su uso en una aplicación.

Esto incluye el parseo de la cadena JSON a un objeto javascript mediante **JSON.parse()** o **json()** al usar **Fetch**. Posteriormente, se accede a los datos mediante notacion de puntos o corchetes. En casos donde se necesita cambiar el formato de los datos, se emplean métodos como **.map()**, **.filter()** o **.reduce()**.

Finalmente, los datos parseados y posiblemente transformados se almacenan en el estado del componente para su uso en el proceso de renderizado.

Gracias

Tibu Mayo González

Alicia López Vázquez

Pablo Márquez Gómez

Adrián Martínez Segura

Carmen Sánchez Martín

José Antonio Vergara Sánchez