«Talento Tech»

React JS

Clase 08

Buenos Aires Ciudad

Clase N° 8 | Introducción a Context API

Índice:

- Creación de Context API para el manejo de estado global.
- Uso de useContext para compartir datos entre componentes.
- Implementación del estado global para el carrito de compras.

Objetivos de la Clase:

- Comprender qué es Context API y cuándo es útil.
- Crear un contexto global para manejar el estado compartido en una aplicación React.
- Utilizar el hook useContext para acceder y actualizar datos globales desde diferentes componentes.
- Implementar un estado global para gestionar el carrito de compras.

Introducción a Context API

¿Qué es Context API?

Context API es una herramienta integrada en React que permite manejar estados globales sin necesidad de pasar props manualmente entre componentes. Es ideal para compartir datos o estados entre componentes que no tienen una relación directa en el árbol de la aplicación.

¿Cuándo usar Context API?

- Cuando varios componentes necesitan acceder al mismo estado (por ejemplo, autenticación, carrito de compras).
- Para evitar el "prop drilling", es decir, pasar props a través de múltiples niveles de componentes.

Creación de Context API para Manejo de Estado Global

Paso 1: Crear un contexto

En la raíz de tu proyecto, crea una carpeta llamada context y dentro de ella, un archivo CarritoContext.js.

```
import React, { createContext, useState } from 'react';

// Crear el contexto
export const CarritoContext = createContext();

// Proveedor del contexto
```

```
export function CarritoProvider({ children }) {
    const [carrito, setCarrito] = useState([]);

    const agregarAlCarrito = (producto) => {
        setCarrito([...carrito, producto]);
    };

    const vaciarCarrito = () => {
        setCarrito([]);
    };

    return (
        <CarritoContext.Provider value={{ carrito, agregarAlCarrito,
vaciarCarrito }}>
            {children}
        </CarritoContext.Provider>
    );
}
```

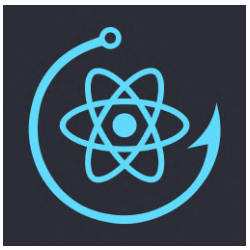
Paso 2: Envolver tu aplicación con el proveedor del contexto

En el archivo principal `index.js` o `App.js`, envuelve tu aplicación con el proveedor del contexto.

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { CarritoProvider } from './context/CarritoContext';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <CarritoProvider>
      <App />
    </CarritoProvider>
  </React.StrictMode>
);
```

Uso de useContext para Compartir Datos entre Componentes



El hook useContext permite consumir el contexto creado. Vamos a usarlo en componentes que necesitan acceder al estado del carrito.

Componente para mostrar el carrito:

```
import React, { useContext } from 'react';

import { CarritoContext } from '../context/CarritoContext';

function Carrito() {

    const { carrito, vaciarCarrito } = useContext(CarritoContext);

    return (

        <div>

            <h1>Carrito de Compras</h1>

            {carrito.length > 0 ? (

                <ul>

                    {carrito.map((producto, index) => (

                        <li key={index}>{producto.nombre} -
${producto.precio}</li>
```

```
                    ))}

                </ul>

            ) : (

                <p>El carrito está vacío.</p>

            )}

            {carrito.length > 0 && <button
onClick={vaciarCarrito}>Vaciar Carrito</button>}

        </div>

    );

}

export default Carrito;
```

Componente para agregar productos al carrito:

```
import React, { useContext } from 'react';

import { CarritoContext } from '../context/CarritoContext';



function Producto({ producto }) {

    const { agregarAlCarrito } = useContext(CarritoContext);

    return (

        <div>
```

```
            <h2>{producto.nombre}</h2>

            <p>Precio: ${producto.precio}</p>

            <button onClick={() => agregarAlCarrito(producto)}>Agregar
al Carrito</button>

        </div>

    );

}

export default Producto;
```

Implementación del Estado Global para el Carrito

Lista de Productos y Componente Principal:

```
import React from 'react';
import Producto from './Producto';
import Carrito from './Carrito';

function App() {
    const productos = [
        { id: 1, nombre: 'Producto 1', precio: 100 },
        { id: 2, nombre: 'Producto 2', precio: 200 },
        { id: 3, nombre: 'Producto 3', precio: 300 },
    ];
```

```
    return (
        <div>
            <h1>Tienda Online</h1>
            <div style={{ display: 'flex', justifyContent:
'space-between' }}>
                <div>
                    <h2>Productos</h2>
                    {productos.map((producto) => (
                        <Producto key={producto.id} producto={producto}
/>
                    ))}
                </div>
                <Carrito />
            </div>
        </div>
    );
}

export default App;
```

Consignas Pre-Entrega de Proyecto Final

¡Hemos llegado a un momento clave! Es el momento de realizar la pre-entrega del proyecto trabajado hasta el momento. Como es de costumbre en varias empresas, nos toca pronto juntarnos con nuestro cliente para ver como va creciendo su sitio web. Esto nos va a permitir estar más en sintonía con las necesidades de nuestro cliente esperando su feedback y nos

comentará por donde seguir. Por lo que venís trabajando te detallo lo que se espera tener esta primera presentación del sitio a nuestro cliente.

Requerimiento #1: Crear una funcionalidad básica para el manejo de un carrito de compras.

- Crear un componente para listar los productos disponibles.
- Usar el hook **useState** para manejar el estado del carrito.
- Implementar un evento de clic que permita agregar productos al carrito.
- Mostrar el carrito con los productos seleccionados en otro componente.
- Crear un Layout del eCommerce

Requerimiento #2: Conectar la aplicación a una API que provea información sobre los productos

- Integración con una API
- Gestión del estado con useState
- Manejo de efectos secundarios con useEffect
- Estado de carga y errores
- Actualizar el diseño del eCommerce
- Ampliación del carrito

Requerimiento #3: Integración de rutas

- Implementación de rutas
- Crear componente para cada sección
- Navegar entre productos
- Estado de carga y manejo de errores

Requerimiento #4: implementar rutas dinámicas y protegidas

- Rutas Dinámicas
- Rutas Protegidas
- Interactividad
- Navbar

Reflexión final

Aprendimos a usar Context API para manejar estados globales en aplicaciones React. Vimos cómo crear y consumir un contexto con useContext, y cómo implementar el estado global para gestionar un carrito de compras.

El uso de Context API nos ayuda a mantener una arquitectura más limpia y escalable, especialmente en aplicaciones complejas donde múltiples componentes necesitan compartir información. Esta herramienta es fundamental para crear aplicaciones React modernas y eficientes.

Materiales y Recursos Adicionales:

★ Documentación oficial de Context API
★ Context API en tus proyectos

Preguntas para Reflexionar:

- ¿Qué ventajas tiene usar Context API frente a pasar props entre componentes?
- ¿En qué situaciones sería más conveniente usar Redux en lugar de Context API?
- ¿Qué problemas podrías enfrentar al manejar múltiples contextos en una misma aplicación?
- ¿Cómo podrías optimizar el rendimiento de una aplicación que usa Context API intensivamente?

Próximos Pasos:

- Introducción a la Autenticación de Usuarios
- Implementación de formulario de login.
- Manejo de autenticación con tokens (simulada).
- Protección de rutas usando Context API para la autenticación.


Buenos Aires
aprende
Agencia de Habilidades para el Futuro
Buenos
Aires
Ciudad