«Talento Tech»

React JS

Clase 05

Buenos Aires Ciudad

Clase N° 5 | Uso de useEffect y Peticiones a APIs

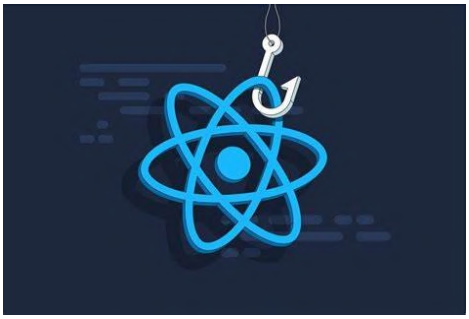
Índice:

- Introducción al manejo de efectos secundarios con useEffect.
- Realización de peticiones a APIs para cargar productos.
- Manejo del estado de carga y errores en aplicaciones React.

Objetivos de la Clase:

- Comprender qué son los efectos secundarios y cómo gestionarlos con el hook **useEffect**.
- Realizar peticiones a APIs para obtener datos dinámicamente.
- Integrar MockAPI como una herramienta para simular datos en tiempo real.
- Manejar estados de carga y errores para mejorar la experiencia del usuario.

Introducción a useEffect



¿Qué son los efectos secundarios en React? Los efectos secundarios son operaciones que ocurren fuera del flujo principal de renderizado de un componente. Algunos ejemplos comunes incluyen:

- Obtener datos desde una API.
- Manipular el DOM directamente.
- Establecer suscripciones o temporizadores.

El hook **useEffect** nos permite gestionar estos efectos secundarios en los componentes funcionales de React. Al usar **useEffect**, podemos controlar cuándo se ejecutan estas operaciones y cómo afectan al ciclo de vida del componente.

Sintaxis básica de useEffect:

```
useEffect(() => {
  // Código del efecto secundario
  return () => {
      // Limpieza del efecto (opcional)
  };
}, [dependencias]);
```

- **Efecto:** Se define en el primer parámetro como una función.
- **Limpieza:** Es opcional y se ejecuta antes de que el componente se desmonte o se actualice el efecto.
- **Dependencias:** Es un array opcional. Indica cuándo ejecutar el efecto (p. ej., al cambiar una variable).

Ejemplo:

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

function Mensaje() {
```

```
    useEffect(() => {
        console.log('El componente se ha montado.');

        return () => {
            console.log('El componente se ha desmontado.');
        };
    }, []);

    return <h1>Hola, React!</h1>;
}
```

En este ejemplo:

- El mensaje "El componente se ha montado." se muestra una vez al cargar el componente.
- El mensaje "El componente se ha desmontado." aparece al eliminarlo del DOM.
- El array de dependencias vacío ([]) asegura que el efecto se ejecute solo una vez, al montar el componente.

Casos comunes de uso:

1. Obtener datos de una API al montar el componente.
2. Actualizar el título del documento del navegador.
3. Establecer y limpiar temporizadores.

Realización de Peticiones a APIs.

Las peticiones a APIs permiten cargar datos dinámicamente desde un servidor. Esto es fundamental para crear aplicaciones que interactúen con datos en tiempo real o contenidos dinámicos.

Pasos principales:

1. Usar **fetch** o librerías como **axios** para realizar la petición.
2. Gestionar el estado local para almacenar los datos obtenidos.
3. Usar **useEffect** para realizar la petición cuando el componente se monte.

Ejemplo:

Obtener productos desde una API:

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

function Productos() {
    const [productos, setProductos] = useState([]);

    useEffect(() => {
        fetch('https://api.ejemplo.com/productos')
            .then((respuesta) => respuesta.json())
            .then((datos) => setProductos(datos))
            .catch((error) => console.error('Error:', error));
    }, []);

    return (
        <ul>
            {productos.map((producto) => (
                <li key={producto.id}>{producto.nombre}</li>
            ))}
```

```
        </ul>

    );

}

export default Productos;
```

En este ejemplo:

- Se realiza una petición a la API cuando el componente se monta.
- Los datos obtenidos se almacenan en el estado `productos`.
- La función `catch` gestiona posibles errores en la petición.

Introducción a MockAPI.

¿Qué es MockAPI? MockAPI es una herramienta en línea que permite simular un backend RESTful. Es ideal para proyectos de desarrollo y aprendizaje donde un servidor real no está disponible.

Características principales:

- Creación rápida de endpoints para realizar peticiones HTTP.
- Interfaz gráfica sencilla para gestionar datos.
- Simulación de operaciones CRUD (Crear, Leer, Actualizar, Eliminar).

Ventajas de usar MockAPI:

- Permite trabajar con datos realistas sin necesidad de configurar un servidor.
- Facilita pruebas y desarrollo rápidos.
- Compatible con **fetch** y **axios**.

Configuración de MockAPI:

1. Accede a Mock API y crea una cuenta.
2. Crea un proyecto y define un recurso (p. ej., "productos").
3. MockAPI generará un endpoint, como: “https://63f123.mockapi.io/productos”
4. Usa este endpoint para realizar peticiones HTTP desde tu aplicación.

Ejemplo con MockAPI:

Cargar productos desde MockAPI:

```
useEffect(() => {
  fetch('https://63f123.mockapi.io/productos')
      .then((respuesta) => respuesta.json())
      .then((datos) => setProductos(datos))
      .catch((error) => console.error('Error:', error));
}, []);
```

Manejo del Estado de Carga y Errores

Cuando realizamos peticiones a APIs, es importante manejar estados de carga y errores para mejorar la experiencia del usuario.

Estado de carga

El estado de carga indica si los datos están en proceso de ser obtenidos.

```
const [cargando, setCargando] = useState(true);
```

- Muestra un spinner o mensaje mientras se cargan los datos.

Estado de error

El estado de error guarda información sobre posibles fallos.

```
const [error, setError] = useState(null);
```

- Informa al usuario si ocurre un problema.

Ejemplo:

```
export default function Productos() {
  const [productos, setProductos] = useState([]);
  const [cargando, setCargando] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
      fetch('https://63f123.mockapi.io/productos')
          .then((respuesta) => respuesta.json())
          .then((datos) => {
              setProductos(datos);
              setCargando(false);
          })
          .catch((error) => {
              setError('Hubo un problema al cargar los productos.');
              setCargando(false);
          });
  }, []);

  if (cargando) return <p>Cargando productos...</p>;
  if (error) return <p>{error}</p>;

  return (
      <ul>
          {productos.map((producto) => (
              <li key={producto.id}>{producto.nombre}</li>
          ))}
      </ul>
  );
}
```

TalentoLab - Proyecto Final

¡Felicitaciones nuevamente! 🎉 Ahora que has dominado la primera etapa de tu tarea en **TalentoLab**, el equipo ha decidido asignarte una responsabilidad más avanzada. Esta vez, deberás mejorar el eCommerce integrando una API para obtener datos reales de productos y optimizar la experiencia del usuario con manejo de estados de carga y errores.

Descripción de tu tarea:

Tu objetivo será conectar la aplicación a una API que provea información sobre los productos. Deberás mostrar los datos en tiempo real, manejar estados de carga y errores, y continuar utilizando un diseño atractivo para el eCommerce.

Requisitos:

1. **Integración con una API:**
    - Usa **MockAPI** para obtener una lista de productos desde un endpoint.
    - Asegúrate de configurar correctamente la URL del recurso.
2. **Gestión del estado con useState:**
    - Almacena los productos obtenidos de la API en un estado local.
    - Usa otro estado para gestionar si la aplicación está en proceso de carga.
3. **Manejo de efectos secundarios con useEffect:**
    - Realiza la petición a la API cuando el componente principal de productos se monte.
    - Gestiona posibles errores y muestra un mensaje adecuado si ocurre algún problema.
4. **Estado de carga y errores:**
    - Mientras los productos están cargando, muestra un mensaje o un spinner con "Cargando productos...".
    - Si ocurre un error, muestra un mensaje como: "Error al cargar productos. Inténtalo más tarde."

5. **Actualizar el diseño del eCommerce:**
    - Crea una sección de productos dinámicos que se cargue desde la API.
    - Asegúrate de mantener el diseño atractivo y responsivo.
6. **Ampliación del carrito:**
    - Permite agregar al carrito productos que ahora se obtendrán desde la API.
    - Si el carrito está vacío, muestra el mensaje "El carrito está vacío", como en el ejercicio anterior.

Pautas Generales:

- **Estructura del proyecto:**
    - Continúa trabajando con el proyecto creado en la clase anterior.
    - Agrega una nueva funcionalidad en el componente principal para realizar la integración con la API.
- **Componentes:**
    - Mantén los componentes organizados y reutilizables. Por ejemplo:
        - Un componente para el **listado de productos**.
        - Un componente para el **carrito de compras**.
- **Código limpio:**
    - Evita redundancias y organiza tus estados y efectos de forma clara.
- **Diseño:**
    - Usa **Bootswatch** o una alternativa para mantener un diseño profesional.

Reflexión final

En esta clase, aprendimos a gestionar efectos secundarios con **useEffect**, realizar peticiones a APIs para cargar datos y manejar estados de carga y errores. También descubrimos cómo utilizar MockAPI para simular un backend, lo que nos permite trabajar con datos realistas sin necesidad de configurar un servidor.

Estas habilidades son esenciales para construir aplicaciones React que interactúen con datos externos, brindando una experiencia de usuario fluida y profesional. Al dominar estos conceptos, estarás preparado para desarrollar aplicaciones más completas y robustas.

Materiales y Recursos Adicionales:

- ★ Documentación oficial de useEffect
- ★ Guía introductoria a las promesas en JavaScript
- ★ MockAPI: Plataforma de simulación de datos
- ★ Estado y ciclo de vida en React
- ★ Cómo manejar errores en JavaScript

Preguntas para Reflexionar:

- ¿Qué diferencia a **useEffect** de otros hooks como **useState**?
- ¿Cómo decidirías qué variables incluir en el array de dependencias de **useEffect**?
- ¿Por qué es importante manejar estados de carga y errores al trabajar con datos de APIs?
- ¿Qué ventajas tiene usar una herramienta como MockAPI frente a configurar un servidor propio?
- ¿Cómo podrías integrar múltiples peticiones a APIs en un solo componente sin afectar el rendimiento?

Próximos Pasos:

- Instalación y configuración de React Router.
- Creación de rutas estáticas (inicio, lista de productos).
- Uso de Link para navegar entre componentes.

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad