«Talento Tech»

React JS

Clase 11

Buenos Aires Ciudad

Clase N° 11 | Leer, Actualizar y Eliminar productos

Índice:

- Implementación de la funcionalidad para leer productos desde el estado o API.
- Actualización y eliminación de productos.
- Validación de los formularios de edición

Objetivos de la Clase:

- Leer y renderizar productos almacenados en MockAPI.
- Implementar la funcionalidad de edición de productos mediante formularios controlados.
- Permitir la eliminación de productos con confirmación del usuario.
- Validar los datos en el formulario de edición para garantizar la consistencia.

Implementación de la funcionalidad para leer productos desde el estado o API

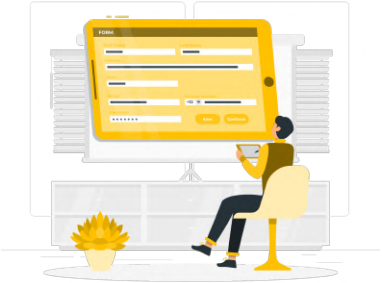
Primero, configuraremos la funcionalidad para obtener y renderizar productos almacenados en MockAPI. Este proceso se realizará con el hook `useEffect` y el método `fetch`. Este enfoque es útil para inicializar el estado de nuestra aplicación con datos provenientes de una API.

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

function ListaProductos() {

  const [productos, setProductos] = useState([]);

  useEffect(() => {

    const fetchProductos = async () => {

      try {

        const respuesta = await
fetch('https://mockapi.io/api/v1/productos');

        if (!respuesta.ok) {

          throw new Error('Error al obtener los productos.');

        }

        const data = await respuesta.json();

        setProductos(data);

      } catch (error) {

        console.error(error.message);

      }

    };

    fetchProductos();
```

```
  }, []);

  return (
    <div>
      <h2>Lista de Productos</h2>
      <ul>
        {productos.map((producto) => (
          <li key={producto.id}>
            <strong>{producto.nombre}</strong>: ${producto.precio}
            <p>{producto.descripcion}</p>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default ListaProductos;
```

Este componente obtendrá los productos desde MockAPI al cargarse y los renderizará en una lista. Es importante manejar posibles errores durante la obtención de datos para evitar que fallos en la red afecten la funcionalidad de la aplicación.

Actualización y eliminación de productos



La edición de productos se realizará mediante un formulario que permita actualizar sus datos. Este formulario se abrirá al seleccionar un producto para editar. La práctica de formularios controlados es clave para garantizar que los datos del formulario estén siempre sincronizados con el estado del componente.

Pasos para implementar:

1. Crear un formulario controlado similar al de creación de productos.
2. Poblar el formulario con los datos existentes del producto seleccionado.
3. Validar los datos antes de enviarlos.
4. Realizar una solicitud PUT a MockAPI para actualizar el producto.

Código del formulario de edición:

```
function FormularioEdicion({ productoSeleccionado, onActualizar }) {
  const [producto, setProducto] = useState(productoSeleccionado);

  useEffect(() => {
    setProducto(productoSeleccionado);
  }, [productoSeleccionado]);

  const handleChange = (e) => {
    const { name, value } = e.target;
    setProducto({ ...producto, [name]: value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const respuesta = await
fetch(`https://mockapi.io/api/v1/productos/${producto.id}`, {
```

```
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(producto),
    });

    if (!respuesta.ok) {
      throw new Error('Error al actualizar el producto.');
    }

    const data = await respuesta.json();
    onActualizar(data);
    alert('Producto actualizado correctamente.');
  } catch (error) {
    console.error(error.message);
    alert('Hubo un problema al actualizar el producto.');
  }
};

return (
  <form onSubmit={handleSubmit}>
    <h2>Editar Producto</h2>
    <div>
      <label>Nombre:</label>
      <input
        type="text"
        name="nombre"
        value={producto.nombre || ''}
        onChange={handleChange}
        required
      />
    </div>
    <div>
      <label>Precio:</label>
      <input
        type="number"
        name="precio"
        value={producto.precio || ''}
        onChange={handleChange}
        required
        min="0"
```

```
        />
      </div>
      <div>
        <label>Descripción:</label>
        <textarea
          name="descripcion"
          value={producto.descripcion || ''}
          onChange={handleChange}
          required
        />
      </div>
      <button type="submit">Actualizar Producto</button>
    </form>
  );
}
```

Este formulario también es útil para demostrar cómo reutilizar el mismo componente para distintos propósitos, como creación y edición, con solo pequeños ajustes.

Al hacer clic en un botón de "Editar" en la lista de productos, se abrirá el formulario de edición con los datos del producto seleccionado. Esto permite al usuario actualizar fácilmente los datos sin abandonar la página principal.

Eliminación de productos

Para eliminar un producto, implementaremos un botón que solicite confirmación al usuario antes de realizar una solicitud `DELETE` a MockAPI. La confirmación evita eliminaciones accidentales, lo cual es una buena práctica de diseño.

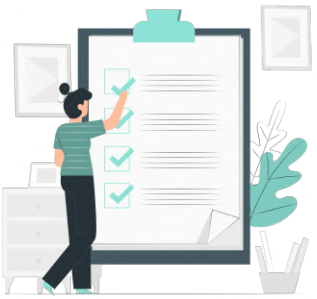
```
const eliminarProducto = async (id) => {

  const confirmar = window.confirm('¿Estás seguro de que deseas
eliminar este producto?');

  if (confirmar) {

    try {

      const respuesta = await
fetch(`https://mockapi.io/api/v1/productos/${id}`, {

        method: 'DELETE',

      });

      if (!respuesta.ok) {

        throw new Error('Error al eliminar el producto.');

      }

      alert('Producto eliminado correctamente.');

    } catch (error) {

      console.error(error.message);

      alert('Hubo un problema al eliminar el producto.');

    }

  }

};
```

En la lista de productos, añadiremos un botón de "Eliminar" que invoque esta función al ser clicado. Es importante mostrar mensajes al usuario para confirmar la acción realizada.

Validación de los formularios de edición

Para garantizar la consistencia de los datos, reutilizaremos las validaciones del formulario de creación:

- **Nombre obligatorio:** Esto asegura que cada producto tenga un identificador claro.
- **Precio mayor a 0:** Evita errores relacionados con precios inválidos o negativos.
- **Descripción con al menos 10 caracteres:** Esto fomenta descripciones más detalladas y útiles.



Los mensajes de error se mostrarán junto a los campos correspondientes para mejorar la experiencia del usuario y ayudarle a corregir errores de manera inmediata.

Nueva Tarea en Talento Lab

El cliente de **Talento Lab** ha quedado impresionado con el progreso del equipo y ahora quiere mejorar aún más la plataforma. Para ello, ha solicitado la funcionalidad de **editar y eliminar productos** del catálogo. Es fundamental que los usuarios puedan **modificar la información** de un producto cuando sea necesario y también **eliminar aquellos que ya no sean relevantes**. Además, se deben incluir validaciones para evitar errores en la edición y una confirmación antes de eliminar un producto.

Objetivos:

1. **Integrar el formulario de agregar/editar productos** con el estado global de la aplicación utilizando **Context API**.
2. **Validar los datos del formulario** para asegurar que la información sea consistente.
3. **Manejar errores** y proporcionar retroalimentación clara al usuario.

Requerimientos

1. **Conectar el formulario con el estado global**

- Implementar **Context API** para manejar el estado de los productos en un solo lugar.
- Crear un **ProductsProvider** que almacene la lista de productos y las funciones para **agregar, editar y eliminar** productos.
- Envolver la aplicación con el `ProductsProvider` en `main.jsx` para que todos los componentes tengan acceso al estado global.

2. **Formulario controlado de agregar/editar productos**

- Crear un formulario reutilizable en React para agregar y editar productos.
- El formulario debe manejar el estado del producto mediante **useState**.
- Implementar la lógica para diferenciar entre **modo "agregar" y "editar"**.
- Al enviar el formulario, actualizar el estado global llamando a las funciones correspondientes (`agregarProducto` o `editarProducto`).

3. **Validaciones del formulario**

- Todos los campos son **obligatorios**.
- **El precio** debe ser un número mayor a **0**.
- **La descripción** debe tener al menos **10 caracteres**.
- Mostrar mensajes de error junto a los campos que no cumplan con las validaciones.

4. **Manejo de errores y feedback al usuario**

- Si los datos ingresados no cumplen con las reglas, mostrar un mensaje de error.

- Si la operación de agregar/editar es exitosa, limpiar el formulario y mostrar un mensaje de confirmación.

Reflexión final

En esta clase aprendimos a:

- Obtener y renderizar datos desde una API.
- Implementar la funcionalidad de edición y eliminación de productos.
- Validar formularios para mantener la calidad de los datos.

Estos conceptos refuerzan las habilidades necesarias para construir aplicaciones completas y funcionales que interactúan con APIs. Además, permiten que los estudiantes comprendan el flujo de datos entre el cliente y el servidor.

Materiales y Recursos Adicionales:

★ Documentación de MockAPI
★ Uso del método fetch
★ Validación de Formularios en React

Preguntas para Reflexionar:

- ¿Qué beneficios y desafíos has encontrado al trabajar con APIs como MockAPI para manejar datos dinámicos en tus aplicaciones?
- ¿Cómo podrías mejorar la experiencia de usuario en las funcionalidades de edición y eliminación de productos para que sean más intuitivas y visualmente atractivas?
- ¿Qué estrategias podrías implementar para manejar errores de manera más eficiente cuando las solicitudes a la API fallan?

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad