«Talento Tech»

React JS

Clase 12

Buenos Aires Ciudad

Clase N° 12 | Implementación del CRUD Completo

Índice:

- Conectar el formulario de agregar/editar productos con el estado global.
- Validación de formularios y manejo de errores.

Objetivos de la Clase:

- Integrar el formulario de agregar/editar productos con el estado global de la aplicación.
- Validar los datos del formulario para garantizar su calidad.
- Manejar errores y proporcionar retroalimentación al usuario.

Conexión del formulario

Para manejar el estado de los productos a nivel global, utilizaremos el Context API. Esto nos permitirá compartir el estado entre componentes sin necesidad de pasarlo como props.

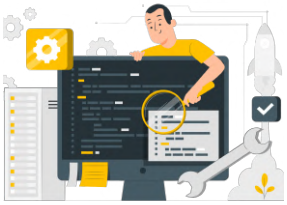
Pasos para implementar el Context API:

1. **Crear el contexto global de productos:**
   Configuraremos un `ProductsProvider` que almacenará el estado global y las funciones para manipular los productos.

Código del Contexto Global:

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

export const ProductsContext = createContext();

export const ProductsProvider = ({ children }) => {

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

  const agregarProducto = (nuevoProducto) => {

    setProductos([...productos, nuevoProducto]);

  };

  const editarProducto = (productoActualizado) => {

    setProductos(

      productos.map((producto) =>

        producto.id === productoActualizado.id ? productoActualizado :
producto
```

```
      )
    );
  };
  const eliminarProducto = (id) => {
    setProductos(productos.filter((producto) => producto.id !== id));
  };
  return (
    <ProductsContext.Provider
      value={{ productos, agregarProducto, editarProducto,
eliminarProducto }}
    >
      {children}
    </ProductsContext.Provider>
  );
};
```



2. **Integrar el Contexto en la aplicación:**
Envolvemos la aplicación con el `ProductsProvider` para que los componentes puedan acceder al estado global.

Código en `main.jsx`:

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

import { ProductsProvider } from './context/ProductsContext';

ReactDOM.createRoot(document.getElementById('root')).render(
```

```
  <ProductsProvider>
    <App />
  </ProductsProvider>
);
```

3. **Conectar el formulario con el estado global:**
   Usaremos el contexto en los componentes `FormularioEdicion` y `ListaProductos` para leer y actualizar productos.

Formulario de Agregar/Editar conectado al estado global:

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

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

function FormularioProducto({ productoInicial = {}, modo = 'agregar',
onCerrar }) {

  const [producto, setProducto] = useState(productoInicial);

  const { agregarProducto, editarProducto } =
useContext(ProductsContext);

  const handleChange = (e) => {

    const { name, value } = e.target;

    setProducto({ ...producto, [name]: value });

  };

  const handleSubmit = (e) => {

    e.preventDefault();

    if (modo === 'agregar') {

      agregarProducto({ ...producto, id: Date.now() });

    } else {

      editarProducto(producto);

    }

    onCerrar();

  };

  return (
```

```
    <form onSubmit={handleSubmit}>

      <h2>{modo === 'agregar' ? 'Agregar Producto' : '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">{modo === 'agregar' ? 'Agregar' :
'Actualizar'}</button>
    </form>
  );
}
export default FormularioProducto;
```

Validación de formularios y manejo de errores

Es crucial garantizar que los datos ingresados sean válidos y que el usuario reciba retroalimentación en caso de errores. Implementaremos validaciones y mensajes de error para el formulario.

Validación de datos:

1. **Campos requeridos:** Todos los campos deben completarse.
2. **Restricciones específicas:**
    - El precio debe ser mayor a 0.
    - Descripción con al menos 10 caracteres.

Código con validaciones:

```
const [errores, setErrores] = useState({});

const validarFormulario = () => {

  const nuevosErrores = {};

  if (!producto.nombre) nuevosErrores.nombre = 'El nombre es
obligatorio.';

  if (producto.precio <= 0) nuevosErrores.precio = 'El precio debe ser
mayor a 0.';

  if (!producto.descripcion || producto.descripcion.length < 10)

    nuevosErrores.descripcion = 'La descripción debe tener al menos 10
caracteres.';

  return nuevosErrores;

};

const handleSubmit = (e) => {

  e.preventDefault();

  const nuevosErrores = validarFormulario();

  if (Object.keys(nuevosErrores).length > 0) {

    setErrores(nuevosErrores);

    return;

  }

  if (modo === 'agregar') {

    agregarProducto({ ...producto, id: Date.now() });

  } else {

    editarProducto(producto);

  } onCerrar();

};
```

Los errores se mostrarán junto a los campos correspondientes:

```
<div>
  <label>Nombre:</label>
  <input
    type="text"
    name="nombre"
    value={producto.nombre || ''}
    onChange={handleChange}
    required
  />
  {errores.nombre && <p className="error">{errores.nombre}</p>}
</div>
<div>
  <label>Precio:</label>
  <input
    type="number"
    name="precio"
    value={producto.precio || ''}
    onChange={handleChange}
    required
    min="0"
  />
  {errores.precio && <p className="error">{errores.precio}</p>}
</div>
<div>
```

```
  <label>Descripción:</label>

  <textarea

    name="descripcion"

    value={producto.descripcion || ''}

    onChange={handleChange}

    required

  />

  {errores.descripcion && <p
classNAme="error">{errores.descripcion}</p>}

</div>
```

Ruta de avance

Este apartado detalla el progreso esperado del proyecto hasta el momento. Si en tu desarrollo actual notas que falta alguna funcionalidad o que hay áreas por mejorar, este es un buen momento para revisarlo y ajustarlo.

1️⃣ Manejo del Estado Global

- **Carrito de Compras**: Se creó `CarritoContext.js` para gestionar productos en el carrito, con funciones para agregar y vaciar. El carrito solo es accesible para usuarios autenticados mediante rutas protegidas.
- **Autenticación**: `AuthContext.js` maneja el login/logout con localStorage. Se implementó un formulario de inicio de sesión y redirección tras autenticación.

2️⃣ Gestión de Productos

- **Agregar Productos**: Se desarrolló un formulario controlado con validaciones en tiempo real. Se implementó una solicitud `POST` para enviar datos a MockAPI y manejar respuestas.

3️⃣ Edición y Eliminación de Productos

- **Estado Global**: `ProductsProvider` gestiona la lista de productos.

- **Edición**: El formulario reutilizable diferencia entre agregar y editar, enviando una solicitud `PUT` a MockAPI.
- **Eliminación**: Se implementó una función con confirmación del usuario antes de ejecutar la solicitud `DELETE`.

📌 Si aún no implementaste la edición o eliminación, asegúrate de agregar estas funcionalidades antes de seguir optimizando.

Nueva Tarea en Talento Lab

El cliente de **Talento Lab** está emocionado con los avances y ahora necesita que la aplicación maneje el **CRUD completo** de productos. Para lograrlo, debemos conectar el formulario de agregar y editar productos con el estado global, validar los datos ingresados y manejar errores para mejorar la experiencia del usuario.

Objetivos:

1. Diseñar un formulario controlado en React para agregar productos.
2. Implementar validaciones dinámicas para los datos ingresados.
3. Conectar la aplicación con MockAPI para almacenar los nuevos productos.

Requerimientos

1. Crear un Formulario Controlado

- Diseña un formulario en React que permita ingresar:
    - Nombre del producto.
    - Precio (en números).

    - Descripción (mínimo 10 caracteres).
- Asegúrate de manejar el estado del formulario mediante el hook `useState`.

2. Validaciones del Formulario

- Implementa las siguientes reglas de validación:
    - Todos los campos son obligatorios.
    - El precio debe ser mayor a 0.
    - La descripción debe tener al menos 10 caracteres.
- Muestra mensajes de error junto a los campos correspondientes.

3. Conectar con MockAPI

- Configura una función para enviar los datos del producto mediante una solicitud POST a MockAPI.
- Si el producto se agrega correctamente:
    - Limpia el formulario.
    - Muestra un mensaje de éxito.
- Si ocurre un error:
    - Muestra un mensaje de error en pantalla.

Reflexión final

En esta clase completamos la implementación del CRUD conectando los formularios al estado global, validando los datos y proporcionando retroalimentación clara al usuario en caso de errores. Esto asegura que la aplicación sea funcional, robusta y fácil de usar.

Materiales y Recursos Adicionales:

★ Sitio oficial de React Hook Form
★ React Context API - The Basics (YouTube)

Preguntas para Reflexionar:

- ¿Qué ventajas ofrece el uso del Context API frente a pasar props manualmente en aplicaciones React más grandes?
- ¿Cómo mejorarías la validación del formulario para manejar errores más complejos, como límites máximos o valores específicos?
- ¿Qué técnicas o librerías adicionales podrías implementar para manejar de manera más eficiente los errores y notificaciones en la interfaz de usuario?

Próximos Pasos:

- Introducción a Bootstrap o styled-components para estilizar componentes.
- Creación de un diseño básico y responsive.
- Aplicación de estilos en componentes (botones, formularios, productos).

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad