«Talento Tech»

React JS

Clase 10

Buenos Aires Ciudad

Clase N° 10 | Creación de Productos - Formulario y Validación

Índice:

- Creación de formulario para agregar productos.
- Validación de datos de formularios.
- Implementación de la funcionalidad para agregar productos.

Objetivos de la Clase:

- Diseñar un formulario en React para agregar nuevos productos.
- Validar los datos ingresados en el formulario para garantizar consistencia y prevenir errores.
- Implementar la funcionalidad de agregar productos mediante solicitudes a una API usando MockAPI.
- Comprender cómo manejar el estado del formulario y realizar validaciones dinámicas.

Creación de formulario para agregar productos

En esta clase, vamos a construir una funcionalidad completa para agregar productos a nuestra lista mediante un formulario interactivo.

Para hacerlo:

- Crearemos un formulario que permita al usuario ingresar los datos del producto.
- Validaremos esos datos antes de enviarlos para asegurarnos de que cumplen con los requisitos esperados.
- Realizaremos una solicitud a MockAPI para agregar el producto y confirmar que se creó correctamente.

El primer paso será crear un formulario en React que permita al usuario ingresar información como el nombre, precio y descripción del producto.

Formularios en React

Un formulario en React no es solo un conjunto de campos para ingresar datos, sino que requiere manejar su estado de manera controlada. Esto significa que cada cambio en un campo del formulario debe reflejarse inmediatamente en el estado de la aplicación.

Para lograr esto:

1. Creamos un estado local que almacene los datos del formulario.
2. Manejamos los eventos `onChange` para actualizar este estado.
3. Implementamos un evento `onSubmit` para procesar los datos al enviar el formulario.

Implementación paso a paso:

Primero, necesitamos definir un estado local para almacenar los datos ingresados por el usuario en los campos del formulario. Este estado será un objeto con propiedades como `nombre`, `precio` y `descripcion`.

```
const [producto, setProducto] = useState({
  nombre: '',
  precio: '',
  descripcion: '',
});
```

Cada vez que el usuario escriba algo en un campo del formulario, capturaremos ese cambio y actualizaremos el estado. Esto se logra mediante el evento `onChange` y una función que actualice el estado basado en el nombre del campo.

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

Al enviar el formulario, los datos del estado `producto` estarán listos para ser procesados.

Estructura completa del formulario:

El formulario tendrá campos para el nombre, precio y descripción del producto. Además, incluirá un botón para enviar los datos.

Aquí está la estructura final:

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

function FormularioProducto({ onAgregar }) {
  const [producto, setProducto] = useState({
    nombre: '',
    precio: '',
    descripcion: '',
  });

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

  const handleSubmit = (e) => {
    e.preventDefault();
    onAgregar(producto); // Llamada a la función para agregar el
producto
    setProducto({ nombre: '', precio: '', descripcion: '' }); //
Limpiar el formulario
  };

  return (
    <form onSubmit={handleSubmit}>
      <h2>Agregar 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">Agregar Producto</button>
    </form>
  );
}

export default FormularioProducto;
```

Cada elemento del formulario está conectado al estado mediante `value` y `onChange`. Esto asegura que React controle completamente los datos ingresados.

Validación de datos de formularios

¿Por qué validar datos?

La validación de datos asegura que los productos creados sean consistentes, con los valores esperados y sin errores que puedan afectar la funcionalidad o la presentación de la aplicación.

Validaciones implementadas:

1. **Requerimientos básicos:** Todos los campos son obligatorios.
2. **Validación de precio:** Debe ser un número mayor o igual a 0.
3. **Largo mínimo y máximo:** Los campos deben cumplir con límites de caracteres para garantizar consistencia.

Cómo implementar validaciones

Antes de procesar el formulario en el evento `onSubmit`, verificaremos los datos y mostraremos errores si no cumplen los requisitos. Usaremos un estado adicional para almacenar los errores y mostrarlos al usuario.

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

  if (!producto.nombre.trim()) {
    nuevosErrores.nombre = 'El nombre es obligatorio.';
  }
  if (!producto.precio || producto.precio <= 0) {
    nuevosErrores.precio = 'El precio debe ser mayor a 0.';
  }
  if (!producto.descripcion.trim() || producto.descripcion.length < 10)
{
    nuevosErrores.descripcion = 'La descripción debe tener al menos 10
caracteres.';
  }

  setErrores(nuevosErrores);
  return Object.keys(nuevosErrores).length === 0;
};

const handleSubmit = (e) => {
  e.preventDefault();
  if (validarFormulario()) {
    onAgregar(producto);
    setProducto({ nombre: '', precio: '', descripcion: '' });
    setErrores({});
  }
};
```

Renderizado de errores en el formulario:

```
{errores.nombre && <p style={{ color: 'red' }}>{errores.nombre}</p>}
{errores.precio && <p style={{ color: 'red' }}>{errores.precio}</p>}
{errores.descripcion && <p style={{ color: 'red'
}}>{errores.descripcion}</p>}
```

Al enviar el formulario, si las validaciones no se cumplen, los errores se mostrarán junto a los campos.

Implementación de la funcionalidad para agregar productos

Ahora que tenemos el formulario con validación, procederemos a realizar solicitudes a MockAPI para agregar productos.

Conexión con MockAPI

MockAPI es una herramienta excelente para simular una API real. Vamos a configurar nuestra aplicación para realizar solicitudes HTTP mediante `fetch`.

Explicación del flujo:

1. Al enviar el formulario, llamaremos a una función `agregarProducto`.
2. Esta función enviará una solicitud `POST` a MockAPI con los datos del producto.
3. Si la solicitud es exitosa, el producto será almacenado en MockAPI.

Código para agregar productos a MockAPI:

```
const agregarProducto = async (producto) => {

  try {
```

```
    const respuesta = await
fetch('https://mockapi.io/api/v1/productos', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(producto),
    });
    if (!respuesta.ok) {
      throw new Error('Error al agregar el producto.');
    }
    const data = await respuesta.json();
    console.log('Producto agregado:', data);
    alert('Producto agregado correctamente');
  } catch (error) {
    console.error(error.message);
    alert('Hubo un problema al agregar el producto.');
  }
};
```

Integración en el componente principal:

```
import React from 'react';
```

```
import FormularioProducto from './FormularioProducto';

function App() {

  const agregarProducto = async (producto) => {

    try {

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

        method: 'POST',

        headers: {

          'Content-Type': 'application/json',

        },

        body: JSON.stringify(producto),

      });

      if (!respuesta.ok) {

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

      }

const data = await respuesta.json();

      console.log('Producto agregado:', data);

      alert('Producto agregado correctamente');

    } catch (error) {

      console.error(error.message);

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

```
    }
  };
  return (
    <div>
      <h1>Gestión de Productos</h1>
      <FormularioProducto onAgregar={agregarProducto} />
    </div>
  );
}
export default App;
```

Nueva Tarea en Talento Lab

¡El cliente de Talento Lab está emocionado con los avances! Ahora necesita una funcionalidad que permita agregar nuevos productos al catálogo de su sitio. Este proceso debe incluir un formulario amigable con validaciones para garantizar que los datos ingresados sean consistentes.

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

Hoy aprendimos a construir una funcionalidad completa para agregar productos en React. Este flujo incluyó:

1. Crear un formulario con manejo de estado.
2. Validar los datos ingresados por el usuario.
3. Enviar estos datos a una API (MockAPI) para almacenarlos.

Este proceso no solo refuerza los conceptos básicos de React, sino que también introduce prácticas fundamentales para desarrollar aplicaciones interactivas que interactúan con APIs.

Materiales y Recursos Adicionales:

★ Documentación de MockAPI
★ Uso del método fetch

Preguntas para Reflexionar:

- ¿Qué validaciones podrías agregar para mejorar la calidad de los datos ingresados?
- ¿Qué harías si la API tarda mucho en responder o devuelve un error?
- ¿Cómo manejarías la edición de un producto ya existente?

Próximos Pasos:

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


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