



Buenos
Aires
Ciudad

Agencia de Habilidades
para el Futuro

<Talento
Tech />

ReactJS

**Clase 11 | Leer, Actualizar y Eliminar
productos**

¡Les damos la bienvenida!



Vamos a comenzar a grabar la clase.

Índice

1**2****3**

Creación de Productos - Formulario y Validación

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

Leer, Actualizar y Eliminar productos

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

Implementación del CRUD Completo

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

Objetivos de la Clase

1

Leer y renderizar productos almacenados en MockAPI.

2

Implementar la funcionalidad de edición de productos mediante formularios controlados.

3

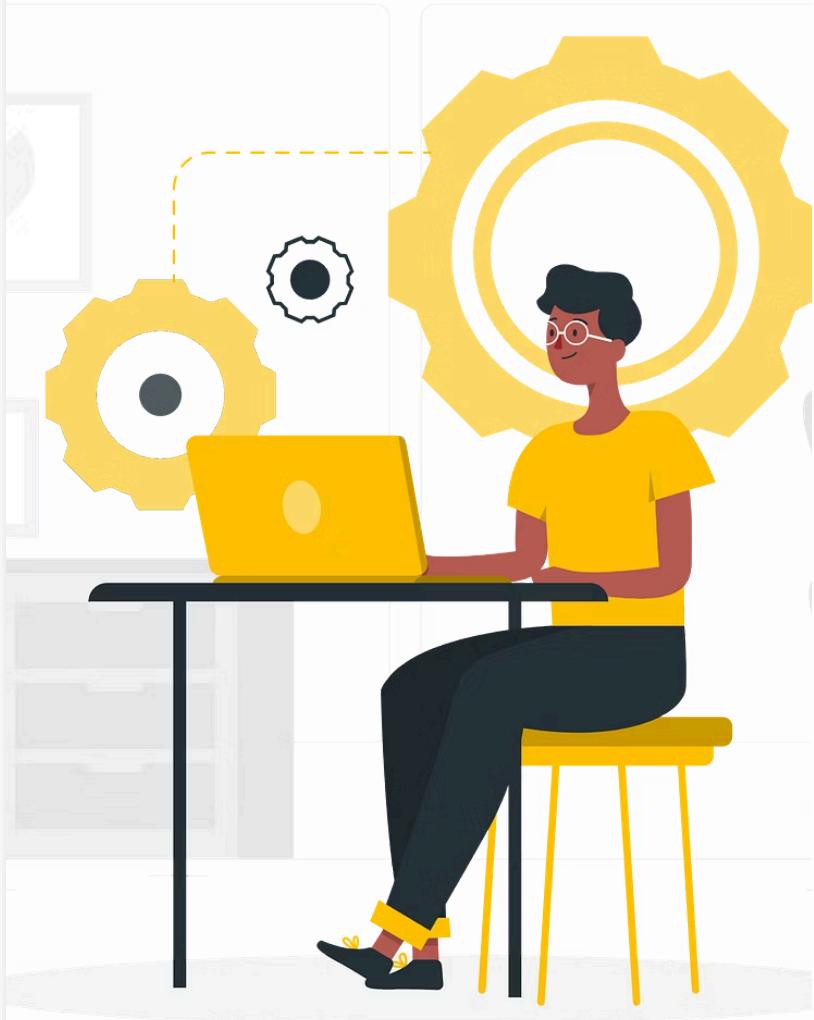
Permitir la eliminación de productos con confirmación del usuario.

4

Validar los datos en el formulario de edición para garantizar la consistencia.

Implementación de la funcionalidad para leer productos desde el estado o

API



Lectura de Productos

Configurar useEffect

Utilizar para inicializar el estado con datos de la API.

Realizar fetch

Obtener productos de MockAPI.

Actualizar estado

Almacenar productos obtenidos en el estado del componente.

Componente ListaProductos

```
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;
```

Manejo de Errores

Try-Catch

Envolver la lógica de fetch en un bloque try-catch.

Mensajes de Error

Mostrar mensajes claros al usuario en caso de fallo.

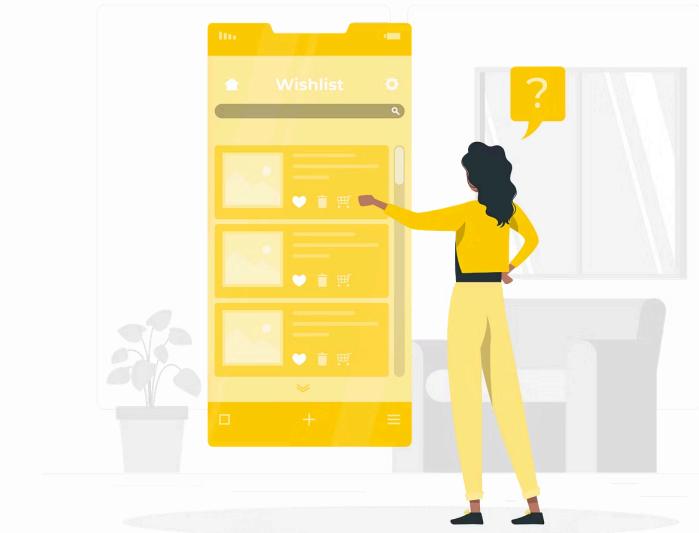
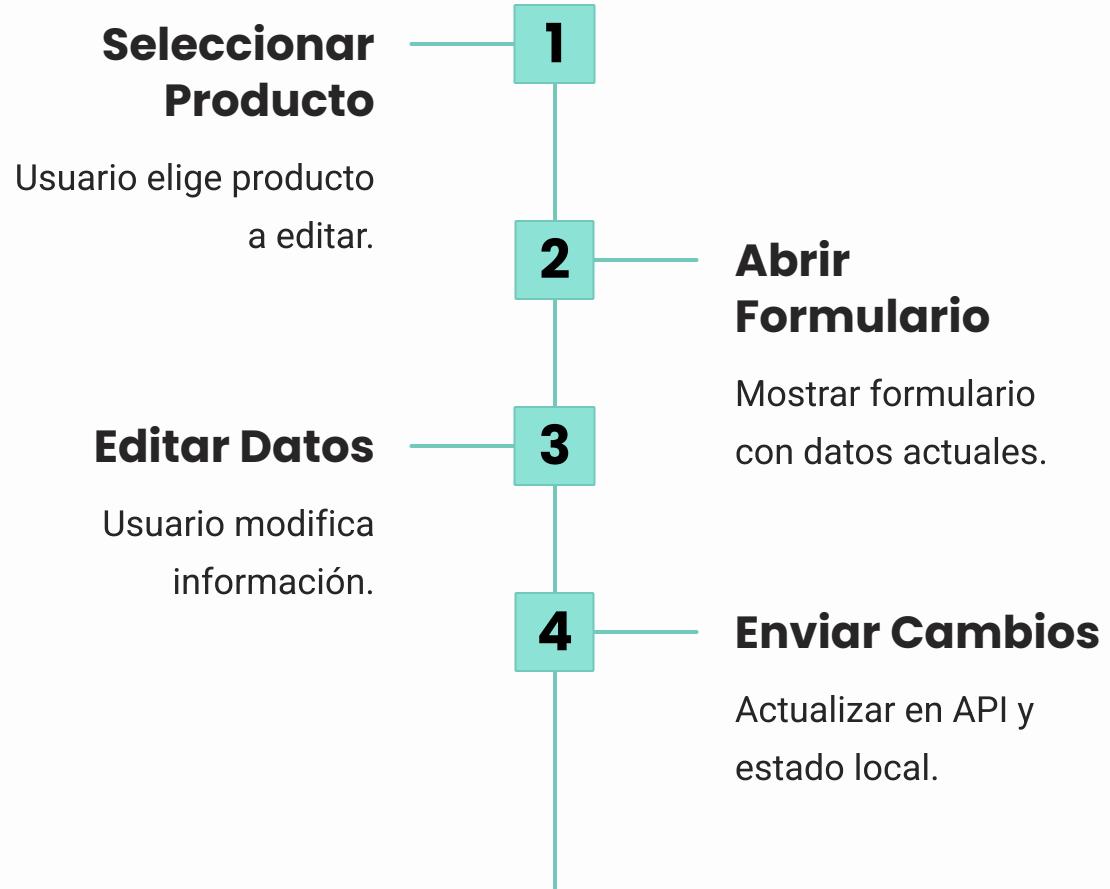
Estado de Carga

Implementar un estado de carga para mejorar la UX.



Actualización y eliminación de productos

Actualización de Productos



Formulario de Edición

Estado Controlado

Usar useState para manejar datos del formulario.

Población de Datos

Llenar formulario con datos existentes del producto.

Validación

Verificar datos antes de enviar.

Código del Formulario

```
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 || 0}
                onChange={handleChange}
                required
            />
        </div>
        <div>
            <label>Stock:</label>
            <input
                type="number"
                name="stock"
                value={producto.stock || 0}
                onChange={handleChange}
                required
            />
        </div>
        <div>
            <button type="submit">Actualizar</button>
        </div>
    </form>
)
```

```
        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>
);
}
```

Actualización en API



Preparar Datos

Formatear datos del formulario.

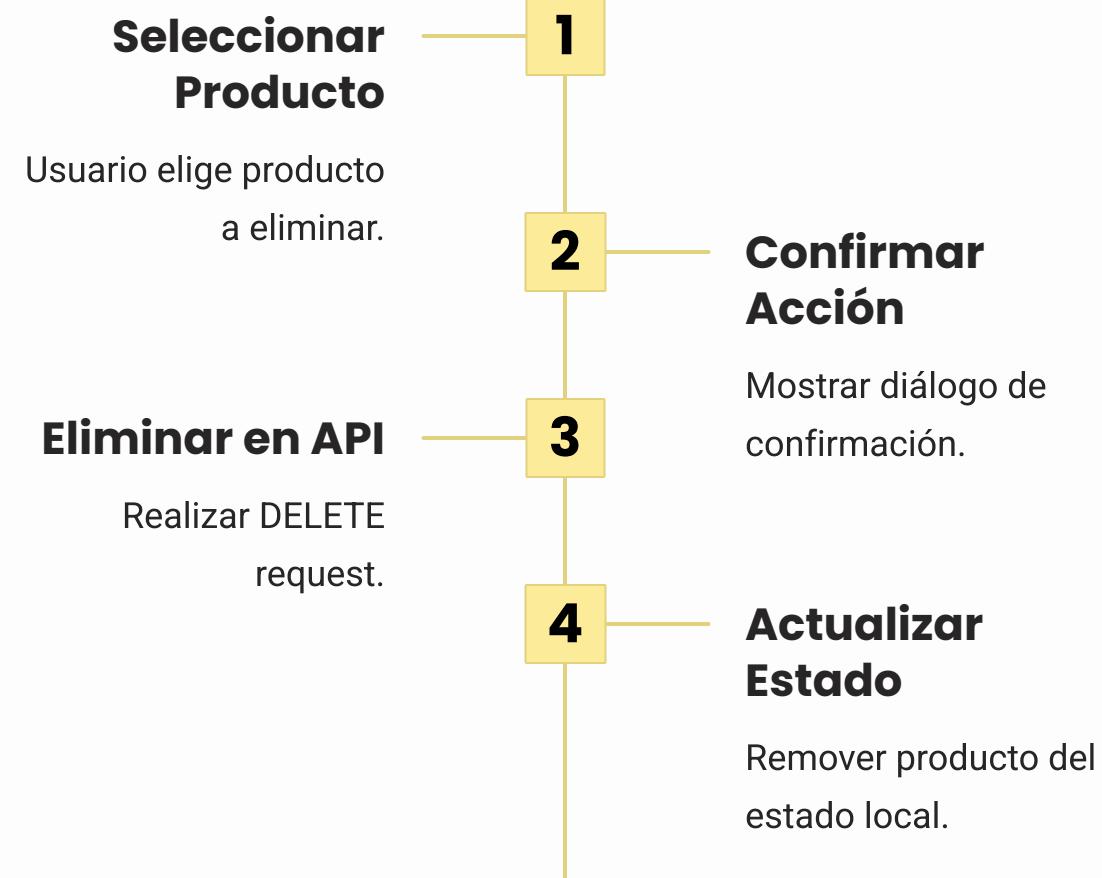
Enviar Solicitud

Realizar PUT request a MockAPI.

Manejar Respuesta

Actualizar estado local con datos recibidos.

Eliminación de Productos



Código de Eliminación

```
const eliminarProducto = async (id) => {
  const confirmar = window.confirm('¿Estás seguro de eliminar?');
  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');
      alert('Producto eliminado correctamente.');
    } catch (error) {
      console.error(error.message);
      alert('Hubo un problema al eliminar el producto.');
    }
  }
};
```

Validación de los formularios de edición

Validación de Formularios

1

Nombre Obligatorio

Asegurar que cada producto tenga un identificador claro.

2

Precio Mayor a 0

Evitar errores con precios inválidos o negativos.

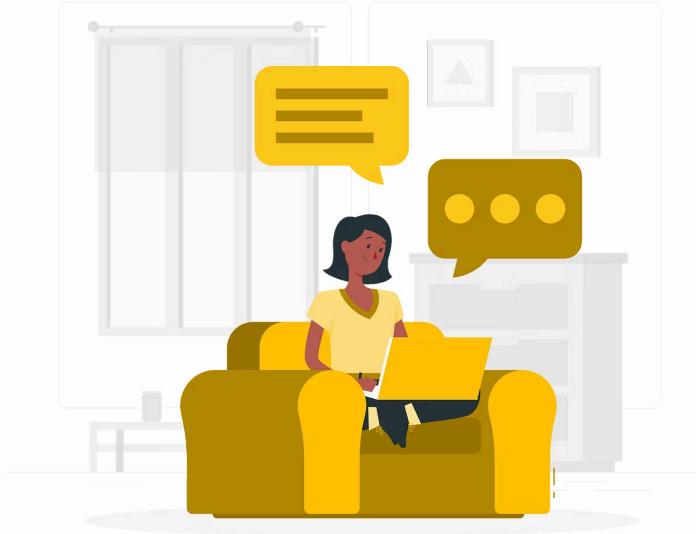
3

Descripción Mínima

Fomentar descripciones detalladas y útiles.



Mensajes de Error



Ubicación

Mostrar junto a los campos correspondientes.

Claridad

Usar lenguaje claro y específico.

Estilo

Diseño que destaque sin ser intrusivo.

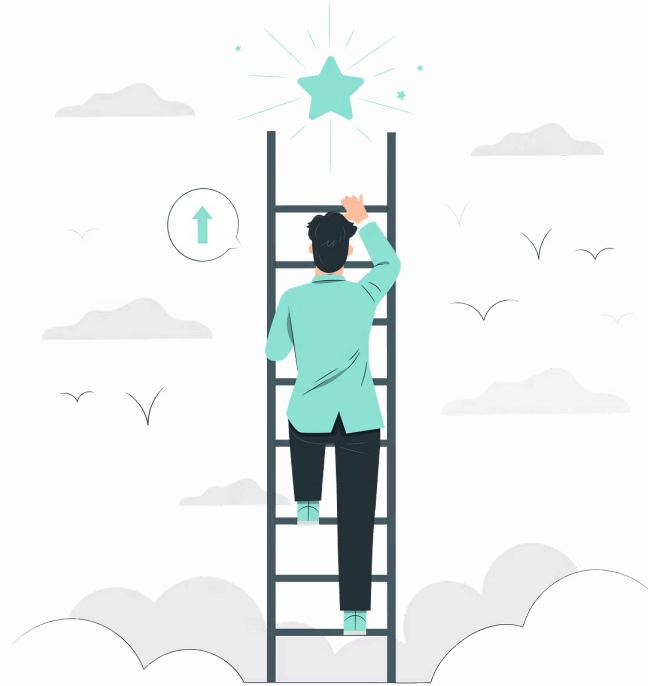
Reflexión Final

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.



Preguntas para Reflexionar



- 1** ¿Qué beneficios y desafíos has encontrado al trabajar con APIs como MockAPI para manejar datos dinámicos en tus aplicaciones?

- 2** ¿Cómo podrías mejorar la experiencia de usuario en las funcionalidades de edición y eliminación de productos para que sean más intuitivas y visualmente atractivas?

- 3** ¿Qué estrategias podrías implementar para manejar errores de manera más eficiente cuando las solicitudes a la API fallan?

Próximos Pasos

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

Materiales Adicionales



Documentación MockAPI

Recurso oficial para trabajar con MockAPI.



Método Fetch

Guía sobre el uso de fetch en JavaScript.



Formularios en React

Documentación sobre manejo de formularios.



Ejercicios Prácticos



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.



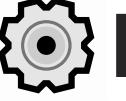
Ejercicio Práctico

Obligatorio

Descripción de tu tarea inicial:



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.



Ejercicio Práctico

Obligatorio

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.



Ejercicio Práctico

Obligatorio

Requisitos del Proyecto:

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.

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).



Ejercicio Práctico

Obligatorio

Requisitos del Proyecto:

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.

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.