



Agencia de Habilidades
para el Futuro

<Talento
Tech />

ReactJS

**Clase 12 | Implementación del CRUD
Completo**

¡Les damos la bienvenida!



Vamos a comenzar a grabar la clase.

Índice

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.**

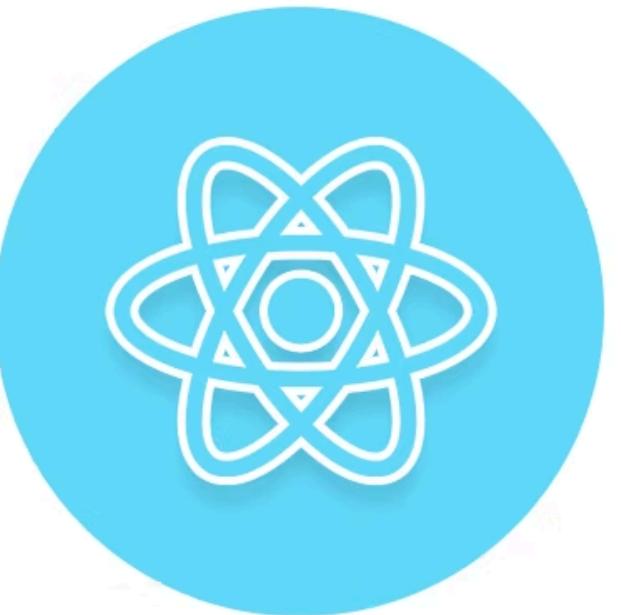
Estilización con Bootstrap o styled-components

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

Objetivos de la Clase

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

Coneectar el formulario de agregar/editar productos con el estado global.



Context API

Context API para Estado Global

Crear Contexto

Configurar ProductsProvider para almacenar el estado global de productos.

Integrar en la App

Envolver la aplicación con ProductsProvider para acceso al estado global.

Conectar Componentes

Usar el contexto en FormularioEdicion y ListaProductos para gestionar productos.

Creación del Contexto Global

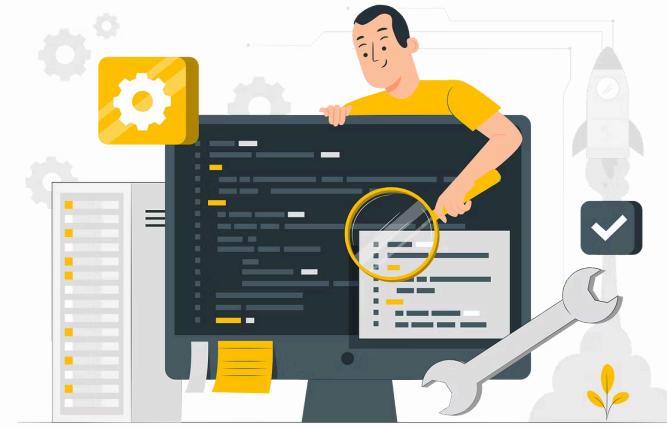
```
import React, { createContext, useState } from 'react';
export const ProductsContext = createContext();
export const ProductsProvider = ({ children }) => {
  const [productos, setProductos] = useState([]);
  const agregarProducto = (nuevoProducto) => {
    setProductos([...productos, nuevoProducto]);
  };
  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>  
);  
};
```

Integración del Contexto en la App

```
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>
);
```



Formulario Conectado al Estado Global



Importaciones

Importar useContext y ProductsContext.

Estado Local

Usar useState para manejar el estado del formulario.

Contexto Global

Acceder a funciones globales con useContext.

Estructura del Formulario

```
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' ? 'Aregar' : 'Actualizar'}</button>
  </form>
);
}

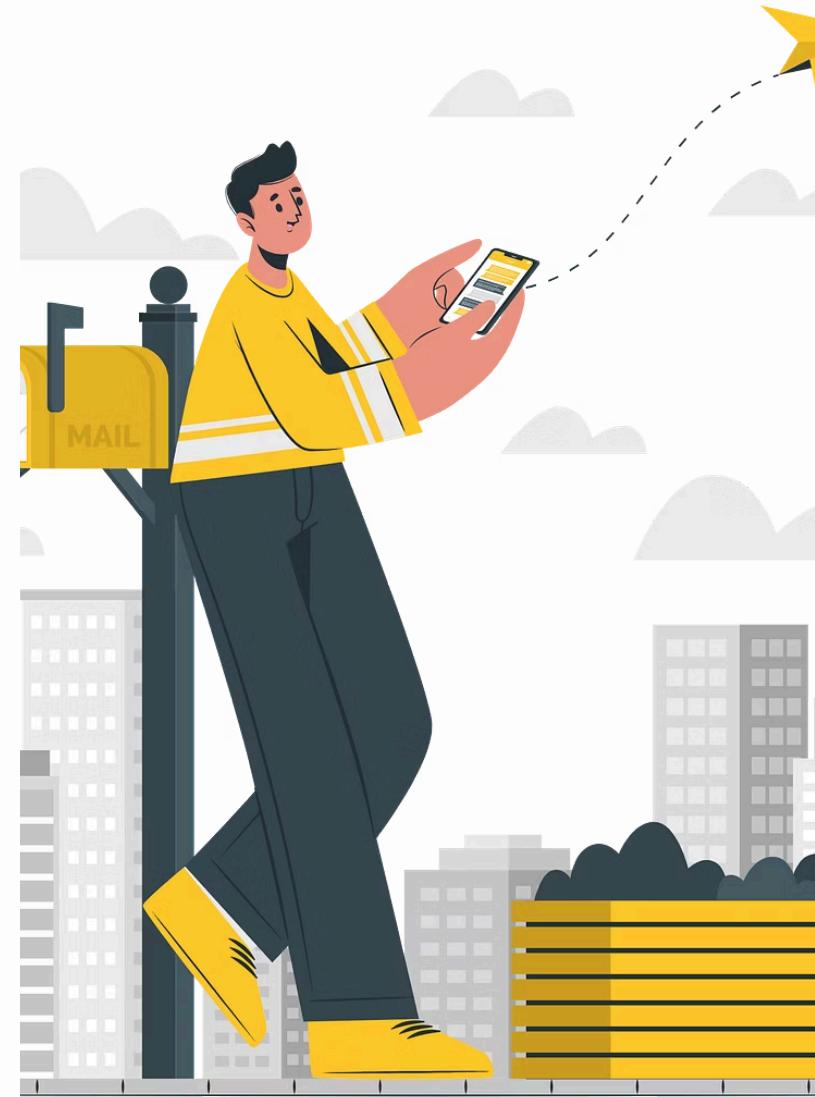
export default FormularioProducto;
```

Manejo de Cambios en el Formulario

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


Envío del Formulario

```
const handleSubmit = (e) => {
  e.preventDefault();
  if (modo === 'agregar') {
    agregarProducto({ ...producto, id: Date.now() });
  } else {
    editarProducto(producto);
  }
  onCerrar();
};
```



Validación de formularios y manejo de errores



Validación de Formularios

1 Campos Requeridos

Todos los campos deben completarse.

2 Precio

Debe ser mayor a 0.

3 Descripción

Al menos 10 caracteres.

Implementación de Validaciones

```
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;
};
```

Manejo de Errores en el Envío



```
onst 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();
};
```

Mostrar Errores en el Formulario

```
<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>
```



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



[React Hook Form](#)

Sitio oficial para manejo avanzado de formularios en React.



[Context API](#)

Tutorial en YouTube sobre los fundamentos de React Context API.



Preguntas para Reflexionar

1

Context API vs Props

¿Qué ventajas ofrece el Context API en aplicaciones React más grandes?

2

Validación Avanzada

¿Cómo mejorarías la validación para manejar errores más complejos?

3

Manejo de Errores

¿Qué técnicas adicionales implementarías para manejar errores y notificaciones?

Próximos Pasos



- 1
- 2
- 3
- 4

Introducción a Bootstrap

O styled-components para estilizar componentes.

Diseño Responsive

Creación de un diseño básico y adaptable.

Estilos en Componentes

Aplicación de estilos en botones, formularios y productos.

Barra de búsqueda

Implementación de una barra de búsqueda en el eCommerce.

Ruta de avance



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.

1

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.

2

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.

3

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.



Ejercicios Prácticos

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.





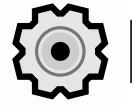
Ejercicio Práctico

Obligatorio

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.



Ejercicio Práctico

Obligatorio

Requisitos del Proyecto:

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.

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.



Ejercicio Práctico

Obligatorio

Requisitos del Proyecto:



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.



¡Nuevo cuestionario en Campus!

-  No olvides que los cuestionarios son de carácter obligatorio para poder avanzar con la cursada.