

# Curso de Desarrollo Frontend con React



# CRUD en el Frontend con React

# ¿Qué aprenderemos hoy?



## 1 Fundamentos CRUD

Conceptos básicos y su importancia en el desarrollo frontend



## 2 Estado en React

Manejo de datos con useState para gestionar usuarios



## 3 Operaciones completas

Create, Read, Update y Delete paso a paso



## 4 Navegación

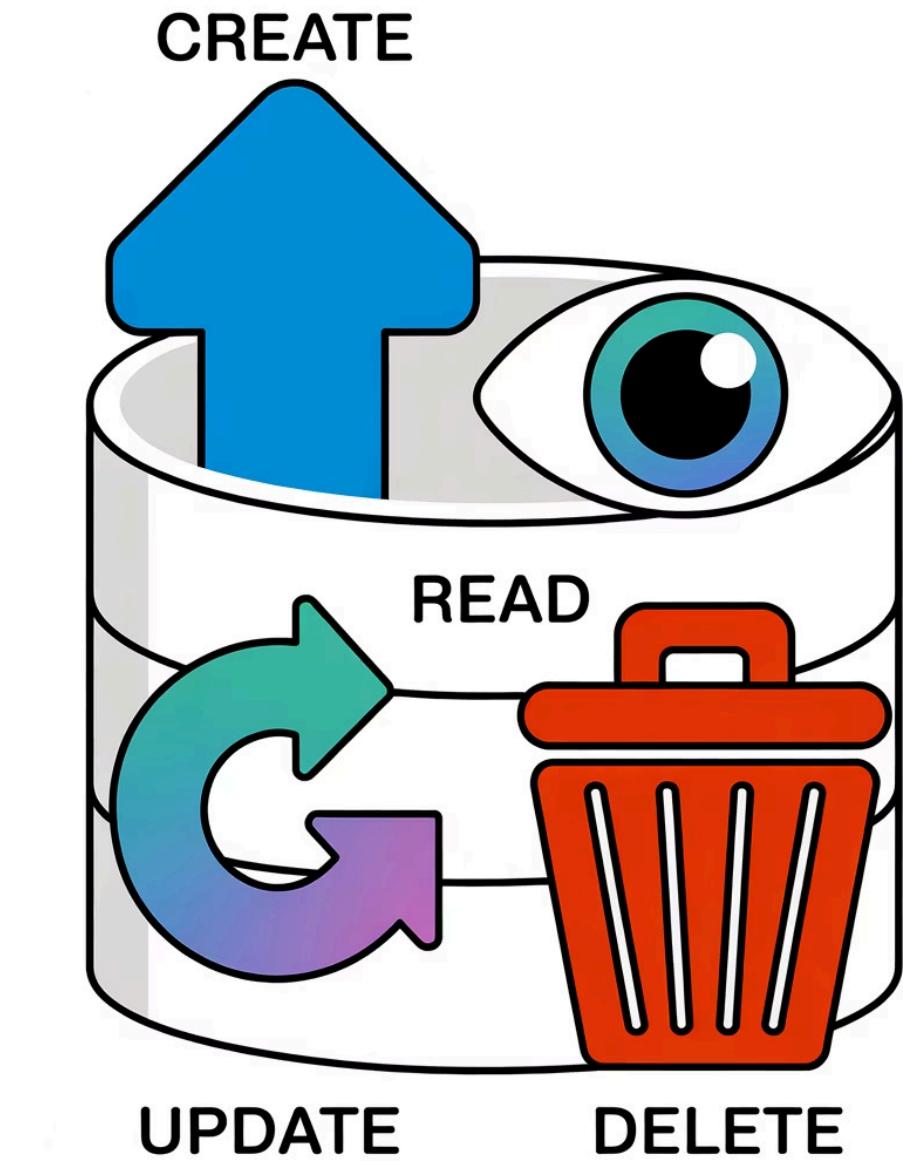
Rutas y vistas organizadas con React Router



## 5 Buenas prácticas

Componentes, validaciones y código limpio

# ¿Qué es CRUD?



# Las cuatro operaciones fundamentales

## Create

**Crear** nuevos registros de usuarios en nuestra aplicación

## Read

**Leer** y mostrar la lista completa de usuarios existentes

## Update

**Actualizar** información de usuarios ya registrados

## Delete

**Eliminar** usuarios de nuestra base de datos

# ¿Por qué es importante el CRUD?

El CRUD es la base de casi cualquier aplicación web moderna.

Dominar estas operaciones te permitirá:

- Entender la lógica fundamental de las aplicaciones
- Crear interfaces de usuario interactivas
- Prepararte para integrar con APIs reales
- Desarrollar aplicaciones completas y funcionales



# Tipos de CRUD en Frontend

## CRUD con datos estáticos

Los datos se almacenan en el estado del componente usando `useState`. Perfecto para aprender y prototipar.

## CRUD conectado a backend

Los datos se envían y reciben desde un servidor real mediante APIs REST. Para aplicaciones en producción.

Comenzaremos con datos estáticos para dominar los conceptos fundamentales.

```
= scnte>,
= usestate();
= Jeatascoatreat();
=
/
<JavaScreibl((/ /))>-
<JavaScript>ace st e estusetteal stl>.
```

# usestate

## Manejo de Estado en React

# useState: El corazón de nuestro CRUD

El hook `useState` nos permite almacenar y actualizar la lista de usuarios en nuestra aplicación.

```
const [usuarios, setUsuarios] = useState([
  { id: 1, nombre: 'Ana García', correo: 'ana@email.com' },
  { id: 2, nombre: 'Carlos López', correo: 'carlos@email.com' },
  { id: 3, nombre: 'María Silva', correo: 'maria@email.com' }
]);
```

# Estructura de nuestros datos



## ID único

Identificador numérico que permite distinguir cada usuario de manera única en nuestra aplicación.



## Nombre

Campo de texto que almacena el nombre completo del usuario para mostrar en la interfaz.

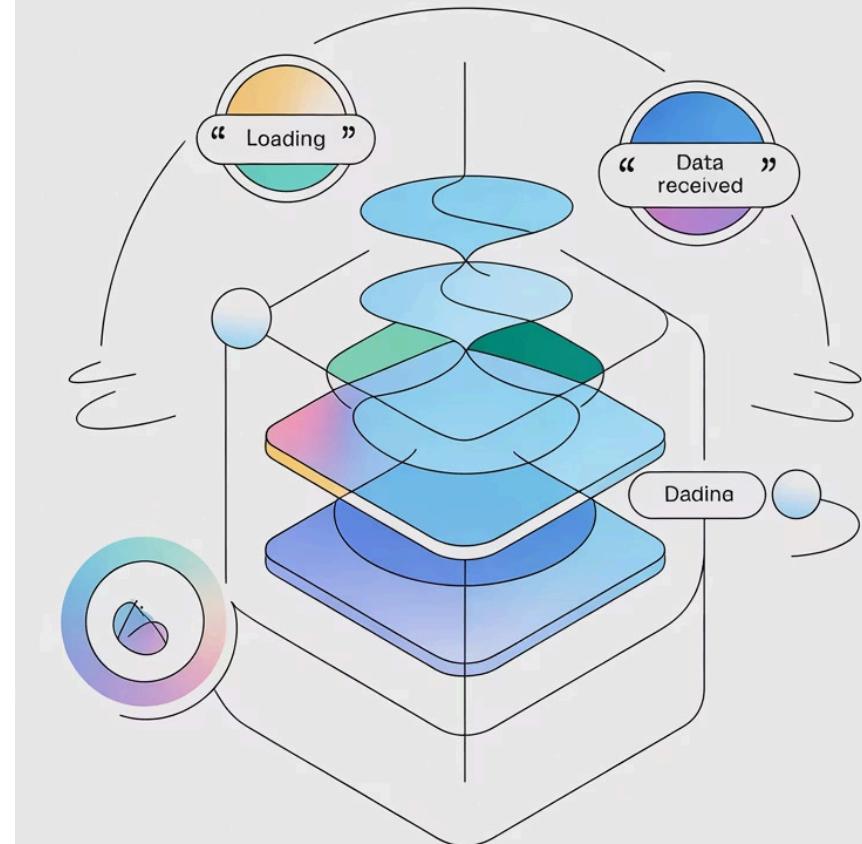


## Correo electrónico

Campo de email que sirve como contacto y puede ser usado para validaciones futuras.

# Estado inicial de nuestra aplicación

Iniciamos con algunos usuarios de ejemplo para poder ver inmediatamente el funcionamiento de nuestras operaciones CRUD.



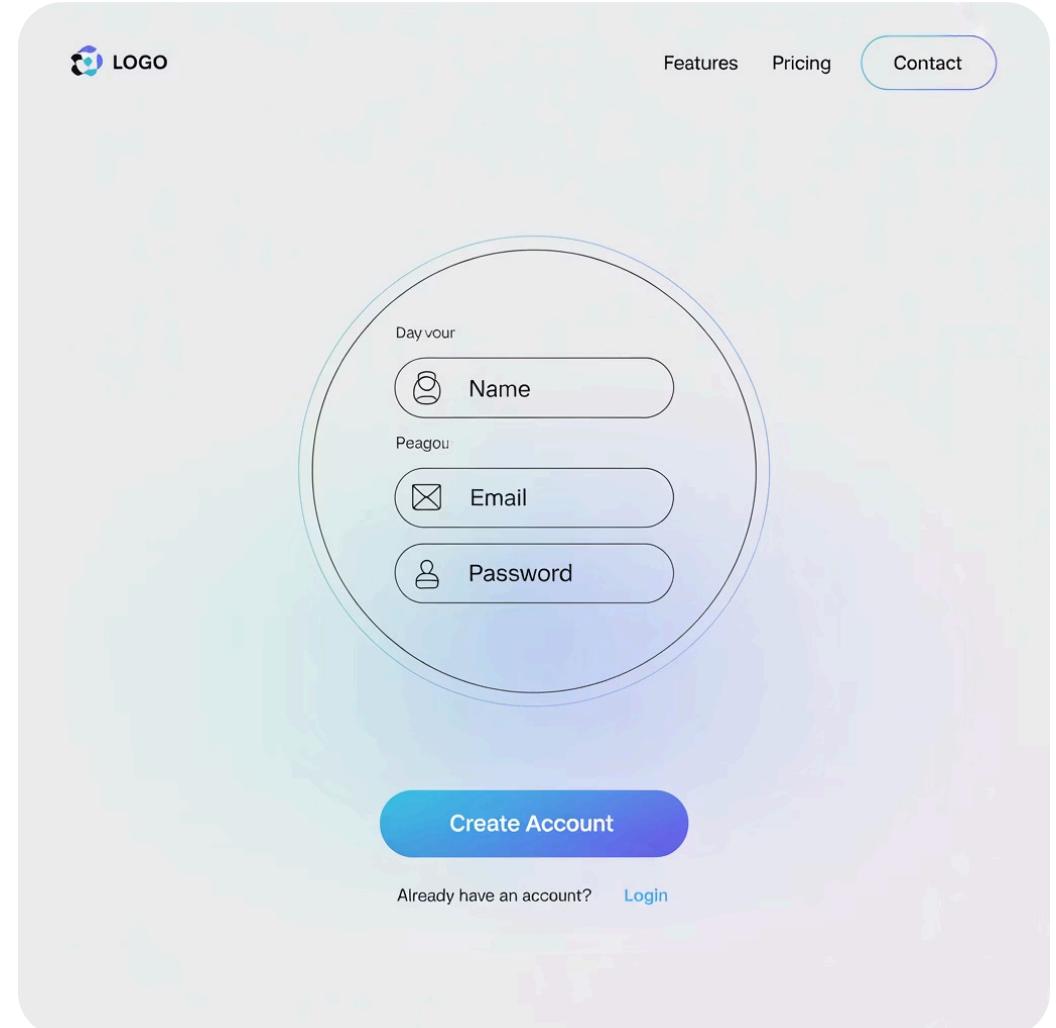
# **Operación CREATE**

# Creando nuevos usuarios

## Formulario controlado

Usamos inputs controlados con `useState` para capturar los datos del nuevo usuario:

- Campo nombre (texto)
- Campo correo (email)
- Botón para enviar



# Código para crear usuarios

```
const [nombre, setNombre] = useState("");
const [correo, setCorreo] = useState("");

const crearUsuario = () => {
  const nuevoUsuario = {
    id: Date.now(),
    nombre: nombre,
    correo: correo
  };

  setUsuarios([...usuarios, nuevoUsuario]);
  setNombre("");
  setCorreo("");
};
```

# Puntos clave del CREATE

## → ID automático

Usamos Date.now() para generar un ID único

## → Spread operator

Mantenemos usuarios existentes con [...usuarios, nuevo]

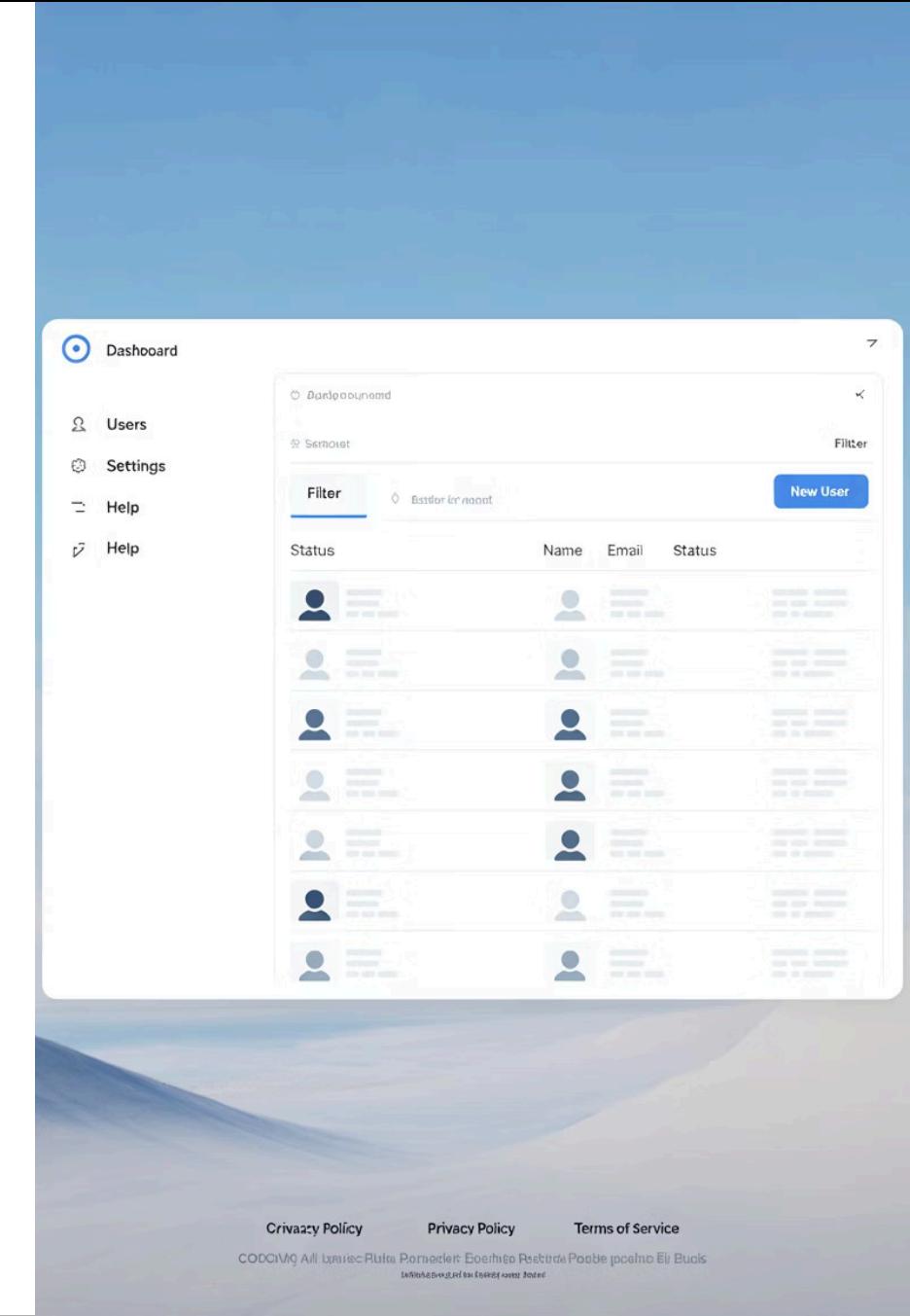
## → Limpieza de formulario

Reiniciamos los campos después de crear el usuario

# Operación READ

# Mostrando la lista de usuarios

La operación READ es la más simple: renderizamos la lista completa de usuarios almacenados en nuestro estado.



# Renderizando usuarios con map()

A continuación, se muestra un ejemplo limpio y funcional de cómo renderizar una lista de usuarios utilizando el método `map()` en React.

```
import React from 'react';

const ListaUsuarios = () => {
  // Datos de ejemplo para la demostración
  const usuarios = [
    { id: 1, nombre: "Alice", correo: "alice@example.com" },
    { id: 2, nombre: "Bob", correo: "bob@example.com" },
    { id: 3, nombre: "Charlie", correo: "charlie@example.com" },
  ];

  return (
    <div>{/* Contenedor para la lista de usuarios */}
      <h2>Nuestros Usuarios</h2>
      {usuarios.map(usuario => (
        <div key={usuario.id}>
          <h3>{usuario.nombre}</h3>
          <p>Correo: {usuario.correo}</p>
          <p>ID: {usuario.id}</p>
        </div>
      ))}
    </div>
  );
};

export default ListaUsuarios;
```

En este ejemplo, definimos un array de `usuarios` y luego usamos `map()` para iterar sobre cada usuario, creando un elemento `<div>` por cada uno. Es crucial usar la prop `key` (en este caso, `usuario.id`) cuando se renderizan listas para ayudar a React a identificar qué elementos han cambiado, se han añadido o se han eliminado, optimizando el rendimiento.

# Elementos importantes del READ

## Key única

Siempre usar el `id` como key en el `map()` para optimizar el renderizado de React.

## Información completa

Mostrar todos los datos relevantes: nombre, correo e ID del usuario.

## Diseño responsive

Crear cards o tabla que se adapte bien a diferentes tamaños de pantalla.

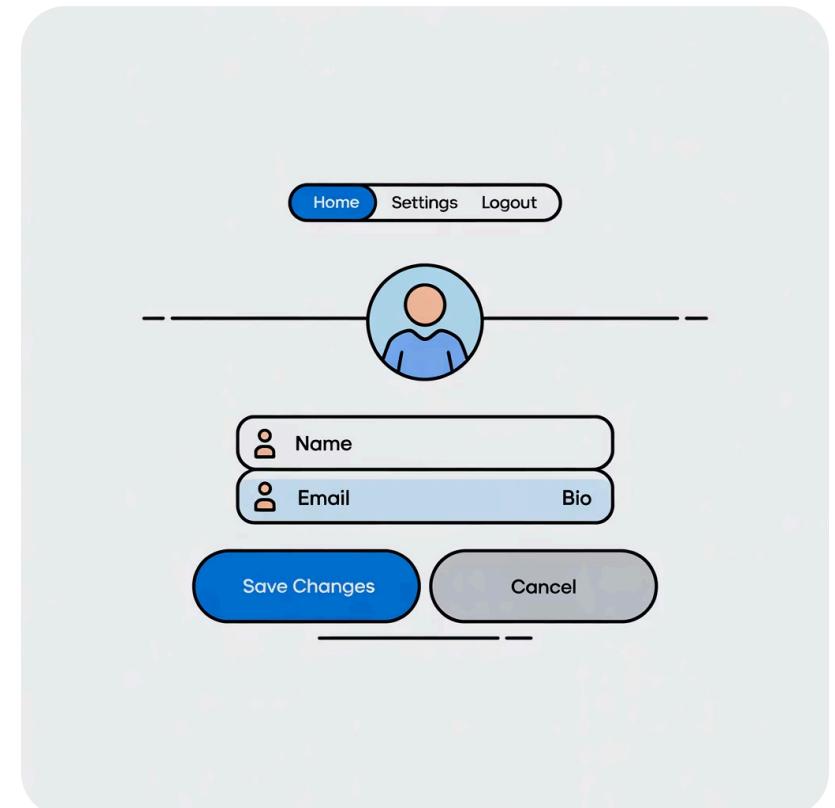
# Operación UPDATE

# Actualizando información existente

El UPDATE requiere dos pasos importantes:

1. Cargar los datos actuales del usuario en el formulario
2. Actualizar el usuario en la lista manteniendo el mismo ID

Es más complejo que CREATE porque debemos encontrar y reemplazar un elemento específico.



# Código para actualizar usuarios

```
const [usuarioEditando, setUsuarioEditando] = useState(null);

const editarUsuario = (id) => {
  const usuario = usuarios.find(u => u.id === id);
  setUsuarioEditando(usuario);
  setNombre(usuario.nombre);
  setCorreo(usuario.correo);
};

const actualizarUsuario = () => {
  setUsuarios(usuarios.map(u =>
    u.id === usuarioEditando.id
    ? { ...u, nombre, correo }
    : u
  ));
};
```

# Flujo del UPDATE

## Seleccionar usuario

El usuario hace clic en "Editar" junto a un usuario específico

## Modificar información

El usuario cambia los datos necesarios en el formulario

## Cargar datos

Pre-llenar el formulario con la información actual del usuario

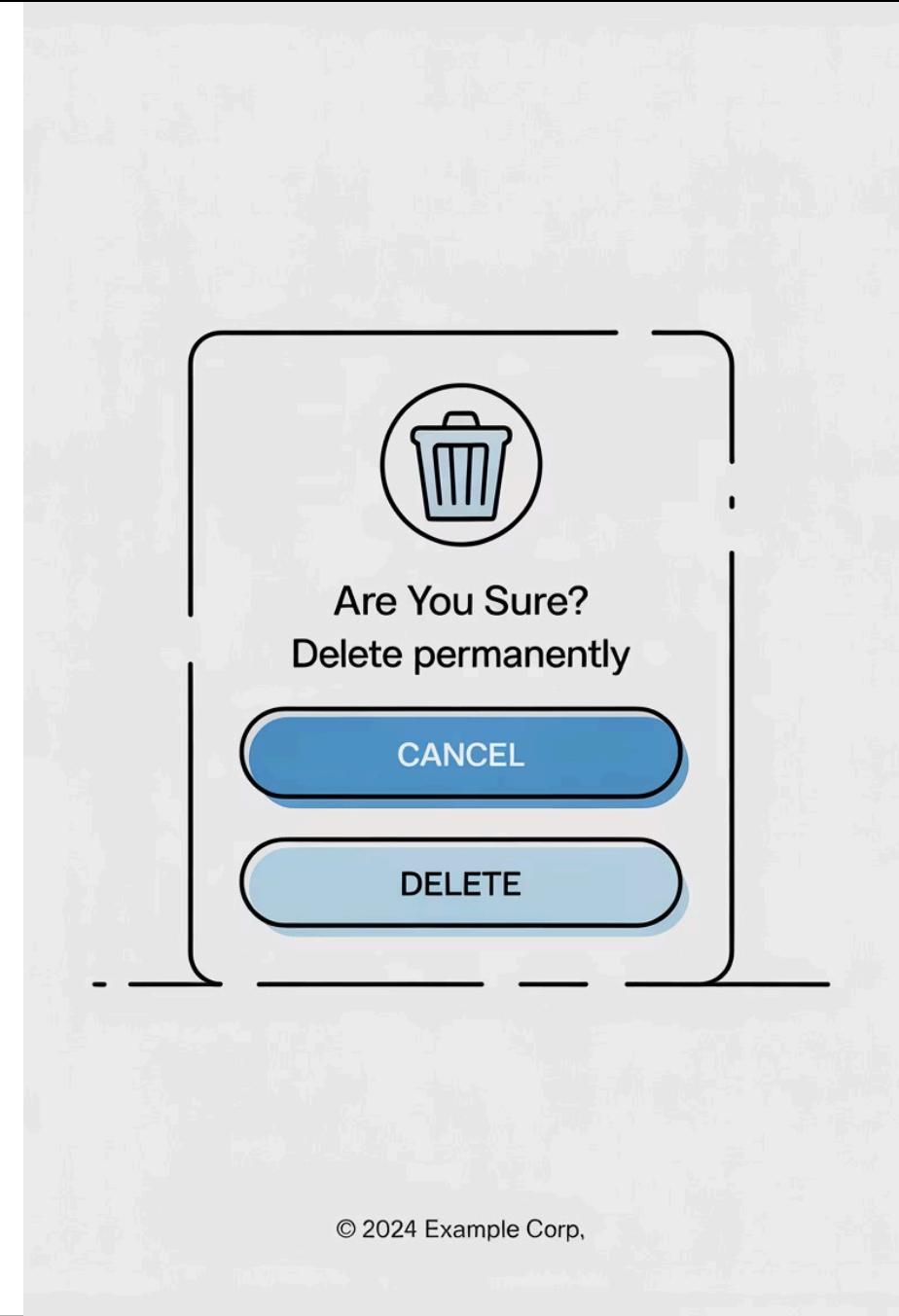
## Guardar cambios

Actualizar el estado con la nueva información del usuario

# Operación DELETE

# Eliminando usuarios

DELETE es la operación más directa: removemos un usuario específico de la lista usando su ID único.



# Implementando DELETE

```
const eliminarUsuario = (id) => {
  const confirmacion = window.confirm(
    '¿Estás seguro de eliminar este usuario?'
  );

  if (confirmacion) {
    setUsuarios(usuarios.filter(usuario => usuario.id !== id));
  }
};
```

Utilizamos `filter()` para crear una nueva lista sin el usuario eliminado.

# Mejores prácticas para DELETE



## Confirmación

Siempre pedir confirmación antes de eliminar para evitar accidentes



## Filter() correcto

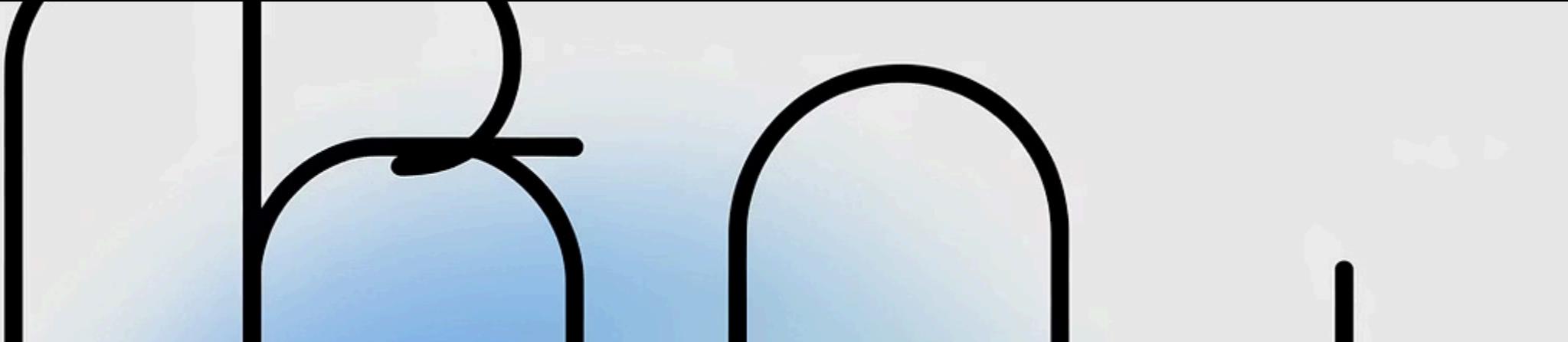
Usar `filter()` para crear un nuevo array sin el elemento



## Retroalimentación

Mostrar mensaje de confirmación cuando la eliminación sea exitosa

# Navegación y Rutas



# Organizando nuestra aplicación

Una aplicación CRUD bien estructurada necesita diferentes vistas para cada operación. React Router nos ayuda a organizar esta navegación.

# Estructura de rutas



## **/ (Raíz)**

Listado completo de usuarios - Vista principal READ



## **/nuevo**

Formulario para crear un nuevo usuario - Vista CREATE



## **/editar/:id**

Formulario de edición con datos precargados - Vista UPDATE

# Configuración de React Router

```
import { BrowserRouter, Routes, Route } from 'react-router';

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/nuevo" element={<NewUser />} />
        <Route path="/editar/:id" element={<EditUser />} />
      </Routes>
    </BrowserRouter>
  );
}


```

# CRUD Completo

# Integrando todas las operaciones

## CREATE

Usuario completa formulario y se agrega a la lista

## READ

Nueva entrada aparece inmediatamente en el listado

## DELETE

Usuario puede eliminar registros cuando sea necesario

## UPDATE

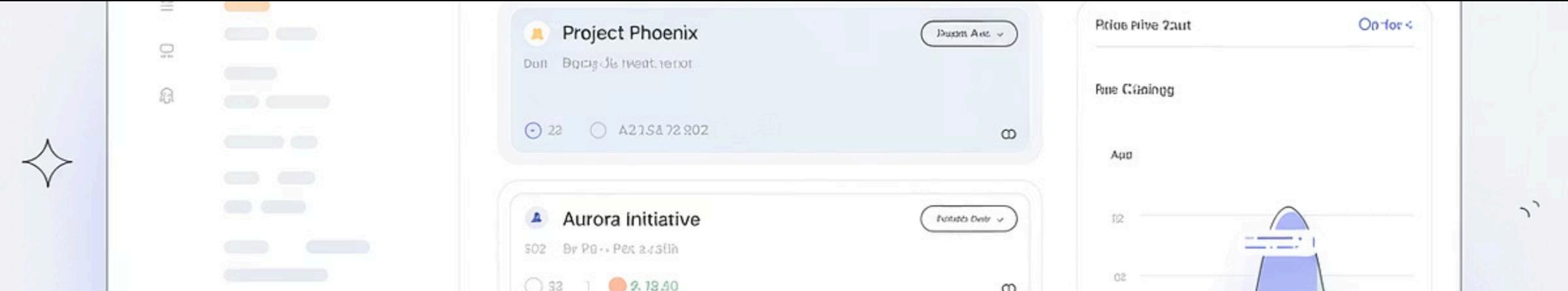
Usuario puede editar información desde la lista



# Flujo de usuario completo

Imaginemos a María, nuestra usuaria, interactuando con nuestra aplicación:



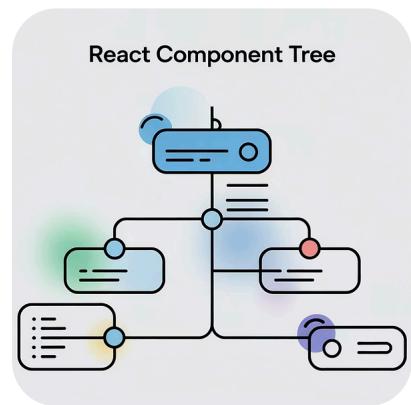


# Coherencia de la aplicación

La navegación fluida entre vistas es crucial para mantener una experiencia de usuario coherente y profesional.

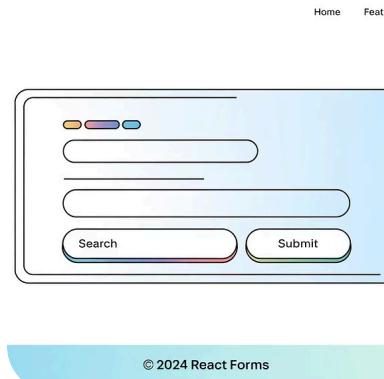
# Buenas Prácticas

# Arquitectura de componentes



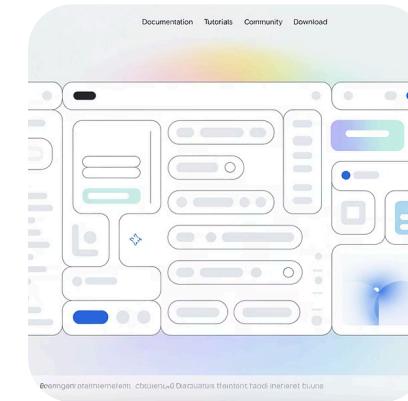
## UserList

Componente dedicado a mostrar la lista completa de usuarios con opciones de edición y eliminación.



## UserForm

Formulario reutilizable para crear y editar usuarios con validaciones incorporadas.



## UserEdit

Componente especializado que combina UserForm con lógica de edición específica.

# Principios fundamentales



## IDs únicos siempre

Cada usuario debe tener un identificador único para operaciones seguras



## Inputs controlados

Todos los formularios deben usar `value` y `onChange`



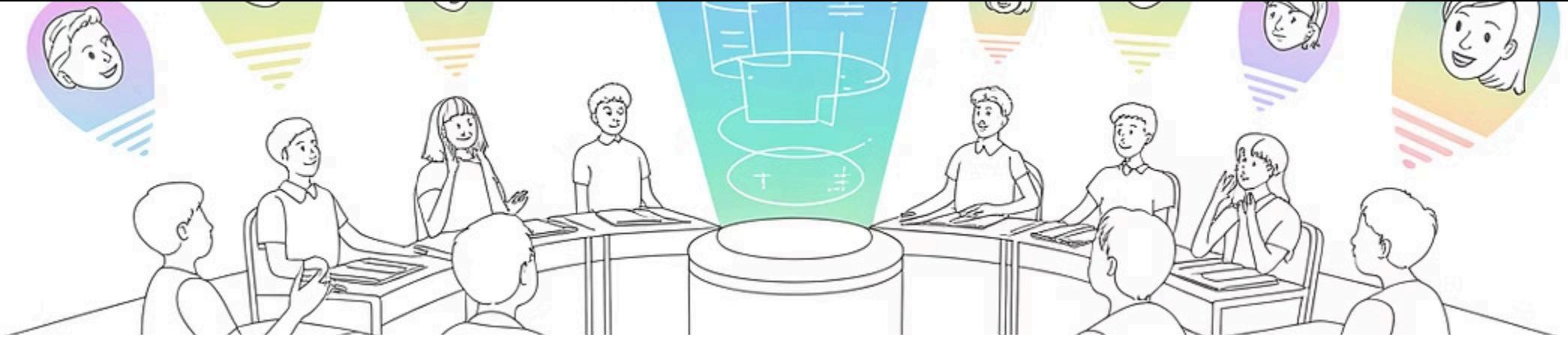
## Validaciones básicas

Verificar que los campos requeridos estén completos antes de enviar



## Feedback al usuario

Mostrar mensajes de confirmación para todas las operaciones



# Pregunta de Reflexión

# ¿Por qué empezar con datos estáticos?

¿Qué ventajas tiene practicar CRUD en el frontend con datos estáticos antes de conectarlo con un backend real?

## Enfoque en lógica

Concentrarse en React sin complicaciones de APIs o red

## Aprendizaje gradual

Dominar useState y flujo de datos antes de añadir complejidad

## Prototipado rápido

Crear y probar interfaces sin dependencias externas

## Base sólida

Fundamentos que facilitan la transición a APIs reales

# #EDCOUNIANDES

<https://educacioncontinua.uniandes.edu.co/>

Contacto: [educacion.continua@uniandes.edu.co](mailto:educacion.continua@uniandes.edu.co)

© - Derechos Reservados: La presente obra, y en general todos sus contenidos, se encuentran protegidos por las normas internacionales y nacionales vigentes sobre propiedad Intelectual, por lo tanto su utilización parcial o total, reproducción, comunicación pública, transformación, distribución, alquiler, préstamo público e importación, total o parcial, en todo o en parte, en formato impreso o digital y en cualquier formato conocido o por conocer, se encuentran prohibidos, y solo serán lícitos en la medida en que se cuente con la autorización previa y expresa por escrito de la Universidad de los Andes.