



# Estados Globales en React

Centro de Servicios y Gestión Empresarial  
SENA Regional Antioquia



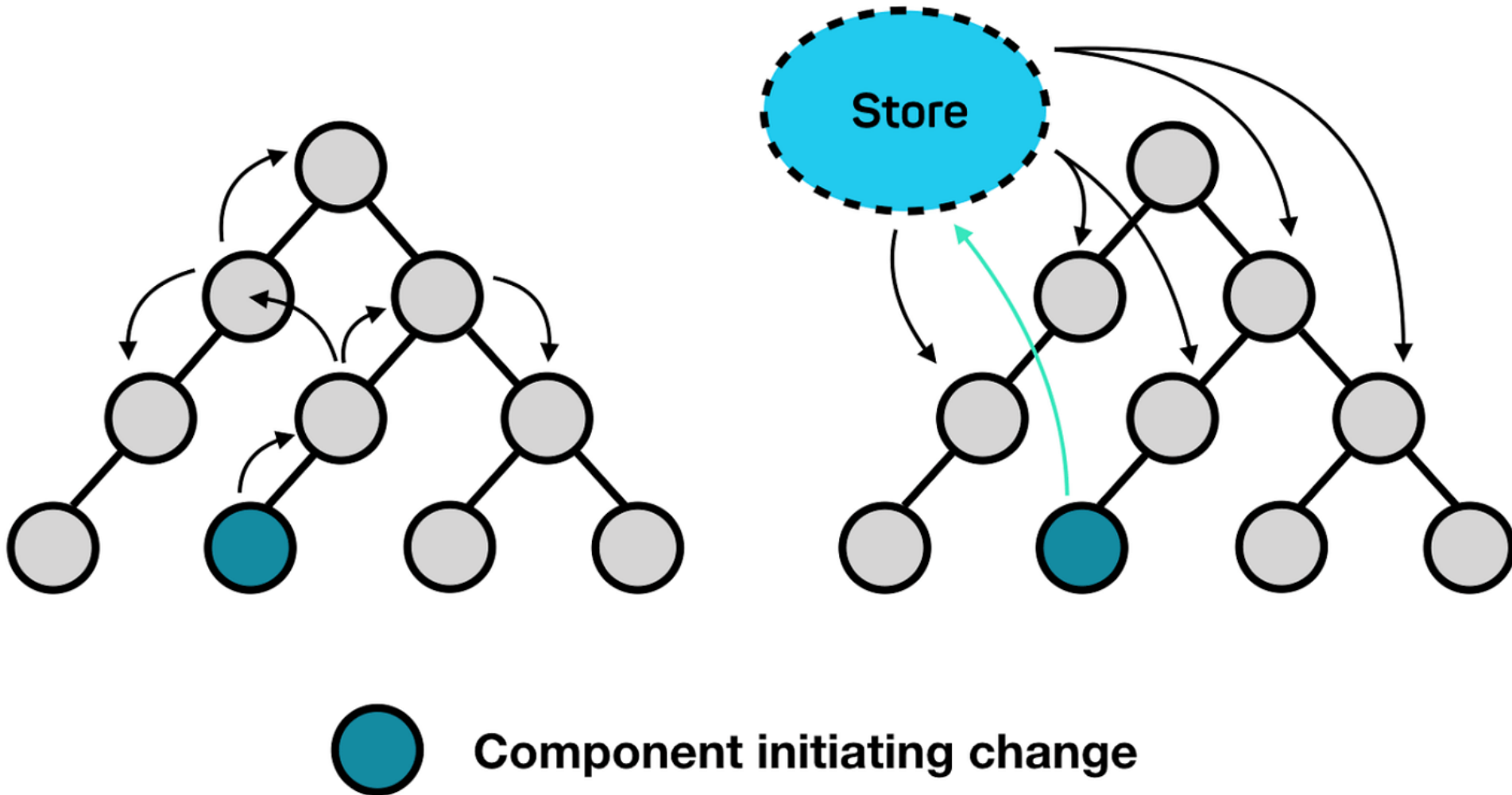
[www.sena.edu.co](http://www.sena.edu.co)

# Concepto – Estados globales

En React, los "***estados globales***" se refieren a estados que se comparten entre varios componentes, permitiendo gestionar datos comunes en diferentes niveles de la jerarquía de la aplicación.

Esto es útil cuando múltiples componentes necesitan acceder o actualizar los mismos datos. React cuenta con alternativas nativas y bibliotecas de administración para gestionar los estados globales.

# Concepto – Estados globales





# React Context API



# 1. React Context API

La **Context API** de React permite manejar el estado global compartiendo valores y funciones entre múltiples componentes sin necesidad de pasar props en cada nivel de la jerarquía.

Cuando se crea un contexto con **createContext**, se define un "árbol de contexto" con un **Provider** que contiene los componentes hijos. Estos componentes pueden acceder a los datos y funciones del contexto mediante **Provider** y **Consumer** o con el hook `useContext`.

El contexto solo es accesible dentro del árbol donde fue "**proveído**". Para que un componente fuera de este árbol acceda al contexto, el **Provider** debe envolver un nivel superior que abarque todos los árboles que requieran el estado, o bien, se debe crear un contexto nuevo según las necesidades. Esto implica que los contextos en React no se comparten entre árboles separados sin una estructura común de Provider.

# React Context API – Ventajas

A vertical line with four circles is positioned on the left side of the slide. Each circle is connected to a horizontal bar containing text. The circles are white with a thin blue outline, and the bars are blue, teal, green, and olive green from top to bottom.

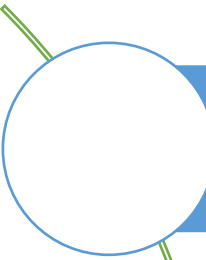
**Evita del "Prop Drilling":** Permite pasar datos de nivel superior a cualquier nivel inferior sin pasar props en cada nivel intermedio, simplificando el código.

**Simplicidad y Accesibilidad:** Fácil de configurar y no requiere instalación de dependencias externas, lo cual es ideal para aplicaciones que no tienen grandes necesidades de manejo de estado.

**Desempeño Adecuado en Escalas Reducidas:** Funciona bien en aplicaciones donde solo unos pocos componentes necesitan el estado compartido, ya que es rápido de implementar y fácil de entender.

**Encapsulación de Lógica Común:** Permite centralizar funciones o datos que son usados por múltiples componentes, como configuraciones, temas, autenticación o preferencias de usuario.

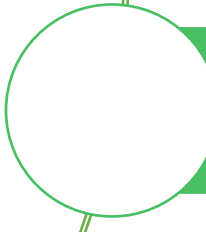
# React Context API – Desventajas




**Rendimiento:** No está optimizado para aplicaciones grandes o con muchos componentes que cambian frecuentemente. Cuando se actualiza un valor en el contexto, todos los componentes que lo consumen se vuelven a renderizar, lo que puede generar problemas de rendimiento en aplicaciones complejas.



**Dificultad de Escalabilidad:** El uso de múltiples contextos para diferentes secciones de la aplicación puede volverse difícil de gestionar y llevar a una estructura de componentes compleja y poco manejable.



**Dependencia en la Jerarquía:** Como los valores de un contexto solo están disponibles en el árbol en el que se usa el Provider, puede ser difícil de mantener si varios árboles o partes de la aplicación necesitan esos datos. Esto requiere estructurar bien la jerarquía o duplicar datos, lo cual puede reducir la eficiencia.



**Ausencia de Herramientas Avanzadas:** A diferencia de bibliotecas como Redux, la Context API carece de herramientas avanzadas para manejar la depuración y el seguimiento de estado, lo que puede ser una limitación en aplicaciones que requieren un manejo detallado del flujo de datos.

# React Context API - Ejemplo

```
import React, { createContext, useContext, useState } from 'react';
import './App.css';

// 1. Crear un contexto
const UserContext = createContext();

// Componente padre
export function App() { ...
}

// Componentes hijos
function Profile() { ...
}

// Componentes hijos
function Settings() { ...
}
```

En este ejemplo, el contexto **UserContext** permite que tanto **Profile** como **Settings** accedan y modifiquen el **estado user**, sin necesidad de pasar **user** y **setUser** manualmente por cada nivel de componentes.





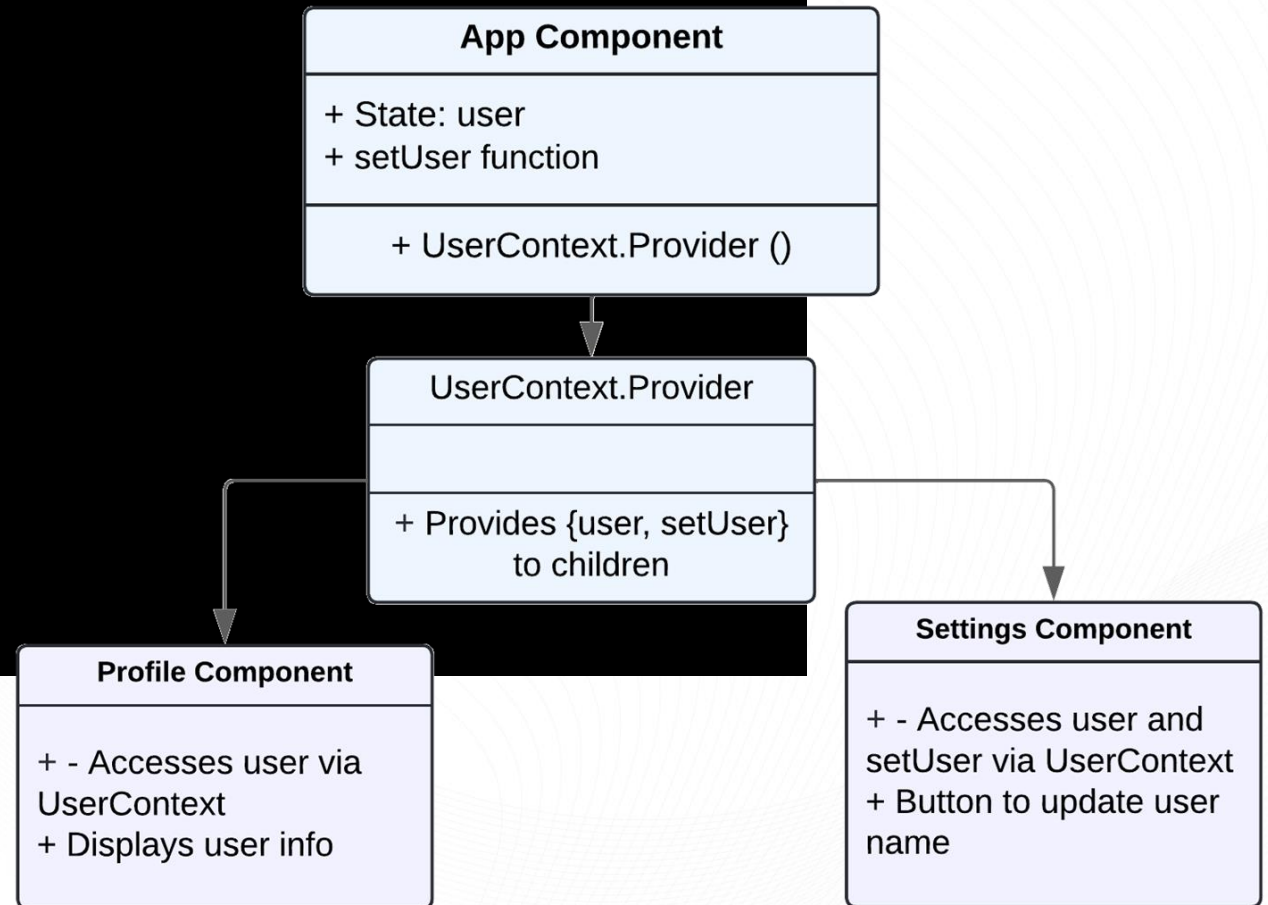
```
import React, { createContext, useContext, useState } from 'react';  
import './App.css';
```

```
// 1. Crear un contexto  
const UserContext = createContext();
```

```
// Componente padre  
export function App() { ...  
}
```

```
// Componentes hijos  
function Profile() { ...  
}
```

```
// Componentes hijos  
function Settings() { ...  
}
```





```
import React, { createContext, useContext, useState } from 'react';
// 1. Crear un contexto
const UserContext = createContext();
// Componente padre
export function App() {
  const [user, setUser] = useState({ name: 'Alice', age: 25 });
  return (
    // 2. Proveer el contexto a los componentes hijos
    <div className='card-uno'>
      <UserContext.Provider value={{ user, setUser }}>
        <Profile />
        <Settings />
      </UserContext.Provider>
    </div>
  );
}
```

```
// Componentes hijos
function Profile() {
  // 3. Acceder al contexto
  const { user } = useContext(UserContext);
  return (
    // 4. Consumir el contexto
    <div className='card-dos'>
      <h1>Perfil: {user.name}</h1>
      <p>Edad: {user.age}</p>
    </div>
  );
}
```





```
// Componentes hijos
function Settings() {
  const { user, setUser } = useContext(UserContext);

  const updateName = () => {
    setUser((prevUser) => ({ ...prevUser, name: 'Bob', age: 30 }));
  };

  return (
    // 5. Modificar el contexto
    <div className='card-dos'>
      <h2>Configuración</h2><hr/>
      <p>Usuario: {user.name}</p>
      <p>Edad: {user.age}</p>
      <button onClick={updateName}>Actualizar Perfil</button>
    </div>
  );
}
```

## Perfil: Alice

Edad: 25

### Configuración

Usuario: Alice

Edad: 25

Actualizar Perfil



# Bibliotecas de gestión de estados



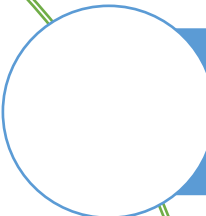
## 2. Bibliotecas de gestión de estados

Para **aplicaciones grandes**, donde el estado es complejo y necesita ser **compartido entre muchos componentes**, bibliotecas de administración de estado como **Redux**, Zustand o Recoil son más adecuadas que la Context API.


**Redux** permite gestionar el estado global con una arquitectura predecible basada en el patrón de "***acción-reductor-estado***".

- **Acciones** representan eventos que describen algo que ha ocurrido.
- **Reductores** son funciones que toman el estado actual y una acción, y devuelven un nuevo estado.
- Storage almacena el **estado** de la aplicación y lo proporciona a los componentes.


# Redux - Ventajas

A white circle with a thin blue outline, connected by a thin blue line to the text box.

**Previsibilidad del Estado:** Al tener un flujo de datos unidireccional y ser inmutable, el estado en Redux es más predecible, lo que facilita la depuración y el rastreo de cambios.

A white circle with a thin teal outline, connected by a thin teal line to the text box.

**Escalabilidad:** Redux es ideal para aplicaciones grandes, donde múltiples componentes y módulos requieren un acceso constante y sincronizado al estado global.

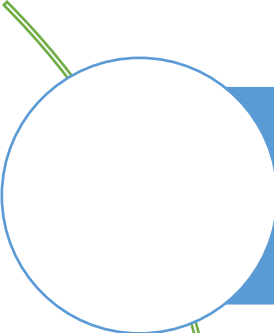
A white circle with a thin green outline, connected by a thin green line to the text box.

**Debugging y Herramientas:** Herramientas como Redux DevTools facilitan la visualización, el historial y la depuración del estado en tiempo real, algo que es muy útil en aplicaciones complejas.

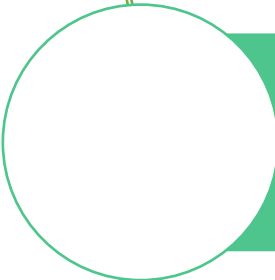
A white circle with a thin olive-green outline, connected by a thin olive-green line to the text box.

**Consistencia y Organización:** Redux ayuda a mantener una arquitectura limpia y organizada, especialmente cuando el flujo de datos en la aplicación es complejo.

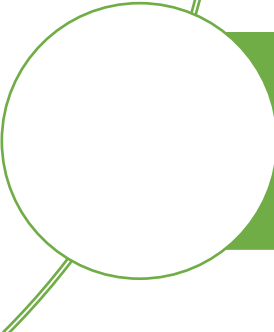
# Redux- Desventajas

A white circle with a thin blue outline, connected by a thin blue line to the top-left corner of the first text box.

**Curva de Aprendizaje Pronunciada:** La estructura de Redux requiere un entendimiento de sus conceptos, lo que puede hacer que la curva de aprendizaje sea alta para principiantes.

A white circle with a thin green outline, connected by a thin green line to the top-left corner of the second text box.

**Verboso:** Comparado con soluciones más simples como la Context API, Redux requiere una mayor cantidad de código para implementar una funcionalidad simple debido a la estructura de acciones, reducers y el store.

A white circle with a thin green outline, connected by a thin green line to the top-left corner of the third text box.

**Rendimiento:** En aplicaciones pequeñas, el uso de Redux puede ser un exceso y reducir el rendimiento debido a la sobrecarga de sus procesos.





# Redux- Instalación

**Paso 1:** Instalar las dependencias necesarias de Redux y React-Redux

```
npm install redux react-redux
```

**Paso 2:** Crear el Store, Actions y Reducer

→ Para crear el store de Redux y poder guardar los estados globales; se debe ejecutar el siguiente comando:

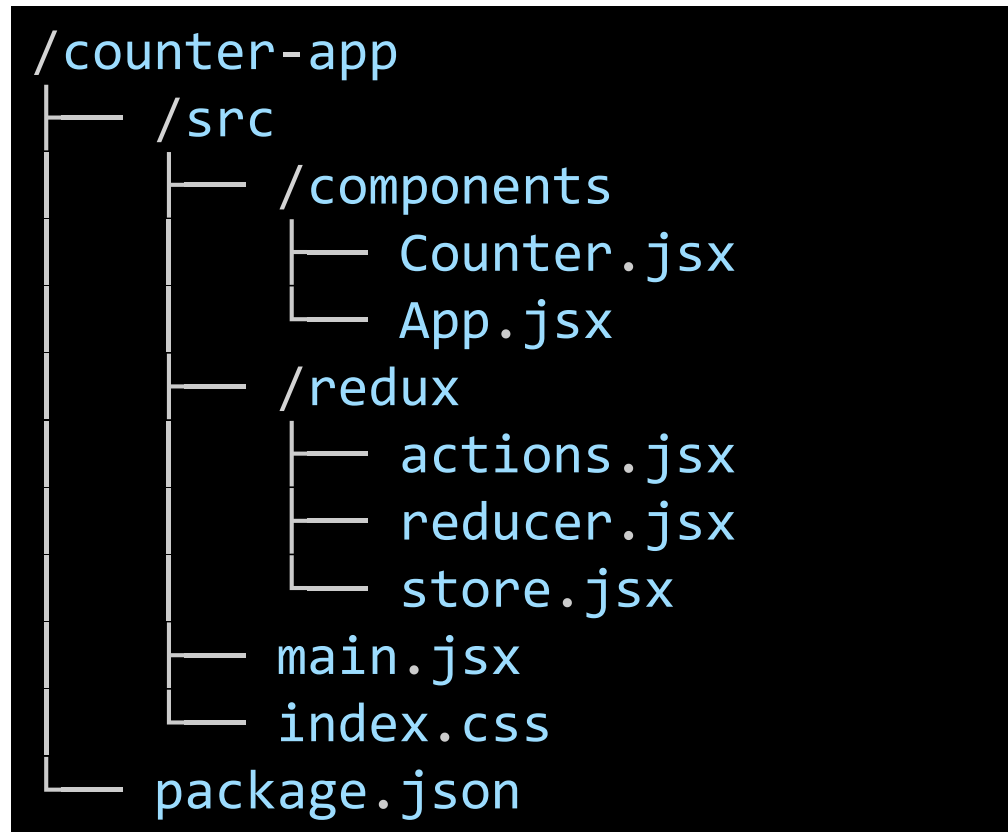
```
npm install @reduxjs/toolkit
```

**Paso 3:** Configurar el Proveedor de Redux en el Componente Principal



# Redux- Instalación

**Paso 4:** Configurar el Consumer de Redux en el Componente Secundario



**Paso 5:** Organizar los scripts bajo una estructura de carpetas para una mejor organización y modularidad de la información.

**Ejercicio:** Crear una aplicación en React para gestionar un contador utilizando Redux para el manejo de su estado. Redux permitirá controlar el valor del contador de manera centralizada, facilitando que cualquier componente en la aplicación acceda al estado actual del contador y realice actualizaciones para aumentarlo o disminuirlo.

# Redux- Ejercicio



## Contador con Redux

Incrementar

23

Decrementar

## Contador con Redux

Incrementar

0

Decrementar

## Contador con Redux

Incrementar

-17

Decrementar

# Redux- Ejercicio



## Componentes Redux

- Archivo **redux/actions.jsx** - Definimos las acciones que manipularán el estado (incrementar y decrementar el contador).
- Archivo **redux/reducer.jsx** - Creamos el reductor que manejará las acciones de Redux.
- Archivo **redux/store.jsx** - Creamos el store de Redux donde vamos a guardar el estado global del contador

# Redux- Ejercicio



## Componentes Visuales

- Archivo **components/App.jsx** – Componente que envuelve la aplicación con el Provider de Redux para que todos los componentes puedan acceder al store.
- Archivo **components/Counter.jsx** – El componente Counter usará useSelector para obtener el valor del contador desde el store, y useDispatch para disparar las acciones de incrementar y decrementar el contador.



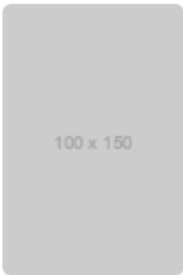
# Ejercicio de Aplicación

# Redux- Ejercicio



## Gestión de Productos

### Lista de Productos



#### Apple

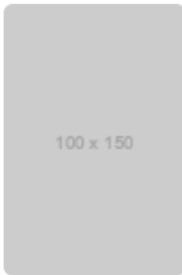
Precio: \$1

Cantidad: 1

Categoría: Fruits

Add

Remove



#### Banana

Precio: \$0.5

Cantidad: 1

Categoría: Fruits

Add

Remove



#### Broccoli

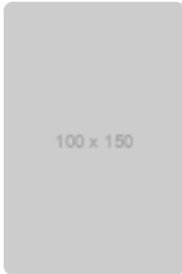
Precio: \$1.5

Cantidad: 1

Categoría: Vegetables

Add

Remove



#### Chicken

Precio: \$5

Cantidad: 1

Categoría: Meats

Add

Remove

### Total Categoría: 3

#### Fruits

Cantidad Total: 2

#### Meats

Cantidad Total: 1

#### Vegetables

Cantidad Total: 1



# Redux- Ejercicio

```
/product-app
├── /src
│   ├── /components
│   │   ├── ProductCard.jsx
│   │   ├── CartSummary.jsx
│   │   └── App.jsx
│   ├── /redux
│   │   ├── actions.jsx
│   │   ├── reducer.jsx
│   │   └── store.jsx
│   ├── main.jsx
│   └── index.css
└── package.json
```

## Ejemplo de Aplicación de Gestión de Productos con Redux

Esta aplicación de ejemplo incluye:

- Una tarjeta de **productos** donde se pueden añadir o eliminar productos.
- Una tarjeta de **categorías** que muestra el total de productos por categoría.
- La cantidad de productos se gestiona **utilizando un estado global en Redux**.

### Instalar Redux y React-Redux:

```
npm install redux react-redux
```

# Primer Paso: Componentes Redux



- **actions.jsx:** Las acciones en Redux representan eventos que indican a la aplicación qué tipo de cambio queremos realizar en el estado. En este caso, vamos a crear acciones para **agregar** y **eliminar productos**.
- **reducer.jsx:** El reductor es una función que define cómo cambia el estado en respuesta a una acción. Aquí, el reductor manejará el **estado** de **productos** (agregar y eliminar) y la cantidad de productos por categoría.
- **store.jsx:** El store de Redux **almacena el estado de la aplicación (agregar y eliminar productos)** y permite acceder a él desde cualquier componente.

```
npm install @reduxjs/toolkit
```



# Segundo Paso: Componentes App



- **ProductCard.jsx:** Este componente representa los productos disponibles. Los botones de "Añadir" y "Quitar" permiten activar acciones de Redux, como `addProduct` o `removeProduct`, que modifican el estado global de los productos.
- **CategoryCard.jsx:** Este componente muestra un resumen, agrupado por categorías y cantidad total de productos. Utiliza el estado global de Redux para acceder a los datos de los productos y cantidades. Cada vez que el estado global cambia, este componente se renderiza nuevamente con los datos actualizados.
- **App.jsx:** El componente principal integra los productos y las categorías.

# Tercer Paso: Elementos Principales



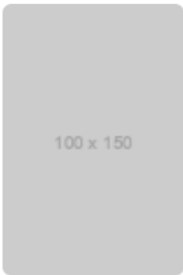
- **main.jsx:** Este archivo es el punto de entrada de la aplicación y donde se envuelve la aplicación con el Provider de Redux, conectando así la store al resto de la app.
- **styles.css:** Este archivo contiene los estilos globales de la aplicación.

# Redux- Ejercicio



## Gestión de Productos

### Lista de Productos



#### Apple

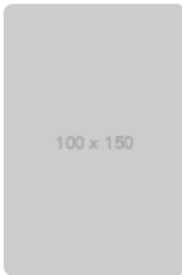
Precio: \$1

Cantidad: 1

Categoría: Fruits

Add

Remove



#### Banana

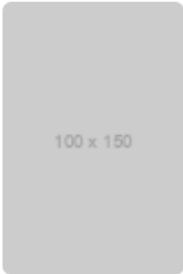
Precio: \$0.5

Cantidad: 1

Categoría: Fruits

Add

Remove



#### Broccoli

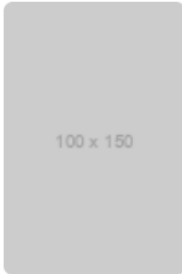
Precio: \$1.5

Cantidad: 1

Categoría: Vegetables

Add

Remove



#### Chicken

Precio: \$5

Cantidad: 1

Categoría: Meats

Add

Remove

### Total Categoría: 3

#### Fruits

Cantidad Total: 2

#### Meats

Cantidad Total: 1

#### Vegetables

Cantidad Total: 1



# GRACIAS

Presentó: Alvaro Pérez Niño  
Instructor Técnico

Correo: [aperezn@misena.edu.co](mailto:aperezn@misena.edu.co)

<http://centrodeserviciosygestionempresarial.blogspot.com/>

Línea de atención al ciudadano: 01 8000 910270

Línea de atención al empresario: 01 8000 910682



@SENAComunica

[www.sena.edu.co](http://www.sena.edu.co)