**🏠 Introducción a Gestión de Estados**

**👋 Introducción**

**¡Te damos la bienvenida!**

En el proceso de desarrollar interfaces de usuario con React, la gestión de estados emerge como un componente crucial. Los estados son fundamentales para permitir a los componentes reaccionar y adaptarse a la interacción del usuario y otros eventos. En este escenario, entra en juego el hook useState, una herramienta esencial especialmente diseñada para los componentes funcionales de React, brindando una forma efectiva de gestionar datos dinámicos. Con useState, los componentes funcionales pueden ahora mantener y actualizar estados de manera eficiente, desbloqueando así la capacidad de crear interfaces dinámicas y altamente interactivas. Este es un componente clave en la caja de herramientas de React que nos permite llevar nuestras aplicaciones al siguiente nivel.

**📖 Material Descargable**

A continuación, te proporcionamos el contenido teórico pertinente: 👉[**Teoría Manejo de Estados**](https://drive.google.com/file/d/1GXwoCCaYJF8M_npkLSoKgj3z0rDZqvfx/view?usp=drive_link)

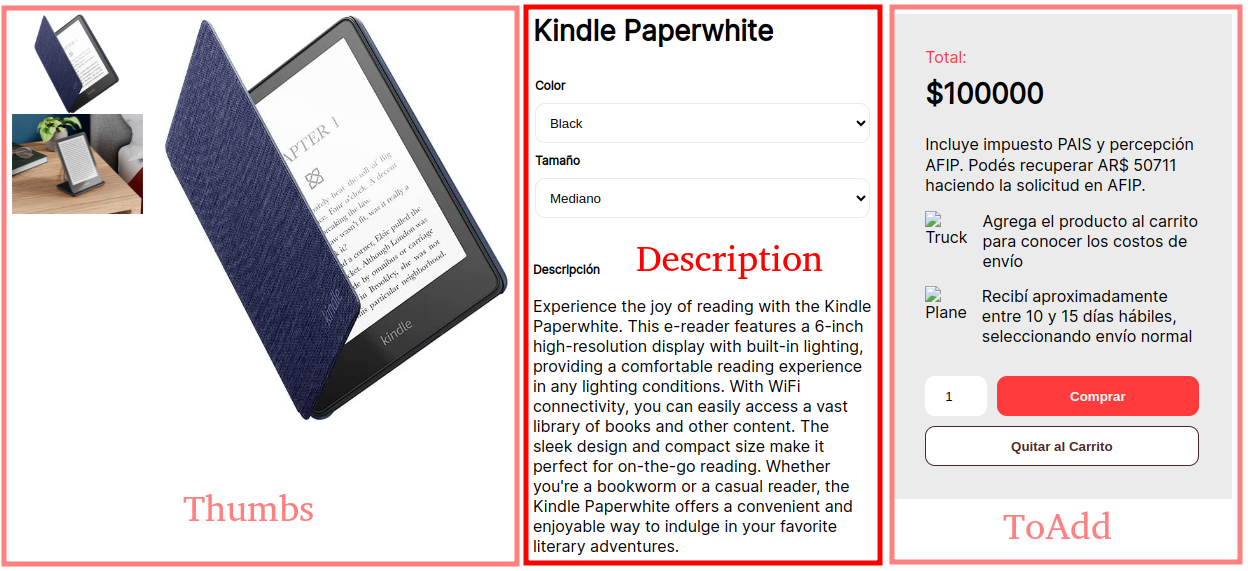
# 👣 Manejar estados para el subtotal del detalle

Este paso se centra en la componentización de la vista de detalle del producto en React. Vamos a dividir la vista en componentes más pequeños y manejables, lo cual es una práctica clave en React para mejorar la organización del código y facilitar la reutilización de componentes.

Nos enfocaremos en una tarea esencial en React: la configuración y gestión del estado de un componente. Específicamente, trabajaremos con el componente ToAdd para implementar un estado que maneje la cantidad de productos a añadir al carrito. Esta tarea es fundamental para entender cómo React permite la interacción del usuario con la aplicación y cómo dichas interacciones afectan el estado de los componentes.

### ✏️ Actividad:  Componentizar la página de de detalles

En esta actividad, descompondrás la vista 'Product' en componentes más pequeños con el objetivo de explorar la modularización de una interfaz compleja. Tu tarea será programar los siguientes componentes:



1. **Crear los componentes Thumbs, Description y ToAdd**:
   * Crear los archivos Thumbs.tsx, Description.tsx y ToAdd.tsx en la carpeta /src/components
   * Definir la estructura básica de cualquier componente.
   * Cortar los elementos que correspondan del componente Cart.
   * Pegar el contenido dentro del retorno return de cada componente.
2. **Definir los estilos de Thumbs, Description y ToAdd**:
   * Crear los archivos Thumbs.module.css, Description.module.css y ToAdd.module.css en la carpeta /src/components
   * Mover los estilos relevantes desde /Cart.module.css a estos archivos.
   * Importar los estilos en cada componente y modificar las clases correspondientes.
3. **Implementar los componentes de Thumbs, Description y ToAdd en la vista**:
   * Importar Thumbs, Description y ToAdd en Cart.
   * Llamar a los componentes <Thumbs /> <Description /> y <ToAdd /> donde estaban las etiquetas pertinentes.
   * Pasar los atributos con las propiedades que necesiten. Debería quedar así:

/\* /src/views/Cart.tsx \*/

<Thumbs product={product} />

<Description product={product} />

<ToAdd product={product} />

1. **Verificar el correcto renderizado de la vista.**

### ✏️ Actividad:  Configurar el estado y su funcións

Aprenderás a utilizar el Hook useState para establecer un estado inicial y una función para actualizarlo. Esta habilidad es crucial en el desarrollo de aplicaciones React, ya que permite crear interfaces dinámicas que responden a la interacción del usuario.

1. **Definir un estado con un número**:
   * Importar el hook useState en el componente ToAdd.
   * Desestructurar el hook con el estado y su función "seteadora".
   * El estado inicial del componente será 1 para indicar que la cantidad mínima a agregar al carrito.

const [quantity, setQuantity] = useState(1);

1. **Configurar el input**:
   * Cambiar el atributo value por defaultValue del input numérico.
   * Configurar el evento para setear el nuevo estado numérico. Cada cambio en el input numérico debe configurar el correspondiente cambio de estado de quantity para luego poder modificar el cálculo del subtotal del producto.

<input

        id="input-quantity"

        type="number"

        min="1"

        defaultValue={quantity}

        onChange={(event) => setQuantity(Number(event?.target.value))}

/>

1. **Cambiar la visualización del subtotal**:
   * Cambiar el product?.price por defaultValue del input numérico.
   * Configurar el evento para setear el nuevo estado numérico. Cada cambio en el input numérico debe configurar el correspondiente cambio de estado de quantity para luego poder modificar el cálculo del subtotal del producto.

<input

        id="input-quantity"

        type="number"

        min="1"

        defaultValue={quantity}

        onChange={(event) => setQuantity(Number(event?.target.value))}

/>

* Realizar los cambios necesarios para que el componente se visualice según la imagen:



* Implementar el método toLocaleString para las "comas" de los miles.
* Interpolar los estados necesarios para calcular el valor del subtotal.

{(product?.price \* quantity).toLocaleString()}

1. **Verificar el correcto renderizado y cambio de estados.**

**👣 Manejar estados para añadir al carrito**

Este paso es fundamental para cualquier desarrollador de React que busque crear interfaces interactivas y responsivas. No solo habrás mejorado la funcionalidad de tu aplicación, sino que también tendrás una comprensión más profunda de cómo React maneja los cambios de estado y actualiza la interfaz de usuario en consecuencia.

**✏️ Actividad:  Cambiar estados del botón de "Añadir al carrito"**

En esta actividad, explorarás la interactividad en React implementando cambios de estado en el botón de "Añadir al Carrito". Comprenderás cómo los cambios en el estado de un componente pueden influir significativamente en su presentación y comportamiento.

1. **Definir un estado con un booleano**:
   * Importar el hook useState en el componente ToAdd.
   * Desestructurar el hook con el estado y su función "seteadora".
   * El estado inicial del componente será false para indicar que el producto todavía no fue agregado al carrito.

const [button, setButton] = useState(false);

1. **Configurar el botón con los ternarios correspondiente**:
   * Si el estado button está en false se debe mostrar el botón rojo y debe decir "Añadir al carrito".
   * Interpolar con el ternario el atributo className para cambiar la clase del botón.
   * Si el estado button está en true se debe mostrar el botón marrón y debe decir "Quitar al carrito".
   * Interpolar con el ternario el texto del botón.

<button

      className={button ? styles.btnOff : styles.btnOutline}

      id="btn-add-to-cart"

      onClick={() => setButton(!button)}

> {button ? "Quitar" : "Añadir"} al Carrito </button>

**✏️ Actividad:  Gestionar los datos del localStorage**

En esta parte, manejarás el almacenamiento local del navegador para guardar la información de los productos añadidos al carrito, y también para retirarlos cuando el usuario así lo decida.

1. **Almacenar Productos en localStorage:**
   * Utiliza localStorage para guardar los detalles de cada producto añadido al carrito.
   * Asegúrate de actualizar localStorage cuando los productos sean añadidos o quitados.
2. **Retirar Productos del localStorage:**
   * Implementa la lógica para eliminar los datos de los productos del localStorage cuando se quiten del carrito.
   * Esta acción debe sincronizarse con el cambio de estado del botón de "Añadir/Quitar al Carrito".