Taller de React: Dashboard de Administración de E-Commerce

¡Bienvenido! En este taller, construirás un Dashboard de Administración funcional para gestionar el inventario de una tienda. Este proyecto no es solo una "lista de tareas", sino una aplicación de página única (SPA) con un *layout* profesional, enrutamiento anidado y un modelo de datos híbrido.

El Desafío: Modo Híbrido (API + Estado Local)

Este taller está diseñado para enseñarte una arquitectura de prototipado muy común:

1. **READ (Leer) → Desde la API :** Cargarás la lista inicial de productos y los detalles de un producto específico consultando una API real (`dummyjson.com`). Esto te obligará a manejar `useEffect`, `fetch`, y los estados de `loading`/`error`.
2. **CREATE, UPDATE, DELETE (CUD) → En Estado Local :** Para una experiencia de usuario instantánea (y para simplificar la lógica del backend), *simularás* las operaciones de creación, actualización y eliminación. En lugar de enviar `POST`, `PUT` o `DELETE` a la API, modificarás directamente el array de productos que tienes en tu estado de React.

Conceptos Clave que Aplicarás

- **React Router:** Enrutamiento anidado con un `<Layout />` persistente, rutas dinámicas (`:productId`), y paso de contexto (`<Outlet context={...} />`).
- **Manejo de Estado:** "Lifting State Up" (levantar el estado) a un componente padre (`App.jsx`).
- **Hooks:** `useEffect`, `useState`, `useNavigate`, `useParams`, `useOutletContext`.
- **Formularios:** Creación de un componente de formulario controlado y reutilizable.
- **Manipulación de Datos:** Uso de `.map()`, `.filter()` para actualizar el estado local de forma inmutable.

Objetivo

Tu trabajo es construir toda la lógica de React. Se te proporciona **únicamente el archivo CSS** al final de este documento. Deberás crear los componentes, gestionar el estado, definir las rutas y conectar todo usando los `className` que se proveen en el CSS.

HINT 1:

```
const sleep = (ms) => new Promise(resolve => setTimeout(resolve, ms))
```

colocar esta instrucción en la lógica del componente para que puedan simular tiempos de carga.

```
// El código MODIFICADO con async/await
const { setProducts, setLoading } = useOutletContext();

const handleDelete = async (productId) => { // 1. Marca la función como 'async'
  if (window.confirm("¿Seguro que quieres eliminar?")) {
    try {
      // 2. Inicia el loading
      setLoading(true);

      // 3. Espera 1.5 segundos (simulando la llamada a la API)
      await sleep(1500);

      // 4. Ejecuta tu lógica
      setProducts(prevProducts => prevProducts.filter(p => p.id !== productId));

    } catch (error) {
      console.error("Error al eliminar", error);
      // Aquí también deberías manejar el estado de error
    } finally {
      // 5. Detén el loading (esto se ejecuta siempre, incluso si hay un error)
      setLoading(false);
    }
  }
};
```

HINT 2:

```
const willFail = Math.random() < 0.4;
```

```
// helpers (fuera del componente)
const sleep = (ms) => new Promise(resolve => setTimeout(resolve, ms));

// Dentro de ProductListPage.jsx
const { setProducts, setLoading, error, setError } = useOutletContext(); // <-- Añadimos setError

const handleDelete = async (productId) => {
  if (window.confirm("¿Seguro que quieres eliminar este producto?")) {

    try {
      // 1. Limpia errores antiguos e inicia la carga
      setError(null);
      setLoading(true);

      // 2. Simula el tiempo de espera de la red
      await sleep(1500);

      // 3. ¡El dado de la suerte! (Simulamos un 40% de probabilidad de fallo)
      const willFail = Math.random() < 0.4;

      if (willFail) {
        // 4. Lanzamos un error simulado
        throw new Error("¡Error de red simulado! No se pudo conectar con el servidor.");
      }

      // 5. Si no falló, ejecuta la lógica de éxito
      setProducts(prevProducts => prevProducts.filter(p => p.id !== productId));

    } catch (err) { // 6. El 'throw' será atrapado aquí
      console.error("Error simulado:", err.message);
      setError(err); // <-- ¡Actualiza el estado de error global!

    } finally {
      // 7. Esto se ejecuta siempre (con éxito O con error)
      setLoading(false);
    }
  }
};
```

Esta instrucción nos permite simular los errores que podrían pasar por cualquier razón.

Documentación Útil:

1. Componente Outlet de React Router: qué es y cómo usarlo 2025
2. useOutletContext | React Router

Paso 0: Configuración del Proyecto

1. **Crea tu proyecto** (Vite es recomendado):
2. Bash
3. npm create vite@latest dashboard-ecommerce -- --template react
4. cd dashboard-ecommerce
5. **Instala React Router:**
6. Bash
7. npm install react-router-dom
8. **Crea tu estructura de archivos:**

```
src/
├── components/
│   ├── Layout.jsx        # (El shell de la app: sidebar + área de contenido)
│   └── ProductForm.jsx   # (Formulario reutilizable para Crear/Editar)
├── pages/
│   ├── ProductListPage.jsx  # (La tabla de productos)
│   ├── NewProductPage.jsx   # (La página que usa el ProductForm para crear)
│   └── EditProductPage.jsx  # (La página que usa el ProductForm para editar)
├── App.jsx               # (Componente raíz: define rutas y estado global)
└── App.css               # (Copia y pega el CSS que te doy al final)
```

Copia el CSS: Toma todo el código CSS al final de este documento y pégalo en tu src/App.css.

Inicia tu app: npm run dev

Paso 1: Estado Global y Enrutamiento (El Cerebro)

En lugar de que cada página maneje su propio estado, vamos a "levantar el estado" a App.jsx. Esto permite que todas las páginas compartan y modifiquen la misma lista de productos.

src/App.jsx

1. **Importa** los componentes de react-router-dom y tus páginas.
2. **Define el Estado Global:** Crea los estados para products, setProducts, loading, setLoading, error, y setError aquí, en el componente padre.
3. **Carga de Datos:** Crea un useEffect (con []) aquí mismo para hacer fetch a https://dummyjson.com/products.
4. **Maneja la Carga:** En el fetch, actualiza loading, error, y finalmente setProducts(data.products).
5. **Define las Rutas:**
    - Configura <Routes> para tener una **ruta padre** (path="/") que renderice tu componente <Layout />.
    - Dentro de esta ruta padre, **anida las rutas hijas**:
        - <Route index ... /> para ProductListPage.
        - <Route path="new" ... /> para NewProductPage.
        - <Route path="edit/:productId" ... /> para EditProductPage.
        - <Route path="*" ... /> para una página 404.

src/components/Layout.jsx

3. Este componente es el "shell" visual. Debe renderizar el <nav className="sidebar">...</nav> y el <main className="main-content">...</main>.
4. Usa <NavLink> de React Router para los enlaces de la barra lateral (Inventario y Añadir Producto).
5. **Hint :** Para renderizar las rutas hijas (ProductListPage, NewProductPage, etc.) dentro del <main>, necesitarás usar el componente <Outlet />.
6. **Hint :** Para pasar tu estado global (products, loading, etc.) a todas las rutas hijas, usa la prop context del <Outlet />.
    - Ejemplo: <Outlet context={{ products, setProducts, loading, error }} />

Paso 2: READ - Mostrar los Productos (API)

src/pages/ProductListPage.jsx

1. Esta página no necesita su propio estado, ¡lo recibirá del Layout!
2. **Obtén el Contexto:** Usa el hook useOutletContext() para acceder a { products, setProducts, loading, error }.
3. **Renderizado Condicional:**
    - Si loading es true, muestra un ... .
    - Si error existe, muestra un ... .
4. **Renderizado de Tabla:**
    - Si hay products, usa .map() para renderizarlos en una <table> (usa las clases product-table-wrapper y product-table).
    - Incluye columnas para Imagen (product-thumbnail), Nombre, Categoría, Precio y Acciones.
5. **Botones de Acción:** En la columna "Acciones", renderiza un <Link> que lleve a /edit/${product.id} y un <button> para "Eliminar".

Paso 3: DELETE - Eliminar un Producto (Local)

src/pages/ProductListPage.jsx (Continuación)

1. **Crea la función handleDelete:** Esta función recibirá el productId.
2. **Pide Confirmación:** Usa window.confirm() para asegurarte de que el usuario quiere eliminar.
3. **Actualiza el Estado (Local):** Si el usuario confirma, actualiza el estado global (que obtuviste del contexto) usando setProducts().
    - Hint : La forma inmutable de eliminar un ítem de un array es con .filter(): setProducts(prevProducts => prevProducts.filter(p => p.id !== productId))
4. Pasa esta función al onClick del botón "Eliminar" en tu .map().

Paso 4: CREATE - Añadir un Producto (Local)

src/components/ProductForm.jsx (El Formulario Reutilizable)

1. Este es un **componente controlado**. Deberá tener su *propio* estado para manejar los campos del formulario (ej. const [formData, setFormData] = useState(...)).
2. **Props:** Debe aceptar initialData (para edición), onSubmit (la función a ejecutar), y isLoading.
3. **Estado Inicial:** Usa useState(initialData) para pre-llenar el formulario.
    - **Hint :** Para manejar la edición, necesitarás un useEffect que *actualice* formData si la prop initialData cambia. useEffect(() => setFormData(initialData), [initialData]).
4. Crea los div.form-group con label e input.form-control para title, description, price, category, etc.
5. Maneja el onChange de cada input para actualizar formData.
6. En el onSubmit del <form>, llama a e.preventDefault() y luego ejecuta la prop onSubmit(formData).

src/pages/NewProductPage.jsx

1. **Obtén el Contexto:** Usa useOutletContext() para acceder a { setProducts }.
2. **Navegación:** Usa el hook useNavigate().
3. **Define handleCreate:** Esta es la función que pasarás al onSubmit de tu <ProductForm />.
4. Lógica de handleCreate(formData):
   a. No llames a la API.
   b. Crea un objeto de producto nuevo: const newProduct = { ...formData, id: Date.now() } (Usa Date.now() para un ID temporal único).
   c. Actualiza el estado global: setProducts(prevProducts => [newProduct, ...prevProducts]).
   d. Redirige al usuario a la lista: Maps('/').
5. **Renderiza:** <ProductForm onSubmit={handleCreate} initialData={{ title: '', description: '', price: 0, ... }} />

Paso 5: UPDATE - Editar un Producto (Híbrido)

Esta es la tarea más avanzada que combina todo.

src/pages/EditProductPage.jsx

1. **Hooks:** Importa useParams, useNavigate, useOutletContext, useState, y useEffect.
2. **Obtén Contexto y Parámetros:**
   - const { setProducts } = useOutletContext().
   - const { productId } = useParams().
   - const navigate = useNavigate().
3. **Estado Local (para Carga):** Crea un estado local *solo para este componente*: const [productData, setProductData] = useState(null).
4. **Paso 1: Leer (API)**
   - Crea un useEffect que se ejecute cuando productId cambie ([productId]).
   - Dentro, haz fetch a https://dummyjson.com/products/${productId}.
   - Guarda el resultado en setProductData(data).
5. **Paso 2: Actualizar (Local)**
   - Define la función handleUpdate(formData).
   - **No llames a la API.**
   - Actualiza el estado global: setProducts(prevProducts => prevProducts.map(p => p.id === Number(productId) ? { ...p, ...formData } : p)).
   - **Hint :** productId viene de la URL como *string*. Asegúrate de compararlo correctamente (ej. Number(productId)).
   - Redirige al usuario: Maps('/').
6. **Renderizado:**
   - Si productData es null, muestra "Cargando datos del producto...".
   - Si productData existe, renderiza:
     <ProductForm onSubmit={handleUpdate} initialData={productData} />

Recurso: Estilos (CSS)

Copia todo este código en tu archivo `src/App.css`

CSS

```
/* --- Reseteo Básico y Variables --- */
:root {
  --primary-color: #007bff;
  --primary-hover: #0056b3;
  --danger-color: #dc3545;
  --danger-hover: #a71d2a;
  --secondary-color: #6c757d;
  --secondary-hover: #545b62;
  --bg-color: #f8f9fa;
  --sidebar-bg: #ffffff;
  --border-color: #dee2e6;
  --text-color: #212529;
  --shadow: 0 1px 4px rgba(0, 21, 41, 0.08);
}

* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}

body {
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, 'Helvetica Neue', Arial,
sans-serif;
  background-color: var(--bg-color);
  color: var(--text-color);
}

a {
  text-decoration: none;
  color: var(--primary-color);
}

/* --- 1. Layout Principal --- */
.app-layout {
  display: flex;
  min-height: 100vh;
}

.sidebar {
  width: 240px;
  background-color: var(--sidebar-bg);
  border-right: 1px solid var(--border-color);
  padding: 1.5rem;
```

```
  box-shadow: var(--shadow);
  flex-shrink: 0;
}

.sidebar-header {
  font-size: 1.5rem;
  font-weight: bold;
  margin-bottom: 2rem;
  color: var(--primary-color);
}

.sidebar-nav ul {
  list-style: none;
}

.sidebar-nav li {
  margin-bottom: 1rem;
}

.nav-link {
  font-size: 1.1rem;
  color: var(--secondary-color);
  display: block;
  padding: 0.5rem;
  border-radius: 5px;
  transition: background-color 0.2s, color 0.2s;
}

.nav-link:hover {
  background-color: #f1f1f1;
}

/* React Router pasará esta clase automáticamente a NavLink */
.nav-link.active {
  background-color: var(--primary-color);
  color: #ffffff;
}

.main-content {
  flex-grow: 1;
  padding: 2rem;
  overflow-y: auto;
}

/* --- 2. Encabezados de Página --- */
.page-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
```

```
  margin-bottom: 2rem;
}

.page-title {
  font-size: 2rem;
  font-weight: 600;
}

/* --- 3. Botones --- */
.btn {
  padding: 0.75rem 1.25rem;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  font-size: 1rem;
  font-weight: 500;
  transition: background-color 0.2s;
}

.btn-primary {
  background-color: var(--primary-color);
  color: #fff;
}
.btn-primary:hover {
  background-color: var(--primary-hover);
}

.btn-danger {
  background-color: var(--danger-color);
  color: #fff;
}
.btn-danger:hover {
  background-color: var(--danger-hover);
}

.btn-secondary {
  background-color: var(--secondary-color);
  color: #fff;
}
.btn-secondary:hover {
  background-color: var(--secondary-hover);
}

.btn:disabled {
  background-color: #ccc;
  cursor: not-allowed;
}
```

```css
/* --- 4. Tabla de Productos --- */
.product-table-wrapper {
  background-color: #fff;
  border: 1px solid var(--border-color);
  border-radius: 8px;
  box-shadow: var(--shadow);
  overflow-x: auto; /* Para responsive */
}

.product-table {
  width: 100%;
  border-collapse: collapse;
}

.product-table th,
.product-table td {
  padding: 1rem;
  text-align: left;
  border-bottom: 1px solid var(--border-color);
  vertical-align: middle;
}

.product-table th {
  font-weight: 600;
  background-color: #f9f9f9;
}

.product-table tr:last-child td {
  border-bottom: none;
}

.product-table tr:hover {
  background-color: #f1f1f1;
}

.product-thumbnail {
  width: 60px;
  height: 60px;
  object-fit: cover;
  border-radius: 5px;
}

.product-actions {
  display: flex;
  gap: 0.5rem;
}

/* Estilo para Link que parece botón */
.btn-link {
```

```
  display: inline-block;
  padding: 0.4rem 0.8rem;
  font-size: 0.9rem;
  text-align: center;
  border-radius: 5px;
  text-decoration: none;
}

/* Clases para los botones de acción */
.btn-edit {
  background-color: var(--secondary-color);
  color: #fff;
}
.btn-edit:hover {
  background-color: var(--secondary-hover);
}

.btn-delete {
  padding: 0.4rem 0.8rem;
  font-size: 0.9rem;
  background-color: var(--danger-color);
  color: #fff;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}
.btn-delete:hover {
  background-color: var(--danger-hover);
}


/* --- 5. Formularios --- */
.form-container {
  max-width: 800px;
  background-color: #fff;
  padding: 2rem;
  border: 1px solid var(--border-color);
  border-radius: 8px;
  box-shadow: var(--shadow);
}

.form-group {
  margin-bottom: 1.5rem;
}

.form-group label {
  display: block;
  font-weight: 500;
  margin-bottom: 0.5rem;
```

```
}

.form-control {
  width: 100%;
  padding: 0.75rem;
  font-size: 1rem;
  border: 1px solid var(--border-color);
  border-radius: 5px;
}

.form-control:focus {
  outline: none;
  border-color: var(--primary-color);
  box-shadow: 0 0 0 2px rgba(0, 123, 255, 0.25);
}

textarea.form-control {
  min-height: 120px;
  resize: vertical;
}

/* --- 6. Loader y Mensajes --- */
.loader-container,
.error-container {
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 4rem;
  font-size: 1.25rem;
}

.error-message {
  color: var(--danger-color);
  font-weight: 500;
  background-color: #fdd;
  border: 1px solid var(--danger-color);
  padding: 1rem;
  border-radius: 5px;
}
```