**Estructura básica del proyecto con ReactJS y Vite**

**1. Crear el proyecto con Vite**

Ejecuta los siguientes comandos en tu terminal para inicializar el proyecto con Vite:

npm create vite@latest 2000s-style --template react

cd 2000s-style

npm install

**2. Organizar la estructura del proyecto**

La estructura del proyecto será similar a la anterior, pero adaptada para Vite:

src/

├── components/

│ ├── Header.jsx

│ ├── Footer.jsx

│ ├── Navbar.jsx

├── pages/

│ ├── Contacto.jsx

│ ├── Productos.jsx

│ ├── Resenias.jsx

├── App.jsx

├── main.jsx

├── styles.css

**3. Instalar React Router**

Para manejar las rutas entre las páginas, instala react-router-dom:

npm install react-router-dom

**4. Configurar las rutas en App.jsx**

Usaremos React Router para definir las rutas de las páginas principales (Contacto, Productos, Reseñas).

**Código inicial**

**main.jsx**

Este archivo es el punto de entrada de la aplicación:

import React from 'react';

import ReactDOM from 'react-dom/client';

import { BrowserRouter } from 'react-router-dom';

import App from './App';

import './styles.css';

ReactDOM.createRoot(document.getElementById('root')).render(

<React.StrictMode>

<BrowserRouter>

<App />

</BrowserRouter>

</React.StrictMode>

);

**App.jsx**

Aquí configuramos las rutas principales:

import React from 'react';

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

import Navbar from './components/Navbar';

import Footer from './components/Footer';

import Contacto from './pages/Contacto';

import Productos from './pages/Productos';

import Resenias from './pages/Resenias';

const App = () => {

return (

<div>

<Navbar />

<Routes>

<Route path="/" element={<Productos />} />

<Route path="/contacto" element={<Contacto />} />

<Route path="/resenias" element={<Resenias />} />

</Routes>

<Footer />

</div>

);

};

export default App;

**components/Navbar.jsx**

El componente de navegación:

import React from 'react';

import { Link } from 'react-router-dom';

const Navbar = () => {

return (

<nav>

<ul>

<li><Link to="/">Productos</Link></li>

<li><Link to="/contacto">Contacto</Link></li>

<li><Link to="/resenias">Reseñas</Link></li>

</ul>

</nav>

);

};

export default Navbar;

**components/Footer.jsx**

El pie de página:

import React from 'react';

const Footer = () => {

return (

<footer>

<p>© 2025 - Todos los derechos reservados</p>

</footer>

);

};

export default Footer;

**pages/Productos.jsx**

Página de productos:

import React from 'react';

const Productos = () => {

return (

<div>

<h1>Productos</h1>

<p>Aquí se mostrarán los productos.</p>

</div>

);

};

export default Productos;

**pages/Contacto.jsx**

Página de contacto:

import React from 'react';

const Contacto = () => {

return (

<div>

<h1>Contacto</h1>

<p>Formulario de contacto aquí.</p>

</div>

);

};

export default Contacto;

**pages/Resenias.jsx**

Página de reseñas:

import React from 'react';

const Resenias = () => {

return (

<div>

<h1>Reseñas</h1>

<p>Aquí se mostrarán las reseñas.</p>

</div>

);

};

export default Resenias;

**styles.css**

Archivo de estilos globales:

body {

font-family: Arial, sans-serif;

margin: 0;

padding: 0;

}

nav ul {

list-style: none;

display: flex;

justify-content: space-around;

padding: 10px;

background-color: #4930cd;

}

nav ul li a {

color: white;

text-decoration: none;

}

footer {

text-align: center;

padding: 10px;

background-color: #f6f8fa;

}

**Próximos pasos**

1. **Migrar la lógica de los archivos JavaScript (productos.js, resenias.js, validarContacto.js) a los componentes correspondientes.**
2. **Integrar los estilos específicos en los componentes.**
3. **Probar la aplicación en el navegador ejecutando:**

npm run dev

¡Perfecto! Vamos a comenzar con la migración de la lógica de \*\*productos.js\*\* a ReactJS. Esto incluirá:

1. **Cargar los productos desde la API (https://dummyjson.com/products)**.
2. **Mostrar los productos en una lista o cuadrícula**.
3. **Agregar funcionalidad para excluir productos de la categoría "groceries"**.
4. \*\*Preparar el estado para manejar los datos de los productos usando useState y useEffect.

**Migración de productos.js a ReactJS**

**Código del componente Productos.jsx**

import React, { useState, useEffect } from 'react';

const Productos = () => {

const [productos, setProductos] = useState([]); // Estado para almacenar los productos

const [loading, setLoading] = useState(true); // Estado para manejar el cargando

// Función para cargar los productos desde la API

useEffect(() => {

const fetchProductos = async () => {

try {

const response = await fetch('https://dummyjson.com/products');

if (!response.ok) {

throw new Error('Error al cargar los productos');

}

const data = await response.json();

const productosValidos = data.products.filter(

(producto) => producto.category !== 'groceries' // Excluir productos de la categoría "groceries"

);

setProductos(productosValidos);

setLoading(false);

} catch (error) {

console.error('Error al cargar los datos:', error);

setLoading(false);

}

};

fetchProductos();

}, []);

// Renderizar los productos

return (

<div>

<h1>Productos</h1>

{loading ? (

<p>Cargando productos...</p>

) : (

<div className="productos-grid">

{productos.map((producto) => (

<div key={producto.id} className="producto-card">

<img

src={producto.thumbnail}

alt={producto.title}

className="producto-imagen"

/>

<h3>{producto.title}</h3>

<p>{producto.description}</p>

<p>Precio: ${producto.price}</p>

</div>

))}

</div>

)}

</div>

);

};

export default Productos;

**Explicación del código**

1. **Estado (useState)**:
   * productos: Almacena los datos de los productos.
   * loading: Indica si los datos están siendo cargados.
2. **Efecto (useEffect)**:
   * Se utiliza para cargar los datos desde la API cuando el componente se monta.
   * Filtra los productos para excluir la categoría "groceries".
3. **Renderización**:
   * Si los datos están cargando, muestra un mensaje de "Cargando productos...".
   * Una vez cargados, muestra los productos en una cuadrícula.

**Estilos para la cuadrícula de productos**

Agrega estos estilos en styles.css para mejorar la presentación:

.productos-grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));

gap: 20px;

padding: 20px;

}

.producto-card {

border: 1px solid #ddd;

border-radius: 8px;

padding: 10px;

text-align: center;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

}

.producto-imagen {

max-width: 100%;

border-radius: 8px;

}

**Próximos pasos**

1. **Probar el componente**: Ejecuta el proyecto con npm run dev y verifica que los productos se carguen correctamente.
2. **Agregar funcionalidad adicional**:
   * ¿Te gustaría agregar un carrito de compras o filtros para los productos?

¡Avísame cómo deseas proceder! 😊

¡Perfecto! Vamos a migrar la lógica de \*\*resenias.js\*\* a ReactJS. Esto incluirá:

1. **Cargar las reseñas desde la API (https://dummyjson.com/comments)**.
2. **Mostrar las reseñas en una lista**.
3. **Agregar un filtro por categoría o palabra clave**.

**Migración de resenias.js a ReactJS**

**Código del componente Resenias.jsx**

import React, { useState, useEffect } from 'react';

const Resenias = () => {

const [resenias, setResenias] = useState([]); // Estado para almacenar las reseñas

const [filtro, setFiltro] = useState(''); // Estado para manejar el filtro

const [loading, setLoading] = useState(true); // Estado para manejar el cargando

// Función para cargar las reseñas desde la API

useEffect(() => {

const fetchResenias = async () => {

try {

const response = await fetch('https://dummyjson.com/comments');

if (!response.ok) {

throw new Error('Error al cargar las reseñas');

}

const data = await response.json();

setResenias(data.comments); // Guardar las reseñas en el estado

setLoading(false);

} catch (error) {

console.error('Error al cargar los datos:', error);

setLoading(false);

}

};

fetchResenias();

}, []);

// Filtrar las reseñas según el texto ingresado en el filtro

const reseniasFiltradas = resenias.filter((resenia) =>

resenia.body.toLowerCase().includes(filtro.toLowerCase())

);

return (

<div>

<h1>Reseñas</h1>

<input

type="text"

placeholder="Filtrar por palabra clave..."

value={filtro}

onChange={(e) => setFiltro(e.target.value)}

style={{

padding: '10px',

marginBottom: '20px',

width: '100%',

maxWidth: '400px',

borderRadius: '5px',

border: '1px solid #ddd',

}}

/>

{loading ? (

<p>Cargando reseñas...</p>

) : (

<div className="resenias-lista">

{reseniasFiltradas.length > 0 ? (

reseniasFiltradas.map((resenia) => (

<div key={resenia.id} className="resenia-card">

<h3>{resenia.user.username}</h3>

<p>{resenia.body}</p>

</div>

))

) : (

<p>No se encontraron reseñas que coincidan con el filtro.</p>

)}

</div>

)}

</div>

);

};

export default Resenias;

**Explicación del código**

1. **Estado (useState)**:
   * resenias: Almacena las reseñas obtenidas de la API.
   * filtro: Almacena el texto ingresado por el usuario para filtrar las reseñas.
   * loading: Indica si los datos están siendo cargados.
2. **Efecto (useEffect)**:
   * Se utiliza para cargar las reseñas desde la API cuando el componente se monta.
3. **Filtrado**:
   * Se filtran las reseñas en función del texto ingresado en el campo de búsqueda.
4. **Renderización**:
   * Si los datos están cargando, muestra un mensaje de "Cargando reseñas...".
   * Si no hay reseñas que coincidan con el filtro, muestra un mensaje indicando que no se encontraron resultados.

**Estilos para las reseñas**

Agrega estos estilos en styles.css para mejorar la presentación:

.resenias-lista {

display: flex;

flex-direction: column;

gap: 20px;

padding: 20px;

}

.resenia-card {

border: 1px solid #ddd;

border-radius: 8px;

padding: 10px;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

}

.resenia-card h3 {

margin: 0;

color: #4930cd;

}

.resenia-card p {

margin: 5px 0 0;

color: #333;

}

**Próximos pasos**

1. **Probar el componente**: Ejecuta el proyecto con npm run dev y verifica que las reseñas se carguen correctamente y que el filtro funcione.
2. **Agregar funcionalidad adicional**:
   * ¿Te gustaría agregar un sistema de clasificación (por ejemplo, estrellas)?
   * ¿O prefieres implementar paginación para las reseñas?

Perfecto! Vamos a migrar la lógica de \*\*validarContacto.js\*\* a ReactJS. Esto incluirá:

1. **Crear un formulario de contacto con validación en tiempo real**.
2. **Validar campos como nombre, correo electrónico y mensaje**.
3. **Mostrar mensajes de error si los campos no cumplen con los requisitos**.

**Migración de validarContacto.js a ReactJS**

**Código del componente Contacto.jsx**

import React, { useState } from 'react';

const Contacto = () => {

const [formulario, setFormulario] = useState({

nombre: '',

correo: '',

mensaje: '',

});

const [errores, setErrores] = useState({});

const [enviado, setEnviado] = useState(false);

// Función para manejar los cambios en los campos del formulario

const manejarCambio = (e) => {

const { name, value } = e.target;

setFormulario({ ...formulario, [name]: value });

};

// Función para validar el formulario

const validarFormulario = () => {

const nuevosErrores = {};

if (!formulario.nombre.trim()) {

nuevosErrores.nombre = 'El nombre es obligatorio.';

}

if (!formulario.correo.trim()) {

nuevosErrores.correo = 'El correo electrónico es obligatorio.';

} else if (!/\S+@\S+\.\S+/.test(formulario.correo)) {

nuevosErrores.correo = 'El correo electrónico no es válido.';

}

if (!formulario.mensaje.trim()) {

nuevosErrores.mensaje = 'El mensaje es obligatorio.';

} else if (formulario.mensaje.length < 10) {

nuevosErrores.mensaje = 'El mensaje debe tener al menos 10 caracteres.';

}

return nuevosErrores;

};

// Función para manejar el envío del formulario

const manejarEnvio = (e) => {

e.preventDefault();

const nuevosErrores = validarFormulario();

if (Object.keys(nuevosErrores).length === 0) {

setEnviado(true);

setErrores({});

console.log('Formulario enviado:', formulario);

// Aquí podrías enviar los datos a un servidor o API

} else {

setErrores(nuevosErrores);

setEnviado(false);

}

};

return (

<div>

<h1>Contacto</h1>

<form onSubmit={manejarEnvio} className="formulario-contacto">

<div>

<label htmlFor="nombre">Nombre:</label>

<input

type="text"

id="nombre"

name="nombre"

value={formulario.nombre}

onChange={manejarCambio}

/>

{errores.nombre && <p className="error">{errores.nombre}</p>}

</div>

<div>

<label htmlFor="correo">Correo Electrónico:</label>

<input

type="email"

id="correo"

name="correo"

value={formulario.correo}

onChange={manejarCambio}

/>

{errores.correo && <p className="error">{errores.correo}</p>}

</div>

<div>

<label htmlFor="mensaje">Mensaje:</label>

<textarea

id="mensaje"

name="mensaje"

value={formulario.mensaje}

onChange={manejarCambio}

></textarea>

{errores.mensaje && <p className="error">{errores.mensaje}</p>}

</div>

<button type="submit">Enviar</button>

{enviado && <p className="exito">¡Formulario enviado con éxito!</p>}

</form>

</div>

);

};

export default Contacto;

**Explicación del código**

1. **Estado (useState)**:
   * formulario: Almacena los valores de los campos del formulario.
   * errores: Almacena los mensajes de error para cada campo.
   * enviado: Indica si el formulario se envió correctamente.
2. **Validación**:
   * La función validarFormulario verifica que los campos no estén vacíos y que cumplan con los requisitos (por ejemplo, formato de correo electrónico y longitud mínima del mensaje).
3. **Manejo de eventos**:
   * manejarCambio: Actualiza el estado del formulario cuando el usuario escribe en los campos.
   * manejarEnvio: Valida el formulario y muestra errores si los hay. Si no hay errores, marca el formulario como enviado.
4. **Renderización**:
   * Muestra los mensajes de error debajo de cada campo si no cumplen con los requisitos.
   * Muestra un mensaje de éxito si el formulario se envió correctamente.

**Estilos para el formulario**

Agrega estos estilos en styles.css para mejorar la presentación:

.formulario-contacto {

max-width: 400px;

margin: 0 auto;

display: flex;

flex-direction: column;

gap: 15px;

}

.formulario-contacto label {

font-weight: bold;

}

.formulario-contacto input,

.formulario-contacto textarea {

width: 100%;

padding: 10px;

border: 1px solid #ddd;

border-radius: 5px;

}

.formulario-contacto button {

padding: 10px;

background-color: #4930cd;

color: white;

border: none;

border-radius: 5px;

cursor: pointer;

}

.formulario-contacto button:hover {

background-color: #3725a1;

}

.error {

color: red;

font-size: 0.9em;

}

.exito {

color: green;

font-size: 1em;

text-align: center;

}

**Próximos pasos**

1. **Probar el componente**: Ejecuta el proyecto con npm run dev y verifica que la validación funcione correctamente.
2. **Agregar funcionalidad adicional**:
   * ¿Te gustaría enviar los datos del formulario a una API o servidor?
   * ¿O prefieres agregar más campos al formulario?

¡Avísame cómo deseas proceder! 😊

Para convertir la página \*\*"Quiénes somos"\*\* en ReactJS y agregarla al menú de navegación, seguiremos estos pasos:

1. **Crear un componente para la página "Quiénes somos".**
2. **Crear un componente para el menú de navegación.**
3. **Configurar React Router para manejar la navegación entre las páginas.**

**Migración de la página "Quiénes somos" a ReactJS**

**Instalación de React Router**

Primero, asegúrate de instalar React Router para manejar la navegación:

npm install react-router-dom

**Código del componente QuienesSomos.jsx**

import React from 'react';

const QuienesSomos = () => {

return (

<main>

<article className="main-container">

<section className="section-container">

<h1>¿Quién es 2000s Style?</h1>

<br />

<h2>Bienvenida/o a 2000s STYLE en Línea</h2>

<br />

<p>

Nacimos en 1960 con el concepto de <b><i>moda rápida:</i></b> todo tipo de objetos personales asequibles y

listos para usar basados en los últimos diseños de alta gama.

</p>

<p>

Proveemos <b><i>lo mejor</i></b> de los objetos y accesorios que anticipan las tendencias futuras{' '}

<b><i>a precios alcanzables para todos</i></b>.

</p>

<p>¡Disfrute!</p>

<p>Diego Leonel Bochini</p>

<p>Presidente, SportsPro</p>

</section>

<div className="embed-container">

<iframe

src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d9288.92505041785!2d-58.39111493845885!3d-34.600850143177766!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x95bccac630121623%3A0x53386f2ac88991a9!2sTeatro%20Col%C3%B3n!5e0!3m2!1ses!2sar!4v1710158476038!5m2!1ses!2sar"

frameBorder="0"

allowFullScreen

title="Ubicación"

></iframe>

</div>

</article>

</main>

);

};

export default QuienesSomos;

**Código del componente Navbar.jsx**

import React from 'react';

import { Link } from 'react-router-dom';

const Navbar = () => {

return (

<header>

<nav className="navbar navbar-expand-lg navbar-light bg-light">

<div className="container-fluid">

{/\* Logo \*/}

<Link className="navbar-brand" to="/" style={{ color: 'var(--bg-colortext1)' }}>

2000s STYLE

</Link>

{/\* Botón para el menú colapsable \*/}

<button

className="navbar-toggler"

type="button"

data-bs-toggle="collapse"

data-bs-target="#navbarNav"

aria-controls="navbarNav"

aria-expanded="false"

aria-label="Toggle navigation"

>

<span className="navbar-toggler-icon"></span>

</button>

{/\* Enlaces del menú \*/}

<div className="collapse navbar-collapse" id="navbarNav">

<ul className="navbar-nav">

<li className="nav-item">

<Link className="nav-link" to="/" style={{ color: 'var(--bg-colortext1)' }}>

Inicio

</Link>

</li>

<li className="nav-item">

<Link className="nav-link" to="/quienes-somos" style={{ color: 'var(--bg-colortext1)' }}>

Quiénes somos

</Link>

</li>

<li className="nav-item">

<Link className="nav-link" to="/productos" style={{ color: 'var(--bg-colortext1)' }}>

Productos

</Link>

</li>

<li className="nav-item">

<Link className="nav-link" to="/resenias" style={{ color: 'var(--bg-colortext1)' }}>

Reseñas

</Link>

</li>

<li className="nav-item">

<Link className="nav-link" to="/contacto" style={{ color: 'var(--bg-colortext1)' }}>

Contacto

</Link>

</li>

</ul>

</div>

</div>

</nav>

<figure className="contenedorImagen">

<img id="imagen-index" src="/img/PersonalStyle-Banner.jpg" alt="Banner" />

</figure>

</header>

);

};

export default Navbar;

**Configuración de React Router en App.jsx**

import React from 'react';

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

import Navbar from './components/Navbar';

import QuienesSomos from './components/QuienesSomos';

import Productos from './components/Productos';

import Resenias from './components/Resenias';

import Contacto from './components/Contacto';

const App = () => {

return (

<Router>

<Navbar />

<Routes>

<Route path="/" element={<h1>Inicio</h1>} />

<Route path="/quienes-somos" element={<QuienesSomos />} />

<Route path="/productos" element={<Productos />} />

<Route path="/resenias" element={<Resenias />} />

<Route path="/contacto" element={<Contacto />} />

</Routes>

</Router>

);

};

export default App;

**Explicación del código**

1. **Componente QuienesSomos.jsx**:
   * Contiene el contenido de la página "Quiénes somos".
   * Incluye el texto y el iframe de Google Maps.
2. **Componente Navbar.jsx**:
   * Implementa el menú de navegación utilizando React Router.
   * Usa el componente Link para manejar la navegación entre las páginas.
3. **Configuración de React Router**:
   * En App.jsx, se define la estructura de las rutas con el componente Routes y Route.
   * Cada ruta apunta a un componente específico.

**Próximos pasos**

1. **Probar la navegación**: Ejecuta el proyecto con npm run dev y verifica que el menú de navegación funcione correctamente.
2. **Agregar estilos adicionales**: Si deseas personalizar el diseño, puedes modificar los estilos en styles.css.
3. **Agregar más páginas o funcionalidades**: ¿Te gustaría agregar más contenido o mejorar la experiencia del usuario?

¡Avísame cómo deseas proceder! 😊

Para lograr que el componente `

` no se muestre en la página de inicio (`/`), puedes usar el hook `useLocation` de `react-router-dom`. Este hook te permite acceder a la ruta actual y condicionar la renderización del `` en función de la misma.

**Solución: Usar useLocation para condicionar el renderizado del <Header />**

Aquí está el código actualizado:

import React from 'react';

import { BrowserRouter as Router, Routes, Route, useLocation } from 'react-router-dom';

import Navbar from './components/Navbar';

import Header from './components/Header';

import Footer from './components/Footer';

import Inicio from './components/Inicio';

import QuienesSomos from './components/QuienesSomos';

import Productos from './components/Productos';

import Resenias from './components/Resenias';

import Contacto from './components/Contacto';

const App = () => {

const location = useLocation(); // Hook para obtener la ruta actual

return (

<>

<Navbar />

{/\* Mostrar el Header solo si no estamos en la página de Inicio \*/}

{location.pathname !== '/' && <Header />}

<Routes>

<Route path="/" element={<Inicio />} />

<Route path="/quienes-somos" element={<QuienesSomos />} />

<Route path="/productos" element={<Productos />} />

<Route path="/resenias" element={<Resenias />} />

<Route path="/contacto" element={<Contacto />} />

</Routes>

<Footer />

</>

);

};

const AppWrapper = () => {

return (

<Router>

<App />

</Router>

);

};

export default AppWrapper;

**Explicación del código**

1. **Uso de useLocation**:
   * El hook useLocation de react-router-dom proporciona información sobre la ruta actual.
   * location.pathname contiene la ruta actual (por ejemplo, '/' para la página de inicio).
2. **Condicional para el <Header />**:
   * Se verifica si location.pathname es igual a '/'.
   * Si la ruta es '/', el <Header /> no se renderiza.
3. **Componente AppWrapper**:
   * Se utiliza un componente contenedor (AppWrapper) para envolver el componente App con el Router. Esto es necesario porque useLocation solo funciona dentro de un contexto de Router.

**Alternativa: Usar una lista de rutas donde no se muestre el <Header />**

Si en el futuro necesitas ocultar el <Header /> en más de una ruta, puedes usar una lista de rutas y verificar si la ruta actual está incluida en esa lista:

const App = () => {

const location = useLocation();

const rutasSinHeader = ['/']; // Lista de rutas donde no se muestra el Header

return (

<>

<Navbar />

{!rutasSinHeader.includes(location.pathname) && <Header />}

<Routes>

<Route path="/" element={<Inicio />} />

<Route path="/quienes-somos" element={<QuienesSomos />} />

<Route path="/productos" element={<Productos />} />

<Route path="/resenias" element={<Resenias />} />

<Route path="/contacto" element={<Contacto />} />

</Routes>

<Footer />

</>

);

};

**Próximos pasos**

1. **Probar la solución**:
   * Verifica que el <Header /> no se muestre en la página de inicio (/) pero sí en las demás rutas.
2. **Agregar más rutas a la lista** (si usas la alternativa):
   * Si necesitas ocultar el <Header /> en otras páginas, simplemente agrégalas a la lista rutasSinHeader.
3. **Personalizar el diseño**:
   * Si deseas que el espacio del <Header /> no afecte el diseño de la página de inicio, asegúrate de ajustar los estilos en tu archivo CSS.

¡Avísame si necesitas más ayuda! 😊

Para convertir el código original de `productos.js` en un componente React llamado `Productos.jsx` y mantener todas las funcionalidades y formatos actuales, podemos dividir el código en varias partes y usar el estado y los efectos de React para manejar la lógica. A continuación, te proporcionaré una implementación completa.

**Código de Productos.jsx**

import React, { useState, useEffect } from "react";

import "bootstrap/dist/css/bootstrap.min.css";

import "bootstrap/dist/js/bootstrap.bundle.min.js";

const Productos = () => {

const [productos, setProductos] = useState([]);

const [categorias, setCategorias] = useState([]);

const [filtroCategoria, setFiltroCategoria] = useState("all");

const [carrito, setCarrito] = useState(() => {

// Inicializar el carrito desde localStorage

return JSON.parse(localStorage.getItem("carrito")) || [];

});

const [modalDescripcion, setModalDescripcion] = useState("");

// Cargar productos desde la API

useEffect(() => {

fetch("https://dummyjson.com/products")

.then((response) => {

if (!response.ok) {

throw new Error("Error al cargar el JSON");

}

return response.json();

})

.then((data) => {

const productosFiltrados = data.products.filter(

(producto) => producto.category !== "groceries"

);

setProductos(productosFiltrados);

// Obtener categorías únicas

const categoriasUnicas = [

...new Set(productosFiltrados.map((producto) => producto.category)),

];

setCategorias(categoriasUnicas);

})

.catch((error) => console.error("Error al cargar los datos:", error));

}, []);

// Guardar el carrito en localStorage

useEffect(() => {

localStorage.setItem("carrito", JSON.stringify(carrito));

}, [carrito]);

// Actualizar la cantidad de un producto en el carrito

const actualizarCantidadCarrito = (id, cantidad) => {

setCarrito((prevCarrito) =>

prevCarrito.map((producto) =>

producto.id === id ? { ...producto, quantity: cantidad } : producto

)

);

};

// Eliminar un producto del carrito

const eliminarDelCarrito = (id) => {

setCarrito((prevCarrito) => prevCarrito.filter((producto) => producto.id !== id));

};

// Vaciar el carrito

const vaciarCarrito = () => {

setCarrito([]);

};

// Añadir un producto al carrito

const agregarAlCarrito = (producto, cantidad) => {

if (cantidad > 0) {

setCarrito((prevCarrito) => {

const productoEnCarrito = prevCarrito.find((item) => item.id === producto.id);

if (productoEnCarrito) {

return prevCarrito.map((item) =>

item.id === producto.id ? { ...item, quantity: cantidad } : item

);

} else {

return [...prevCarrito, { ...producto, quantity: cantidad }];

}

});

alert(`Añadido al carrito: ${producto.title} (x${cantidad})`);

} else {

alert("Por favor, ingresa una cantidad válida.");

}

};

// Filtrar productos por categoría

const productosFiltrados =

filtroCategoria === "all"

? productos

: productos.filter((producto) => producto.category === filtroCategoria);

return (

<div className="container mt-4">

{/\* Filtro de categorías \*/}

<div className="mb-4">

<label htmlFor="categoriaFiltro" className="form-label">

Seleccione la categoría o Todas:

</label>

<select

id="categoriaFiltro"

className="form-select"

value={filtroCategoria}

onChange={(e) => setFiltroCategoria(e.target.value)}

>

<option value="all">Todas</option>

{categorias.map((categoria) => (

<option key={categoria} value={categoria}>

{categoria}

</option>

))}

</select>

</div>

{/\* Lista de productos \*/}

<div className="row">

{productosFiltrados.map((producto) => (

<div key={producto.id} className="col-sm-12 col-md-6 col-lg-4 mb-4">

<div className="card h-100">

<img

src={producto.thumbnail}

className="card-img-top"

alt={producto.title}

style={{ cursor: "pointer" }}

onClick={() => setModalDescripcion(producto.description)}

data-bs-toggle="modal"

data-bs-target="#descripcionModal"

/>

<div className="card-body">

<p>{producto.title}</p>

<p className="card-text">

<strong>Precio:</strong> ${producto.price.toFixed(2)}

</p>

<div className="d-flex align-items-center">

<label htmlFor={`cantidadProducto${producto.id}`} className="me-2">

Cantidad:

</label>

<input

type="number"

id={`cantidadProducto${producto.id}`}

className="form-control me-2"

style={{ width: "120px" }}

min="1"

defaultValue="1"

/>

<button

className="btn btn-sm btn-primary flex-shrink-0"

style={{ width: "100px" }}

onClick={() =>

agregarAlCarrito(

producto,

parseInt(

document.getElementById(`cantidadProducto${producto.id}`).value,

10

)

)

}

>

Comprar

</button>

</div>

</div>

</div>

</div>

))}

</div>

{/\* Modal de descripción ampliada \*/}

<div

className="modal fade"

id="descripcionModal"

tabIndex="-1"

aria-labelledby="descripcionModalLabel"

aria-hidden="true"

>

<div className="modal-dialog">

<div className="modal-content">

<div className="modal-header">

<h5 className="modal-title" id="descripcionModalLabel">

Descripción Ampliada

</h5>

<button

type="button"

className="btn-close"

data-bs-dismiss="modal"

aria-label="Cerrar"

></button>

</div>

<div className="modal-body">

<p>{modalDescripcion}</p>

</div>

<div className="modal-footer">

<button

type="button"

className="btn btn-secondary"

data-bs-dismiss="modal"

>

Cerrar

</button>

</div>

</div>

</div>

</div>

{/\* Carrito de compras \*/}

<div className="mt-4">

<h3>Carrito de Compras</h3>

<table className="table">

<thead>

<tr>

<th>Producto</th>

<th>Cantidad</th>

<th>Precio</th>

<th>Subtotal</th>

<th>Acciones</th>

</tr>

</thead>

<tbody>

{carrito.map((producto) => (

<tr key={producto.id}>

<td>{producto.title}</td>

<td>

<input

type="number"

min="1"

value={producto.quantity}

className="form-control"

onChange={(e) =>

actualizarCantidadCarrito(producto.id, parseInt(e.target.value, 10))

}

/>

</td>

<td>${producto.price.toFixed(2)}</td>

<td>${(producto.price \* producto.quantity).toFixed(2)}</td>

<td>

<button

className="btn btn-danger btn-sm"

onClick={() => eliminarDelCarrito(producto.id)}

>

Eliminar

</button>

</td>

</tr>

))}

</tbody>

</table>

<button className="btn btn-warning" onClick={vaciarCarrito}>

Vaciar Carrito

</button>

</div>

</div>

);

};

export default Productos;

**Explicación**

1. **Estado y efectos**:
   * useState se usa para manejar el estado de los productos, categorías, carrito y descripción del modal.
   * useEffect se usa para cargar los datos desde la API y sincronizar el carrito con localStorage.
2. **Modal**:
   * Se utiliza Bootstrap para mostrar la descripción ampliada.
3. **Carrito**:
   * Se actualiza dinámicamente y se guarda en localStorage.
4. **Filtro**:
   * Permite filtrar productos por categoría.

**Próximos pasos**

1. Copia este código en tu archivo Productos.jsx.
2. Asegúrate de tener Bootstrap instalado en tu proyecto.
3. Prueba la funcionalidad y ajusta los estilos si es necesario.

¿Hay algo más que necesites ajustar o agregar? 😊

Para que el código de React sea lo más similar posible al código original que compartiste (que utiliza JavaScript puro), debemos realizar los siguientes cambios:

1. **Incorporar el filtro por categorías**: En lugar de filtrar por texto, implementaremos un filtro por categorías.
2. **Renderizar reseñas asociadas a productos**: En lugar de usar comentarios genéricos, cargaremos productos y sus reseñas desde la API.
3. **Generar estrellas para las calificaciones**: Implementaremos la función para mostrar estrellas según el rating.
4. **Excluir productos de la categoría "groceries"**: Filtraremos los productos para excluir esta categoría.
5. **Agregar un selector de categorías dinámico**: Permitiremos al usuario seleccionar una categoría para filtrar las reseñas.

Aquí está el código actualizado:

**Código actualizado de Resenias.jsx**

import React, { useState, useEffect } from 'react';

const Resenias = () => {

const [productos, setProductos] = useState([]); // Estado para almacenar los productos

const [categorias, setCategorias] = useState([]); // Estado para almacenar las categorías

const [filtroCategoria, setFiltroCategoria] = useState('all'); // Estado para manejar el filtro de categorías

const [loading, setLoading] = useState(true); // Estado para manejar el cargando

// Función para generar estrellas a partir del rating

const generarEstrellas = (rating) => '⭐'.repeat(Math.round(rating));

// Función para cargar los productos desde la API

useEffect(() => {

const fetchProductos = async () => {

try {

const response = await fetch('https://dummyjson.com/products');

if (!response.ok) {

throw new Error('Error al cargar los productos');

}

const data = await response.json();

// Excluir productos de la categoría "groceries"

const productosValidos = data.products.filter(

(producto) => producto.category !== 'groceries'

);

// Guardar los productos en el estado

setProductos(productosValidos);

// Obtener categorías únicas

const categoriasUnicas = [

...new Set(productosValidos.map((producto) => producto.category)),

];

setCategorias(categoriasUnicas);

setLoading(false);

} catch (error) {

console.error('Error al cargar los datos:', error);

setLoading(false);

}

};

fetchProductos();

}, []);

// Filtrar productos por categoría

const productosFiltrados =

filtroCategoria === 'all'

? productos

: productos.filter((producto) => producto.category === filtroCategoria);

return (

<div className="container mt-4">

<h1>Reseñas</h1>

{/\* Filtro de categorías \*/}

<div className="mb-4">

<label htmlFor="categoriaFiltro" className="form-label">

Filtrar por categoría:

</label>

<select

id="categoriaFiltro"

className="form-select"

value={filtroCategoria}

onChange={(e) => setFiltroCategoria(e.target.value)}

>

<option value="all">Todas</option>

{categorias.map((categoria) => (

<option key={categoria} value={categoria}>

{categoria}

</option>

))}

</select>

</div>

{loading ? (

<p>Cargando reseñas...</p>

) : (

<div className="reviews-grid">

{productosFiltrados.length > 0 ? (

productosFiltrados.map((producto) => (

<div key={producto.id} className="review card mb-4">

<div className="card-body">

{/\* Encabezado del producto \*/}

<img

src={producto.thumbnail}

alt={producto.title}

className="img-fluid mb-3"

/>

<p>{producto.title}</p>

</div>

{/\* Reseñas del producto \*/}

{producto.reviews && producto.reviews.length > 0 ? (

producto.reviews.map((resena, index) => (

<div key={index} className="card-footer">

<p>{generarEstrellas(resena.rating)}</p>

<p>{resena.comment}</p>

</div>

))

) : (

<div className="card-footer">

<p>No hay reseñas para este producto.</p>

</div>

)}

</div>

))

) : (

<p>No se encontraron productos en esta categoría.</p>

)}

</div>

)}

</div>

);

};

export default Resenias;

**Cambios realizados**

1. **Carga de productos y reseñas**:
   * Se utiliza la API https://dummyjson.com/products para cargar productos y sus reseñas.
   * Se excluyen los productos de la categoría "groceries".
2. **Filtro por categorías**:
   * Se implementa un selector dinámico para filtrar productos por categoría.
   * Se muestra la opción "Todas" para ver todos los productos.
3. **Renderización de reseñas**:
   * Cada producto muestra sus reseñas (si las tiene).
   * Si un producto no tiene reseñas, se muestra un mensaje indicando que no hay reseñas disponibles.
4. **Generación de estrellas**:
   * Se implementa la función generarEstrellas para mostrar estrellas según el rating de cada reseña.
5. **Manejo de estados**:
   * Se utiliza el estado loading para mostrar un mensaje de carga mientras se obtienen los datos.

**Resultado esperado**

* Al cargar la página, se mostrarán todos los productos (excepto los de la categoría "groceries") con sus reseñas.
* El usuario puede filtrar los productos por categoría utilizando el selector.
* Cada producto muestra su imagen, título y las reseñas asociadas (si las tiene).
* Las calificaciones de las reseñas se muestran como estrellas.

¿Hay algo más que te gustaría ajustar o agregar? 😊

Para mostrar los artículos con sus reseñas en tarjetas dispuestas una al lado de la otra, podemos utilizar un diseño basado en \*\*CSS Grid\*\* o \*\*Bootstrap Grid System\*\*. Esto permitirá que las tarjetas se alineen horizontalmente y se ajusten automáticamente según el tamaño de la pantalla.

Voy a ajustar el código para que las tarjetas se muestren en un diseño de cuadrícula utilizando **Bootstrap Grid System**, ya que es una solución común y flexible.

**Código actualizado de Resenias.jsx**

import React, { useState, useEffect } from 'react';

const Resenias = () => {

const [productos, setProductos] = useState([]); // Estado para almacenar los productos

const [categorias, setCategorias] = useState([]); // Estado para almacenar las categorías

const [filtroCategoria, setFiltroCategoria] = useState('all'); // Estado para manejar el filtro de categorías

const [loading, setLoading] = useState(true); // Estado para manejar el cargando

// Función para generar estrellas a partir del rating

const generarEstrellas = (rating) => '⭐'.repeat(Math.round(rating));

// Función para cargar los productos desde la API

useEffect(() => {

const fetchProductos = async () => {

try {

const response = await fetch('https://dummyjson.com/products');

if (!response.ok) {

throw new Error('Error al cargar los productos');

}

const data = await response.json();

// Excluir productos de la categoría "groceries"

const productosValidos = data.products.filter(

(producto) => producto.category !== 'groceries'

);

// Guardar los productos en el estado

setProductos(productosValidos);

// Obtener categorías únicas

const categoriasUnicas = [

...new Set(productosValidos.map((producto) => producto.category)),

];

setCategorias(categoriasUnicas);

setLoading(false);

} catch (error) {

console.error('Error al cargar los datos:', error);

setLoading(false);

}

};

fetchProductos();

}, []);

// Filtrar productos por categoría

const productosFiltrados =

filtroCategoria === 'all'

? productos

: productos.filter((producto) => producto.category === filtroCategoria);

return (

<div className="container mt-4">

<h1>Reseñas</h1>

{/\* Filtro de categorías \*/}

<div className="mb-4">

<label htmlFor="categoriaFiltro" className="form-label">

Filtrar por categoría:

</label>

<select

id="categoriaFiltro"

className="form-select"

value={filtroCategoria}

onChange={(e) => setFiltroCategoria(e.target.value)}

>

<option value="all">Todas</option>

{categorias.map((categoria) => (

<option key={categoria} value={categoria}>

{categoria}

</option>

))}

</select>

</div>

{loading ? (

<p>Cargando reseñas...</p>

) : (

<div className="row">

{productosFiltrados.length > 0 ? (

productosFiltrados.map((producto) => (

<div key={producto.id} className="col-sm-12 col-md-6 col-lg-4 mb-4">

<div className="card h-100">

<div className="card-body">

{/\* Encabezado del producto \*/}

<img

src={producto.thumbnail}

alt={producto.title}

className="img-fluid mb-3"

/>

<p className="fw-bold">{producto.title}</p>

</div>

{/\* Reseñas del producto \*/}

{producto.reviews && producto.reviews.length > 0 ? (

producto.reviews.map((resena, index) => (

<div key={index} className="card-footer">

<p>{generarEstrellas(resena.rating)}</p>

<p>{resena.comment}</p>

</div>

))

) : (

<div className="card-footer">

<p>No hay reseñas para este producto.</p>

</div>

)}

</div>

</div>

))

) : (

<p>No se encontraron productos en esta categoría.</p>

)}

</div>

)}

</div>

);

};

export default Resenias;

**Cambios realizados**

1. **Diseño de cuadrícula con Bootstrap**:
   * Se utiliza la clase row para el contenedor principal de las tarjetas.
   * Cada tarjeta está envuelta en un div con las clases col-sm-12 col-md-6 col-lg-4, lo que asegura que:
     + En pantallas pequeñas (sm), las tarjetas ocupen toda la fila.
     + En pantallas medianas (md), se muestren dos tarjetas por fila.
     + En pantallas grandes (lg), se muestren tres tarjetas por fila.
2. **Estilo de las tarjetas**:
   * Se utiliza la clase card para cada producto.
   * La clase h-100 asegura que todas las tarjetas tengan la misma altura, lo que mejora la alineación.
3. **Espaciado entre tarjetas**:
   * Se agrega la clase mb-4 a cada tarjeta para añadir un margen inferior y evitar que las tarjetas se amontonen.
4. **Imágenes y encabezados**:
   * Las imágenes de los productos se muestran en la parte superior de cada tarjeta.
   * El título del producto se muestra como un texto destacado (fw-bold).
5. **Reseñas dentro de las tarjetas**:
   * Las reseñas se muestran en el pie de la tarjeta (card-footer).
   * Si no hay reseñas, se muestra un mensaje indicando que no hay reseñas disponibles.

**Resultado esperado**

* Las tarjetas de productos se mostrarán en un diseño de cuadrícula.
* En pantallas grandes, se mostrarán tres tarjetas por fila; en pantallas medianas, dos; y en pantallas pequeñas, una.
* Cada tarjeta incluirá la imagen del producto, su título y las reseñas asociadas (si las tiene).
* Si no hay reseñas, se mostrará un mensaje en el pie de la tarjeta.

¿Es este el diseño que tenías en mente? 😊

**components/FiltroCategorias.js**

jsx

CopyEdit

const FiltroCategorias = ({ categorias, filtro, onChange }) => (

<div className="mb-4">

<label htmlFor="categoriaFiltro" className="form-label">Seleccione la categoría o Todas:</label>

<select

id="categoriaFiltro"

className="form-select"

value={filtro}

onChange={(e) => onChange(e.target.value)}

>

<option value="all">Todas</option>

{categorias.map((categoria) => (

<option key={categoria} value={categoria}>{categoria}</option>

))}

</select>

</div>

);

export default FiltroCategorias;

**✅ 2. components/ProductoCard.js**

jsx

CopyEdit

const ProductoCard = ({ producto, agregarAlCarrito, setModalDescripcion }) => {

const handleClick = () => {

const input = document.getElementById(`cantidadProducto${producto.id}`);

const cantidad = parseInt(input.value, 10);

agregarAlCarrito(producto, cantidad);

};

return (

<div className="col-sm-12 col-md-6 col-lg-4 mb-4">

<div className="card h-100">

<img

src={producto.thumbnail}

className="card-img-top"

alt={producto.title}

style={{ cursor: "pointer" }}

onClick={() => setModalDescripcion(producto.description)}

data-bs-toggle="modal"

data-bs-target="#descripcionModal"

/>

<div className="card-body">

<p>{producto.title}</p>

<p><strong>Precio:</strong> ${producto.price.toFixed(2)}</p>

<div className="d-flex align-items-center">

<label htmlFor={`cantidadProducto${producto.id}`} className="me-2">Cantidad:</label>

<input

type="number"

id={`cantidadProducto${producto.id}`}

className="form-control me-2"

style={{ width: "120px" }}

min="1"

defaultValue="1"

/>

<button className="btn btn-sm btn-primary" onClick={handleClick}>Comprar</button>

</div>

</div>

</div>

</div>

);

};

export default ProductoCard;

**✅ 3. components/ModalDescripcion.js**

jsx

CopyEdit

const ModalDescripcion = ({ descripcion }) => (

<div className="modal fade" id="descripcionModal" tabIndex="-1" aria-labelledby="descripcionModalLabel" aria-hidden="true">

<div className="modal-dialog">

<div className="modal-content">

<div className="modal-header">

<h5 className="modal-title" id="descripcionModalLabel">Descripción Ampliada</h5>

<button type="button" className="btn-close" data-bs-dismiss="modal" aria-label="Cerrar"></button>

</div>

<div className="modal-body">

<p>{descripcion}</p>

</div>

<div className="modal-footer">

<button type="button" className="btn btn-secondary" data-bs-dismiss="modal">Cerrar</button>

</div>

</div>

</div>

</div>

);

export default ModalDescripcion;

**✅ 4. components/ModalCarrito.js**

jsx

CopyEdit

const ModalCarrito = ({

carrito,

actualizarCantidadCarrito,

eliminarDelCarrito,

vaciarCarrito,

}) => {

const total = carrito.reduce(

(acc, prod) => acc + prod.price \* prod.quantity,

0

);

return (

<div className="modal fade" id="carritoModal" tabIndex="-1" aria-labelledby="carritoModalLabel" aria-hidden="true">

<div className="modal-dialog modal-lg">

<div className="modal-content">

<div className="modal-header">

<h5 className="modal-title" id="carritoModalLabel">Carrito de Compras</h5>

<button type="button" className="btn-close" data-bs-dismiss="modal" aria-label="Cerrar"></button>

</div>

<div className="modal-body">

{carrito.length > 0 ? (

<table className="table">

<thead>

<tr>

<th>Producto</th>

<th>Cantidad</th>

<th>Precio</th>

<th>Subtotal</th>

<th>Acciones</th>

</tr>

</thead>

<tbody>

{carrito.map((producto) => (

<tr key={producto.id}>

<td>{producto.title}</td>

<td>

<input

type="number"

min="1"

value={producto.quantity}

className="form-control"

onChange={(e) =>

actualizarCantidadCarrito(

producto.id,

parseInt(e.target.value, 10)

)

}

/>

</td>

<td>${producto.price.toFixed(2)}</td>

<td>${(producto.price \* producto.quantity).toFixed(2)}</td>

<td>

<button

className="btn btn-danger btn-sm"

onClick={() => eliminarDelCarrito(producto.id)}

>

Eliminar

</button>

</td>

</tr>

))}

</tbody>

</table>

) : (

<p className="text-center">El carrito está vacío.</p>

)}

</div>

<div className="modal-footer d-flex justify-content-between">

{carrito.length > 0 && (

<h5>Total: ${total.toFixed(2)}</h5>

)}

<div>

<button className="btn btn-warning me-2" onClick={vaciarCarrito}>Vaciar Carrito</button>

<button type="button" className="btn btn-secondary" data-bs-dismiss="modal">Cerrar</button>

</div>

</div>

</div>

</div>

</div>

);

};

export default ModalCarrito;

**✅ 5. Productos.js (componente principal)**

Este archivo ahora solo gestiona la lógica y renderiza los componentes:

jsx

CopyEdit

import React, { useState, useEffect } from "react";

import FiltroCategorias from "./components/FiltroCategorias";

import ProductoCard from "./components/ProductoCard";

import ModalDescripcion from "./components/ModalDescripcion";

import ModalCarrito from "./components/ModalCarrito";

const Productos = () => {

const [productos, setProductos] = useState([]);

const [categorias, setCategorias] = useState([]);

const [filtroCategoria, setFiltroCategoria] = useState("all");

const [carrito, setCarrito] = useState(() => JSON.parse(localStorage.getItem("carrito")) || []);

const [modalDescripcion, setModalDescripcion] = useState("");

useEffect(() => {

fetch("https://dummyjson.com/products")

.then((res) => res.json())

.then((data) => {

const filtrados = data.products.filter(p => p.category !== "groceries");

setProductos(filtrados);

setCategorias([...new Set(filtrados.map(p => p.category))]);

});

}, []);

useEffect(() => {

localStorage.setItem("carrito", JSON.stringify(carrito));

}, [carrito]);

const actualizarCantidadCarrito = (id, cantidad) => {

setCarrito(prev =>

prev.map(p => (p.id === id ? { ...p, quantity: cantidad } : p))

);

};

const eliminarDelCarrito = (id) => {

setCarrito(prev => prev.filter(p => p.id !== id));

};

const vaciarCarrito = () => setCarrito([]);

const agregarAlCarrito = (producto, cantidad) => {

if (cantidad > 0) {

setCarrito(prev => {

const existente = prev.find(item => item.id === producto.id);

return existente

? prev.map(item =>

item.id === producto.id ? { ...item, quantity: cantidad } : item

)

: [...prev, { ...producto, quantity: cantidad }];

});

}

};

const productosFiltrados = filtroCategoria === "all"

? productos

: productos.filter(p => p.category === filtroCategoria);

return (

<div className="container mt-4">

<FiltroCategorias

categorias={categorias}

filtro={filtroCategoria}

onChange={setFiltroCategoria}

/>

<div className="row">

{productosFiltrados.map(producto => (

<ProductoCard

key={producto.id}

producto={producto}

agregarAlCarrito={agregarAlCarrito}

setModalDescripcion={setModalDescripcion}

/>

))}

</div>

<ModalDescripcion descripcion={modalDescripcion} />

<ModalCarrito

carrito={carrito}

actualizarCantidadCarrito={actualizarCantidadCarrito}

eliminarDelCarrito={eliminarDelCarrito}

vaciarCarrito={vaciarCarrito}

/>

</div>

);

};

export default Productos;

¿Querés que te lo empaquete todo en un ZIP para descargar, o te gustaría que te dé alguna mejora extra (como usar useRef en vez de document.getElementById, o usar context para el carrito)?

¡Buena pregunta! 👌 Actualmente tu carrito **se guarda en localStorage**, lo cual **persiste incluso si cierras el navegador o recargas la página**. Entonces, **no se vacía al salir de la aplicación**.

**🧹 ¿Querés que se vacíe al cerrar la pestaña o navegador?**

Podés lograr eso usando sessionStorage en lugar de localStorage. Te explico la diferencia:

| **Almacenamiento** | **¿Se borra al cerrar el navegador?** |
| --- | --- |
| localStorage | ❌ No |
| sessionStorage | ✅ Sí |

**🔄 Cambio sencillo para que el carrito NO persista**

Solo tenés que reemplazar localStorage por sessionStorage.

**Cambios en Productos.js:**

js

CopyEdit

const [carrito, setCarrito] = useState(() =>

JSON.parse(sessionStorage.getItem("carrito")) || []

);

useEffect(() => {

sessionStorage.setItem("carrito", JSON.stringify(carrito));

}, [carrito]);

**✅ Resultado:**

Con ese cambio, el carrito:

* **Se conserva mientras la pestaña está abierta**
* **Se borra automáticamente cuando cerrás la pestaña o navegador**

¡Sí, totalmente! 💥 Separar el modal como componente mejora mucho la organización del código y hace que sea más reutilizable y limpio. Te muestro cómo hacerlo paso a paso:

**✅ Paso 1: Crear el componente ModalConfirmacion.js**

jsx

CopyEdit

import React, { useEffect } from "react";

const ModalConfirmacion = ({ producto }) => {

useEffect(() => {

if (producto) {

const modal = new window.bootstrap.Modal(document.getElementById("confirmacionModal"));

modal.show();

}

}, [producto]);

return (

<div

className="modal fade"

id="confirmacionModal"

tabIndex="-1"

aria-labelledby="confirmacionModalLabel"

aria-hidden="true"

>

<div className="modal-dialog">

<div className="modal-content">

<div className="modal-header">

<h5 className="modal-title" id="confirmacionModalLabel">Producto agregado</h5>

<button type="button" className="btn-close" data-bs-dismiss="modal" aria-label="Cerrar"></button>

</div>

<div className="modal-body">

<p className="mb-2">✅ El producto fue agregado al carrito.</p>

{producto && (

<>

<p><strong>{producto.title}</strong></p>

<p className="text-muted">{producto.description}</p>

</>

)}

</div>

<div className="modal-footer">

<button type="button" className="btn btn-secondary" data-bs-dismiss="modal">

Cerrar

</button>

</div>

</div>

</div>

</div>

);

};

export default ModalConfirmacion;

**✅ Paso 2: En Productos.js**

1. Importalo arriba:

js

CopyEdit

import ModalConfirmacion from "./ModalConfirmacion";

1. Agregá un nuevo useState para manejar el producto recientemente agregado:

js

CopyEdit

const [productoAgregado, setProductoAgregado] = useState(null);

1. Pasá la función setProductoAgregado a ProductoCard:

jsx

CopyEdit

<ProductoCard

key={producto.id}

producto={producto}

agregarAlCarrito={agregarAlCarrito}

setModalDescripcion={setModalDescripcion}

setProductoAgregado={setProductoAgregado}

/>

1. Agregá el componente del modal al final del return:

jsx

CopyEdit

<ModalConfirmacion producto={productoAgregado} />

**✅ Paso 3: Modificá ProductoCard.js**

Agregá setProductoAgregado como prop y modificá handleClick:

js

CopyEdit

const ProductoCard = ({ producto, agregarAlCarrito, setModalDescripcion, setProductoAgregado }) => {

const handleClick = () => {

const input = document.getElementById(`cantidadProducto${producto.id}`);

const cantidad = parseInt(input.value, 10);

if (cantidad > 0) {

agregarAlCarrito(producto, cantidad);

setProductoAgregado(producto); // Esto activa el modal

} else {

alert("Por favor, ingresa una cantidad válida.");

}

};

**✅ Resultado**

* Al hacer clic en **Comprar**, se agrega el producto al carrito.
* Se muestra el modal con nombre y descripción.
* Todo modularizado y limpio 💅

¿Querés que el modal desaparezca automáticamente después de unos segundos también?

4o

**You said:**

me podria dar el codigo de producto.js modificado con lo anterior? este es el codigo actual: import React, { useState, useEffect } from "react"; import FiltroCategorias from "../components/FiltroCategorias"; import ProductoCard from "../components/ProductoCard"; import ModalDescripcion from "../components/ModalDescripcion"; import ModalCarrito from "../components/ModalCarrito"; const Productos = () => { const [productos, setProductos] = useState([]); const [categorias, setCategorias] = useState([]); const [filtroCategoria, setFiltroCategoria] = useState("all"); const [carrito, setCarrito] = useState(() => { return JSON.parse(sessionStorage.getItem("carrito")) || []; }); const [modalDescripcion, setModalDescripcion] = useState(""); useEffect(() => { fetch("https://dummyjson.com/products") .then((res) => res.json()) .then((data) => { const filtrados = data.products.filter(p => p.category !== "groceries"); setProductos(filtrados); setCategorias([...new Set(filtrados.map(p => p.category))]); }); }, []); useEffect(() => { sessionStorage.setItem("carrito", JSON.stringify(carrito)); }, [carrito]); const actualizarCantidadCarrito = (id, cantidad) => { setCarrito(prev => prev.map(p => (p.id === id ? { ...p, quantity: cantidad } : p)) ); }; const eliminarDelCarrito = (id) => { setCarrito(prev => prev.filter(p => p.id !== id)); }; const vaciarCarrito = () => setCarrito([]); const agregarAlCarrito = (producto, cantidad) => { if (cantidad > 0) { setCarrito(prev => { const existente = prev.find(item => item.id === producto.id); return existente ? prev.map(item => item.id === producto.id ? { ...item, quantity: cantidad } : item ) : [...prev, { ...producto, quantity: cantidad }]; }); } }; const productosFiltrados = filtroCategoria === "all" ? productos : productos.filter(p => p.category === filtroCategoria); return ( <div className="container mt-4"> <FiltroCategorias categorias={categorias} filtro={filtroCategoria} onChange={setFiltroCategoria} /> <div className="row"> {productosFiltrados.map(producto => ( <ProductoCard key={producto.id} producto={producto} agregarAlCarrito={agregarAlCarrito} setModalDescripcion={setModalDescripcion} /> ))} </div> <ModalDescripcion descripcion={modalDescripcion} /> <ModalCarrito carrito={carrito} actualizarCantidadCarrito={actualizarCantidadCarrito} eliminarDelCarrito={eliminarDelCarrito} vaciarCarrito={vaciarCarrito} /> </div> ); }; export default Productos;

**ChatGPT said:**

¡Perfecto! A continuación te dejo el código actualizado de Productos.js con el modal de confirmación integrado como componente separado (ModalConfirmacion). También verás cómo se pasa setProductoAgregado a ProductoCard.

**✅ Código modificado de Productos.js**

jsx

CopyEdit

import React, { useState, useEffect } from "react";

import FiltroCategorias from "../components/FiltroCategorias";

import ProductoCard from "../components/ProductoCard";

import ModalDescripcion from "../components/ModalDescripcion";

import ModalCarrito from "../components/ModalCarrito";

import ModalConfirmacion from "../components/ModalConfirmacion"; // nuevo modal

const Productos = () => {

const [productos, setProductos] = useState([]);

const [categorias, setCategorias] = useState([]);

const [filtroCategoria, setFiltroCategoria] = useState("all");

const [carrito, setCarrito] = useState(() => {

return JSON.parse(sessionStorage.getItem("carrito")) || [];

});

const [modalDescripcion, setModalDescripcion] = useState("");

const [productoAgregado, setProductoAgregado] = useState(null); // nuevo estado

useEffect(() => {

fetch("https://dummyjson.com/products")

.then((res) => res.json())

.then((data) => {

const filtrados = data.products.filter(p => p.category !== "groceries");

setProductos(filtrados);

setCategorias([...new Set(filtrados.map(p => p.category))]);

});

}, []);

useEffect(() => {

sessionStorage.setItem("carrito", JSON.stringify(carrito));

}, [carrito]);

const actualizarCantidadCarrito = (id, cantidad) => {

setCarrito(prev =>

prev.map(p => (p.id === id ? { ...p, quantity: cantidad } : p))

);

};

const eliminarDelCarrito = (id) => {

setCarrito(prev => prev.filter(p => p.id !== id));

};

const vaciarCarrito = () => setCarrito([]);

const agregarAlCarrito = (producto, cantidad) => {

if (cantidad > 0) {

setCarrito(prev => {

const existente = prev.find(item => item.id === producto.id);

return existente

? prev.map(item =>

item.id === producto.id ? { ...item, quantity: cantidad } : item

)

: [...prev, { ...producto, quantity: cantidad }];

});

setProductoAgregado(producto); // mostrar modal de confirmación

}

};

const productosFiltrados = filtroCategoria === "all"

? productos

: productos.filter(p => p.category === filtroCategoria);

return (

<div className="container mt-4">

<FiltroCategorias

categorias={categorias}

filtro={filtroCategoria}

onChange={setFiltroCategoria}

/>

<div className="row">

{productosFiltrados.map(producto => (

<ProductoCard

key={producto.id}

producto={producto}

agregarAlCarrito={agregarAlCarrito}

setModalDescripcion={setModalDescripcion}

setProductoAgregado={setProductoAgregado} // nuevo prop

/>

))}

</div>

<ModalDescripcion descripcion={modalDescripcion} />

<ModalCarrito

carrito={carrito}

actualizarCantidadCarrito={actualizarCantidadCarrito}

eliminarDelCarrito={eliminarDelCarrito}

vaciarCarrito={vaciarCarrito}

/>

<ModalConfirmacion producto={productoAgregado} />

</div>

);

};

export default Productos;

Con eso, solo te falta tener el componente ModalConfirmacion.js como te lo pasé antes, y listo 🎉

¿Querés que el modal se cierre solo después de unos segundos también?