

Documentación del Componente Header

El componente **Header** representa la cabecera principal de la página web. Su función es ofrecer navegación, mostrar el estado del carrito de compras, permitir la búsqueda de productos y proporcionar accesos rápidos hacia secciones clave de la plataforma.

Importaciones

```
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
import { faCartShopping, faCircleUser } from "@fortawesome/free-solid-svg-icons";
import { Link } from "react-router-dom";
import { CarritoContext } from "../context/CarritoContext";
import { BusquedaContext } from "../context/BusquedaContext";
import { useContext } from "react";
```

- ② **FontAwesomeIcon**: Se utiliza para renderizar los íconos dentro del encabezado.
- ② **faCartShopping / faCircleUser**: Íconos importados desde FontAwesome para representar el carrito y el perfil de usuario.
- ② **Link**: Componente de React Router que permite navegar sin recargar la página.
- ② **CarritoContext**: Proporciona acceso al estado global del carrito de compras.
- ② **BusquedaContext**: Permite actualizar el término de búsqueda ingresado por el usuario.
- ② **useContext**: Hook necesario para consumir valores de los dos contextos.

Obtención de Datos desde los Contextos

```
const { carrito } = useContext(CarritoContext);
const { setTermino } = useContext(BusquedaContext);
```

- ② **carrito**: Contiene todos los productos que el usuario ha agregado.
- ② **setTermino**: Función para actualizar el término de búsqueda compartido en la aplicación.

Cálculo de la Cantidad Total de Productos

```
const totalItems = carrito.reduce((acc, p) => acc + p.cantidad, 0);
```

- ② Se utiliza el método **reduce()** para calcular el total de productos dentro del carrito.

- ② Se suman todas las cantidades almacenadas en cada elemento del arreglo carrito.

Estructura General del Header

El componente retorna un encabezado formado por tres áreas principales:

1. Sección de Promoción

```
<div className="box-promocion">
  <p>
    Regístrate y obtén un 20% en tu primer compra. Regístrate ahora
  </p>
</div>
```

- ② Muestra un mensaje promocional en la parte superior del sitio.

- ② Es un texto estático, pensado para captar la atención del usuario.

2. Barra de Navegación Principal

```
<nav>
  <Link to="/">
    
  </Link>
```

- Contiene el logotipo, que sirve como acceso directo al inicio mediante un Link.

3. Menú de Navegación

```
<ul>
  <li><Link to="/">Inicio</Link></li>
  <li><Link to="/categoria">Productos</Link></li>
</ul>
```

- Proporciona enlaces rápidos hacia la página principal y hacia la sección de productos.

4. Barra de Búsqueda

```
<input  
  className="search"  
  type="search"  
  placeholder="Buscar productos..."  
  onChange={(e) => setTermino(e.target.value)}  
/>
```

- ② Permite realizar búsquedas en tiempo real.
- ② Cada vez que el usuario escribe, se actualiza el término de búsqueda mediante setTermino.

5. Iconos de Carrito y Perfil

```
<div className="box-icons">  
  <Link to="/carrito" className="box-contador">  
    <FontAwesomeIcon icon={faCartShopping} />  
    {totalItems > 0 && <span className="contador-carrito">{totalItems}</span>}  
  </Link>  
  
  <Link to="Perfil"><FontAwesomeIcon icon={faCircleUser} /></Link>  
</div>
```

② Carrito

- Muestra el ícono del carrito.
- Si existen productos, se muestra un contador con el número total.

② Perfil del Usuario

- Enlace directo a la sección de perfil.

Consideraciones Finales

1. El componente utiliza **React Router**, por lo que los elementos <Link> reemplazan las etiquetas <a> para evitar recargas completas.
2. El manejo del carrito y de la búsqueda se realiza mediante **Context API**, lo cual hace que estos valores sean accesibles en toda la aplicación.
3. La estructura general del header está pensada para que permanezca visible y funcional en todas las páginas del sitio.

Este componente corresponde al **Footer** de la página web. Su función principal es ofrecer información de contacto, enlaces importantes, suscripción al boletín de noticias y mostrar los métodos de pago disponibles. Está construido con **React** y utiliza **FontAwesome** para los íconos de las tarjetas de pago.

```
import { faCcApplePay, faCcMastercard, faCcPaypal, faCcVisa } from "@fortawesome/free-brands-svg-icons";
import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";
```

- ② **faCcApplePay, faCcMastercard, faCcPaypal, faCcVisa:** Se importan los íconos de marcas de pago de FontAwesome para mostrarlos en el footer.
- ② **FontAwesomeIcon:** Componente de FontAwesome que permite renderizar los íconos importados en React.

Definición del Componente

```
export default function Footer() {
  return (
    <div>
```

- ② Se define una función de React que exporta el componente Footer.
- ② Este componente no recibe props y se encarga de renderizar todo el contenido del pie de página.

Estructura del Footer

1. Sección de Suscripción

```
<div className="box-suscribete">
  <h3>Mantente al dia con <br/>las ultimas ofertas</h3>
  <form className="form-suscribete">
    <input type='email' name='email' placeholder='Ingresa tu correo electronico'/>
    <input type='submit' value="Susribete"/>
  </form>
</div>
```

- ② Contiene un **encabezado** invitando a los usuarios a mantenerse al día con ofertas.
- ② Incluye un **formulario** con:
 - Campo de correo electrónico (`input type='email'`) para suscribirse al boletín.
 - Botón de envío (`input type='submit'`) para completar la suscripción.

2. Sección de Enlaces del Footer

```
<div className="box-footer">
  <!-- Varias columnas con enlaces -->
</div>
```

El footer está organizado en **5 columnas**, cada una con enlaces relevantes:

1. Primera columna

- Logo de la empresa.
- Enlaces a términos de uso y política de privacidad.

2. Segunda columna (Empresa)

- Información sobre la empresa: Sobre nosotros, Características, Trabajo, Carrera.

3. Tercera columna (Ayuda)

- Enlaces de soporte y detalles legales: Soporte, Envíos, Términos y condiciones, Política y privacidad.

4. Cuarta columna (FAQ)

- Preguntas frecuentes relacionadas con la cuenta, envíos, órdenes y pagos.

5. Quinta columna (Recursos)

- Enlaces a recursos adicionales: Catálogo, Canal de YouTube, Blog.

3. Sección de Derechos y Métodos de Pago

```
<div className="box-derecho">
  <p>©2025 Grupo 4 | Derechos reservados</p>
  <div className="box-tiposPagos">
    <FontAwesomeIcon icon={faCcMastercard}/>
    <FontAwesomeIcon icon={faCcVisa}/>
    <FontAwesomeIcon icon={faCcPaypal}/>
    <FontAwesomeIcon icon={faCcApplePay}/>
  </div>
</div>
```

- ② Contiene un mensaje de copyright.
- ② Muestra los métodos de pago aceptados, usando los íconos de FontAwesome para Mastercard, Visa, PayPal y Apple Pay.

Observaciones Importantes

1. Uso de className:
Todas las clases CSS utilizan className en lugar de class, siguiendo la sintaxis de React.
2. Formulario de suscripción:
Actualmente el formulario no tiene lógica de envío; se puede integrar más adelante usando state o librerías de manejo de formularios.
3. Enlaces:
Todos los enlaces usan href="#" como marcador de posición; deben actualizarse con las rutas reales de la aplicación.
4. Responsabilidad del Componente:
Este componente se encarga exclusivamente de la presentación del footer y no maneja lógica compleja ni estados.

Documentación del Componente NewArrivals

El componente **NewArrivals** se encarga de mostrar una sección destacada con los productos más recientes que han sido agregados al catálogo. Su objetivo principal es presentar visualmente los nuevos ingresos para captar la atención del usuario y redirigirlo a la vista detallada de cada producto o a la página general de categorías.

Importaciones

```
import { obtenerProductos } from "../api/productos";
import { useState, useEffect } from "react";
import { Link } from "react-router-dom";
```

- ② **obtenerProductos:** Función encargada de obtener la lista de productos desde la API definida en el proyecto.
- ② **useState:** Hook de React utilizado para almacenar el listado de productos en el estado local.
- ② **useEffect:** Permite ejecutar la petición de productos cuando el componente se monta.

🔗 **Link**: Componente de React Router para navegar entre vistas sin recargar la página.

Estado y Efecto Principal

```
const [productos, setProductos] = useState([]);  
useEffect(()=> {  
  obtenerProductos().then(setProductos);  
}, []);
```

- Se declara un estado llamado **productos**, que inicia como un arreglo vacío.
- Dentro del **useEffect**, se ejecuta obtenerProductos() solo una vez al cargar el componente.
- Cuando la promesa se resuelve, la lista de productos obtenida se almacena en el estado mediante setProductos.

Este mecanismo permite que el componente se renderice con los datos provenientes del backend.

Renderización del Contenido

El componente estructura la sección de nuevos ingresos dentro de un elemento <article>.

1. Título de la sección

```
<h2>Nuevo Ingreso</h2>
```

2. Muestra de Productos

```
{productos.slice(0,4).map((p)=>(
  <Link className="link-newArrivals" to={`/producto/${p.id_producto}`}>
    <div key={p.id} className="tarjeta-newArrivals">
      <div className="box-img-newArrivals">
        <img className="img-newArrivals" src={p.imagen} alt={p.nombre}/>
      </div>
      <div className="box-info-newArrivals">
        <h3>{p.nombre}</h3>
        <p>{p.descripcion}</p>
        <span>Q{p.precio}</span>
      </div>
    </div>
  </Link>
))}
```

Esta sección realiza las siguientes funciones:

1. Limitación de productos mostrados

Se utiliza slice(0,4) para mostrar únicamente los primeros cuatro productos obtenidos desde la API.

2. Generación dinámica de tarjetas

Por cada producto se crea una tarjeta con:

- Imagen del producto.
- Nombre.
- Descripción.
- Precio.

3. Enlace a la vista individual

Cada tarjeta está envuelta en un Link que dirige a la ruta /producto/{id_producto}, lo cual permite acceder al detalle del producto específico.

4. Estructura visual y clases CSS

Las clases definidas (.tarjeta-newArrivals, .box-img-newArrivals, .box-info-newArrivals, etc.) permiten dar estilo y mejorar la presentación visual.

Botón para ver todos los productos

```
<div className="box-btn-newArrivals">
  <Link to={'/categoria'}>
    <button className="btn-newArrivals">Ver Todo</button>
  </Link>
</div>
```

- ② Esta sección incluye un botón que redirige al usuario a la página general de categorías.
- ② Su función es permitir que el usuario explore el catálogo completo más allá de los cuatro productos mostrados inicialmente.

Consideraciones Finales

1. El componente depende completamente de la API proporcionada por obtenerProductos, por lo que la correcta estructura de los datos es fundamental.
2. La limitación a cuatro elementos mejora la estética y evita saturación en la página principal.
3. Está diseñado como un componente totalmente presentacional, siendo su única lógica la obtención inicial de datos.
4. Utiliza React Router para una navegación fluida sin recarga de página.
5. La estructura del JSX está organizada para favorecer la lectura, el mantenimiento y la extensión del componente.

Documentación del Componente Productos

El componente **Productos** tiene como función principal mostrar el catálogo de productos disponible en la tienda, permitiendo aplicar filtros avanzados y realizar paginación. Asimismo, incluye la opción de agregar elementos al carrito mediante el uso de un contexto global.

```
import { obtenerProductos } from "../api/productos";
import { useEffect, useState, useContext } from "react";
import { Link } from "react-router-dom";
import { CarritoContext } from "../context/CarritoContext";
```

- **obtenerProductos**: Función encargada de consumir la API y obtener todos los productos almacenados en la base de datos.
- **useState / useEffect**: Hooks utilizados para manejar el estado del componente y ejecutar operaciones al montar la vista.
- **useContext**: Permite acceder a funciones del contexto global, como el manejo del carrito.
- **Link**: Herramienta de navegación proporcionada por React Router.
- **CarritoContext**: Proporciona acceso a la función agregarAlCarrito.

Estado Inicial y Contexto

```
const [productos, setProductos] = useState([]);
const { agregarAlCarrito } = useContext(CarritoContext);
```

- **productos** almacena la lista completa de productos obtenidos desde la API.
- **agregarAlCarrito** permite añadir un producto al carrito desde cualquier tarjeta.

Carga de Productos Desde la API

```
useEffect(() => {
  obtenerProductos().then(setProductos);
}, []);
```

- Se ejecuta una única vez cuando el componente se monta.
- obtenerProductos() retorna la lista completa de productos y se almacena en el estado.
- Esto permite que la vista se renderice dinámicamente con los datos reales.

Filtrado de Productos

El componente recibe un objeto **filtros** como propiedad, con diversas condiciones que el usuario selecciona desde la interfaz.

```
const productosFiltrados = productos.filter(p => {
  if (filtros.categoría && String(p.id_categoria) !== filtros.categoría) return false;
  if (filtros.tipo_prenda && p.tipo_prenda !== filtros.tipo_prenda) return false;
  if (filtros.estilo && p.estilo !== filtros.estilo) return false;
  if (filtros.color && p.color !== filtros.color) return false;
  if (filtros.talla && p.talla !== filtros.talla) return false;
  if (p.precio < filtros.minPrice || p.precio > filtros.maxPrice) return false;

  if (filtros.busqueda && !p.nombre.toLowerCase().includes(filtros.busqueda)) {
    return false;
  }

  return true;
});
```

El filtrado evalúa:

1. **Categoría del producto**
2. **Tipo de prenda**
3. **Estilo**
4. **Color**
5. **Talla**
6. **Rango de precios**
7. **Término de búsqueda** (comparación con el nombre en minúsculas)

Si alguno de los filtros no coincide con el producto, este se excluye.

Sistema de Paginación

El componente implementa paginación para mejorar la experiencia de usuario, mostrando únicamente nueve productos por página.

```
const productosPorPagina = 9;
const [paginaActual, setpaginaActual] = useState(1);

const totalPaginas = Math.ceil(productosFiltrados.length / productosPorPagina);
```

② **productosPorPagina** define cuántos elementos se muestran por página.

- ❑ **paginaActual** almacena el número de la página actual.
- ❑ **totalPaginas** calcula cuántas páginas se necesitan en total.

Cálculo de los índices

```
const indiceInicial = (paginaActual - 1) * productosPorPagina;
const indiceFinal = indiceInicial + productosPorPagina;

const productosPagina = productosFiltrados.slice(indiceInicial, indiceFinal);
```

- Se determina el rango de productos que deben mostrarse en cada página.
- slice extrae únicamente los elementos correspondientes.

Función de cambio de página

```
const cambiarPagina = (num) => {
  if (num < 1 || num > totalPaginas) return;
  setPaginaActual(num);
};
```

- ❑ Controla el avance entre páginas.
- ❑ Evita números fuera de rango.

Renderización de Productos

Los productos visibles según la página seleccionada se recorren mediante map.

```
{productosPagina.map(p => (
  <div key={p.id_producto} className="tarjeta-newArrivals">
```

Cada tarjeta incluye:

1. Imagen y enlace al detalle del producto

```
<Link to={`/producto/${p.id_producto}`}>
  <img src={p.imagen} alt={p.nombre}/>
</Link>
```

2. Información del producto

```
<h3>{p.nombre}</h3>  
<p>{p.descripcion}</p>  
<span>Q{p.precio}</span>
```

3. Botón para agregar al carrito

```
<button className="add-produ" onClick={() => agregarAlCarrito(p)}>  
  Agregar al carrito  
</button>
```

- Llama a agregarAlCarrito, enviando el producto completo como parámetro.

Sistema de Navegación por Páginas

```
<div className="pagination">  
  
  <a className="prev" ... > Anterior </a>  
  
  {[...Array(totalPaginas)].map((_, i) => (  
    <a key={i} className={`page-num ${paginaActual === i + 1 ? "active" : ""}`}>  
      {i + 1}  
    </a>  
  ))}  
  
  <a className="next" ... > Siguiente </a>  
  
</div>
```

Incluye:

1. Botón para ir a la página anterior.
2. Botones numerados para cada página.
3. Botón para avanzar a la página siguiente.

Además, se deshabilitan los botones cuando corresponde, tanto visual como funcionalmente.

Consideraciones Finales

1. El componente es altamente dinámico, adaptándose a filtros, cantidad de productos, y métodos de navegación.
2. Se integra completamente con el **CarritoContext**, permitiendo agregar elementos sin recargar la página.
3. La estructura permite escalar añadiendo nuevos filtros o ajustando el sistema de paginación.
4. Utiliza React Router para realizar navegación interna entre vistas de producto.
5. Está diseñado para ser reutilizable, recibiendo filtros como propiedades externas.

Descripción General

El componente **Reviews** se encarga de gestionar y mostrar las reseñas asociadas a un producto específico. Permite obtener reseñas desde la API, mostrarlas en pantalla y registrar nuevas reseñas por parte de usuarios autenticados. Este componente implementa gestión de estado local, llamadas a la API, validación de sesión de usuario y renderizado dinámico.

Importaciones

```
import { useState, useEffect, useContext } from "react";
import { AuthContext } from "../context/AuthContext";
```

② **useState**: Manejo de estados locales del componente.

② **useEffect**: Ejecución de efectos secundarios, principalmente la carga inicial de reseñas.

② **useContext**: Acceso al contexto global de autenticación.

② **AuthContext**: Proporciona información del usuario autenticado.

Definición del Componente

```
export default function Reviews({ id_producto }) {
```

Componente funcional que recibe como propiedad el **id_producto**, utilizado para consultar y registrar reseñas relacionadas al producto seleccionado.

Configuración Inicial

```
const API_URL = process.env.REACT_APP_API_URL;
```

- Define la URL base de la API, obtenida desde las variables de entorno del proyecto.

```
const { usuario } = useContext(AuthContext);
```

- Obtiene la información del usuario autenticado, necesaria para validar el envío de reseñas.

Estados del Componente

```
const [reviews, setReviews] = useState([]);  
const [comentario, setComentario] = useState("");  
const [calificacion, setCalificacion] = useState(5);
```

- **reviews**: Almacena la lista de reseñas recuperadas desde la API.
- **comentario**: Contiene el texto ingresado por el usuario.
- **calificacion**: Guarda el valor de calificación seleccionada (1–5).

Función: obtenerReviews

```
const obtenerReviews = async () => {  
  try {  
    const res = await fetch(` ${API_URL}/api_reviews.php?id_producto=${id_producto}`);  
    const data = await res.json();  
    setReviews(data);  
  } catch (err) {  
    console.error("Error cargando reseñas:", err);  
  }  
};
```

- ② Realiza una solicitud **GET** a la API para obtener las reseñas del producto.
- ② Actualiza el estado **reviews** con los datos recibidos.
- ② Maneja errores de conexión mediante `console.error`.

Carga Inicial de Reseñas

```
useEffect(() => {
  obtenerReviews();
}, [id_producto]);
```

- Ejecuta la función obtenerReviews cuando el componente se monta o cuando cambia el id_producto.

- Garantiza que las reseñas correspondan siempre al producto seleccionado.

Función: enviarReview

```
const enviarReview = async () => {
```

Función encargada de procesar y enviar una nueva reseña al servidor.

Validación del Usuario

```
if (!usuario) {
  alert("Debes iniciar sesión para dejar una reseña.");
  return;
}
```

- Evita que usuarios no autenticados envíen reseñas.

Preparación de Datos

```
const datos = {
  id_producto,
  id_usuario: usuario.id_usuario,
  calificacion,
  comentario
};
```

- Construye el objeto con la información necesaria para registrar la reseña.

Solicitud POST

```
const res = await fetch(` ${API_URL}/api_reviews.php`, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify(datos)
});
```

- Envía los datos al servidor en formato JSON mediante una solicitud POST.

Respuesta del Servidor

```
const data = await res.json();

if (data.success) {
  setComentario("");
  obtenerReviews();
} else {
  alert("Error al guardar reseña");
}
```

② Verifica si la operación fue exitosa.

③ Limpia el campo de comentario.

④ Recarga la lista de reseñas para reflejar el cambio.

Renderizado del Componente

El componente retorna una estructura compuesta por:

Listado de Reseñas

```
<div className="reviews-list">
  {reviews.length === 0 ? (
    <p>No hay reseñas todavía.</p>
  ) : (
    reviews.map(r => (
      <div key={r.id_review} className="review-tarjeta">
        <h3><strong>{r.nombre}</strong></h3>
        <p>{"★".repeat(r.calificacion)}</p>
        <p>{r.comentario}</p>
        <span>{r.fecha}</span>
      </div>
    )))
  )}
</div>
```

② Muestra un mensaje si no existen reseñas.

③ Renderiza cada reseña con nombre, calificación, comentario y fecha.

④ Genera estrellas dinámicamente mediante "★".repeat().

Formulario para Agregar Reseña

```
<div className="review-form">
  <h3>Agregar reseña</h3>
```

Calificación

```
<label>Calificación</label>
<select required value={calificacion} onChange={e => setCalificacion(e.target.value)}>
```

Comentario

```
<label>Comentario</label>
<textarea
  required
  rows="3"
  value={comentario}
  onChange={e => setComentario(e.target.value)}
></textarea>
```

El componente **Reviews** gestiona de forma completa el ciclo de vida de reseñas de un producto. Integra comunicación con la API, validación de usuario, almacenamiento de estado y renderizado dinámico, proporcionando una interfaz funcional para consultar y agregar reseñas dentro de la aplicación.

Documentación del Contexto CarritoContext

Descripción General

El módulo **CarritoContext** implementa un contexto global para la administración del carrito de compras dentro de la aplicación. Permite almacenar, actualizar y persistir productos agregados al carrito, así como exponer funciones para modificarlos desde cualquier componente. Utiliza Context API, useState y useEffect para gestionar estado global con persistencia en localStorage.

Importaciones

```
import { createContext, useState, useEffect } from "react";
```

② **createContext**: Crea un contexto para el manejo global del carrito.

② **useState**: Administra el estado local del carrito.

② **useEffect**: Sincroniza el estado del carrito con el almacenamiento local.

Creación del Contexto

```
export const CarritoContext = createContext();
```

Declaración del contexto que será utilizado por los componentes que necesiten acceder a los datos y funciones del carrito.

Definición del Provider

```
export function CarritoProvider({ children }) {
```

Componente proveedor encargado de envolver la aplicación y proporcionar acceso global al carrito y sus funciones.

Estado Inicial con Persistencia

```
const [carrito, setCarrito] = useState(() => {
  const almacenado = localStorage.getItem("carrito");
  return almacenado ? JSON.parse(almacenado) : [];
});
```

- Inicializa el carrito verificando si existe información previamente almacenada en **localStorage**.
- Si existe, se recupera; si no, se crea un arreglo vacío.
- Esto evita pérdida de datos al recargar la página.

Sincronización Automática con LocalStorage

```
useEffect(() => {
  localStorage.setItem("carrito", JSON.stringify(carrito));
}, [carrito]);
```

- ❑ Cada vez que el estado del carrito cambia, se actualiza el almacenamiento local.
- ❑ Mantiene persistencia continua entre sesiones.

Funciones del Carrito

Agregar Producto al Carrito

```

function agregarAlCarrito(producto) {
  setCarrito(prev => {
    const existe = prev.find(p => p.id_producto === producto.id_producto);

    if (existe) {
      return prev.map(p =>
        p.id_producto === producto.id_producto
          ? { ...p, cantidad: p.cantidad + 1 }
          : p
      );
    } else {
      return [...prev, { ...producto, cantidad: 1 }];
    }
  });
}

```

② Verifica si el producto ya existe en el carrito:

- **Si existe:** incrementa la cantidad.
- **Si no existe:** lo agrega con cantidad inicial de 1.

③ Utiliza el estado previo para asegurar actualizaciones correctas.

Reducir Cantidad

```

function reducirCantidad(id_producto) {
  setCarrito(prev =>
    prev.map(p =>
      p.id_producto === id_producto
        ? { ...p, cantidad: Math.max(p.cantidad - 1, 1) }
        : p
    )
  );
}

```

② Disminuye la cantidad de un producto sin permitir que baje de 1.

③ Evita que existan productos con cantidad 0 en el carrito.

Aumentar Cantidad

```
function aumentarCantidad(id_producto) {
  setCarrito(prev =>
    prev.map(p =>
      p.id_producto === id_producto
        ? { ...p, cantidad: p.cantidad + 1 }
        : p
    )
  );
}
```

- ② Incrementa la cantidad del producto especificado.
- ② Mantiene la estructura del carrito sin duplicar productos.

Eliminar Producto del Carrito

```
function eliminarDelCarrito(id_producto) {
  setCarrito(prev => prev.filter(p => p.id_producto !== id_producto));
}
```

Elimina completamente un producto del carrito según su ID.

Vaciar Carrito

```
function vaciarCarrito() {
  setCarrito([]);
  localStorage.removeItem("carrito");
}
```

- Limpia el carrito y elimina su registro en localStorage.
- Restaura al estado inicial vacío.

Exposición del Contexto

```
<CarritoContext.Provider value={{  
    carrito,  
    agregarAlCarrito,  
    eliminarDelCarrito,  
    vaciarCarrito,  
    reducirCantidad,  
    aumentarCantidad  
}}>  
    {children}  
</CarritoContext.Provider>
```

Proporciona a todos los componentes hijos acceso al:

- Estado global del carrito.
- Funciones de modificación

CarritoContext implementa un sistema completo de gestión del carrito con persistencia, funciones de modificación y acceso global. Garantiza que los componentes puedan interactuar con el carrito de manera consistente, evitando estados duplicados o pérdida de datos entre recargas.

Documentación del Componente Carrito.jsx

Descripción General

El componente **Carrito** corresponde a la página principal del carro de compras. Administra la visualización de los productos seleccionados por el usuario, los controles de edición de cantidades, el resumen del pedido y el proceso de creación del pedido previo al pago. Se integra con los contextos **CarritoContext**, **PedidoContext** y **AuthContext**, además de comunicarse con el servidor mediante API para generar pedidos.

Importaciones Principales

```
import '../estilos/carrito-productos.css';
import { useContext, useState } from 'react';
import { CarritoContext } from '../context/CarritoContext';
import { PedidoContext } from '../context/PedidoContext';
import { AuthContext } from '../context/AuthContext';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faTrash, faPlus, faMinus } from '@fortawesome/free-solid-svg-icons';
import { Link, useNavigate } from 'react-router-dom';
```

- ② **Hojas de estilo:** estilos específicos para la página de carrito.
- ② **useContext / useState:** gestión de estado y acceso a contextos globales.
- ② **Contextos:** importación de CarritoContext, PedidoContext y AuthContext.
- ② **FontAwesome:** iconos utilizados en controles de cantidad y eliminación.
- ② **react-router-dom:** navegación interna mediante Link y useNavigate.

Variables Iniciales

```
const API_URL = process.env.REACT_APP_API_URL;
```

URL base de la API, extraída de variables de entorno.

```
const { setIdPedido } = useContext(PedidoContext);
const { usuario } = useContext(AuthContext);
```

- Acceso al ID del pedido y datos del usuario autenticado.

```
const idUsuario = usuario ? Number(usuario.id_usuario) : null;
```

- Conversión del ID de usuario a número, o null si no hay sesión activa.

```
const navigate = useNavigate();
```

- Hook para redirigir al usuario hacia otras rutas.

```
const [mensaje, setMensaje] = useState("");
```

- Estado encargado de mostrar alertas temporales en la página.

```
const { carrito, eliminarDelCarrito, reducirCantidad, aumentarCantidad, vaciarCarrito } = useContext(CarritoContext);
```

- Acceso a los productos del carrito y sus funciones de gestión.

Cálculos del Pedido

```
const subtotal = carrito.reduce((acc, p) => acc + p.precio * p.cantidad, 0);
```

- Suma total de los precios multiplicados por sus cantidades.

```
const descuento = subtotal * 0.20;
```

- Descuento fijo aplicado (20%).

```
const envio = subtotal >= 1000 ? 0 : 30;
```

- Envío gratuito si el subtotal supera Q1000.

```
const total = subtotal - descuento + envio;
```

- Total general con descuento y envío incluido.

Función de Procesamiento de Pago

```

async function procesarPago() {
    try {
        if (carrito.length === 0) {
            setMensaje("Debe de seleccionar un producto");
            setTimeout(() => {
                navigate("/categoria");
            }, 2000);
            return;
        }

        if (!idUsuario) {
            setMensaje("Debes iniciar sesión primero");
            setTimeout(() => {
                navigate("/Login?redirect=Pago");
            }, 1500);
            return;
        }

        const response = await fetch(`${API_URL}/crearPedido.php`, {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({
                id_usuario: idUsuario,
                total: total
            })
        });

        const data = await response.json();

        if (data.status === "success") {
            setIdPedido(data.id_pedido);
            navigate("/Pago");
        } else {
            setMensaje("Error al crear el pedido");
        }
    } catch (error) {
        setMensaje("Error de conexión con el servidor");
    }
}

```

Responsabilidades:

1. Validar que el carrito no esté vacío.

2. Validar que el usuario haya iniciado sesión.
3. Enviar un **POST** hacia crearPedido.php con:
 - o id_usuario
 - o total
4. Recibir y procesar la respuesta de la API.
5. Guardar el id_pedido en el PedidoContext.
6. Redirigir al usuario hacia la página de pago.

Renderizado del Componente

Estructura Principal

```
<article className='cart-page'>
```

Contenedor general de la página del carrito.

Alertas Temporales

```
{mensaje && (  
    <div className="alerta-compra">  
        {mensaje}  
    </div>  
)}
```

- Se muestra cuando la variable mensaje contiene algún texto.

Listado de Productos

```
{carrito.map(p => (  
    <div key={p.id_producto} className="tarjeta-carrito">  
        ...  
    </div>  

```

- Se genera una tarjeta por cada producto del carrito.

Elementos de cada tarjeta

1. **Imagen con enlace al producto**
2. **Nombre, descripción y precio**
3. **Botón de eliminar**

4. Controles de cantidad

- Decrementar (faMinus)
- Mostrar cantidad
- Incrementar (faPlus)
-

componente Categoria.jsx

Imports

```
import { useState, useContext, useEffect } from "react";
import Productos from "../Components/Productos";
import { BusquedaContext } from "../context/BusquedaContext";
```

- ② Se importan hooks de React.
- ② Se importa el componente que mostrará los productos.
- ② Se obtiene el contexto donde viene el término de búsqueda.

Obtener el término de búsqueda

```
const { termino } = useContext(BusquedaContext);
```

- Extrae el texto que el usuario escribe en el buscador global.

Estado de filtros

```
const [filtros, setFiltros] = useState({
  categoria: "",
  tipo_prenda: "",
  estilo: "",
  talla: "",
  color: "",
  minPrice: 0,
  maxPrice: 9999,
  busqueda: ""
});
```

- Guarda todos los filtros que el usuario selecciona.
- busqueda se actualizará automáticamente cuando el usuario escriba algo en el buscador.

Función para actualizar un filtro

```
const actualizarFiltro = (campo, valor) => {
  setFiltros(prev => ({
    ...prev,
    [campo]: valor
  }));
};
```

- Toma el nombre del filtro y el valor, y actualiza solo ese campo sin perder lo demás.

useEffect para sincronizar el buscador

```
useEffect(() => {
  const seguro = typeof termino === "string" ? termino : "";

  setFiltros(prev => ({
    ...prev,
    busqueda: seguro.toLowerCase().trim()
  }));
}, [termino]);
```

- Cada vez que el usuario escriba algo en el buscador → se actualiza el filtro busqueda.
- Se limpia el texto (minúsculas + sin espacios extras).

Aside de filtros

Cada <select> actualiza un filtro:

Ejemplo:

```
<select onChange={({e}) => actualizarFiltro("categoria", e.target.value)}>
```

Lo mismo para:

- tipo_prenda
- estilo
- color
- talla
- precios (minPrice y maxPrice)

Todo actualiza el estado de filtros.

componente DetalleProducto.jsx

```
import NewArrivals from "../Components/NewArrivals";
import Reviews from "../Components/Reviews";
import { useEffect, useState, useContext } from "react";
import { useParams } from "react-router-dom";
import { obtenerProductosId } from "../api/productosID";
import { CarritoContext } from "../context/CarritoContext";
```

- ② Se importan componentes (NewArrivals y Reviews).
- ② Se traen hooks de React.
- ② useParams para obtener el **ID del producto desde la URL**.
- ② obtenerProductosId() para hacer una petición a la API.
- ② CarritoContext para poder agregar productos al carrito.

Obtener función del carrito

```
const { agregarAlCarrito } = useContext(CarritoContext);
```

- Permite agregar productos al carrito desde esta página.

Obtener el ID de la URL

```
const { id } = useParams();
```

- Si la ruta es /producto/12, entonces id = "12".

Estado para guardar el producto

```
const [producto, setProducto] = useState(null);
```

- ② Al inicio no hay producto → null.
- ② Cuando la API responde → se guarda el objeto del producto.

useEffect para cargar el producto

```
useEffect(() => {
  obtenerProductosId(id).then(data => {
    setProducto(data);
  });
}, [id]);
```

- ② Cada vez que cambie el ID en la URL, se vuelve a cargar.
- ② Llama a la API y actualiza el estado.

Función para agregar al carrito

```
function agregarProducto() {
  agregarAlCarrito({
    id_producto: producto.id_producto,
    nombre: producto.nombre,
    descripcion: producto.descripcion,
    precio: producto.precio,
    imagen: producto.imagen
  });
}
```

Prepara la estructura del producto que el carrito necesita.

Llama a agregarAlCarrito().

Render principal del detalle del producto

- Muestra la imagen principal.
- Nombre, precio, descripción.
- Selectores de color y talla (estáticos por ahora).
- Botón "Agregar al carrito".

Reviews del product

```
{producto && <Reviews id_producto={producto.id_producto} />}
```

② Solo se muestra si ya existe un producto cargado.

② Envía el ID para cargar sus reseñas.

Mostrar nuevos ingresos

<NewArrivals/>

- Componente que muestra productos recientes.

componente HomePage.jsx

1. Imports

```
import NewArrivals from "../Components/NewArrivals";
import HappyCustomers from "../Components/HappyCustomers";
import TopSelling from "../Components/TopSelling";
import { Link } from "react-router-dom";
```

② **NewArrivals** → muestra productos nuevos.

② **TopSelling** → muestra los más vendidos.

② **HappyCustomers** → opiniones de clientes.

② **Link** → navegación interna con React Router.

2. Estructura principal del componente

```
export default function HomePage() {
  return (
    <section className="section-home">
```

- El componente devuelve toda la estructura de la página de inicio.

3. Banner principal

```
<article className="art-banner">
  
```

- Imagen promocional en la parte superior.

```
<div className="box-info-banner">
  <h2>Encuentra la ropa...</h2>
  <Link to={'/categoria'}><button>Compra ahora</button></Link>
```

- Texto destacado + botón que lleva a la sección de categorías.

```
<div className="box-datos-banner">
  <div><p>200+</p><span>Marcas internacionales</span></div>
  <div><p>2000+</p><span>Calidad de productos</span></div>
  <div><p>30000+</p><span>Clientes satisfechos</span></div>
</div>
```

- Pequeñas estadísticas de marca y clientes.
- Título de la sección.

```
<div className="box-categorias-estilos">
  <div className="box-img-estilos box-img-c"><h3>Casual</h3></div>
  <div className="box-img-estilos box-img-g"><h3>Formal</h3></div>
  <div className="box-img-estilos box-img-g"><h3>Fiesta</h3></div>
  <div className="box-img-estilos box-img-c"><h3>Gym</h3></div>
</div>
```

❑ Cuatro categorías visuales: Casual, Formal, Fiesta y Gym.

❑ Sirven como inspiración al usuario.

Registro.jsx

```
import { useState } from "react";
import "../estilos/registro.css";
import { useNavigate, Link } from "react-router-dom";
```

💡 **useState** → para manejar el estado del formulario y mensajes.

💡 **CSS** → estilos del formulario.

💡 **useNavigate** → para redirigir al usuario después del registro.

💡 **Link** → para navegar a la página de login.

Componente Registro

```
export const Registro = () => {
```

Define el componente funcional Registro.

API URL desde variables de entorno

```
const API_URL = process.env.REACT_APP_API_URL;
```

Usa la URL del backend guardada en .env.

Estados

```
const [form, setForm] = useState({
  nombre: "",
  apellido: "",
  correo: "",
  telefono: "",
  pass: ""
});
```

Guarda los valores de los inputs del formulario.

```
const [error, setError] = useState("");
const [success, setSuccess] = useState("");
```

Manejan mensajes de error y éxito.

```
const navigate = useNavigate();
```

Permite navegar a otra ruta desde JS.

Actualizar inputs

```
const handleChange = (e) => {
  setForm({
    ...form,
    [e.target.name]: e.target.value
  });
};
```

② Toma el nombre del input

② Actualiza su valor dentro del estado **form**

Enviar formulario

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setError("");
  setSuccess("");
```

Evita recargar la página y limpia mensajes previos.

Petición al backend

```
const response = await fetch(`${API_URL}/registro.php`, {
  method: "POST",
  headers: { "Content-Type": "application/json" },
  body: JSON.stringify(form)
});
```

- Envía los datos del formulario como JSON al backend.

Procesar respuesta

```
const data = await response.json();

if (data.error) {
  setError(data.error);
} else {
  setSuccess("Registro exitoso, puedes iniciar sesión ✓");
  setTimeout(() => navigate("/Login"), 2000);
}
```

② Si PHP devuelve un error → lo muestra.

② Si no → muestra éxito y redirige al login.

Estructura visual del formulario

```
return (
  <div className="login-container">
```

Inputs

Cada input usa **name="campo"** y se actualiza con **onChange**.

```
<input className="login-input" name="nombre" placeholder="Nombre"
onChange={handleChange}/>
```

Botón de registro

```
<button type="submit" className="login-btn">Registrarme</button>
```

Link para acceder al login

```
<Link to="/Login" className="login-btn">Ya tengo cuenta</Link>
```

Mensajes de error y éxito

```
{error && <p style={{ color: "red" }}>{error}</p>}
```

```
{success && <p style={{ color: "green" }}>{success}</p>}
```

APIS:

Función exportada

```
export async function enviarPedido(datos) {
```

- Exporta la función para usarla en otros componentes.
- Es `async` porque hará una petición asíncrona al backend.

URL de la API

```
const API_URL = process.env.REACT_APP_API_URL;
```

Toma la URL del backend desde las variables de entorno.

Intento de enviar datos

```
try {
```

Comienza el bloque donde puede ocurrir un error.

Petición al backend

```
const res = await fetch(` ${API_URL}/api_pago.php` , {  
    method: "POST",  
    headers: {  
        "Content-Type": "application/json"  
    },  
    body: JSON.stringify(datos)  
});
```

- Envía una petición **POST** al archivo `api_pago.php`.
- Los datos del pedido se envían en formato **JSON**.
- `"Content-Type": "application/json"` indica al backend que el cuerpo del mensaje viene en JSON.

Convertir respuesta

```
const data = await res.json();  
return data;
```

- Convierte la respuesta del servidor a JSON.

- Devuelve la respuesta para usarla en el componente que llame esta función.

Captura de errores

```
} catch (e) {  
  
    console.error("Error al enviar pedido:", e);  
  
    throw new Error("Error al conectar con el servidor");  
  
}
```

- Si ocurre un error de conexión o la API está caída, entra aquí.
- Muestra un mensaje en consola.
- Lanza un error para que el componente lo maneje (con un mensaje entendible para el usuario).
- Envía un pedido al backend con fetch POST.
- Convierte respuesta en JSON.
- Devuelve la respuesta.
- Si algo falla → lanza un error de conexión.

obtenerProductos()

```
export async function obtenerProductos() {
```

- Se exporta para poder usarla en otros componentes.
- Es async porque utiliza await dentro.

Obtener la URL del backend

```
const API_URL = process.env.REACT_APP_API_URL;
```

- Toma la URL base de tu API desde las variables de entorno.

Petición al servidor

```
const res = await fetch(` ${API_URL}/api_productos.php`);
```

Llama al archivo PHP api_productos.php.

Como es un GET, no necesita opciones adicionales.

await espera la respuesta del backend.

Convertir la respuesta en JSON

```
const data = await res.json();
```

- Convierte la respuesta en formato JSON (lista de productos).

Devolver los datos

```
return data;
```

- Los productos se devuelven para ser usados en tu componente (por ejemplo en un useEffect).
- Hace un fetch GET a api_productos.php.
- Convierte la respuesta a JSON.
- Devuelve la lista de productos.

Config

1. Definición de URL principal

```
export const API_URL = "http://localhost/Ejemplos.PHP/crud_sql/productos/api_productos.php";
```

💡 export const → La variable se puede importar desde cualquier archivo.

💡 API_URL → Es la URL completa que apunta directamente al archivo PHP api_productos.php.

💡 Esta dirección se usa normalmente para obtener productos, porque apunta exactamente al recurso que devuelve la lista.

2. Definición de URL base (para otras rutas)

```
export const API_URL_CARRITO = "http://localhost/Ejemplos.PHP/crud_sql/productos";
```

- Es la **ruta base** de la carpeta donde están tus archivos PHP.
- No apunta a un script específico.
- Te sirve para construir rutas como:
 - \${API_URL_CARRITO}/api_carrito.php
 - \${API_URL_CARRITO}/api_pago.php
 - \${API_URL_CARRITO}/api_producto_id.php

Es básicamente la **carpeta padre** para tus diferentes APIs.

- API_URL → apunta a **un archivo exacto**: api_productos.php
- API_URL_CARRITO → apunta a la **carpeta**, útil para armar otras rutas.

obtenerProductosId

```
export async function obtenerProductosId(id_producto){
```

② export → permite importar esta función desde otros archivos.

② async → la función usará await dentro.

② id_producto → es el ID que se enviará al backend para obtener un producto específico.

```
const API_URL = process.env.REACT_APP_API_URL;
```

② Obtiene la URL base desde las variables de entorno de React.

② Esto permite cambiar la URL sin modificar el código.

```
const res = await fetch(` ${API_URL}/api_productos.php?id_producto=${id_producto}`);
```

- Hace una petición HTTP GET al archivo api_productos.php.
- Envía el id_producto por la URL como parámetro.
- await espera la respuesta del servidor.

Ejemplo final de URL generada:

http://tu-servidor/api_productos.php?id_producto=5

```
const data = await res.json();
```

Convierte la respuesta del servidor en formato JSON.

data es el producto obtenido.

```
return data;
```

```
}
```

- La función devuelve los datos del producto al componente que la llamó.

- Toma un **id de producto**.
- Llama a **api_productos.php** enviando ese id.
- Espera la respuesta.
- La convierte a JSON.
- Devuelve el producto.