Desarrollo del Backend - Carrito de Compras API REST

---

1) Inicio del Proyecto

El backend se inició desde cero utilizando **Node.js**, **Express** y **SQLite** como base de datos. La estructura del proyecto se organizó en carpetas separadas para rutas, modelos, controladores y middlewares.

Se instalaron los siguientes paquetes:

- `express`: servidor web.
- `better-sqlite3`: cliente para la base de datos SQLite.
- `bcrypt`: para encriptar contraseñas.
- `jsonwebtoken`: para autenticación con JWT.
- `dotenv`: para manejar variables de entorno.
- `chalk`: para mostrar mensajes de consola con colores.

---

2) Autenticación de Usuarios [usuario]

Se desarrolló un sistema de autenticación básico con dos endpoints:

- **POST** `/auth/register`: Crea un nuevo usuario con email, contraseña encriptada mediante `bcrypt` y rol (por defecto "usuario").
- **POST** `/auth/login`: Verifica las credenciales y retorna un token JWT. Este token se usará para acceder a rutas protegidas.

La tabla `usuarios` incluye los campos: `id`, `email`, `password` y `rol`.

---

3) Middlewares [log, auth, rol]

- **authMiddleware**: Verifica que el token JWT sea válido. Extrae los datos del usuario autenticado y los adjunta al `req` para usarlos en las rutas.
- **adminMiddleware**: Permite el acceso a ciertas rutas sólo si el usuario tiene el rol "admin" (por ejemplo: crear productos, acceder a logs).
- **logMiddleware**: Middleware global compuesto por `logRequest`, `logResponse` y `auditLogger`. Registra en la base de datos cada acción del usuario (endpoint accedido, método, usuario, fecha y hora).

---

4) Gestión de Productos [producto]

Se implementaron las siguientes rutas para manejar productos:

- **GET** `/api/productos`: Lista todos los productos.
- **GET** `/api/productos/:id`: Trae un producto específico por su ID.
- **POST** `/api/productos`: Crea un nuevo producto (solo para admins).

- **PUT** `/api/productos/:id`: Actualiza un producto (solo para admins).
- **DELETE** `/api/productos/:id`: Elimina un producto (solo para admins).

Los productos se almacenan con los campos `id`, `nombre` y `precio`.

---

5) Carrito de Compras [carrito]

Cada usuario tiene un carrito asociado. Las rutas para gestionarlo son:

- **POST** `/api/carrito`: Agrega un producto al carrito del usuario autenticado.
- **GET** `/api/carrito`: Muestra los productos actuales del carrito.
- **DELETE** `/api/carrito/:id`: Elimina un producto puntual del carrito.
- **DELETE** `/api/carrito`: Vacía completamente el carrito.

La tabla `carrito` asocia productos con `cantidad` y el `user_id` correspondiente.

---

6) Finalizar Compra [compra]

Cuando un usuario finaliza su compra:

1. Se verifica que el carrito no esté vacío.
2. Se calcula el total a pagar.
3. Se guarda la compra en la tabla `compras`.
4. Se guarda el detalle de cada producto (producto, cantidad, precio unitario) en la tabla `compras_detalles`.
5. Se vacía el carrito.
6. Se registra la acción en los logs.

Ruta correspondiente:

- **POST** `/api/compra`

---

7) Historial de Compras [compra]

Cada usuario puede consultar su historial de compras:

- Fecha de cada compra.
- Total gastado.
- Detalle de los productos comprados.

Ruta correspondiente:

- **GET** `/api/compra`

El backend realiza un JOIN entre las tablas `compras`, `compras_detalles` y `productos` para retornar toda la información en una sola respuesta estructurada.