Parcial Integrador – Bases de Datos 2

Tema: API REST para un e-commerce

Modelado de la Base de Datos

1. **Usuarios**
    - Datos básicos del cliente (nombre, email, dirección, teléfono, rol, etc.)
    - Podrá incluir información de perfil o direcciones.
    - Los usuarios pueden ser clientes o administradores.
2. **Productos**
    - Información del producto: nombre, descripción, categoría, precio, stock, etc.
    - un campo de reseñas o calificaciones.
3. **Categorías**
    - Nombre y descripción de la categoría.
    - Los productos se agrupan por categoría.
4. **Pedidos (Orders)**
    - Información general del pedido: fecha, estado, total, método de pago.
    - Debe incluir los ítems comprados (producto, cantidad, subtotal).
    - Asociado a un usuario (cliente que realiza la compra).
5. **Carritos (Carts)**
    - Cada usuario puede tener un carrito activo.
    - Contiene productos seleccionados y cantidades.
6. **Resenas (Reviews)**

- Opiniones de usuarios sobre productos.
- Calificación numérica y comentario.
- Asociadas a un usuario y un producto.

🌐 Rutas requeridas

🔹 Usuarios (`/api/usuarios`)

- CRUD de usuarios
- `GET /api/users` → listar todos los usuarios
- `GET /api/users/:id` → detalle de un usuario.
- `POST /api/users` → registrar usuario.
- `DELETE /api/users/:id` → eliminar usuario y su carrito.

🔹 Productos (`/api/productos`)

- CRUD de productos
- `GET /api/productos` → listar productos con su categoría .
- `GET /api/productos/filtro` → filtrar por rango de precio y marca.
- `GET /api/productos/top` → productos más reseñados .
- `PATCH /api/productos/:id/stock` → actualizar stock.

🔹 Categorías (`/api/categorias`)

- CRUD de categorías
- `GET /api/categorias/stats` → cantidad de productos por categoría.

🔹 Carrito (`/api/carrito`)

- CRUD de carrito

- `GET /api/carrito/:usuarioId` → mostrar carrito con productos
- `GET /api/carrito/:usuarioId/total` → calcular total y subtotal del carrito .

🔹 **Pedidos (`/api/ordenes`)**

- CRUD de pedidos
- `GET /api/ordenes` → listar pedidos con datos de usuario.
- `GET /api/ordenes/stats` → total de pedidos por estado.
- `GET /api/ordenes/user/:userId` → pedidos de un usuario.
- `PATCH /api/ordenes/:id/status` → actualizar estado .

🔹 **Reseñas (`/api/resenas`)**

- CRUD de reseñas
- `GET /api/resenas` → listar todas las reseñas con datos de usuario y producto.
- `GET /api/resenas/product/:productId` → reseñas de un producto.
- `GET /api/resenas/top` → promedio de calificaciones por producto
- `POST /api/resenas` → crear reseña solo si el usuario compró el producto.

Operadores que deben utilizarse

Los alumnos deberán usar en distintas rutas:

- **Comparación:** `$eq`, `$ne`, `$gte`, `$lte`, `$and`, `$or`
- **Modificación:** `$set`, `$push`, `$pull`
- **Agregación:** `$lookup`, `$group`, `$match`, `$sort`, `$unwind`, `$count`, `$avg`, `$sum`

Qué rutas proteger con JWT (sugerencia)

- Requieren token (cliente o admin): `/api/carrito/*`, `POST /api/ordenes`, `GET /api/ordenes/user/:userId` (solo dueño o admin), `POST /api/resenas`.

- Requieren **admin**: crear/editar/eliminar productos y categorías, listar todos los usuarios, `/api/ordenes/stats`, cambiar estado de orden (`PATCH`

`/api/ordenes/:id/status`).

- Rutas públicas: listar productos, ver detalles, ver reseñas.

Manejo de errores y respuestas JSON

- Estándar: `{ success: true/false, data: ..., error: ... }` o `{ message, ... }`.
- Validar inputs y devolver 4xx con mensaje claro.
- Atrapar errores con `try/catch` y middleware global de error (`next(err)` → `res.status(500).json({error: err.message})`).

Tests y comprobaciones

- Usar Postman / Insomnia para todas las rutas, separar cada ruta por carpetas segun el modelo
- Para probar expiración, cambiar `JWT_EXPIRES_IN` a `1s` y verificar.

🧠 Puntos a Evaluar

1. Correcto modelado (referencias y embebidos).
2. Utilización de JWT y verificación de rol en las rutas de POST PUT DELETE
3. Organización de la API RESTful.
4. Uso adecuado de Mongoose.
5. Implementación de operadores y agregaciones.
6. Respuestas JSON bien formateadas y manejo de errores.