**1️⃣ Registro de usuario**

**Endpoint:** POST /auth/register  
**Payload de ejemplo:**

{

"email": "usuario@correo.com",

"full\_name": "Juan Perez",

"password": "contraseña123"

}

**Flujo interno:**

1. Se crea un registro en la tabla User.
2. Se genera user\_uuid automáticamente.
3. No hay carrito ni historial aún.

**Respuesta ejemplo:**

{

"user\_uuid": "f3a1c6e2-1234-4567-890a-abcdef123456",

"email": "usuario@correo.com",

"full\_name": "Juan Perez"

}

**2️⃣ Login de usuario**

**Endpoint:** POST /auth/login  
**Payload de ejemplo:**

{

"email": "usuario@correo.com",

"password": "contraseña123"

}

**Flujo interno:**

1. Se valida email y contraseña.
2. Se devuelve un access\_token para autenticar futuras solicitudes.
3. El token se usará en headers Authorization: Bearer <token>.

**3️⃣ Agregar productos al carrito**

**Endpoint:** POST /cart/add  
**Payload de ejemplo:**

{

"product\_uuid": "3d50d33c-01cc-439b-b481-56d14d2eae84",

"quantity": 1

}

**Flujo interno:**

1. Se llama a add\_to\_cart(user\_uuid, product\_uuid, quantity, price, db).
2. Si el producto ya está en el carrito, se suma la cantidad.
3. Si no está, se crea un nuevo CartItem.

**Respuesta ejemplo:**

{

"message": "Producto agregado al carrito"

}

**4️⃣ Listar items en el carrito**

**Endpoint:** GET /cart/list

**Flujo interno:**

* Devuelve todos los items del carrito del usuario.
* Útil para revisar antes del pago.

**Respuesta ejemplo:**

[

{

"product\_id": "3d50d33c-01cc-439b-b481-56d14d2eae84",

"product\_name": "Mouse",

"quantity": 1,

"price": 180000,

"subtotal": 180000

}

]

**5️⃣ Checkout / Pago**

**Endpoint:** POST /checkout/payment  
**Payload de ejemplo:**

{

"card\_number": "4111111111111111",

"holder\_name": "Juan Perez",

"expiration\_date": "12/26",

"cvv": "123"

}

**Flujo interno (process\_checkout):**

1. **Validación de tarjeta**
   * Se envía a MOCK\_PAYMENT\_URL/cards/validate.
2. **Cálculo del total**
   * Se obtiene el total del carrito (get\_cart\_total).
3. **Pago en mock**
   * Se envía account\_uuid, amount y description a /payments.
4. **Registro de items comprados**
   * Se genera la lista de PurchasedProduct a devolver.
5. **Limpieza del carrito**
   * Se vacía con clear\_cart.

**Respuesta ejemplo:**

{

"message": "Pago procesado exitosamente",

"total": 180000,

"items": [

{

"product\_id": "3d50d33c-01cc-439b-b481-56d14d2eae84",

"product\_name": "Mouse",

"quantity": 1,

"price": 180000,

"subtotal": 180000

}

]

}

**6️⃣ Comprar de nuevo**

* Después de una compra, el carrito queda vacío.
* Se pueden agregar nuevos productos y repetir el flujo de pago.
* Cada compra es independiente y segura.

**Opcional: Historial de compras – NO SE INCLUYE EN EL PROYECTO.**

Si quieres mantener un historial de todas las compras:

1. Crear tabla Order y OrderItem.
2. Antes de limpiar el carrito, guardar todos los items en OrderItem.
3. Así puedes consultar todas las compras de un usuario sin perder la información.

Describo paso paso

**Detalle paso a paso:**

1. **Registro de Usuario**
   * Se crea user\_uuid.
   * No hay carrito ni compras.
2. **Login**
   * Se obtiene access\_token.
   * Autenticación para endpoints protegidos.
3. **Agregar Productos al Carrito**
   * add\_to\_cart(user\_uuid, product\_uuid, quantity, price, db)
   * Carrito dinámico, se actualiza si ya existe el producto.
4. **Listar Carrito**
   * list\_cart\_items(user\_uuid, db)
   * Muestra subtotal de cada item y total.
5. **Checkout / Pago**
   * Validación de tarjeta.
   * Cálculo total (get\_cart\_total).
   * Pago mock (/payments).
   * Limpieza del carrito (clear\_cart).
   * Devuelve lista de PurchasedProduct.
6. **Carrito Limpio**
   * Puede agregar nuevos productos.
   * Compra posterior es independiente.
7. **Historial de Compras (Opcional)**
   * Guardar items en Order / OrderItem antes de limpiar el carrito.
   * Permite consultar todas las compras pasadas.