Taller 3 - Integracion de Backend con FastAPI

Pola Escobar, Víctor Núñez, José Salazar, Sofía Sepúlveda

13 de noviembre de 2025

1. Resumen

Documento que describe la arquitectura de datos, los endpoints principales de la API desarrollada y las decisiones técnicas tomadas para el proyecto "Mini Marketplace UNAB".

2. Configuración y servidor

La API se ejecuta con FastAPI (ver `backend/API/app/main.py`). La conexión a la base de datos se configura en `backend/API/app/database.py` mediante la variable `DATABASE_URL`.

3. Modelo de datos

Se exponen a continuación las tablas principales implementadas en SQLAlchemy y reflejadas en la base de datos PostgreSQL.

3.1. Usuarios

Tabla: usuarios (modelo: `models.Usuario`)

id	UUID, PK, default uuid_generate_v4()
email	VARCHAR(255), UNIQUE, NOT NULL
username	VARCHAR(100), UNIQUE, NOT NULL
password_hash	VARCHAR(255), NOT NULL
nombre	VARCHAR(100) (opcional)
apellido	VARCHAR(100) (opcional)
activo	BOOLEAN, default true
rol	VARCHAR(50), default 'user'
metadata	JSONB, default {}
fecha_creacion	TIMESTAMP, default CURRENT_TIMESTAMP
fecha_actualizacion	TIMESTAMP, default CURRENT_TIMESTAMP, ON UPDATE

3.2. Documentos

Tabla: documentos (modelo: `models.Documento`)

id	UUID, PK
usuario_id	UUID, FK a usuarios(id), ON DELETE CASCADE
estado	VARCHAR(50), default 'borrador'

monto_total	FLOAT, default 0
fecha_creacion	TIMESTAMP
fecha_actualizacion	TIMESTAMP

3.3. Detalle de documentos

Tabla: detalle_documentos (modelo: `models.DetalleDocumento`)

id	UUID, PK
documento_id	UUID, FK a documentos(id), ON DELETE CASCADE
producto	VARCHAR(255), NOT NULL
precio	FLOAT/NUMERIC, NOT NULL
cantidad	INTEGER, default 1
fecha_creacion	TIMESTAMP
fecha_actualizacion	TIMESTAMP

3.4. Direcciones de despacho

Tabla: direcciones_despacho (modelo: `models.DireccionDespacho`)

id	UUID, PK, default uuid_generate_v4()
usuario_id	UUID, FK a usuarios(id), ON DELETE CASCADE
direccion	VARCHAR(255), NOT NULL
comuna	VARCHAR(100), NOT NULL
ciudad	VARCHAR(100), NOT NULL
codigo_postal	VARCHAR(10) (opcional)
es_principal	BOOLEAN, default false
activa	BOOLEAN, default true
metadata	JSONB, default {}
fecha_creacion	TIMESTAMP, default CURRENT_TIMESTAMP
fecha_actualizacion	TIMESTAMP, default CURRENT_TIMESTAMP, ON UPDATE

3.5. Índices y triggers

El script de inicialización `backend/BD/init-db.sql` crea índices GIN sobre JSONB y triggers que actualizan `fecha_actualizacion` y recalculan `monto_total` cuando cambian los detalles del documento. También inserta un usuario admin de prueba.

4. Esquemas (Pydantic)

Los esquemas usados por los endpoints (Pydantic) están en `backend/API/app/schemas.py`. Se destacan:

- `Token` — respuesta del endpoint /token
- `TokenData` — datos internos del token
- `UsuarioCreate`, `UsuarioResponse`
- `DocumentoCreate`, `DocumentoResponse`
- `DetalleDocumentoCreate`, `DetalleDocumentoResponse`

5. Autenticación

Flujo basada en OAuth2 password grant y JWT.

- Hashing de contraseñas con `passlib` (bcrypt): funciones en `backend/API/app/auth.py`.
- Generación de token JWT: `create_access_token(data, expires_delta)`.
- Validación de token y recuperación de usuario actual: `get_current_user`.
- Variables de configuración: `SECRET_KEY`, `ALGORITHM`, `ACCESS_TOKEN_EXPIRE_MINUTES`.

6. Endpoints principales

A continuación se listan rutas, método HTTP, autenticación requerida y esquemas de entrada/salida.

6.1. Autenticación

- POST /token — Obtener token JWT (OAuth2 password flow).
  Entrada: form-urlencoded `username`, `password`.
  Respuesta: `Token`. (Implementación: `main.login`)
- POST /register — Registrar nuevo usuario.
  Entrada: `UsuarioCreate`.
  Respuesta: `UsuarioResponse`. (Implementación: `main.register_user`)
- GET /users/me — Obtener usuario actual. Requiere Authorization Bearer.
  Respuesta: `UsuarioResponse`. (Implementación: `main.read_users_me`)

6.2. Usuarios

- GET /usuarios — Listar usuarios (auth).
  Respuesta: List[`UsuarioResponse`]. (`main.get_usuarios`)
- GET /usuarios/usuario_id — Obtener usuario por ID (auth).
  Respuesta: `UsuarioResponse`. (`main.get_usuario`)
- PUT /usuarios/me — Actualizar información del usuario actual.
  Entrada: `UsuarioUpdate`. Requiere auth.
  Respuesta: `UsuarioResponse`. (Implementación: `main.update_usuario_me`)
- GET /usuarios/me/compras — Obtener compras anteriores del usuario actual. Requiere auth.
  Parámetros de consulta: `skip` (int, default 0), `limit` (int, default 20, max 100), `estado` (opcional), `orden` (string, default "fecha_desc").
  Respuesta: List[`CompraResponse`]. (Implementación: `main.get_mis_compras`)

6.3. Documentos

- POST /documentos — Crear documento.
  Entrada: `DocumentoCreate`. Requiere auth.
  Respuesta: `DocumentoResponse`. (`main.create_documento`)

- GET /documentos — Listar documentos del usuario actual. Requiere auth.
  Respuesta: List[`DocumentoResponse`]. (`main.get_documentos`)
- GET /documentos/documento_id — Obtener documento por ID. Requiere auth.
  Respuesta: `DocumentoResponse`. (`main.get_documento`)
- PUT /documentos/documento_id — Actualizar documento.
  Entrada: `DocumentoCreate`. Requiere auth.
  Respuesta: `DocumentoResponse`. (`main.update_documento`)
- DELETE /documentos/documento_id — Eliminar documento. Requiere auth.
  Respuesta: 204. (`main.delete_documento`)

6.4. Direcciones de despacho

- POST /usuarios/me/direcciones — Crear nueva dirección de despacho para el usuario actual.
  Entrada: `DireccionDespachoCreate`. Requiere auth.
  Respuesta: `DireccionDespachoResponse`. (Implementación: `main.create_direccion_despacho`)
- GET /usuarios/me/direcciones — Listar direcciones de despacho del usuario actual. Requiere auth.
  Parámetros de consulta: `activa` (bool, opcional).
  Respuesta: List[`DireccionDespachoResponse`]. (Implementación: `main.get_direcciones_despacho`)
- GET /usuarios/me/direcciones/{direccion_id} — Obtener dirección de despacho por ID. Requiere auth.
  Respuesta: `DireccionDespachoResponse`. (Implementación: `main.get_direccion_despacho`)
- PUT /usuarios/me/direcciones/{direccion_id} — Actualizar dirección de despacho.
  Entrada: `DireccionDespachoUpdate`. Requiere auth.
  Respuesta: `DireccionDespachoResponse`. (Implementación: `main.update_direccion_despacho`)
- DELETE /usuarios/me/direcciones/{direccion_id} — Eliminar dirección de despacho. Requiere auth.
  Respuesta: 204. (Implementación: `main.delete_direccion_despacho`)
- PUT /usuarios/me/direcciones/{direccion_id}/principal — Marcar dirección como principal. Requiere auth.
  Respuesta: `DireccionDespachoResponse`. (Implementación: `main.marcar_direccion_principal`)

6.5. Detalles de documento

- POST /documentos/documento_id/detalles — Agregar detalle.
  Entrada: `DetalleDocumentoCreate`. Requiere auth.
  Respuesta: `DetalleDocumentoResponse`. (`main.create_detalle`)
- GET /documentos/documento_id/detalles — Obtener detalles. Requiere auth. Respuesta: List[`DetalleDocumentoResponse`]. (`main.get_detalles`)

6.6. Checkout

- POST /checkout — Procesar checkout y crear compra.
  Entrada: `CheckoutRequest`. Requiere auth.
  Respuesta: `DocumentoResponse`. (Implementación: `main.checkout`)

7. Decisiones técnicas tomadas

- Migración de Angular a React: Se tomó la decisión de migrar tempranamente a React en esta iteración para facilitar la transición a React Native en la siguiente entrega para el desarrollo de la parte Mobile.
- Uso de PostgreSQL: Se decidió utilizar PostresSQL por su integración de ciertas características de NoSQL, con soporte nativo para 3 tipos de datos no relacionales (XML, JSON, JSONB). Además de una gran cantidad de extensiones que podrán utilizarse para la migración de una base de datos relacional a una base de datos no relacional.
- Modelado de la base de datos: Se definió una base de datos sencilla, con 4 tablas (`Usuario, documentos, detalle_documentos` y `direcciones_despacho`. Usuarios necesita de un correo, usuario y contraseña que no pueden ser nulos, pero también se guarda su rut, nombre, apellido y teléfono para agilizar el proceso de compra con autollenado de datos en el checkout. Los documentos solamente tienen el atributo de estado (completado, borrador) y el monto_total. El detalle de los documentos almacena producto, precio y cantidad, que luego se utiliza para autocalcular el monto total de la compra. Un documento puede tener varios detalles, esto para que las compras puedan contener más de un producto. Las direcciones de despacho se enlazan directamente con el usuario, y solo se necesitan en caso de elegir algún medio de despacho que la necesite.
- Endpoints y documentación de usuarios y documentos: Los endpoints de autenticación usan JWT para sesiones stateless (/token, /register), recursos de usuario y documentos exponen operaciones CRUD claras (p. ej. /usuarios/me, /documentos) y los detalles de documento se anidan bajo su documento padre. Se añadió un endpoint transaccional de checkout (/checkout) que agrupa validaciones, cálculo de totales y creación de ordenes para garantizar atomicidad; además se usan esquemas Pydantic para validar entradas (RUT, email, teléfono) y permisos basados en el usuario autenticado.
- Validaciones en frontend: Se hacen validaciones sencillas en los formularios de registro/inicio de sesión, se valida el formato de email, teléfono y rut, los requisitos de las contraseñas, los campos obligatorios y se hace una normalización de los datos antes de enviar. Se hizo con el fin de mejorar la experiencia del usuario, tener feedback inmediato y reducir peticiones innecesarias.
