

UNIVERSIDAD NACIONAL DE COLOMBIA

FACULTAD DE INGENIERÍA

Asignatura: Programación Orientada a Objetos

FiApp

De la libreta al clic: tus finanzas en la palma de tu mano

Manual del Backend

Guía de Instalación, Arquitectura y Uso

Autores:

Santiago López Murcia

Andrés Mauricio Cepeda Villanueva

José Luis Cancelado Castro

Juan Diego Cuartas Casas

3 de diciembre de 2025

Índice

1. Introducción	2
2. Requisitos e Instalación	2
2.1. Prerrequisitos	2
2.2. Proceso de Instalación	2
3. Configuración y Ejecución	2
3.1. Variables de Entorno	2
4. Estructura del Proyecto	3
5. Descripción Lógica del Código (Arquitectura Detallada)	3
5.1. Paquete: app	3
5.2. Paquete: database	4
5.3. Paquete: static (Frontend Logic)	4
5.4. Variables Globales y de Entorno Importantes	4
6. Modelos de Datos (JSON)	4
6.1. Usuarios	5
6.2. Locales	5
6.3. Proveedores	5
7. Servicios Clave	5
7.1. AuthService	5
7.2. DBService	5
8. Rutas HTTP y API	6
8.1. Rutas Generales	6
8.2. Rutas de Tendero	6
8.3. API del Asistente IA	6
9. Guía del Asistente IA (Chat)	6
10.Solución de Problemas (Troubleshooting)	6
11.Despliegue en Producción	7

1. Introducción

Este manual describe cómo instalar, configurar, operar y depurar el backend de **FiApp**, una aplicación diseñada para modernizar las finanzas personales y de pequeños negocios, basada en **Flask** y **Firebase Realtime Database**.

El sistema ha sido actualizado recientemente para incluir manejo de imágenes, gestión de proveedores con propietario y un asistente de chat IA orientado a cálculos financieros.

Resumen Técnico

- **Tecnología:** Flask + Firebase Admin SDK.
- **Base de Datos:** Firebase Realtime Database.
- **Archivos Clave:**
 - `app/main.py` (Servidor y rutas)
 - `database/firebase_config.py` (Conexión)
 - `database/auth_service.py` (Autenticación)
 - `database/db_service.py` (CRUD)
 - `static/script.js` (Lógica frontend y Chat)

2. Requisitos e Instalación

2.1. Prerrequisitos

- **Python 3.8** o superior.
- Paquetes listados en `requirements.txt`:
 - Flask
 - firebase-admin
 - python-dotenv
 - requests

2.2. Proceso de Instalación

1. Clone o abra el repositorio y sitúese en la carpeta raíz **FIAPP**. 2. Instale las dependencias ejecutando:

```
1 python -m pip install -r requirements.txt
```

Instalación de dependencias

3. Configuración y Ejecución

3.1. Variables de Entorno

Para ejecutar el backend, es necesario configurar las siguientes variables de entorno. Puede usar un archivo `.env` para desarrollo local.

FIREBASE_CREDENTIALS_PATH Ruta absoluta al archivo JSON de la *Service Account*. **Nota:** No subir este archivo al control de versiones.

FIREBASE_DB_URL URL de su Realtime Database (ej: `https://fiapp-xyz.firebaseio.com`).

USE_LOCAL_AUTH `true` para evitar llamadas a Firebase (modo debug), `false` para producción.

QROQ_API_KEY **Requerido**. Clave API necesaria para habilitar y visualizar el Chatbot del tendero.

Ejemplo completo de configuración y ejecución en PowerShell:

```
1 # Configuración básica de Firebase
2 $Env:FIREBASE_CREDENTIALS_PATH = 'C:\ruta\a\fiapp-adminsdk.json'
3 $Env:FIREBASE_DB_URL = 'https://fiapp-17341-default-rtdb.firebaseio.com'
4 $Env:USE_LOCAL_AUTH = 'false'
5
6 # Configuración del Chatbot (OBLIGATORIO para ver el asistente)
7 $env:QROQ_API_KEY = '
8     gsk_7EUXhWt7BjfYC40fA3imWGdyb3FY7W7eildLZTPlAeFYgAxRzNd'
9
10 # Ejecución del servidor
11 python -m app.main
```

Nota Importante sobre la Ejecución

La línea `$env:QROQ_API_KEY` es indispensable. Si no se configura esta variable antes de iniciar el servidor, el módulo de IA no cargará y el botón del chat no aparecerá en la interfaz del tendero.

4. Estructura del Proyecto

- `app/main.py`: Servidor Flask, definición de rutas HTTP y control de sesiones.
- `database/`:
 - `auth_service.py`: Registro, login, hashing de contraseñas y asignación de roles.
 - `db_service.py`: Operaciones CRUD (Crear, Leer, Actualizar, Borrar) para locales, productos y clientes.
- `ViewModel/use_cases.py`: Capa lógica que orquesta las llamadas entre el controlador y la base de datos.
- `templates/`: Vistas HTML (Jinja2).
- `static/`: Archivos CSS, imágenes y `script.js` (lógica del chat y UI).

5. Descripción Lógica del Código (Arquitectura Detallada)

Siguiendo la arquitectura modular del proyecto, a continuación se describe la lógica implementada en los paquetes y archivos principales.

5.1. Paquete: `app`

Este paquete actúa como el controlador principal de la aplicación web.

- **Archivo:** `main.py`
 - **Método Principal:** Inicia la instancia de Flask y configura la llave secreta para el manejo de sesiones.
 - **Gestión de Rutas:** Define los endpoints HTTP (`@app.route`) que conectan las vistas (HTML) con la lógica de negocio.

- **Lógica de Chatbot:** Contiene el endpoint `/api/ai_chat` que recibe el mensaje del frontend, consulta la API de Groq y retorna la respuesta procesada.

5.2. Paquete: database

Equivalente a la capa de persistencia y modelo de datos.

- **Archivo: `firebase_config.py`**
 - **Método `init_firebase()`:** Es el encargado de leer las variables de entorno (`FIREBASE_CREDENTIALS`) e inicializar la conexión segura con Google Cloud. Valida que no exista una instancia previa para evitar conflictos.
- **Archivo: `auth_service.py`**
 - **Método `login_user(email, password)`:** Verifica las credenciales contra la base de datos. Retorna una tupla con el `user_id` y el `tipo_usuario` para gestionar la sesión.
 - **Método `register_user(...)`:** Crea un nuevo registro en el nodo `usuarios/` usando un hash MD5 del correo como clave única.
- **Archivo: `db_service.py`**
 - **Método `registrar_deuda(local_id, cliente_id, monto)`:** Realiza dos operaciones atómicas: 1. Actualiza el saldo total del cliente. 2. Crea un registro histórico con *timestamp* en el nodo `deudas/`.
 - **Métodos CRUD:** Contiene la lógica para crear, leer, actualizar y borrar Locales y Productos.

5.3. Paquete: static (Frontend Logic)

Contiene los recursos públicos y la lógica del lado del cliente.

- **Archivo: `script.js`**
 - **Evento `DOMContentLoaded`:** Inicializa los componentes visuales y verifica si el usuario tiene rol de tendero para inyectar el botón del chat.
 - **Función `toggleChat()`:** Controla la visibilidad de la ventana flotante del asistente.
 - **Función `sendMessage()`:** Captura el texto del input, lo envía asíncronamente (Fetch API) al backend y renderiza la respuesta de la IA en el historial del chat.

5.4. Variables Globales y de Entorno Importantes

Para garantizar la seguridad y configuración dinámica del sistema se utilizan las siguientes constantes:

- `FIREBASE_DB_URL`: Define el punto de acceso a la base de datos NoSQL.
- `GROQ_API_KEY`: Credencial necesaria para autorizar las peticiones al modelo de lenguaje (LLM).
- Carpeta `static/productos`: Ruta relativa donde se almacenan las imágenes subidas por los usuarios para el inventario visual.

6. Modelos de Datos (JSON)

Los datos se almacenan en Firebase Realtime Database con la siguiente estructura jerárquica:

6.1. Usuarios

Ruta: `usuarios/{email_key}`, donde `email_key` es el MD5 del email.

- Campos: `email`, `password_hash`, `user_id`, `tipo_usuario` (tendero/cliente).

6.2. Locales

Ruta: `locales/{local_id}`. Estructura típica:

```
1 "local_jhose9282_1610000000": {
2   "nombre": "Mi Tienda",
3   "propietario_id": "jhose9282",
4   "productos": {
5     "p1": {
6       "nombre": "Arroz",
7       "precio": 8200,
8       "stock": 10,
9       "imagen_url": "/static/productos/archivo.jpg"
10    }
11  },
12  "clientes": {
13    "cliente123": {
14      "nombre": "Ana",
15      "deuda": 15000,
16      "deudas": { "timestamp": { "monto": 5000 } }
17    }
18  }
19 }
```

Ejemplo de estructura de un Local

6.3. Proveedores

Ruta: `proveedores/{proveedor_id}`.

- Campos: `nombre`, `contacto`, `email`, `propietario_id` (para filtrar por tendero).

7. Servicios Clave

7.1. AuthService

Ubicado en `database/auth_service.py`.

- `register_user`: Crea usuario sin rol.
- `login_user`: Valida credenciales y retorna (`user_id`, `tipo_usuario`).
- `set_user_type`: Asigna el rol (tendero o cliente).

7.2. DBService

Ubicado en `database/db_service.py`. Maneja la persistencia de datos:

- **Locales**: `add_local`, `get_local`, `update_local`.
- **Productos**: `add_producto`, `update_producto`. Ahora soporta enlace a imágenes.
- **Deudas**: `registrar_deuda(local_id, cliente_id, monto)`. Actualiza el acumulado y añade un registro histórico.

- **Proveedores:** CRUD completo con soporte de `propietario_id`.

8. Rutas HTTP y API

8.1. Rutas Generales

- GET `/`: Página de inicio.
- GET, POST `/register`: Registro de usuario.
- GET, POST `/login`: Inicio de sesión.
- GET, POST `/select-type`: Selección de rol tras el registro.

8.2. Rutas de Tendero

Prefijo: `/tendero`

- `/locales`: Listado de tiendas.
- `/locales/create`: Crear tienda.
- `/locales/<id>/inventario`: Gestión de productos.
- `/proveedores`: Gestión de proveedores.

8.3. API del Asistente IA

- **Endpoint:** POST `/api/ai_chat`
- **Payload:** JSON `{ "message": "tu pregunta" }`
- **Respuesta:** JSON `{ reply": "texto de respuesta" }`
- **Restricción:** Solo accesible si `tipo_usuario == 'tendero'`.

9. Guía del Asistente IA (Chat)

El sistema incluye un asistente capaz de realizar cálculos rápidos para el tendero.

Pasos para usarlo:

1. Asegúrese de haber configurado `QROQ_API_KEY` antes de iniciar el servidor.
2. Inicie sesión como **tendero**.
3. Busque el botón circular con el icono `lofofiapp.ico` en la esquina inferior derecha.
4. Haga clic para desplegar el chat y envíe comandos (ej: *"10 % de 250"*).

10. Solución de Problemas (Troubleshooting)

Error: ModuleNotFoundError: No module named 'app'

Este error ocurre porque Python no encuentra el paquete `app`. **Solución:** Debe ejecutar el comando desde la carpeta raíz del proyecto (FIAPP).

Incorrecto: `C:\Usuarios\FIAPP\app>python main.py`

Correcto: `C:\Usuarios\FIAPP>python -m app.main`

Error: Invalid certificate argument "None"

La variable `FIREBASE_CREDENTIALS_PATH` no está definida o la ruta es incorrecta.

Error: invalid_grant (JWT Signature)

El reloj del sistema está desincronizado. Ejecute `w32tm /resync` en Windows o ajuste la hora en la configuración del sistema.

Error: Invalid path (Illegal characters)

Está intentando usar un email como clave en la base de datos (contiene `.` o `@`). Use el `user_id` o un hash MD5.

El chat no aparece

Verifique que: 1. Su usuario tiene el rol `tendero`. 2. Ha definido la variable `QROQ_API_KEY` antes de lanzar el servidor.

11. Despliegue en Producción

Para un entorno productivo, siga estas recomendaciones:

- **Servidor WSGI:** Use Gunicorn tras un proxy reverso (Nginx).
- **Seguridad:**
 - Configure `app.secret_key` mediante variable de entorno.
 - No suba credenciales al repositorio.
 - Use HTTPS.

Ejemplo de ejecución con Gunicorn:

```
1 gunicorn -w 3 -b 127.0.0.1:8000 app.main:app
```