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. Modelos de Datos (JSON) **3**
5.1. Usuarios 3
5.2. Locales 3
5.3. Proveedores 4

6. Servicios Clave **4**
6.1. AuthService 4
6.2. DBService 4

7. Rutas HTTP y API **4**
7.1. Rutas Generales 4
7.2. Rutas de Tendero 5
7.3. API del Asistente IA 5

8. Guía del Asistente IA (Chat) **5**

9. Solución de Problemas (Troubleshooting) **5**

10. Despliegue en Producción **6**

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:

```
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:

```
# Configuraci n  b sica  de  Firebase
$Env:FIREBASE_CREDENTIALS_PATH = 'C:\ruta\a\fiapp-adminsdk.json'
$Env:FIREBASE_DB_URL = 'https://fiapp-17341-default-rtdb.firebaseio.com'
$Env:USE_LOCAL_AUTH = 'false'

# Configuraci n  del  Chatbot  (OBLIGATORIO  para  ver  el  asistente)
$env:QROQ_API_KEY = '
    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'

# Ejecuci n  del  servidor
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. Modelos de Datos (JSON)

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

5.1. Usuarios

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

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

5.2. Locales

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

```
"local_jhose9282_1610000000": {
  "nombre": "Mi Tienda",
  "propietario_id": "jhose9282",
```

```
  "productos": {
    "p1": {
      "nombre": "Arroz",
      "precio": 8200,
      "stock": 10,
      "imagen_url": "/static/productos/archivo.jpg"
    }
  },
  "clientes": {
    "cliente123": {
      "nombre": "Ana",
      "deuda": 15000,
      "deudas": { "timestamp": { "monto": 5000 } }
    }
  }
}
```

Ejemplo de estructura de un Local

5.3. Proveedores

Ruta: `proveedores/{proveedor_id}`.

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

6. Servicios Clave

6.1. 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).

6.2. 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`.

7. Rutas HTTP y API

7.1. 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.

7.2. Rutas de Tendero

Prefijo: `/tendero`

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

7.3. 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'`.

8. 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"*).

9. 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.

10. 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:

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