**Banking Microservices Platform**

Este proyecto es una plataforma de microservicios bancarios implementada con **FastAPI**. Proporciona servicios independientes para:

* **Gestión de Cuentas**: creación y consulta de cuentas.
* **Préstamos**: validación y cálculo de condiciones crediticias.
* **Pagos**: facturación de servicios y transferencias.
* **Notificaciones**: envío de correos de confirmación.
* **Riesgo**: evaluación de score crediticio.

**Estructura del Proyecto**

banking-microservices/

├── api-gateway/ # Gateway y autenticación

├── loans-service/ # Servicio de préstamos

├── accounts-service/ # Servicio de cuentas

├── payments-service/ # Servicio de pagos

├── risk-service/ # Servicio de evaluación de riesgo

├── notifier-service/ # Servicio de notificaciones

├── docker-compose.yml # Orquestación de servicios

├── requirements.txt # Dependencias Python comunes

├── setup.sh # Script Bash para Linux/macOS

├── setup.bat # Script Batch para Windows CMD

└── setup.ps1 # Script PowerShell para Windows PowerShell

**Propósito del Proyecto**

Implementar un ecosistema de microservicios desacoplado y escalable para operaciones bancarias clave, facilitando mantenimiento, despliegues independientes y evolución modular.

**Tecnologías Utilizadas**

* **Python 3.9+** con **FastAPI** y **Uvicorn**
* **Docker & Docker Compose** para contenerización y orquestación
* **Pydantic** (+ **email-validator**) para validación de datos
* **httpx** para llamadas internas
* **python-jose** para autenticación JWT
* **scikit-learn** (opcional) para modelos de riesgo avanzados
* **pytest** para pruebas automatizadas
* **Postman** para pruebas manuales y colecciones
* **Git & GitHub** para control de versiones

**Prerrequisitos**

* **Docker Desktop** (Windows/Mac) o **Docker Engine & Docker Compose** (Linux)
* **Python 3.8+** instalado localmente si se desea ejecutar servicios sin Docker
* **Bash** (Linux/macOS) o **PowerShell/CMD** (Windows) para ejecutar scripts de ayuda

**Requerimientos Python**

El archivo raíz requirements.txt concentra librerías comunes:

fastapi

uvicorn[standard]

pydantic

httpx

python-jose

scikit-learn

pytest

email-validator

Para instalar todas:

pip install --upgrade pip

pip install -r requirements.txt

O bien, instala por servicio copiando requirements.txt en cada carpeta y ejecutando:

cd loans-service

pip install -r requirements.txt

**Instalación y Arranque**

**Con Docker Compose**

1. Clona el repositorio:

git clone <https://github.com/jalducin/banking-microservices.git>  
cd banking-microservices

2. Construye los contenedores (opcional antes de arrancar):

```bash

docker compose build

1. Levanta todos los servicios en segundo plano y fuerza rebuild:

docker compose up -d --build

4. Verifica el estado:

```bash

docker compose ps

**Windows:** Asegúrate de usar contenedores Linux en Docker Desktop y ejecutar en PowerShell/CMD con privilegios.

**Con Scripts de Ayuda**

* **Linux/macOS**:

chmod +x setup.sh  
./setup.sh

- \*\*Windows CMD\*\*:

```batch

setup.bat

* **Windows PowerShell**:

./setup.ps1

---

## Documentación Automática

Cada servicio expone Swagger/OpenAPI en `/docs`:

- API Gateway: `http://localhost:8000/docs`

- Loans Service: `http://localhost:8001/docs`

- Accounts Service: `http://localhost:8002/docs`

- Payments Service: `http://localhost:8003/docs`

- Risk Service: `http://localhost:8004/docs`

- Notifier Service: `http://localhost:8005/docs`

---

## Arquitectura Global

La orquestación con `docker-compose` levanta seis microservicios tras un \*\*API Gateway\*\*.

![Arquitectura Global](docs/architecture.png)

| Servicio | Puerto | Rutas principales |

|----------------------|--------|-----------------------------------------------------------|

| \*\*api-gateway\*\* | 8000 | `/health`, `/loans/validate` |

| \*\*loans-service\*\* | 8001 | `/health`, `/loans/validate`, `/loans/calculate-interest` |

| \*\*accounts-service\*\* | 8002 | `/health`, `/accounts/`, `/accounts/{id}` |

| \*\*payments-service\*\* | 8003 | `/health`, `/payments/bill`, `/payments/transfer` |

| \*\*risk-service\*\* | 8004 | `/health`, `/score/{customer\_id}` |

| \*\*notifier-service\*\* | 8005 | `/health`, `/notify/email` |

\*\*Flujo de petición\*\*:

1. Cliente → \*\*api-gateway\*\*

2. Gateway → Servicio interno

3. Servicio procesa y responde al gateway

4. Gateway retorna al cliente

---

## Diseño de Base de Datos

Modelo relacional con cinco tablas principales y relaciones:

![Diagrama ER](docs/db-diagram.png)

| Tabla | Clave(s) | Campos relevantes |

|--------------- |---------------------------------------|-------------------------------------------------- |

| \*\*accounts\*\* | `id PK` | `owner`, `balance`, `created\_at` |

| \*\*loans\*\* | `id PK`, `customer\_id FK→accounts.id` | `amount`, `term\_months`, `approved`, `calculated\_at` |

| \*\*payments\*\* | `id PK`, `from\_account FK→accounts.id`, `to\_account FK→accounts.id` | `type`, `amount`, `processed\_at` |

| \*\*notifications\*\* | `id PK` | `to\_email`, `subject`, `message`, `queued\_at` |

| \*\*risk\_scores\*\*| `customer\_id PK, FK→accounts.id` | `score`, `risk\_level`, `evaluated\_at` |

---

### Scripts SQL de Creación (`schema.sql`)

```sql

-- Tabla de Cuentas

CREATE TABLE accounts (

id VARCHAR(50) PRIMARY KEY,

owner VARCHAR(100) NOT NULL,

balance DECIMAL(12,2) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Tabla de Préstamos

CREATE TABLE loans (

id SERIAL PRIMARY KEY,

customer\_id VARCHAR(50) NOT NULL REFERENCES accounts(id),

amount DECIMAL(12,2) NOT NULL,

term\_months INT NOT NULL,

approved BOOLEAN,

calculated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Tabla de Pagos

CREATE TABLE payments (

id SERIAL PRIMARY KEY,

type VARCHAR(20) NOT NULL,

from\_account VARCHAR(50) NOT NULL REFERENCES accounts(id),

to\_account VARCHAR(50) REFERENCES accounts(id),

amount DECIMAL(12,2) NOT NULL,

processed\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Tabla de Notificaciones

CREATE TABLE notifications (

id SERIAL PRIMARY KEY,

to\_email VARCHAR(255) NOT NULL,

subject VARCHAR(200) NOT NULL,

message TEXT NOT NULL,

queued\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

-- Tabla de Puntuaciones de Riesgo

CREATE TABLE risk\_scores (

customer\_id VARCHAR(50) PRIMARY KEY REFERENCES accounts(id),

score INT NOT NULL,

risk\_level VARCHAR(10) NOT NULL,

evaluated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**Pruebas con Postman**

Para facilitar las pruebas, puedes usar el Environment creado con variables de host (p.ej. host\_gateway, host\_loans, etc.) o las URLs directas.

**1. API Gateway (localhost:8000)**

* **Health Check**
  + Método: GET
  + URL: {{host\_gateway}}/health
  + Respuesta esperada: { "status": "ok" }
* **Validate Loan (via Gateway)**
  + Método: POST
  + URL: {{host\_gateway}}/loans/validate
  + Body (JSON):
  + {
  + "customer\_id": "CUST-1001",
  + "amount": 5000,
  + "term\_months": 6
  + }
  + Respuesta: JSON con aprobación o rechazo.

**2. Loans Service (localhost:8001)**

* **Health Check**
  + GET {{host\_loans}}/health
* **Validate Loan**
  + POST {{host\_loans}}/loans/validate
  + Body igual al Gateway.
* **Calculate Interest**
  + POST {{host\_loans}}/loans/calculate-interest
  + Body (JSON): mismo formato.
  + Respuesta: { "total\_to\_pay": <number>, ... }

**3. Accounts Service (localhost:8002)**

* **Health Check**
  + GET {{host\_accounts}}/health
* **Create Account**
  + POST {{host\_accounts}}/accounts/
  + Body (JSON):
  + {
  + "id": "ACC-2001",
  + "owner": "Juan Perez",
  + "balance": 1500
  + }
* **Get Account**
  + GET {{host\_accounts}}/accounts/ACC-2001

**4. Payments Service (localhost:8003)**

* **Health Check**
  + GET {{host\_payments}}/health
* **Pay Bill**
  + POST {{host\_payments}}/payments/bill
  + Body:
  + {
  + "account\_id": "ACC-2001",
  + "biller": "Electric Company",
  + "amount": 100
  + }
* **Transfer**
  + POST {{host\_payments}}/payments/transfer
  + Body:
  + {
  + "from\_account": "ACC-2001",
  + "to\_account": "ACC-2002",
  + "amount": 50
  + }

**5. Risk Service (localhost:8004)**

* **Health Check**
  + GET {{host\_risk}}/health
* **Get Score**
  + POST {{host\_risk}}/score/CUST-1001

**6. Notifier Service (localhost:8005)**

* **Health Check**
  + GET {{host\_notify}}/health
* **Notify Email**
  + POST {{host\_notify}}/notify/email
  + Body:
  + {
  + "to\_email": "test@example.com",
  + "subject": "Test Notification",
  + "message": "Hola desde Notifier"
  + }

**Desarrollo**

* Modifica código en cada carpeta de servicio bajo app/.
* Para reconstruir un solo servicio:
* docker compose build <service-name>
* docker compose up -d <service-name>

**Buenas Prácticas**

* Almacena variables sensibles en .env, usa python-dotenv.
* Añade pruebas con pytest y configúralas en CI/CD.
* Exponer métricas y health checks mediante Prometheus/Grafana.
* Aprovecha documentación OpenAPI generada automáticamente.

**Licencia**

Este proyecto es de uso educativo y de práctica personal.

**Comandos para Actualizar en GitHub**

git add .

git commit -m "Documentación extensa y diagramas añadidos"

git push origin main