ChatGPT

Mini-Neobank-Go

Documentación Funcional y de Negocio
Versión inicial – 4 jun 2025

---

1. Visión del producto

Mini-Neobank-Go es un backend educativo que replica las piezas esenciales de un neobanco latinoamericano.
Su meta es permitir que desarrolladores entiendan **cómo** y **por qué** fluyen los fondos dentro de una entidad regulada, mientras practican las mejores técnicas de ingeniería en Go.

Objetivos de negocio

- Custodiar dinero de clientes con exactitud matemática (ledger de doble entrada).
- Cumplir requisitos KYC/AML y límites regulados de transacción.
- Monetizar mediante spread de intereses y comisiones por pagos.
- Exponer métricas operativas (P95 latencia, NIM, fraude) para decisiones ejecutivas.

Público objetivo

- Desarrolladores fintech en formación.
- Estudiantes de ingeniería de software interesados en sistemas financieros de misión crítica.
- Equipos de innovación que necesiten un prototipo de core bancario liviano.

---

2. Principios de diseño

Principio	Descripción breve	Beneficio clave
Fuente única de la verdad	Todo movimiento de dinero reside en el *ledger-service*; otros servicios no almacenan saldos persistentes.	Contabilidad consistente y auditable.
Separación de dominios	Datos personales y reglas de producto viven fuera del ledger; el ledger solo conoce IDs y montos.	Privacidad y menor acoplamiento.
Event-driven	Cada acción relevante publica un evento (`account.created`, `transfer.completed`).	Integraciones desacopladas y escalables.
Observabilidad from day-0	Logs JSON, métricas Prometheus, trazas OpenTelemetry en todos los servicios.	Detección temprana de fallos y medición de KPIs.

Principio	Descripción breve	Beneficio clave
Invariantes testeables	Pruebas fuzz/propiedad garantizan suma cero, idempotencia y atomicidad.	Confianza operativa y facilidad de auditoría.

3. Servicios principales

3.1 Modelo de doble cuenta (Cliente vs Ledger)

Aspecto	Cuenta-cliente (perfil de usuario)	Cuenta-ledger (bolsillo contable)
Propósito	Almacenar datos personales, reglas de producto, límites y estado KYC.	Registrar movimientos de dinero y mantener el balance exacto.
Quién la crea	`account-service` (Onboarding)	`ledger-service` mediante RPC `CreateAccount`.
Datos que guarda	nombre, documento, email, tipo de producto, límites, flags.	`id`, `currency`, `created_at`, flags contables (ej. `is_shadow`).
Quién la consulta	Operaciones, Customer Support, AML, Reporting.	Cualquier servicio que necesite mover dinero (Payments, Intereses, Riesgo).
Privacidad	Contiene PII → encripción + control de acceso estricto.	No contiene PII → se replica sin riesgo para auditorías.
Ciclo de vida	Puede cambiar (actualizar límites, estado KYC)	Inmutable una vez creada; el saldo se deriva de los asientos.

> **Analogía rápida:** la cuenta-cliente es tu *tarjeta de socio* (quién eres y qué plan tienes). La cuenta-ledger es la *taquilla numerada* donde guardas tu mochila (dinero). La tarjeta puede cambiar de plan; la taquilla conserva solo el número y su contenido.

Flujo de creación paso a paso

1. **Onboarding** supera KYC → crea registro `cliente_id` y solicita `CreateAccount(currency="USD")` al ledger.
2. **ledger-service** devuelve `ledger_id` y registra saldo inicial 0.
3. `account-service` vincula ambos IDs.
4. Desde ese momento, cualquier operación que implique dinero usa únicamente el `ledger_id`.

```
sequenceDiagram
  participant App as App Móvil
  participant ACC as account-service
  participant LED as ledger-service
  App->>ACC: Completa KYC (POST /clients)
  ACC->>LED: CreateAccount(currency="USD")
  LED-->>ACC: ledger_id = 550e8400...
  ACC-->>App: Cuenta creada (cliente_id+ledger_id)
```

Así se garantiza que los datos sensibles (PII) se gestionen en un dominio protegido, mientras que las operaciones monetarias ocurren en un contexto minimalista y altamente auditable.

---

3. Servicios principales

Servicio	Responsabilidad central	Protocolos / Entradas	Salidas / Eventos
ledger-service	Registrar operaciones contables de doble entrada y exponer balances en tiempo real.	gRPC v1 (CreateAccount, PostTransfer, GetBalance).	`account.created`, `transfer.completed`, métricas `ledger_*`.
account-service	Gestionar perfiles de cliente y reglas de producto; invoca al ledger para crear cuentas contables.	REST v1 (POST / clients), gRPC al ledger.	`client.onboarded`, `kyc.verified`.
payments-service	Autorización de pagos internos (P2P) y externos (ISO 8583 subset).	gRPC + ISO 8583 TCP.	`payment.authorized`, `payment.failed`.
risk-service	Motor antifraude y scoring transaccional, alimentado por eventos Kafka.	Suscripción Kafka, reglas DSL.	`risk.blocked`, `risk.approved`.
interest-service	Cálculo de intereses diarios sobre productos de ahorro; escribe asientos al ledger.	Cron / gRPC.	`interest.accrued`.
reporting-service	Dashboards financieros (NIM, CAC), generación de reportes regulatorios.	SQL/ClickHouse queries.	API Grafana panels, CSV exports.
notification-service	Enviar push/email/ SMS según eventos.	Suscripción Kafka.	Mensajes al proveedor Twilio/Firebase.

> **Nota:** Todos los servicios publican *health checks* HTTP para Kubernetes y exponen un endpoint `/metrics` Prometheus.

---

4. Flujo de programa: Transferencia P2P

A continuación se describe un flujo laboral simplificado para transferir 20 USD del cliente Laura a Juan.

1. **App móvil** → `POST /api/v1/transfers` (body: *from=Laura, to=Juan, amount=20 USD*).
2. **payments-service**:
3. Verifica que ambas cuentas estén activas.
4. Envía mensaje `risk.evaluate` a **risk-service** con hash de la transacción.
5. **risk-service** devuelve `risk.approved` (< 50 ms).
6. **payments-service** invoca `PostTransfer` en **ledger-service** (`debit=ledger_id_Laura` , `credit=ledger_id_Juan` , `amount=2000`).
7. **ledger-service**:
8. Abre transacción serializable.
9. Inserta dos asientos (+2000, −2000).
10. Confirma `sum(entries)=0` y *commit*.
11. Publica evento `transfer.completed` .
12. **payments-service** responde 200 OK a la app.
13. **notification-service** recibe `transfer.completed` y envía push *"Has enviado 20 USD a Juan"*.

Resultado → Saldos actualizados en tiempo real; auditor puede reconstruir la operación consultando solo el ledger + eventos.

---

5. Estrategia de datos clave

- **Ledger**: tablas `accounts` , `entries` , `transfers` ; saldo calculado on-the-fly (`SELECT SUM(amount)…`).
- **Data warehouse**: replicas en ClickHouse para reportes y slicing sin afectar produ.
- **Storage de compliance**: MinIO bucket para documentos KYC.

---

6. Métricas y KPIs

Indicador	Fuente	Uso de negocio
P95 latencia (PostTransfer)	`ledger_latency_seconds` histogram	Experiencia de usuario y SLA con comercios.
Net Interest Margin (NIM)	query reporting-service	Salud financiera vs spread objetivo.
Fraude %	eventos `risk.blocked` vs `transfer.completed`	Ajuste de reglas y provisión contable.
Uptime ledger	probes Kubernetes + Prometheus	Cumplimiento normativo (> 99.9 %).

---

7. Roadmap de implementación (12 semanas sugeridas)

1. **S1–S2** — Diseño y construcción del ledger-service básico.
2. **S3–S4** — Onboarding + KYC, creación integrada de cuentas.
3. **S5–S6** — Pagos internos y subset ISO 8583 para pagos externos.
4. **S7–S8** — Motor de riesgo y observabilidad completa.
5. **S9–S10** — Interés compuesto y pruebas de carga.
6. **S11–S12** — Paneles de negocio, reportes regulatorios y hardening de seguridad.

---

8. Stack tecnológico

Capa	Tecnología principal	Razón de elección
Lenguaje	Go 1.22+	Concurrencia ligera, binarios estáticos, ecosistema fintech (solidez).
Protocolo servicio-servicio	gRPC + Protobuf	Contract-first, rendimiento, streaming.
Persistencia transaccional	PostgreSQL 15	ACID maduro, extensiones financieras.
Mensajería	Kafka 3.x	Orden global y retención para eventos.
Infraestructura	Docker, Kubernetes, Helm	Deploy reproducible, escalado horizontal.
IaC	Terraform	Provisionamiento declarativo multi-cloud.
Observabilidad	Prometheus, Grafana, OpenTelemetry	Métricas, trazas y alertas unificadas.
Storage objetos	MinIO	Compatible S3 para documentos KYC.
CI/CD	GitHub Actions, Testcontainers-go	Automatización y pruebas en PR.
Análisis estático	GolangCI-Lint, Trivy	Calidad de código y seguridad de imágenes.

---

9. Servicios a desarrollar

1. **ledger-service** (corazón contable, doble entrada).
2. **account-service / onboarding** (datos de cliente, KYC).
3. **payments-service** (autorizador interno y externo).
4. **risk-service** (fraude y límites dinámicos).
5. **interest-service** (acumulación y posting diario).
6. **reporting-service** (analítica y reportes regulatorios).
7. **notification-service** (push, email, SMS).
8. **admin-portal** (frontend opcional para demo y monitoreo).

Próximos pasos inmediatos

1. Validar alcance vs. tiempo disponible.
2. Describir historias de usuario prioritarias.
3. Crear repositorio y aplicar este diseño de carpetas.
4. Configurar entorno local (Docker-Compose con Postgres + Kafka + Prometheus).
5. Iniciar desarrollo del **ledger-service** (S1–S2 del roadmap).
