Documento Explicativo — Desenho de Arquitetura Fintech
Autor: Marcos Roberto Martins
Cargo: Tech Lead / Full-Stack Engineer
Data: outubro de 2025

1 — Visão geral do sistema

Aplicação fintech para: cadastro de usuários, KYC (consultas a terceiros: CNPJ, CNH/RG), integração Open Finance (extratos), solicitação de empréstimos, e orquestração automática de pagamentos/recebimentos.

Objetivos não-funcionais principais:

- Segurança e conformidade (dados sensíveis, PCI/ISO/LGPD).
- Alta disponibilidade e tolerância a falhas.
- Latência baixa para a maioria das interações do usuário.
- Escalabilidade horizontal.
- Auditabilidade e rastreabilidade completa (logs/trace).

2 — Diagrama de alto nível

```
Mobile App (iOS / Android)
        |
   (TLS) Mobile API Gateway (WAF, Rate Limiting, Auth)
        |
   -----------------------------
   |         |         |        |
AuthSvc  API Gateway  |   Payments Orchestrator (Workflow)
(OIDC/OAuth2)         |        (Temporal/Step Functions)
   |                  |
User Service <--> KYC Service <---> Third-Party Connectors
(Profiles, MFA)      (CNPJ/CNH/DOC) (Receita, Serasa, Gov APIs)
   |
Loan Service ------> Risk/Scoring Service (ML/Rules)
   |                        |
   |                    Event Bus (Kafka)
   |                        |
Transaction Service <--> Payments Provider (PSP) / Issuing Bank
(ledger, balances)         (PIX, Card, TED, Open Banking Payments)
   |
Datastores:
 - Postgres (core transactional)
 - Redis (session, cache, rate-limits)
 - Kafka (events, audit stream)
 - S3 (document store: CNH scans, comprovantes)
 - ClickHouse / BigQuery (analytics, BI)
   |
Monitoring & Ops:
 - Prometheus + Grafana (metrics)
 - Jaeger (tracing)
 - ELK / Loki (logs)
 - PagerDuty (alerts)
 - CI/CD (GitHub Actions/GitLab CI + ArgoCD)
Kubernetes (EKS/GKE/AKS) for microservices
```

3 — Principais componentes / serviços

Frontend

- Mobile native (Swift / Kotlin) ou cross-platform (React Native/Flutter).
- Comunicação com backend via API Gateway (TLS + JWT/OIDC).

Edge / Gateway

- API Gateway (e.g., AWS API Gateway / Kong / Envoy) + WAF.
- Rate limiting, request validation, JWT validation, basic auth checks.

Autenticação / Autorização

- Identity Provider (Keycloak / Auth0 / Cognito): OIDC, OAuth2, MFA, device binding.
- Short-lived JWTs + Refresh tokens armazenados com segurança. Revoke list.

Backend (microservices)

- User Service: perfil, onboarding, documentos, KYC status.
- KYC Service: orquestra consultas a terceiros, consolida resultados, armazena evidências.
- Loan Service: solicitações, simulações, regras de elegibilidade, amortização.
- Risk/Scoring Service: regras + ML models, executados offline/online.
- Payments Orchestrator: workflow engine (Temporal / AWS Step Functions) para orquestrar recebimentos e emissões.
- Transaction / Ledger Service: livro-razão (single source of truth para saldos).
- Notification Service: push, email, SMS.
- Admin / Backoffice: painéis para operações/human review.

Mensageria / Eventos

- Kafka (ou AWS Kinesis) como backbone de eventos (event sourcing / audit stream).
- Filas de trabalho (RabbitMQ / SQS) para tarefas demoradas/async.

Cache

- Redis para sessões, locks distribuídos, contadores de rate-limit, cache de dados externos.

Storage

- Object storage (S3) para documentos e evidências; criptografia at-rest.

Analytics

- Pipelines ETL para ClickHouse / BigQuery para relatórios e Power BI.

Infra e Orquestração

- Kubernetes (EKS/GKE) + Horizontal Pod Autoscaler + Cluster Autoscaler.
- IaC: Terraform.
- Secrets: HashiCorp Vault / AWS Secrets Manager.
- Observability stack.

4 — Justificativas tecnológicas (escolhas chave)

Relacional vs Não-Relacional

- Postgres (Relacional) para dados transacionais críticos (usuários, loans, ledger). Razões: ACID, integridade referencial, facilidade de consultas complexas e suporte a stored procedures quando necessário.
- Kafka + Event Store para audit/event sourcing e desacoplamento de subsistemas.
- NoSQL / Wide-column (opcional) para cargas analíticas massivas ou time-series (ClickHouse, Cassandra). Não usar para ledger principal.

Workflow

- Temporal (ou Step Functions) para orquestração de processos longos (ex.: validação KYC -> aguardar documento -> scoring -> decisão). Isso dá visibilidade e retry automático.

Mensageria

- Kafka para alta taxa de eventos, reprocessamento, e integração com BI.

Cache

- Redis para latência microsecond-level de leitura (favorito para status de sessão e caches de consultas externas).

Kubernetes

- Padroniza deploy, facilita autoscaling, isolamento por namespaces.

Segurança

- Use serviços gerenciados para KMS, WAF e IAM para reduzir superfície de ataque.

5 — Plano de Escalabilidade

Estratégia geral

- Design stateless para API e serviços web; estado no Postgres / Redis / Kafka.
- Autoscaling horizontal em pods/instances.
- Segregar cargas: separa serviços de leitura (dashboards/BI) de escrita (transactional).

Banco de dados: particionamento e replicação

- Postgres Primary + Read Replicas para leitura (preferências para consultas analíticas).
- Partitioning por range/hash:
- Particionar tabelas grandes (transactions, payments) por created_at ou user_id modulo N.
- Sharding (se necessário):
- Em escala muito grande, shard por cliente/tenant ou por faixa de user_id — cada shard é um cluster Postgres.
- Replicação síncrona/assíncrona:
- Síncrona para criticals regionais (quando precisar RPO baixo).
- Assíncrona para réplicas de leitura.

Cache & CDN

- Redis (clustered) para cache, locks, sessions. TTL agressivo para evitar stale data.
- CDN (CloudFront) para entregar recursos estáticos e reduzir latência.

Filas & Processamento assíncrono

- Kafka para eventos de negócio e ingestão em massa.
- Worker pools consumindo de topics para fazer processamento paralelizado (scoring, notificações).

Distribuição de tráfego

- Load balancers (ALB/NGINX/Envoy) + autoscale.
- Ingress controller + API Gateway para roteamento por serviço.
- Canary / blue-green para deploys com tráfego controlado.

---

6 — Resiliência e Monitoramento

Estratégia contra falhas

- Idempotência: todos endpoints que modificam estado exigem idempotency-key (especialmente pagamentos).
- Retries com backoff exponencial: para chamadas a terceiros, com circuit-breaker.
- Circuit Breaker: Resilience4j / Envoy filters — evitar sobrecarregar sistemas terceiros.
- Bulkhead pattern: isolar recursos críticos (threads/connections) por serviço.
- Graceful shutdown: drains connections before pod termination.

Falhas de infra

- **Multi-AZ** deploy para tolerância a falhas AZ.
- **Cross-region DR**: replicação assíncrona para região secundária; testes regulares de failover.
- **Backups**:
  - Postgres: snapshots regulares + WAL archiving (PITR).
  - S3 com versioning + lifecycle.
- **Recovery**:
  - Runbooks automatizados para cenários comuns.
  - Testes de DR e simulações (chaos engineering).

Observability

- Metrics: Prometheus + Grafana — latency, error rate, queue length, DB connections.
- Tracing: Jaeger / OpenTelemetry — traçar requests across microservices (importante para troubleshooting de empréstimos/pagamentos).
- Logs: ELK / Loki para logs estruturados (JSON) e correlação por trace-id / request-id.
- Alerting: Alertmanager + PagerDuty / OpsGenie com SLAs claros (P1, P2).
- SLOs/SLIs: definir SLOs (Ex: 99.9% availability para endpoints core, p95 latency < 500ms).

Eventos duplicados e at-least-once

- Deduplication: usar idempotency-key + event dedup store (short lived) para evitar efeitos duplicados.
- Exactly-once semantics: para ledger, prefer reconcile jobs e consistência eventual com compensações.

---

7 — Padrões de desenvolvimento & processos

Processos de entrega

- IaC: Terraform + módulos versionados.
- CI/CD: Pipelines automatizados (test → build → lint → deploy). Use GitFlow ou trunk-based workflow.
- ArgoCD / Flux para GitOps (k8s manifests).

Code Quality & revisão

- Code reviews obrigatórias (2 approvers para mudanças críticas).
- Linters e formatters no pipeline (ESLint, PHP-CS-Fixer, dotnet-format).
- Static analysis e SAST (e.g., SonarQube).

Testes automatizados

- Unit tests com cobertura mínima (e.g., 70%+ para core libs).
- Integration tests para endpoints (usar testcontainers ou infra staging).
- Contract tests (Pact) para dependências entre microservices e 3rd parties (Open Banking).
- E2E tests (Cypress / Detox para mobile).
- Load tests (k6 / Gatling) integrados ao pipeline para release maior.
- Chaos testing (opcional) para validar tolerância.

Versioning

- **SemVer para services.**
- **API versioning: path versioning (/v1/, /v2/) ou header-based for backward compatibility.**

Segurança no SDLC

- Secrets scanning, dependency scanning (Dependabot), pentests regulares, e políticas de least-privilege.

---

8 — DER (Entidade-Relacionamento) — textual

Todos os campos id são UUID (recomendado) ou BIGSERIAL conforme contexto.

users

- id (PK)
- email (unique)
- phone
- name
- birth_date
- created_at
- updated_at
- status (enum: pending, active, blocked)

user_profiles

- user_id (FK -> users.id)
- address
- document_type (RG, CNH, CPF)
- document_number
- kyc_status (enum: pending, approved, declined)
- kyc_completed_at

user_documents

- id (PK)
- user_id (FK)
- document_type
- s3_key
- checksum
- uploaded_at
- kyc_result_id (FK -> kyc_results.id)

kyc_results

- id (PK)
- user_id (FK)
- provider (e.g., Receita, Serasa)
- provider_response (json)
- status
- fetched_at

accounts (contas internas / wallets)

- id (PK)
- user_id (FK)
- balance (numeric)
- currency
- created_at
- updated_at
- ledger_pointer (for event store mapping)

transactions

- id (PK)
- account_id (FK)
- amount (numeric)
- type (credit/debit)
- description
- status
- related_external_id (string) — for PSP reference
- created_at

loans

- id (PK)
- user_id (FK)
- amount_requested
- amount_approved
- term_months
- interest_rate
- status (applied, approved, funded, paid, defaulted)
- created_at
- approved_at

loan_payments (amortization schedule)

- id (PK)
- loan_id (FK)
- due_date
- amount
- status
- paid_at

payments

- id (PK)
- from_account_id (FK)
- to_account_id (FK) or external
- amount
- method (PIX, TED, CARD)
- status
- orchestrator_workflow_id
- created_at

third_party_calls

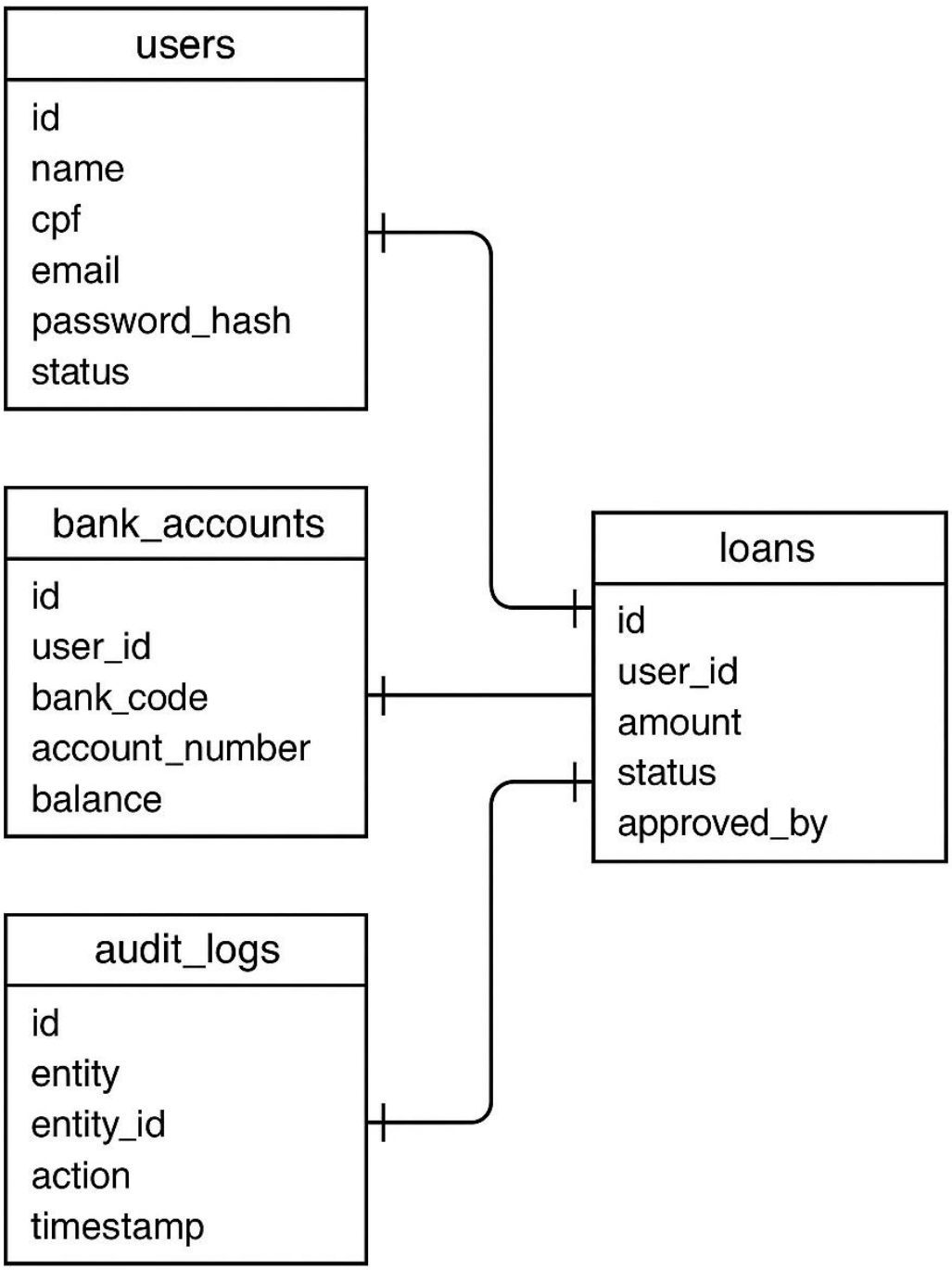
- id (PK)
- user_id (FK)
- provider
- endpoint
- request_payload (json)
- response_payload (json)
- status
- duration_ms
- called_at

audit_logs

- id (PK)
- entity
- entity_id
- action
- data (json)
- performed_by
- timestamp

Observações: para ledger, considere manter também um **immutable event store** (kafka topic + compacted store) que representa todas mudanças de saldo. A tabela transactions é uma projeção desse event store.

Modelagem da Entidade Relacional proposta apenas tabelas principais:



9 — Fluxos críticos (resumidos)

Onboarding / KYC

1. Mobile uploads docs → Mobile API → KYC service enqueues request to kyc-requests topic.
2. Worker puxa topic → chama provedores (gov APIs) com circuit-breaker → armazena evidências em S3 → atualiza kyc_results e user_profile.
3. Se resultado aprovado → disparar webhook / notification.

Solicitação de Empréstimo

4. Mobile solicita simulação → Loan Service calcula condições (utilizando Risk Service).
5. User aceita → Loan Service cria entidade loan (status = applied) e orquestra funding.
6. Funding via Payments Orchestrator → Transaction Service escreve entries no ledger (idempotent).

Emissão / Recebimento de Pagamento

- Orquestrador controla steps: reserve funds → call PSP → confirm settlement → finalize ledger. Todos steps registrados no workflow engine para retry e inspeção.

---

10 — Operações / Governança / Compliance

- **Logs e retenção**: logs de auditoria por 7 anos quando necessário (LGPD / fiscal).
- **Proteção de dados**: criptografia at-rest (KMS) e in-transit (TLS 1.2+). Tokenização de dados sensíveis.
- **Conformidade**: PCI DSS para pagamentos, requisitos Open Banking (se aplicável).
- **Privacidade**: política de consentimento explícito para leitura de extratos (Open Finance).

---

11 — Métricas a monitorar (exemplos de SLIs/SLOs)

- Disponibilidade (uptime) do endpoint /api/v1/*.
- P95 e P99 de latência de endpoints críticos.
- Taxa de erros 5xx.
- Throughput de mensagens Kafka / backlog.
- Tempo médio de processamento de KYC.
- % de retries e circuit-breaker trips.

---

12 — Requisitos opcionais

Geografia / Multi-Region

- Deploy multi-region com leitura local (réplicas read-only) e failover de escrita via leader election.
- Usar CDN e edge-services para reduzir latência global.

Otimização de custo

- Rightsizing instances, usar spot instances para workers não críticos, autoscale por métricas reais, observar custo de retenção em BI (evitar hot storage para tudo).

Alta disponibilidade 99.9%

- Multi-AZ, health checks automatizados, circuit breakers, testes DR e runbooks.

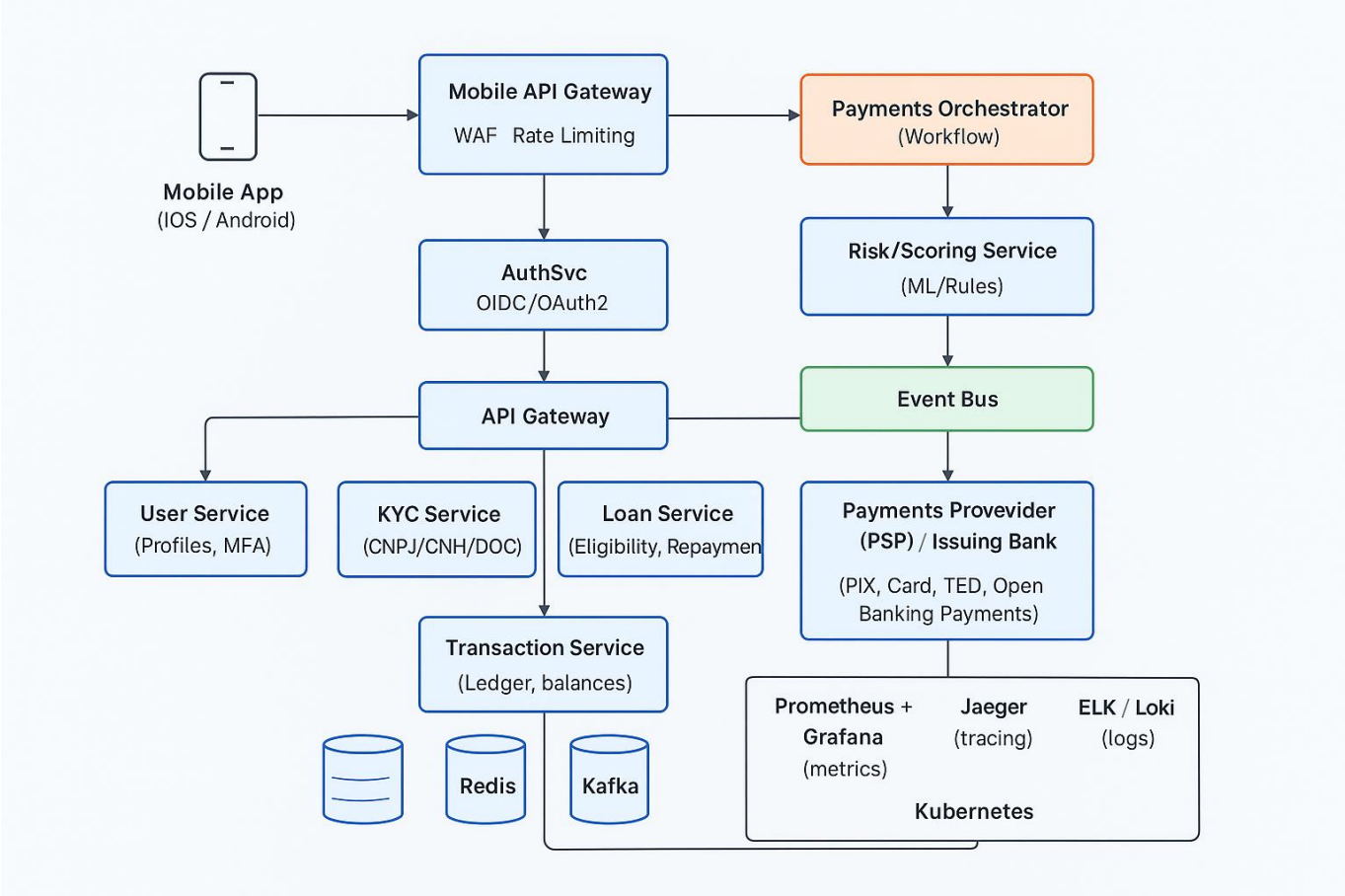
Circuit Breaker

- Implementar na camada de client libs para terceiros (Resilience4j / Envoy).

Camada de "come tolerance" (tolerância a falhas externas)

- Fallbacks, local-caches de última versão, enfileirar requests quando provedores estiverem down e informar usuário do estado.

Diagrama proposto do projeto Fintec:



Marcos Roberto Martins

Campinas Outubro de 2025.