# ■ Documento de Arquitetura – SFU (Sistema Financeiro Universitário) — Versão Final

Revisão: versão expandida incluindo modelo de dados, especificação de APIs com exemplos, fluxos de pagamento e conciliação, armazenamento, backup, segurança, observabilidade (logs, métricas e tracing), runbooks e backlog com épicos e histórias de usuário.

## 1. Introdução

O SFU é uma plataforma para gestão financeira de estudantes e da instituição, cobrindo emissão de faturas, pagamentos (PIX, boleto, cartão), conciliação, gestão de bolsas, relatórios financeiros e integração com SIS/ERP e BI. Este documento descreve a arquitetura backend proposta, requisitos não funcionais, integração, governança e backlog inicial para implementação do MVP e evolução.

## 2. Objetivo do Sistema e KPIs

Objetivo: Centralizar e automatizar a gestão financeira de estudantes e da instituição, reduzindo esforço manual e melhorando a disponibilidade de informações para tomada de decisão e conformidade legal.

**KPIs / Resultados esperados:**

- Redução de inadimplência: até 20% no primeiro ano.
- Redução de esforço manual da equipe financeira: ~30%.
- Relatórios financeiros em 'quase' tempo real (conformidade com latência dos SLAs).
- Tempo médio de conciliação automatizada: < 1h para 90% das transações (meta).

## 3. Usuários e Personas

**Perfis:** Estudante, Financeiro, Coordenação, Admin/SRE, Sistemas integrados (SIS/ERP, Gateways).

Impactos: interfaces simples para estudantes (mobile-first), interfaces ricas e auditáveis para financeiro, segregação de dados por role (RBAC), suporte a picos de carga (início do mês).

## 4. Arquitetura Backend

### 4.1 Tipo de Arquitetura e Módulos

Monolítica modular (MVP) com separação clara de responsabilidades. Módulos

principais: - Faturamento (Invoice Engine)

- Pagamentos (integração com gateways — geração de PIX/Boletos, cartões via provedores tokenizados)
- Conciliação (algoritmos de matching e regras manuais)
- Gestão de Bolsas e Descontos (regras, elegibilidade, aprovações)
- Relatórios e BI (materialized views e APIs de leitura)
- Admin & Auth (usuários, roles, auditoria, secret rotation)

### 4.2 Estrutura de Camadas (exemplo Spring Boot)

- Controller (REST): valida entrada, autentica e encaminha para Services; expõe DTOs.
- Service: orquestra regras de negócio, gerencia transações e publicações de eventos.
- Repository: acesso a dados (Spring Data JPA), queries otimizadas, paginação.
- Integration Adapters: clientes HTTP/gRPC para gateways, SIS, message brokers.

- Domain/Model: entidades, invariantes e agregados (DDD-lite).

- Infra: config, segurança, metrics, logging, health endpoints.

### 4.3 Estrutura de Pastas (proposta)

```
src/main/java/com/sfu/ ■■■ api/ (controllers, dto)
■■■ config/ (security, db, appconfig) ■■■ domain/ (entities, valueobjects)
■■■ service/ (business logic) ■■■ repository/ (jpa repositories)
■■■ integration/ (gateways, clients) ■■■ infra/ (logging, metrics, tracing)
■■■ util/ (mappers, validators) ■■■ Application.java
```

### 4.4 Modelo de Dados - Principais Entidades (resumo)

Entidades principais: Student, User, Invoice, InvoiceLine, Payment, Scholarship, ReconciliationRecord, BankSlip (Boleto), AuditLog.

Exemplo resumo de colunas para 'Invoice' e 'Payment':

```
Invoice(id UUID, student_id UUID, issue_date date, due_date date, total_amount decimal, s tatus enum, created_by UUID, created_at timestamp) Payment(id UUID, invoice_id UUID, stud ent_id UUID, amount decimal, method enum, status enum, gateway_reference varchar, receive d_at timestamp)
```

### 4.5 Estratégia de Persistência e Particionamento

Banco primário: PostgreSQL. Recomendações:

- Particionamento por range (issue_date) para tabelas Invoice e Payment quando volume

alto. - Índices compostos em (student_id, due_date), (invoice_id).

- Read replicas para consultas analíticas; materialized views para relatórios

frequentes. **5. APIs (alto nível) — Endpoints principais e exemplos**

Assumimos uso de JWT Bearer para APIs internas/externas. Documentar via OpenAPI/Swagger.

### 5.1 Autenticação / Usuários

```
POST /auth/login Body: { "username": "...", "password": "..." }
Resp: { "access_token":"", "refresh_token":"", "expires_in":3600 }
```

```
GET /auth/me Headers: Authorization: Bearer
Resp: { "id": "...", "username": "...", "roles":[ "FINANCE" ] }
```

### 5.2 Estudantes / Faturas / Pagamentos (exemplos)

```
POST /students Body: StudentDTO Resp: 201 Location: /students/{id}
```

```
POST /invoices Body: { student_id, issue_date, due_date, lines:
[{description,quantity,un it_price}], metadata } Resp: 201 { invoice_id, status }
```

```
POST /payments Body (internal/reconciler): { invoice_id?, amount, method,
gateway_referen ce?, received_at? } Resp: 201 { payment_id, status }
```

### 5.3 Webhooks e Segurança

Endpoints públicos (e.g., /webhooks/payment-gateway) devem validar assinatura HMAC + timestamp, aceitar apenas conexões TLS e registrar o evento no AuditLog antes de processar.

```
POST /webhooks/payment-gateway Headers: X-SFU-Signature: hmac, X-SFU-Timestamp:
ts Body: gateway_payload
```

### 5.4 Contratos e Versionamento

- Versionar APIs via /v1/ ou headers. Migrar com compatibilidade retroativa onde possível.
- Contratos publicados em OpenAPI; CI deve rodar validação de contrato.

# 6. Segurança e Compliance

## 6.1 Autenticação e Autorização

- OAuth2 + JWT (access + refresh). Roles: STUDENT, FINANCE, COORD, ADMIN, SRE.
- Políticas de least-privilege, scopes e claims mínimos.
- MFA para contas administrativas e operações sensíveis.

## 6.2 Proteção de Dados (LGPD/GDPR)

- Minimizar coleta; pseudonimização quando possível; consent logs para comunicação e marketing.
- Encryption at rest (disk + column-level for sensitive fields).
- Data subject requests: endpoints/processos para export/delete de dados pessoais.

## 6.3 PCI-DSS e Pagamentos

- Não armazenar PAN; usar tokenização via provedores. Comply com SAQ nível aplicável.
- Logs e traces contendo dados sensíveis devem ser mascarados.

# 7. Observabilidade, Logs e Monitoramento (detalhado)

## 7.1 Logs - formato e pipeline

Formato recomendado: JSON estruturado com campos: timestamp, level, service, env, trace_id, span_id, request_id, user_id, path, method, status_code, duration_ms, message, metadata (JSON).

```
{"timestamp":"2025-09-09T22:00:00Z","level":"INFO","service":"sfu-api","trace_id":"...","
request_id":"...","user_id":"...","path":"/invoices","duration_ms":120,"message":"Invoice
created","metadata":{"invoice_id":"inv-01"}}
```

Pipeline sugerido: Application -> Fluentd/Vector -> Elasticsearch (or Loki) -> Kibana/Grafana. Alternativa: ELK + OpenTelemetry collector -> Tempo/Jaeger for traces.

## 7.2 Métricas (Prometheus) e Dashboards (Grafana)

Métricas recomendadas:

- sfu_invoices_created_total{env}
- sfu_payments_received_total{method}
- sfu_payment_failures_total
- http_request_duration_seconds_bucket{endpoint} (histogram)
- db_query_duration_seconds

Dashboards: Overview (throughput, errors, latency), Payments (by method), Reconciliation (success rate), DB replication lag.

## 7.3 Tracing e correlação

- Usar OpenTelemetry para traces; propagar trace_id/span_id nos logs (correlação).
- Amostragem adaptativa em produção (ex.: 1-5% full traces + 100% errors).

## 7.4 Alertas e Runbooks (exemplos)

Exemplos de alertas e playbooks:
- AL: High payment failure rate (>1% por 5m) -> Runbook: validar gateway, checar filas, ativar fallback, notificar stakeholders.

- AL: DB replication lag > 30s -> Promote replica, atualizar config e validar integridade.

Runbooks devem ter passos claros, owners and escalation policies.

### 7.5 Retenção e proteção de logs

- Logs indexados: retenção 90 dias; arquivamento em S3/GCS (Glacier) por 6-12 meses conforme compliance.

- Acesso restrito; masking de dados sensíveis; auditoria de acessos aos logs.

## 8. Armazenamento, Backup e RTO/RPO

- Backups: wal streaming + base backups; snapshots diários + incremental; testar restores trimestralmente.

- RPO objetivo: <= 15 minutos; RTO objetivo: <= 2 horas.

- Estratégia: ponto-in-time recovery (PITR) para PostgreSQL, failover automatizado em caso de perda do primário.

## 9. Conciliação (Reconciliation)

Fluxo automatizado:

1) Recebe webhook -> valida assinatura -> cria Payment (status: pending) -> tenta casamentos automáticos por invoice_id, gateway_reference, amount+date fuzzy match -> se único match: mark confirmed & update invoice; se múltiplos: criar ReconciliationRecord para revisão manual.

Métricas: reconciliation_auto_rate, reconciliation_manual_count,

avg_time_to_reconcile. ## 10. Backlog Inicial (épicos e histórias com AC)

Epic: Gestão de Estudantes

- US-101: Como admin, quero criar estudantes. AC: Validação CPF/email, 201, evento 'student.created' publicado.

- US-102: Como estudante, quero listar minhas faturas. AC: Paginado, filtrar por

status. Epic: Emissão de Faturas

- US-201: Criar faturas com múltiplas linhas. AC: total computado, persistido, evento

publicado. - US-202: Cancelar fatura com histórico. AC: audit log + alteração de status.

Epic: Pagamentos e Conciliação

- US-301: Receber webhooks e registrar pagamentos. AC: validar assinatura, Payment criado, ReconciliationRecord gerado se necessário.

- US-302: Conciliar manualmente via UI/API. AC: operação registrada em AuditLog.

Epic: Bolsas e Descontos

- US-401: Aplicar bolsa fixa/percentual. AC: cálculo aplicado antes da emissão, histórico de alterações.

## 11. Integrações e Mensageria

- Gateways de pagamento: integrações resilientes com retries/exponential backoff. - SIS/ERP: sync

para dados mestres (student_id, matricula) via eventos ou integração batch overnight.

- Mensageria: usar Kafka/RabbitMQ para eventos assíncronos (invoice.created, payment.confirmed) e desacoplamento do processamento de relatórios.

## 12. CI/CD, Testes e Qualidade

- Pipeline: build -> unit tests -> static analysis -> integration tests -> build image -> security scan -> deploy to staging -> integration/e2e -> canary/prod.

- Testes: unitários (>=70% cobertura MVP), integração com Testcontainers (Postgres, Redis), e2e para fluxos críticos (pagamento).

- Security: SAST, DAST, dependency scanning (Trivy/OWASP), secret scanning. **13.**

## Operação, Runbooks e Playbooks (exemplos detalhados)

### Playbook: Pagamentos falhando em massa

1) Confirmar se falha é do gateway (logs + dashboard). 2) Validar se há mudanças em credenciais/URLs. 3) Habilitar modo manutenção de pagamentos (write-through que queueia requisições). 4) Ativar fallback para outro provedor (se existir). 5) Comunicar financeiro e stakeholders. 6) Criar post-mortem com RCA.

### Playbook: Falha completa do serviço API

1) Rota de failover: redirecionar tráfego para read-only ou fallback stateless. 2) Escalar réplicas, checar uso de CPU/mem. 3) Se DB primário comprometido, promover replica e rodar sanity checks. 4) Notificar on-call e stakeholders.

## 14. Diagramas textuais e fluxos críticos

Fluxo de pagamento resumido: Estudante -> API SFU -> Gateway -> Webhook -> Reconciliation -> Invoice update -> Event publish -> BI/ERP.

Componentes: [API], [Postgres], [Redis], [Blob Storage], [Payment Gateway], [Message Broker], [Monitoring Stack].

## 15. Anexos: Exemplos técnicos e SQL (amostra)

Exemplo DDL (simplified):

```
CREATE TABLE student ( id UUID PRIMARY KEY, registration_number VARCHAR, full_name VARCHAR, email VARCHAR, cpf VARCHAR, created_at TIMESTAMP DEFAULT now() );
```

```
CREATE TABLE invoice ( id UUID PRIMARY KEY, student_id UUID REFERENCES student(id),
  issue_date DATE, due_date DATE, total_amount NUMERIC(12,2), status VARCHAR,
  created_at TIMESTAMP DEFAULT now() );
```

Índices sugeridos: CREATE INDEX idx_invoice_student_due ON invoice(student_id, due_date); **16. Próximos Passos e Checklist para MVP**

1) Priorizar épicos do backlog (3 meses). 2) Escolher 1 gateway de pagamento para PoC. 3) Criar repositório com template de infra (IaC), pipeline CI/CD e ambiente staging. 4) Implementar observability básica (metrics + logs) e runbooks. 5) Testes de recuperação (DR) e validação de RTO/RPO.

## 17. Contatos e Responsabilidades (sugestão)

Definir owners: Product Owner, Tech Lead, SRE, Finance Lead, Compliance Lead. Incluir contatos e

SLAs internos de resposta (on-call rota).

## 18. Infraestrutura e Hospedagem

**Ambiente de Hospedagem:**

- O backend do SFU será hospedado na plataforma **Render** (https://sfu-2-r1tw.onrender.com), utilizando o plano gratuito para fins acadêmicos.
- O serviço será configurado como uma aplicação **Spring Boot (Java 17+)**, com build automatizado via GitHub.
- O deploy será feito diretamente a partir do branch principal (`main` ou `master`), com build gradle/maven e start command `java -jar app.jar`.

**Banco de Dados:**

- Para o ambiente acadêmico, será utilizado **H2 Database (modo file)**, dispensando dependência externa.
- Em ambiente de produção, a arquitetura prevê **PostgreSQL gerenciado** com backup automatizado e replicação.

**Configurações de Rede e Segurança:**

- HTTPS gerenciado pelo Render.
- Variáveis de ambiente para credenciais e chaves JWT (sem versionamento no Git).
- Monitoramento básico via logs do Render e métricas Spring Actuator.

**Escalabilidade (futuro):**

- Caso seja necessário migrar para ambientes de maior carga, o deploy pode ser movido para:
    - Docker + AWS ECS / Google Cloud Run;
    - PostgreSQL gerenciado (RDS / Cloud SQL);
    - Cache distribuído com Redis Cloud Free Tier.