DOCUMENTO-GUIA DEFINITIVO DO PROJETO

Plataforma de Monitoramento, Processamento e Segurança de Dados

(Python + Banco Relacional + API + Interface Visual)

Manual completo de execução (do zero à entrega) — Engenharia de Software + Segurança + Qualidade

1. Introdução

O propósito do projeto

- Construir uma plataforma completa para coletar, validar, processar, armazenar, auditar e visualizar dados com foco em confiabilidade, segurança e rastreabilidade ponta a ponta.
- O sistema recebe dados (integrações, uploads controlados, rotinas), aplica validações e regras de negócio, registra trilhas de auditoria, expõe API REST e oferece interface visual simples.

O que ele resolve

- Dados inconsistentes (campos faltando, formatos inválidos, duplicados, valores fora do intervalo).
- Processos manuais (planilhas, e-mails, 'copiar e colar', falta de padronização).
- Ausência de rastreabilidade (quem alterou o quê, quando e por quê).
- Falta de governança (regras, métricas de qualidade, auditoria).
- Riscos de segurança (entrada maliciosa, permissões amplas, logs pobres).

Problema real de mercado atendido

- Finanças/fintechs: processamento de transações e conciliação.
- E-commerce: eventos de pedidos, pagamentos, antifraude.
- Logística: eventos de entrega e rastreamento.
- SaaS B2B: telemetria e eventos de produto.
- SecOps: eventos, alertas e evidências.

Relevância do projeto

- Para empresas: aumenta confiança nos dados, reduz erros e cria governança.
- Para recrutadores: mostra sistema fim a fim (arquitetura, segurança, qualidade).
- Para engenharia: camadas, contratos, versionamento e consistência transacional.
- Para segurança: CIA, validação, auditoria, rastreabilidade e controle de acesso.

2. Cenário do Projeto

Empresa fictícia (realista): NorteLog DataOps

- Empresa de logística com múltiplas cidades, parceiros terceiros e necessidade de relatórios confiáveis para clientes e diretoria.

Problemas enfrentados

- Dados chegam de múltiplas fontes e com formatos variáveis.
- Duplicidade de eventos e inconsistência de estados.
- Sem trilha de auditoria e sem controle de acesso adequado.
- Risco de manipulação de status e falhas operacionais.

Necessidade

Centralizar ingestão, validar e normalizar, separar dado bruto de dado confiável, auditar alterações, proteger acessos, expor API e visualizar indicadores.

3. Visão Geral da Arquitetura

Arquitetura em camadas

- Front-end simples: dashboards e consultas.
- API (Python): contratos REST, validação e autenticação.
- Processamento: normalização, validação semântica, deduplicação.
- Banco relacional: integridade referencial, histórico e auditoria.
- Segurança/Observabilidade: RBAC, logs estruturados e eventos de segurança.

Fluxo de dados

Entrada → validação sintática → registro RAW → normalização → validação semântica → deduplicação → persistência TRUSTED → rejeições/auditoria → visualização.

Princípios

- Baixo acoplamento e alta coesão por módulos.
- Escalabilidade via índices, paginação, limites e isolamento de falhas.
- Separação clara de responsabilidades entre front, API, processamento, banco e segurança.

4. Fase 1 – Levantamento de Requisitos

Entregáveis

- Documento de requisitos (RF, RNF, regras de negócio).
- User stories com critérios de aceitação.
- Casos de uso descritos.
- Definição de zonas: RAW, TRUSTED e REJEIÇÕES.

Objetivo geral e específicos

- Objetivo geral: plataforma que ingere, valida, processa, armazena com integridade, audita, protege e visualiza dados via API e front simples.
- Objetivos específicos: padronização, integridade, deduplicação, auditoria, RBAC, indicadores e rastreabilidade.

Requisitos Funcionais (exemplos)

- RF01 Ingestão autenticada; RF02 validação; RF03 normalização; RF04 deduplicação.
- RF05 persistir RAW; RF06 persistir TRUSTED; RF07 registrar rejeições.
- RF08 consulta paginada; RF09 indicadores; RF10 auditoria; RF11 RBAC; RF12 eventos de segurança.

Requisitos Não Funcionais (exemplos)

- Segurança, performance, disponibilidade, manutenibilidade, observabilidade, confiabilidade transacional e portabilidade.

User stories (exemplos)

- US01 Ingestão segura; US02 visão de qualidade; US03 auditoria completa.

Casos de uso (exemplos)

- UC01 receber/processar; UC02 consultar confiáveis; UC03 auditar mudanças.

5. Fase 2 – Modelagem e Planejamento Técnico

Modelagem de dados e zonas

- RAW: payload original + metadados; TRUSTED: dados normalizados relacionais; REJEIÇÕES: motivos; AUDITORIA: antes/depois; SEGURANÇA: eventos.

Entidades principais (visão ER)

- source_system, raw_ingestion, trusted_event, rejection, user_account, role/permission, audit_log, security_event.

Normalização e integridade

- Aplicar 1FN/2FN/3FN; desnormalizar apenas com justificativa e controle.
- Definir PKs estáveis, FKs, índices (origem, data, status, id externo, hash) e políticas de retenção.

Estrutura do projeto

- Camadas: apresentação (API), aplicação (serviços), domínio, infraestrutura, segurança e observabilidade.
- Padrões de nomenclatura e contratos claros.

6. Fase 3 – Lógica, Algoritmos e Processamento

Pipeline de processamento

Recepção → validação sintática → registro RAW → normalização → validação semântica → deduplicação → persistência TRUSTED → rejeições → auditoria/segurança.

Validações e erros comuns

- Separar validação de contrato vs validação de negócio.

- Registrar rejeições com campo, categoria, severidade e contexto.
- Evitar validar apenas no front; evitar mensagens genéricas; nunca rejeitar sem evidência RAW.

Tratamento de erros

- Distinguir erro do cliente (entrada inválida) de erro interno (servidor).
- Respostas seguras sem vazamento; logs estruturados; isolamento por requisição.

Análise de complexidade (Big-O)

- Deduplicação deve usar chave/índice; evitar comparação 'um contra todos'.
- Paginação e índices para consultas; retenção para tabelas volumosas (RAW/auditoria).

7. Fase 4 – Segurança da Informação

CIA (Confidencialidade, Integridade, Disponibilidade)

- Confidencialidade: RBAC, minimização de dados e segredos fora do código.
- Integridade: validações, transações, integridade referencial e auditoria.
- Disponibilidade: limites, tolerância a falhas e observabilidade.

Boas práticas

- Autenticação forte (tokens, expiração, rotação).
- Autorização por papéis e menor privilégio.
- Logs com correlação (request_id), sem dados sensíveis desnecessários.

8. Fase 5 – API e Integrações

Conceito e padrões

- API REST como contrato oficial para ingestão, consultas, rejeições, auditoria e métricas.
- Padrões: recursos bem nomeados, paginação obrigatória, respostas consistentes e idempotência.

Documentação e versionamento

- Versionar API para evitar quebra de integrações.
- Documentar via OpenAPI/Swagger: endpoints, parâmetros, validações, retornos e permissões.

9. Fase 6 – Front-End (Visualização)

Objetivo

- Acompanhar operação e qualidade com interface simples: KPIs, tabelas e páginas de detalhe.

Diretrizes de UX

- Filtros claros, paginação visível, estados de carregamento/vazio, consistência de termos e foco em clareza.

10. Fase 7 – Entrega e Qualidade

Docker e reprodutibilidade

- Empacotar API, banco e front (se aplicável) com variáveis de ambiente e scripts de migração controlados.

README e versionamento

- README com visão, arquitetura, execução, configuração, testes, permissões e roadmap.
- Commits pequenos, tags de release e changelog.

Testes

- Unitários (validações/regras), integração (fluxos), e segurança (acessos negados, limites).

11. Fase 8 – Evolução e Escalabilidade

Evoluções

- Processamento assíncrono, alertas automáticos, quarentena de eventos suspeitos, exportação para BI e enriquecimento.

Escalabilidade e produção

- Índices, particionamento por data, múltiplas instâncias, retenção/arquivamento, backups e hardening.

Integração futura com IA

- Classificação de rejeições, detecção de anomalias e sumarização de auditoria.

12. Indicadores de Qualidade

KPIs técnicos

- Throughput, latência (média e p95/p99), taxa de erro, disponibilidade.

Qualidade dos dados

- Taxa de rejeição (geral e por origem), top regras violadas, duplicidade e consistência de transições.

Segurança e manutenibilidade

- Tentativas de autenticação falhas, acessos negados, anomalias, cobertura de testes e complexidade em módulos críticos.

13. Conclusão

Valor do projeto

- Sistema fim a fim com pipeline RAW→TRUSTED→REJEIÇÕES, auditoria, segurança, API documentada e visualização operacional.

Diferencial profissional

- Vai além de CRUD: governança de dados, evidências, rastreabilidade e métricas, alinhado a casos reais de mercado.