Svelto SaaS - Blueprint de Arquitetura Backend (v3.0 - "The Core")

Versão do Documento: 3.0
Status da Arquitetura: Estável / Em Produção (Milestone 3 Iniciado)
Stack: NestJS (Node.js LTS), Prisma ORM (v5.22.0), PostgreSQL (Railway)

1. Visão Geral da Arquitetura

O Backend do Svelto não é apenas uma API REST; é um **Motor de Processamento Financeiro Assíncrono** projetado para conciliar grandes volumes de dados com integridade forense. A arquitetura segue os princípios de **Clean Architecture** adaptada para o ecossistema NestJS, com forte separação entre a camada de Ingestão (Providers), a camada de Lógica de Negócio (Services) e a camada de Persistência (Prisma).

1.1. Princípios Fundamentais

1. **Multi-Tenancy Lógico:** Banco de dados compartilhado (Shared Database), isolamento lógico via coluna tenantId em todas as tabelas críticas.
2. **Integridade Financeira (Decimal Math):** Proibido o uso de Float. Todos os valores monetários utilizam Decimal(19,4) no banco e Big.js ou similar nas operações críticas.
3. **Segurança de Credenciais:** Tokens de terceiros (MP/Omie) nunca são salvos em texto plano. Utilizamos **Envelope Encryption** (AES-256-GCM) na camada de aplicação.
4. **Resiliência a Falhas:** Operações de escrita em massa (Sync) utilizam **Batch Processing** (Lotes de 50) para evitar timeouts e deadlocks.

2. Modelagem de Dados (Schema & Source of Truth)

A base do sistema é a estrutura da **"Quádrupla Verdade"**, refletida no schema.prisma.

2.1. Entidades Core

- **Transaction (Verdade Operacional):**
  - Representa a venda capturada no Gateway (Mercado Pago).
  - **PK Lógica:** gatewayId (String).
  - **Característica:** Imutável após conciliação (State Locking), exceto em caso de Chargeback.
  - **Fee Explosion:** Armazena taxas decompostas (amountMdrFee, amountFinancingFee, etc.) para auditoria.
- **ErpReceivable (Verdade Contábil):**
  - Espelho local dos títulos do Omie (Contas a Receber).
  - **PK Lógica:** erpId (ID numérico do Omie convertido para String).
  - **Chave de Ouro:** erpNsu (Armazena o ID da transação original).

    - **Filtro Bancário:** bankAccountId (String) permite segregar títulos por conta corrente.
- **Integration (Configuração):**
    - Armazena credenciais cifradas (credentials) e configurações de estado (settings JSONB) como lastSync.

2.2. Enumeração de Estados (TransactionStatus)

O fluxo de vida financeiro é estrito:

1. PENDING: Ingerido, aguardando par.
2. MATCHED: Par encontrado (NSU ou Fuzzy), aguardando baixa.
3. CONCILIATED: Baixado no ERP (Sucesso final).
4. DIVERGENT: Par encontrado, mas valores não batem (centavos).
5. CHARGEBACK: Estorno/Disputa (Estado de Alerta).
6. ERROR_SYNC: Falha técnica na ingestão/processamento.

3. Motores de Ingestão (Data Ingestion Engines)

3.1. Mercado Pago Engine (MercadoPagoService + IntegrationsController)

- **Estratégia de Busca:**
    - **Carga Inicial:** Busca por date_created (Histórico imutável).
    - **Incremental:** Busca por date_last_updated (Captura mudanças de status).
- **Batch Processing:**
    - O Controller orquestra a gravação usando prisma.$transaction em loops de 50 registros.
    - **Motivo:** Evitar latência excessiva e estouro de memória no driver do PostgreSQL.
- **Mapeamento:**
    - Implementa o método mapToTransactionSchema que explode o array fee_details em colunas individuais para análise de rentabilidade.

3.2. Omie Engine (OmieSyncService)

- **Estratégia de Busca:**
    - Loop do...while paginado (50 registros/página).
    - Filtros fixos: filtrar_apenas_alteracao = 'S', exibir_obs = 'S'.
- **Suporte a Full Sync:**
    - Capacidade de ignorar o cursor lastSync e forçar uma releitura desde uma data de corte (01/01/2025).
- **Tratamento de Dados:**
    - Conversão robusta de datas dd/mm/yyyy para ISO.
    - Captura do codigo_conta_corrente no JSON bruto para inteligência de filtro.

4. Motor de Conciliação ("NSU Hunter" v15)

O cérebro do sistema, residente em ConciliationService. O algoritmo foi refinado para eliminar falsos positivos.

4.1. Estratégia 1: Hard Match (NSU) - Prioridade Máxima

- **Lógica:** Transaction.gatewayId === ErpReceivable.erpNsu.
- **Sanitização:** Aplicação de .trim() e conversão para String em ambos os lados.
- **Segurança:** Verifica duplicidade (se o título já foi usado por outra transação).
- **Resultado:** Confiabilidade de 100%. Transita para status MATCHED.

4.2. Estratégia 2: Fuzzy Match (Fallback)

- **Gatilho:** Apenas se o Hard Match falhar.
- **Critérios (AND):**
  1. **Valor Exato:** amountGross idêntico (Decimal).
  2. **Janela Temporal Estendida:** Data do Evento -2 dias a +7 dias (Cobre atrasos de emissão de NF e fusos horários).
  3. **Filtro Bancário:** Validação estrita do bankAccountId (String) configurado na Integração.
- **Regra de Desambiguação:** O Match só ocorre se houver **exatamente 1 candidato**. Se houver >1, o sistema loga "Ambiguidade" e mantém como PENDING.

5. Scheduler & Automação

O sistema opera de forma autônoma via SyncScheduler.

- **Frequência:** A cada 1 hora (CronExpression.EVERY_HOUR).
- **Escopo:** Itera sobre todas as integrações com status: ACTIVE.
- **Modo:** Sempre Incremental (baseado no cursor lastSync salvo no banco).
- **Locking:** Variável isJobRunning em memória previne sobreposição de jobs se o processamento demorar mais de 1h.

6. Endpoints da API (Superfície de Ataque)

IntegrationsController

- POST /integrations: Criação (Criptografa token).
- PATCH /integrations/:id: Rotação de credenciais e correção de Tenant.
- POST /integrations/:id/sync: Disparo manual do Sync MP (Batch 50).
- POST /integrations/:id/sync-omie: Disparo manual do Sync Omie.
- POST /integrations/:id/match: Disparo manual do "NSU Hunter".

TransactionsController

- GET /transactions: Listagem paginada com filtros (Status, Busca Texto).
- GET /transactions/dashboard: Agregação de KPIs para o Frontend (Count/Sum por Status).

7. Diretrizes de Desenvolvimento Futuro

1. **Escrita no ERP (Milestone 3):**
    - Ao implementar a baixa, usar a API LancarRecebimento do Omie.
    - **Obrigatório:** Enviar valor (Líquido) e desconto (Taxa MDR) separados para garantir o fechamento do caixa correto.
    - **Idempotência:** Tratar erro "Título já baixado" como sucesso.
2. **Monitoramento:**
    - Adicionar logs estruturados em caso de falha crítica nos Schedulers.
    - Monitorar o tamanho da tabela AuditLog (previsão de particionamento futuro).
3. **Performance:**
    - Manter o Batch Size em 50 para operações de escrita.
    - Adicionar índices compostos no Prisma ([tenantId, status, dateEvent]) conforme o volume crescer.

O que fazer agora?

Você tem em mãos:
1. **Blueprint Frontend** (Interface de Alta Fidelidade).
2. **Manual UI/UX** (Regras de Design Financeiro).
3. **Blueprint Backend** (Este documento - O Motor).
4. **Código Validado** (NSU Hunter v15, Batch Controller).

O próximo passo lógico é **Implementar a Ação de Baixa ("Botão Vermelho")** no Backend, conectar com o botão que criamos na interface Web, e fechar o ciclo completo: **Venda -> Conciliação -> Baixa no ERP**.