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). O Svelto é um middleware de Engenharia Financeira Forense. Ele não apenas conecta pontas; ele espelha, audita e reconcilia dados entre Gateways e ERPs.

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.
  - **Colunas Críticas:**
    - amountGross: Valor pago pelo cliente.

        - amountMdrFee: Taxa administrativa explícita.
        - amountFinancingFee: Custo de parcelamento (lojista).
        - amountNetGateway: O líquido esperado.
        -
- **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 que conecta com o gatewayID da tabela Transaction).
    - Propósito: Permitir algoritmos de "Match" local (SQL) sem depender da latência da API do Omie.
    - Colunas Críticas:
    - erpExternalRef: Número do Pedido, ajuda no debug.
    - amountValue: Valor do título.
    - status:
        - "EM ABERTO"
        - “COMBINADO” : quando combinado com um registro da transaction
        - "RECEBIDO": efetuado recebimento com sucesso.
        - “CONCILIADO”: depois do recebimento, e com a conciliação do lançamento na conta corrente dentro do ERP
    - **Filtro Bancário:** bankAccountId (String) permite segregar títulos por conta corrente.

- **Integration (Configuração):**
    - Armazena credenciais cifradas (credentials) e configurações de estado (settings JSONB) como lastSync para cada service relacionado àquela integração.
    - Settings: JSON que armazena o "De-Para" bancário e o timestamp do lastSync.

  Exemplo de settings de um registro de integração do mercado pago:

```
{
        "lastSync": {
                "transactions": "2026-01-18T19:35:36.201Z"
  },
      "omieBankAccount": {
      "nCodCC": 7277285643,
      "descricao": "Conta Mercado Pago"
  }
}
```

  Exemplo de settings de um registro de integração do Omie:

```
{
    "lastSync": {
        "receivables": "2026-01-18T17:41:58.619Z"
    }
}
```

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.
- Provider: MercadoPagoService.
- Lógica: Paginação automática (loop while hasMore).
- Inteligência: Antes de salvar, verifica o status atual no banco. Se o usuário conciliou
- manualmente, o Sync respeita e não sobrescreve (exceto desastres)

3.2. Omie Engine (OmieSyncService)

- **Estratégia de Busca:**
  - Loop do...while paginado (999 registros/página).

- **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.
- Provider: OmieSyncService
- Objetivo: Manter o Svelto atualizado com a realidade do ERP (se alguém baixou um título
- manualmente no Omie, o Svelto deve saber).

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.

8. Guia de Setup e Comandos (Dev Environment)

Infraestrutura Local:

● Node.js: v20 (LTS).

● Gerenciador: npm.

● Banco: PostgreSQL (via Railway).

Comandos de Rotina:

● Atualizar Schema do Banco:

○ Cuidado: No Windows, evite npx global.

○ Comando: .\node_modules\.bin\prisma db push

● Ver Dados (Prisma Studio):

○ Comando: .\node_modules\.bin\prisma studio

● Rodar Backend (API):

○ Pasta: apps/api

○ Comando: npm run start:dev

○ Porta: http://localhost:3000

● Rodar Frontend (Web):

○ Pasta: apps/web

○ Comando: npm run dev

○ Porta: http://localhost:3001

Variáveis de Ambiente (.env na API):

DATABASE_URL="postgresql://postgres:gEEaiOzJTaNBdjAMnVjmXCcxBzeqavJL@shortline.proxy.rlwy.net:19521/railway"

REDIS_URL="redis://default:RgptNHHWRPPeUoJMFncseqxbMCPWhOAv@caboose.proxy.rlwy.net:29678" MASTER_KEY="<GERAR_UMA_HASH_DE_32_BYTES_HEX>"