# Blueprint de Arquitetura Frontend: Sistema Operacional Financeiro Svelto SaaS (v3.0)

## 1. Introdução: O Imperativo da Fidelidade Financeira e a "Quádrupla Verdade"

A concepção da interface do usuário para o Svelto SaaS não obedece às heurísticas tradicionais de design de produtos de consumo, onde o engajamento e a retenção são as métricas norteadoras. No domínio da conciliação financeira automatizada B2B (Business-to-Business), a interface assume o papel crítico de um **Agente de Conformidade Visual**. O objetivo primordial é a *Fidelidade Financeira*: a capacidade do sistema de representar, com precisão matemática e clareza forense, o estado dos ativos de uma empresa através de múltiplos sistemas distribuídos e frequentemente discordantes.[1]

A arquitetura frontend detalhada neste blueprint foi desenvolvida para suportar a complexidade do motor de conciliação "NSU Hunter", que opera sob o paradigma da "Quádrupla Verdade". Este paradigma dita que cada transação financeira não possui um estado único, mas existe simultaneamente em quatro dimensões: a Verdade Operacional (Gateway/Mercado Pago), a Verdade Contábil (ERP/Omie), a Verdade Financeira (Auditoria Svelto) e a Verdade Bancária (Liquidação Efetiva).[1] O desafio de engenharia frontend reside em apresentar essas quatro realidades em uma interface unificada, sem mascarar as discrepâncias que são, por natureza, o foco do trabalho do auditor financeiro.

A escolha da stack tecnológica — Next.js no modelo App Router combinado com Tailwind CSS — não é meramente uma preferência estética, mas uma decisão estratégica de arquitetura. A necessidade de renderização de milhares de linhas de transações com performance nativa, a exigência de segurança na manipulação de chaves de API (via Server Components) e a demanda por feedbacks visuais imediatos em operações de escrita (via Server Actions e Optimistic UI) tornam esta combinação a única viável para escalar o Svelto para o marco de 10.000 clientes (tenants) previsto no roadmap.[1]

Este documento serve como o manual definitivo para a implementação do frontend do Svelto, transcendendo a definição de layouts para estabelecer as regras de engenharia de interface, padrões de interação para tratamento de latência em APIs legadas e diretrizes de acessibilidade para ambientes de alta densidade de dados.

---

# 2. Fundamentos da Engenharia de Interface e Stack Tecnológica

Para atingir a robustez necessária a um "Sistema Operacional Financeiro", a arquitetura rejeita a abordagem de Single Page Application (SPA) puramente client-side em favor de uma arquitetura híbrida server-centric, alavancando o modelo mental do React Server Components (RSC).

## 2.1 O Modelo Mental do Next.js App Router para Finanças

A natureza dos dados financeiros exige uma estratégia de *Data Fetching* que priorize a integridade e a segurança sobre a interatividade superficial. O Svelto adota uma divisão rigorosa entre componentes de servidor e cliente:

- **Server Components como Padrão (80% do Código):** A busca de dados financeiros — seja a consulta de transações no Mercado Pago ou a leitura de títulos no Omie — ocorre exclusivamente no servidor. Isso isola a lógica de negócios e as credenciais de API (chaves criptografadas via Envelope Encryption) longe do navegador do cliente, mitigando vetores de ataque XSS (Cross-Site Scripting).[1] Além disso, permite que o Prisma ORM execute queries complexas diretamente no banco de dados PostgreSQL, reduzindo a latência de rede ("network waterfall") típica de dashboards que fazem múltiplas requisições API client-side.
- **Client Components como "Ilhas de Interatividade":** O uso da diretiva "use client" é restrito a componentes que exigem manipulação direta do DOM ou estado local volátil. No Svelto, isso se manifesta principalmente no **Data Grid Interativo** (TanStack Table), nos formulários de configuração de taxas e nos componentes de feedback visual (Toasts e Modais). Esta separação garante que o "peso" do JavaScript enviado ao navegador seja minimizado, mantendo o *First Contentful Paint* (FCP) instantâneo mesmo em conexões instáveis.[4]
- **Streaming e Suspense:** Dado que as APIs de terceiros (como a do Omie) podem ter tempos de resposta variáveis (2-3 segundos), a interface não pode bloquear o render total. Utilizamos React.Suspense para renderizar instantaneamente a "Shell" da aplicação (Sidebar, Header e Estrutura do Grid) e exibir *Skeleton Screens* (esqueletos de carregamento) nas áreas de dados, proporcionando uma percepção de velocidade e fluidez operacional.[6]

## 2.2 Tailwind CSS: A Lógica Utilitária na Alta Densidade

A escolha do Tailwind CSS é fundamentada na necessidade de controle microscópico sobre o layout. Em interfaces financeiras, a diferença entre padding-4 (16px) e padding-2 (8px) pode determinar se uma coluna crítica de valores é visível ou truncada em um monitor de laptop corporativo.

- **Design System Tokenizado:** O Svelto não utiliza valores arbitrários. Todas as cores,

espaçamentos e tipografia são definidos no tailwind.config.ts, garantindo consistência visual. A paleta de cores é estendida para incluir as cores de marca dos parceiros (Mercado Pago e Omie) e cores semânticas para estados financeiros (Conciliado, Divergente, Chargeback).[8]

- **Responsividade Orientada a Dados:** Diferente de sites de marketing, a responsividade do Svelto não foca em empilhar elementos para mobile, mas em adaptar a densidade da tabela. Utilizamos utilitários como hidden md:table-cell para priorizar colunas essenciais (Valor Líquido, Status) em telas menores, enquanto revelamos metadados técnicos (NSU, IDs de Lote) apenas em viewports maiores (lg ou xl).[10]

### 2.3 Gestão de Estado: URL como Fonte da Verdade

Para um sistema de auditoria, o estado da interface deve ser compartilhável e persistente. Se um auditor encontra uma discrepância na página 5 de um relatório filtrado por "Chargebacks de Janeiro", ele deve ser capaz de copiar a URL e enviá-la a um colega, que verá exatamente a mesma tela.

- **Nuqs (Type-safe Search Params):** Abandonamos o useState para filtros de tabela em favor da biblioteca nuqs. O estado de paginação (page=5), ordenação (sort=date_desc) e filtros (status=chargeback) é sincronizado automaticamente com a URL. Isso garante que o botão "Voltar" do navegador funcione como esperado e que o estado da aplicação seja imune a recargas de página (F5).[11]
- **Cookies para Preferências Globais:** Estados que não devem afetar a URL, mas devem persistir entre sessões (como o estado "Colapsado/Expandido" da Sidebar ou a preferência por "Modo Escuro"), são armazenados em Cookies httpOnly. Isso permite que o servidor leia essa preferência antes de renderizar o HTML, evitando o *Layout Shift* (mudança brusca de layout) durante a hidratação.[13]

---

## 3. Design System Financeiro: Estética da Precisão

O Design System do Svelto, denominado "Ledger UI", é construído sobre os pilares de **Legibilidade**, **Contraste** e **Semântica**. A estética serve à funcionalidade: cada pixel deve contribuir para a clareza do dado financeiro.

### 3.1 Tipografia: O Imperativo Tabular

A escolha tipográfica é a decisão de design mais crítica em um software financeiro. Fontes sans-serif modernas, embora elegantes, geralmente usam larguras proporcionais para números (o glifo "1" ocupa menos espaço que o "8"). Em tabelas financeiras, isso causa o desalinhamento vertical de casas decimais e separadores de milhar, impedindo a comparação visual rápida de ordens de magnitude.

- **Stack Tipográfica:**
  - **Interface Geral:** Inter (via next/font/google) para rótulos, botões e textos descritivos,

oferecendo excelente legibilidade em tamanhos pequenos.
    - **Dados Numéricos e Identificadores:** JetBrains Mono ou Geist Mono para todos os valores monetários, datas, NSUs, CNPJs e IDs de transação.
- **A Classe tabular-nums:** É mandatório o uso da classe Tailwind tabular-nums (que aplica font-variant-numeric: tabular-nums) em qualquer contêiner de dados financeiros. Isso força a fonte a usar glifos de largura fixa para dígitos, garantindo que, em uma lista de 500 transações, a vírgula decimal esteja sempre na mesma coluna vertical pixel-perfect.[1]
- **Escala e Peso:** O tamanho base para dados em tabelas é text-sm (14px). Metadados secundários (como a hora exata de uma transação abaixo da data) são renderizados em text-xs (12px) e cor text-slate-500 para estabelecer hierarquia visual sem poluição.

## 3.2 Paleta de Cores Semântica e Acessível

O sistema de cores do Svelto deve comunicar estados complexos instantaneamente. A paleta foi desenvolvida considerando a acessibilidade para usuários com daltonismo (especialmente deuteranopia, comum em homens), evitando a dependência exclusiva de Vermelho vs. Verde para sinalizar erros ou sucessos.

### Tabela de Cores de Marca e Integração

18

| Entidade | Cor Primária (Hex) | Nome Tailwind (Aproximado) | Uso na Interface |
|---|---|---|---|
| **Svelto (Marca)** | #0F172A | slate-900 | Sidebar, Cabeçalhos, Elementos Neutros |
| **Mercado Pago** | #00BCFF | sky-500 | Badges de Origem, Links para MP, Ícones de Gateway |
| **Omie ERP** | #00E3F5 | cyan-400 | Badges de Destino, Links para ERP, Ícones de Contabilidade |
| **Banco/Liquidez** | #22C55E | green-500 | Indicadores de |

| | | | |
|---|---|---|---|
| | | | Liquidação Efetiva, Dinheiro em Conta |

### Tabela de Cores de Estado (Semântica Financeira)

1

| Estado | Background (Tailwind) | Texto (Tailwind) | Ícone (Lucide) | Significado Operacional |
|---|---|---|---|---|
| **CONCILIADO** | bg-emerald-50 | text-emerald-700 | CheckCircle2 | Match perfeito. "Verdade Quádrupla" alinhada. |
| **DIVERGENTE** | bg-amber-50 | text-amber-700 | AlertTriangle | Match de ID, mas valor difere (taxas incorretas). Ação requerida. |
| **PENDENTE** | bg-slate-100 | text-slate-700 | Clock | Aguardando processamento ou janela temporal. |
| **CHARGEBACK** | bg-rose-50 | text-rose-700 | ShieldAlert | Disputa aberta. Risco financeiro imediato. |
| **ERRO SYNC** | bg-red-100 | text-red-800 | XCircle | Falha técnica na comunicação API (Timeout/Auth) |

| | | | | |
|---|---|---|---|---|
| | | | | . |

*Nota de Implementação:* Para o Modo Escuro (dark:), as cores de fundo devem ser ajustadas para versões translúcidas com opacidade (ex: dark:bg-emerald-900/30), mantendo o texto em tons pastéis mais claros (dark:text-emerald-400) para garantir o contraste conforme diretrizes WCAG AA.[22]

### 3.3 Iconografia e Linguagem Visual

A biblioteca **Lucide React** é adotada por sua consistência de traço e leveza (SVG puro). A iconografia serve como âncora cognitiva para diferenciar as origens dos dados na visualização da "Quádrupla Verdade".

- **Mapeamento de Ícones:**
  - Transação/Venda: CreditCard
  - Lote de Depósito (Settlement): Landmark
  - Entidade ERP (Título): FileText ou Database
  - Ação de Conciliar: Link ou GitMerge (para fusão de dados)
  - Ação de Baixa (Write-back): ArrowDownToLine (metáfora de "baixar" para o livro razão).[24]

---

## 4. Arquitetura de Navegação e Estrutura de Diretórios

A estrutura de diretórios do projeto reflete a arquitetura multi-tenant e a separação de responsabilidades. O Svelto utiliza *Route Groups* para organizar layouts distintos sem afetar a estrutura da URL.

### 4.1 Estrutura de Pastas (Blueprint Detalhado)

[26]

```
src/
├── app/
│   ├── (auth)/                 # Grupo de rotas de Autenticação (Layout Clean)
│   │   ├── login/
│   │   │   └── page.tsx
```

```
│   │   ├── recovery/
│   │   └── layout.tsx            # Layout sem sidebar, focado no formulário
│   ├── (dashboard)/              # Grupo principal da aplicação (Layout App)
│   │   ├── [tenantId]/           # Parâmetro dinâmico de Tenant (Isolamento de Dados)
│   │   │   ├── dashboard/        # "Cockpit" Operacional
│   │   │   │   ├── page.tsx      # Server Component
│   │   │   │   └── loading.tsx   # Skeleton Loader específico
│   │   │   ├── transactions/     # Listagem Geral (Forensic View)
│   │   │   ├── reconciliation/   # Interface do "NSU Hunter"
│   │   │   ├── settings/         # Configurações e Fee Engine
│   │   │   └── layout.tsx        # Layout Persistente (Sidebar + Header + Contexto)
│   │   └── layout.tsx            # Root Layout do Dashboard (Providers)
│   ├── api/                      # Route Handlers (Webhooks, Cron Jobs)
│   │   ├── webhooks/
│   │   │   ├── mercadopago/
│   │   │   └── omie/
│   │   └── cron/                 # Endpoints para o Scheduler de Sync
│   ├── globals.css               # Diretivas Tailwind e CSS Variables
│   └── layout.tsx                # Root Layout da Aplicação (HTML/Body)
├── components/
│   ├── data-grid/                # Componentes do TanStack Table (complexos)
│   │   ├── data-table.tsx        # Wrapper principal
│   │   ├── columns.tsx           # Definições de colunas
│   │   ├── data-table-toolbar.tsx  # Filtros e Ações em Lote
│   │   └── truth-row.tsx         # Componente da "Quádrupla Verdade" expandida
│   ├── financial/                # Componentes de Domínio
│   │   ├── money-input.tsx       # Input com máscara de moeda
│   │   ├── status-badge.tsx      # Badge semântico
│   │   └── fee-breakdown.tsx     # Visualizador de taxas
│   ├── ui/                       # Primitivos Shadcn/Radix (Button, Input, Card)
│   └── layout/                   # Componentes estruturais
│       ├── sidebar.tsx           # Menu lateral
│       ├── tenant-switcher.tsx   # Seletor de clientes
│       └── user-nav.tsx          # Menu de perfil
├── lib/
│   ├── actions/                  # Server Actions (Mutações de Banco de Dados)
│   │   ├── reconcile.ts          # Lógica de conciliação manual
│   │   └── sync.ts               # Gatilhos de sincronização
│   ├── dal/                      # Data Access Layer (Queries Prisma Otimizadas)
│   ├── hooks/                    # Custom Hooks (use-sidebar, use-data-table)
│   └── utils.ts                  # Helpers (Formatadores, Validadores)
└── types/                        # Definições de Tipos TypeScript (Prisma Generated)
```

## 4.2 Estratégia de Menu e Navegação

A navegação lateral (Sidebar) é o centro de comando. Ela deve ser persistente e responsiva. O estado de "colapsado/expandido" é crítico para maximizar a área útil da *Data Table*.

- **Persistência de Estado via Cookies:** Para evitar o "flicker" (a sidebar aparecer expandida e depois colapsar durante o carregamento), o estado da sidebar é salvo em um cookie (sidebar:state). O componente Layout no servidor lê este cookie e renderiza a sidebar com a classe CSS correta (w-64 ou w-16) desde o primeiro byte de HTML enviado. Isso garante uma estabilidade visual absoluta.[13]
- **Menu Hierárquico:**
  - **Dashboard:** Visão geral de KPIs.
  - **Conciliação (Reconciliation):** Onde o trabalho acontece. Sub-menus para "Pendente", "Divergente", "Chargebacks".
  - **Transações (Transactions):** Histórico completo (Pesquisa Forense).
  - **Relatórios:** Exportações para Contabilidade (CSV/PDF).
  - **Configurações:**
    - *Integrações:* Setup de credenciais MP/Omie.
    - *Motor de Taxas:* Definição das regras de contrato (MDR).
- **Tenant Switcher:** Localizado no topo da Sidebar, este componente (um Combobox do Radix UI) permite que contadores troquem entre as empresas gerenciadas. A seleção de um novo tenant dispara uma navegação para a rota //dashboard, re-escopando imediatamente todas as queries de dados.[1]

---

# 5. O Motor de Dados: Data Grid e Visualização Forense

O coração do Svelto é a *Data Table*. Não se trata de uma tabela HTML simples, mas de uma aplicação complexa embutida. Utilizamos **TanStack Table v8** em modo "Headless" para controle total da lógica, renderizando com componentes Tailwind estilizados.[1]

## 5.1 Implementação da Tabela: Virtualização e Paginação Server-Side

Dada a exigência de processar grandes volumes de dados (Milestone 2 - Conciliação Avançada), a renderização de milhares de linhas no DOM travaria o navegador.

- **Paginação Server-Side Obrigatória:** O frontend nunca baixa o dataset completo. A tabela recebe pageIndex, pageSize e sorting via URL Search Params. Esses parâmetros são passados para a Server Action, que retorna apenas o "slice" de dados necessário (ex: 50 registros) e o pageCount total.
- **Virtualização de Linhas:** Para visualizações de "Auditoria em Massa" onde o usuário deseja rolar por centenas de itens, implementamos o @tanstack/react-virtual. Esta biblioteca renderiza apenas as linhas visíveis na viewport (ex: 20 linhas) mais um buffer, reciclando os elementos DOM à medida que o usuário rola. Isso mantém a taxa de

quadros (FPS) estável mesmo em laptops corporativos de baixa performance.[16]

## 5.2 O Padrão "Linha da Verdade" (Truth Row Pattern)

Para resolver o problema da "Quádrupla Verdade" sem sobrecarregar a visualização inicial, adotamos o padrão de linhas expansíveis (*Expandable Rows*).[28]

1. **Estado Contraído (Visão Executiva):** A linha exibe o "Golden Record" — o status consolidado, o NSU principal, a Data e o Valor Líquido. Fontes monoespaçadas e alinhamento à direita para valores são rigorosamente aplicados.
    - *Colunas:* Checkbox, Status (Badge), Data (dd/mm hh:mm), NSU (Truncado), Origem (Ícone MP), Valor Bruto, Valor Líquido, Ações.
2. **Estado Expandido (Visão Forense):** Ao clicar na linha, um painel detalhado se abre (usando getExpandedRowModel do TanStack). Este painel utiliza um CSS Grid de 4 colunas para apresentar as quatro verdades lado a lado:
    - **Coluna 1: Gateway (MP):** Dados brutos da API (status_detail, fee_details). Fundo sutil bg-sky-50 (alusão à marca MP).
    - **Coluna 2: Auditoria Svelto:** Cálculo do "Dever Ser" baseado no motor de taxas. Se houver divergência, destaca-se em bg-amber-100.
    - **Coluna 3: ERP (Omie):** Estado do título a receber (nCodTitulo, Conta Corrente). Fundo sutil bg-cyan-50 (alusão à marca Omie). Link direto para o Omie ("Deep Link").
    - **Coluna 4: Banco:** Dados do Payout/Liquidação (date_released, payout_id). Fundo sutil bg-emerald-50.

Esta disposição permite que o auditor faça um "scan horizontal" para identificar onde a integridade dos dados se quebrou.[1]

## 5.3 Visual Diffing e Highlight de Delta

Dentro da "Linha da Verdade", o sistema realiza uma comparação ativa.

- **Lógica de Diffing:** O componente recebe o objeto transaction e o objeto erpReceivable.
- **Visualização:** Se o valor da taxa no Gateway for R$ 2.50 e o calculado pelo Svelto for R$ 2.49, a célula correspondente recebe uma borda vermelha ou fundo amarelo.
- **Exibição Explícita do Delta:** Ao lado do valor divergente, renderizamos o delta matemático: (Diff: -R$ 0.01) em fonte text-xs text-red-600 font-mono. Isso elimina a necessidade de cálculo mental pelo usuário, acelerando a tomada de decisão.[1]

---

# 6. Blueprint de Telas e Fluxos de Usuário

## 6.1 Dashboard (O Cockpit Operacional)

O objetivo desta tela é fornecer consciência situacional imediata.

- **Cards de KPI:** Usamos o componente Card do Shadcn UI para exibir métricas críticas: "Volume Processado", "Conciliação Pendente" (destaque em Amarelo), "Risco de Chargeback" (destaque em Vermelho).
- **Gráfico de Fluxo de Caixa:** Utilizamos a biblioteca **Tremor** (BarChart ou AreaChart) para plotar "Receita Prevista" vs. "Liquidação Realizada". A Tremor é escolhida por ser construída sobre Tailwind e oferecer componentes financeiros "opinionated" que requerem configuração mínima.[30]
- **Widgets de Ação:** Uma lista "Conciliação Sugerida" mostra as transações onde o "NSU Hunter" encontrou *matches* de alta confiança, permitindo aprovação rápida com um clique.

### 6.2 Tela de Conciliação (O "Matchmaker")

Esta é a interface de trabalho pesado para resolver exceções.

- **Layout Split View:** A tela é dividida verticalmente. À esquerda, uma lista de transações do Gateway "Órfãs" (sem match). À direita, uma lista filtrável de Títulos do ERP "Abertos".
- **Mecânica de Match Manual:** O usuário pode selecionar uma transação à esquerda e um título à direita e clicar no botão de ação central "Vincular Manualmente".
- **Resolução de Ambiguidade:** Quando o algoritmo encontra múltiplos candidatos (Fuzzy Match), a interface apresenta um agrupamento visual. A transação do Gateway aparece em destaque, e abaixo dela, os candidatos do ERP são listados com *Radio Buttons*. O usuário deve selecionar explicitamente qual é o par correto. O candidato mais provável (baseado em data/valor) vem pré-selecionado.[1]

### 6.3 Tela de Configurações (O Motor de Taxas)

Onde o usuário define a "Verdade Contratual".

- **Matriz de Inputs:** Uma tabela complexa onde as linhas são as Bandeiras (Visa, Master, Elo) e as colunas são as condições de parcelamento (Débito, 1x, 2-6x, 7-12x).
- **Input de Alta Precisão:** Utilizamos componentes de input customizados (baseados em react-number-format) que permitem a inserção de até 4 casas decimais (2.4900%). A máscara impede a entrada de caracteres inválidos e formata automaticamente o percentual.
- **Validação em Tempo Real:** Se o usuário inserir uma taxa que parece errada (ex: "50%"), o campo exibe um *warning* visual imediato (borda amarela) sugerindo verificação.[1]

---

## 7. Gestão de Estado e Mutações: O "Botão Vermelho" e Optimistic UI

A ação de escrever no ERP ("Baixar Título") é crítica e lenta (APIs de ERPs podem levar segundos). A UI deve gerenciar essa expectativa.

### 7.1 Server Actions e Feedback Otimista

Utilizamos as Server Actions do Next.js para todas as mutações. Para evitar a sensação de lentidão:

1. **Ação:** O usuário clica em "Baixar no ERP".
2. **Optimistic Update:** O hook useOptimistic atualiza *imediatamente* o estado visual da linha para CONCILIADO (verde) e desabilita o botão. O usuário sente que a ação foi instantânea.[32]
3. **Processamento:** A Server Action executa a chamada à API do Omie em background.
4. **Confirmação/Reversão:**
   - *Sucesso:* O revalidatePath é chamado, confirmando o dado real do servidor.
   - *Falha:* Se a API do Omie falhar, o estado otimista é revertido automaticamente (a linha volta a ser MATCHED/amarela) e um **Toast de Erro** persistente é exibido: "Falha na comunicação com Omie. Tente novamente."

### 7.2 Tratamento de Idempotência na UI

O backend do Svelto implementa verificações de idempotência. Se o usuário tentar baixar um título que já foi baixado (por outro usuário ou processo):

- O backend retorna um código específico ALREADY_SETTLED.
- A UI não mostra um erro vermelho assustador. Em vez disso, exibe um **Toast Azul** (Informativo): "Sincronização Atualizada: Este título já constava como baixado no ERP." e atualiza o estado para Verde. Isso é o padrão de "Sucesso Lógico".[1]

---

## 8. Configurações Técnicas e Boilerplate

Para garantir que o código suporte essa arquitetura, as configurações do projeto devem ser rígidas.

### 8.1 tailwind.config.ts

TypeScript

```
import type { Config } from "tailwindcss"

const config = {
  darkMode: ["class"],
  content: [
    './pages/**/*.{ts,tsx}',
```

```
    './components/**/*.{ts,tsx}',
    './app/**/*.{ts,tsx}',
    './src/**/*.{ts,tsx}',
  ],
  theme: {
    container: {
      center: true,
      padding: "2rem",
      screens: {
        "2xl": "1400px",
      },
    },
    extend: {
      fontFamily: {
        sans: ["var(--font-inter)"], // Inter para UI
        mono: ["var(--font-jetbrains-mono)"], // JetBrains para Dados
      },
      colors: {
        // Cores de Marca
        mercadopago: "#00BCFF",
        omie: "#00E3F5",
        // Cores Semânticas Financeiras (Acessíveis)
        finance: {
            conciliated: {
                bg: "#ECFDF5", // emerald-50
                text: "#047857", // emerald-700
            },
            divergent: {
                bg: "#FFFBEB", // amber-50
                text: "#B45309", // amber-700
            },
            chargeback: {
                bg: "#FFF1F2", // rose-50
                text: "#BE123C", // rose-700
            }
        },
        border: "hsl(var(--border))",
        input: "hsl(var(--input))",
        ring: "hsl(var(--ring))",
        background: "hsl(var(--background))",
        foreground: "hsl(var(--foreground))",
        //... (restante das vars do Shadcn UI)
      },
```

```
    },
  },
  plugins: [require("tailwindcss-animate")],
} satisfies Config

export default config
```

### 8.2 Componentes Chave do Shadcn UI

Para acelerar o desenvolvimento, os seguintes componentes do Shadcn UI devem ser instalados via CLI (npx shadcn-ui@latest add...):

- table (Base para o Grid)
- card (KPIs)
- badge (Status)
- button (Ações)
- toast (Feedback)
- dialog (Modais de detalhe)
- combobox (Tenant Switcher)
- popover (Filtros e DatePicker)
- calendar (Filtros de data)
- skeleton (Loading states)

---

## 9. Segurança e Auditoria no Frontend

A segurança no frontend financeiro vai além da autenticação. Envolve a proteção visual e a rastreabilidade das ações.

- **Mascaramento de Dados Sensíveis:** Campos como CPF/CNPJ dos pagadores ou dados parciais de cartão devem ser mascarados por padrão na UI (***.***.***-00), com um botão "Revelar" (ícone de olho) que exige uma ação consciente do usuário. Isso previne o vazamento de dados em compartilhamentos de tela ou screenshots acidentais.[1]
- **Trilha de Auditoria Visual:** Cada ação de escrita (conciliar, ignorar, baixar) deve deixar um rastro visível. Na tela de detalhes da transação, exibimos um componente "Log de Eventos" (Timeline) que mostra: "Usuário X aprovou conciliação em". Isso aumenta a responsabilidade (*accountability*) do operador.
- **Proteção de Rotas (Middleware):** Utilizamos o Middleware do Next.js para interceptar todas as requisições a /(dashboard)/*. Se o token de sessão não estiver presente ou for inválido, o redirecionamento para /login é imediato, garantindo que nenhuma interface financeira seja renderizada para usuários não autenticados.

---

# 10. Conclusão: O Motor de Confiança

A implementação deste blueprint transformará o Svelto de uma ferramenta de visualização de dados em um verdadeiro **Motor de Confiança**. Ao adotar rigorosamente a tipografia tabular, as cores semânticas acessíveis, e os padrões de interação otimista para lidar com a latência de sistemas legados, o frontend se torna um aliado do auditor, e não um obstáculo.

A arquitetura baseada em Next.js App Router e Server Components garante que o sistema possa escalar para milhares de transações sem degradação de performance, enquanto a estratégia de "Quádrupla Verdade" visualizada através do padrão de "Linhas Expansíveis" resolve o problema fundamental da conciliação: a visibilidade da discrepância. O Svelto não apenas mostra os números; ele conta a história do dinheiro, centavo por centavo, com a fidelidade que a operação financeira exige.

---

*Relatório Técnico elaborado por Svelto Architect | v3.0 Specification*

**Works cited**


1. Svelto SaaS - Master Blueprint v2.3.md.pdf
2. Next.js App Router: common mistakes and how to fix them - Platform.sh, accessed January 19, 2026, https://upsun.com/blog/avoid-common-mistakes-with-next-js-app-router/
3. Next.js SaaS Dashboard Development: Scalability & Best Practices - Ksolves, accessed January 19, 2026, https://www.ksolves.com/blog/next-js/best-practices-for-saas-dashboards
4. Tailwind CSS Advanced Tables - Flowbite, accessed January 19, 2026, https://flowbite.com/blocks/application/advanced-tables/
5. Data Fetching: Server Actions and Mutations | Next.js, accessed January 19, 2026, https://nextjs.org/docs/13/app/building-your-application/data-fetching/server-actions-and-mutations
6. Fintech Design Breakdown: the Most Common Design Patterns - Phenomenon Studio, accessed January 19, 2026, https://phenomenonstudio.com/article/fintech-design-breakdown-the-most-common-design-patterns/
7. Has anyone successfully built a reusable DataTable with ShadCN + Supabase (with optional CRUD)? : r/nextjs - Reddit, accessed January 19, 2026, https://www.reddit.com/r/nextjs/comments/1oay58e/has_anyone_successfully_built_a_reusable/
8. Tailwind Colors - Coolors, accessed January 19, 2026, https://coolors.co/tailwind
9. Colors - Core concepts - Tailwind CSS, accessed January 19, 2026, https://tailwindcss.com/docs/customizing-colors
10. table-layout - Tailwind CSS, accessed January 19, 2026, https://tailwindcss.com/docs/table-layout
11. Pagination on TanStack table under NEXT.JS | by LAI TOCA - Medium, accessed

January 19, 2026, https://tocalai.medium.com/pagination-on-tanstack-table-under-next-js-787ed03198a3
12. Guides: SPAs - Next.js, accessed January 19, 2026, https://nextjs.org/docs/app/guides/single-page-applications
13. Routing: Pages and Layouts | Next.js, accessed January 19, 2026, https://nextjs.org/docs/pages/building-your-application/routing/pages-and-layouts
14. Functions: cookies | Next.js, accessed January 19, 2026, https://nextjs.org/docs/app/api-reference/functions/cookies
15. Cookies vs. Local Storage in Next.js: Which is Best for Your Website? - Shamali Dilrukshi, accessed January 19, 2026, https://mgshamalidilrukshi.medium.com/cookies-vs-local-storage-in-next-js-which-is-best-for-your-website-b3c45199de40
16. Virtualization Guide | TanStack Table Docs, accessed January 19, 2026, https://tanstack.com/table/v8/docs/guide/virtualization
17. How to implement Server Side Pagination with TanStack Table and NextJS 13? - Reddit, accessed January 19, 2026, https://www.reddit.com/r/nextjs/comments/14vm2e3/how_to_implement_server_side_pagination_with/
18. Mercado Pago Logo & Brand Assets (SVG, PNG and vector) - Brandfetch, accessed January 19, 2026, https://brandfetch.com/mercadopago.com
19. Omie Logo & Brand Assets (SVG, PNG and vector) - Brandfetch, accessed January 19, 2026, https://brandfetch.com/omie.com.br
20. Omie Logo & Brand Assets (SVG, PNG and vector) - Brandfetch, accessed January 19, 2026, https://brandfetch.com/omie.com
21. Coloring for Colorblindness - David Nichols, accessed January 19, 2026, https://davidmathlogic.com/colorblind/
22. Accessible Color Palette Generator | WCAG Compliant - Venngage, accessed January 19, 2026, https://venngage.com/tools/accessible-color-palette-generator
23. Seeing Red, or Maybe Green? How Tailwind Helps Colourblind Designers - Liam Hammett, accessed January 19, 2026, https://liamhammett.com/tailwind-for-colourblind-designers
24. Lucide React, accessed January 19, 2026, https://lucide.dev/guide/packages/lucide-react
25. split - Lucide, accessed January 19, 2026, https://lucide.dev/icons/split
26. Getting Started: Project Structure | Next.js, accessed January 19, 2026, https://nextjs.org/docs/app/getting-started/project-structure
27. A complete guide to TanStack Table (formerly React Table) - LogRocket Blog, accessed January 19, 2026, https://blog.logrocket.com/tanstack-table-formerly-react-table/
28. Expanding Guide | TanStack Table Docs, accessed January 19, 2026, https://tanstack.com/table/v8/docs/guide/expanding
29. utkuakyuz/virtual-react-json-diff: A lightweight diff viewer for JSON files in React applications, accessed January 19, 2026,

https://github.com/utkuakyuz/virtual-react-json-diff
30. Tremor: Vercel's React Library for Dashboard Components - Tailkits, accessed January 19, 2026, https://tailkits.com/components/tremor/
31. tremorlabs/tremor: Copy & Paste React components to build modern web applications. - GitHub, accessed January 19, 2026, https://github.com/tremorlabs/tremor
32. Optimistic UI with Server Actions in Next.js: A Smoother User Experience - Medium, accessed January 19, 2026, https://medium.com/@mishal.s.suyog/optimistic-ui-with-server-actions-in-next-js-a-smoother-user-experience-6b779e4293a9
33. useOptimistic for Optimistic UI in Next.js Server Actions (+ Revalidate) - YouTube, accessed January 19, 2026, https://www.youtube.com/watch?v=PPOw-sDeoNw