Manual de Arquitetura de Interface e Experiência do Usuário: Svelto SaaS Financial Operating System

1. Introdução: O Paradigma da Interface de Alta Fidelidade Financeira

No desenvolvimento de sistemas financeiros B2B (Business-to-Business), a interface do usuário transcende a função estética ou meramente operacional; ela se estabelece como a camada primária de mitigação de risco e auditoria corporativa. O Svelto SaaS, concebido sob a arquitetura da "Quádrupla Verdade", exige uma abordagem de design que rejeita as simplificações excessivas comuns em aplicações de consumo (B2C) em favor de uma densidade de informação controlada, precisão tipográfica absoluta e transparência radical dos processos de dados. O usuário alvo — gestores financeiros, auditores e controllers — opera em um ambiente de alta pressão onde a ambiguidade visual pode resultar em prejuízos monetários diretos, passivos fiscais ocultos ou falhas na conformidade regulatória.

A premissa central deste manual é que a confiança no sistema não deriva de interações "encantadoras", mas da previsibilidade e da "solidez percebida" da aplicação. Quando um operador visualiza um status de "Conciliado", a interface deve comunicar, através de cor, forma, tipografia e contexto, que essa afirmação é matematicamente comprovada por quatro fontes de verdade distintas. A adoção da stack tecnológica Next.js com Tailwind CSS oferece as ferramentas necessárias para construir essa experiência com a performance e a modularidade exigidas por uma aplicação de escala empresarial. Este documento disseca as estratégias de UI/UX necessárias para materializar a complexidade do motor de conciliação do Svelto, transformando dados brutos em inteligência financeira acionável.

A análise a seguir detalha não apenas os componentes visuais, mas a lógica de interação profunda necessária para suportar fluxos de trabalho de conciliação de alto volume, gestão de exceções (como chargebacks) e a visualização de linhagem de dados, garantindo que cada pixel na tela sirva ao propósito maior de integridade financeira.

---

2. A Engenharia Visual da 'Quádrupla Verdade'

A arquitetura de "Quádrupla Verdade" impõe um desafio de design singular: a necessidade de apresentar, simultaneamente e sem conflito cognitivo, quatro realidades financeiras que podem, em momentos de divergência, contradizer-se. O sistema não pode simplesmente apresentar uma "média" ou uma "visão unificada" que mascare as discrepâncias; pelo

contrário, a função do Svelto é expor essas discrepâncias. As quatro verdades — Operacional (Gateway), Contábil (ERP), Financeira (Auditoria Svelto) e Bancária (Liquidação) — devem coexistir na interface, permitindo uma comparação lateral (side-by-side) que facilita a auditoria forense instantânea.

2.1. O Padrão de Linha Expandida (The Truth Row Pattern)

Para resolver a tensão entre a necessidade de visualização sintética (para escaneamento rápido) e a necessidade de detalhamento forense, estabelece-se o padrão de "Linha Mestra com Detalhamento em Acordeão" como a unidade fundamental da interface de conciliação. Este padrão permite que a interface opere em dois níveis cognitivos distintos: o nível de gerenciamento de fluxo e o nível de investigação de exceções.

No **Estado Contraído (Visão Executiva)**, a linha da tabela atua como um sumário executivo da transação. Ela deve exibir exclusivamente os identificadores primários e o status consolidado da conciliação. A tipografia aqui deve ser otimizada para legibilidade rápida, utilizando fontes monoespaçadas para valores numéricos e IDs, garantindo que o olho do auditor possa percorrer verticalmente a lista em busca de anomalias de magnitude ou status.[1] Colunas críticas neste estado incluem o ID interno do Svelto, o Valor Bruto Original, o Valor Líquido Final Calculado e, crucialmente, uma coluna de "Indicador de Integridade". Este indicador não é apenas um status, mas um sinal visual que resume a concordância entre as quatro verdades. Se as quatro fontes concordam, o indicador é verde e discreto; se há uma divergência de centavos entre o Gateway e o ERP, o indicador torna-se amarelo e exibe o delta da diferença.

No **Estado Expandido (Visão Forense)**, acionado por um clique ou atalho de teclado, a linha se desdobra para revelar a anatomia completa da "Quádrupla Verdade". O design deste painel expandido deve abandonar a estrutura de tabela e adotar um layout de grid comparativo, dividindo o espaço horizontal em quatro blocos de igual largura, cada um representando uma fonte de verdade:

1. **Bloco da Verdade Operacional (Gateway - Mercado Pago):** Este quadrante deve exibir os dados brutos da captura da transação. Elementos críticos incluem o status_detail (ex: accredited, pending_contingency), o transaction_amount original e a discriminação de taxas retornada pela API (fee_details). A inclusão do ícone do Gateway reforça a origem do dado. A apresentação visual deve refletir a natureza "bruta" deste dado, talvez utilizando uma cor de fundo sutilmente diferente ou uma borda lateral colorida que corresponda à identidade visual do Gateway.[2]
2. **Bloco da Verdade Financeira (Auditoria Svelto):** Adjacente ao bloco operacional, este quadrante exibe o "dever ser". Aqui, o "Fee Engine" do Svelto projeta os valores esperados baseados nas regras contratuais configuradas. É imperativo mostrar a taxa contratada (ex: "MDR 2.49%") *versus* a taxa efetiva calculada. Se houver discrepância, este campo deve ser destacado com um fundo de alerta (bg-amber-50 no Tailwind), chamando a atenção imediata para a perda de receita por cobrança indevida.[2]

3. **Bloco da Verdade Contábil (ERP - Omie):** Este bloco foca no reflexo fiscal e contábil. Deve exibir o nCodTitulo, o status do título no ERP (Aberto, Liquidado, Cancelado) e, vitalmente, a conta corrente de destino vinculada. A interface deve permitir que o usuário verifique se o título foi baixado na conta correta ("Banco Itaú" vs. "Caixinha"), prevenindo erros de conciliação bancária futura. Um link direto para o registro no Omie (deep link) é uma boa prática de UX para facilitar a correção manual.[2]
4. **Bloco da Verdade Bancária (Liquidação/Settlement):** O último quadrante fecha o ciclo, mostrando a materialização do dinheiro. Deve exibir o Payout ID (Lote de transferência), a Data de Liquidação Efetiva (que pode diferir da data de previsão) e o valor líquido final creditado no banco. A visualização deste bloco confirma que o dinheiro saiu do ambiente virtual e tornou-se liquidez real.[2]

A implementação técnica deste padrão em **Tailwind CSS** exige o uso de grids responsivos e utilitários de espaçamento precisos para manter a densidade sem sacrificar a clareza. O uso de grid-cols-4 com gap-4 e bordas divisórias suaves (border-r border-gray-200) cria uma separação visual clara sem adicionar "ruído" desnecessário. O fundo do painel expandido deve diferenciar-se ligeiramente do fundo da tabela (bg-slate-50 em modo claro ou bg-slate-900 em modo escuro) para criar uma hierarquia visual de "profundidade".[3]

2.2. Visualização de Divergências (Visual Diffing e Delta Highlighting)

A detecção de divergências é o valor central do produto Svelto. Portanto, a interface não pode ser passiva ao apresentar conflitos de dados; ela deve ser opinativa e evidente. Quando o sistema detecta que o Gateway cobrou R$ 0,50 a mais de taxa do que o previsto, isso não é apenas um dado; é um alerta de negócio.

O padrão de **Visual Diffing** deve ser aplicado transversalmente. Campos que não coincidem entre as colunas de "Verdade" devem receber tratamento visual imediato.

- **Highlight Semântico:** Utilize cores de fundo semânticas para indicar a severidade da divergência. Uma diferença de arredondamento (ex: R$ 0,01) pode ser sinalizada com um aviso amarelo suave (bg-yellow-50 texto text-yellow-700), enquanto uma diferença de ID ou conta bancária (que quebra a conciliação) deve ser vermelha (bg-red-50 texto text-red-700).
- **Exibição Explícita do Delta:** Ao lado do valor divergente, a interface deve renderizar matematicamente a diferença. Não obrigue o auditor a subtrair R$ 98,50 de R$ 98,45 mentalmente. Exiba (Diff: -R$ 0,05) em uma fonte monoespaçada menor e vermelha. Isso reduz a carga cognitiva e acelera a tomada de decisão.[4]
- **Comparação Side-by-Side:** Em modais de resolução de conflitos, utilize o padrão de comparação lado a lado clássico de ferramentas de versionamento de código (como o diff do Git). Coloque o "Registro Gateway" à esquerda e o "Título ERP Sugerido" à direita, destacando apenas as linhas que diferem. Isso foca a atenção do usuário exclusivamente no problema a ser resolvido.[5]

3. Precisão Tipográfica e Formatação de Dados Financeiros

Em sistemas financeiros, a tipografia é funcionalidade. A escolha da fonte, o alinhamento e a formatação dos números determinam a velocidade com que um auditor pode escanear uma tabela e a precisão com que pode detectar erros. O Svelto lida com dados de alta precisão (taxas de 4 casas decimais) e valores monetários que variam de centavos a milhões; a interface deve acomodar essa amplitude com rigor.

3.1. O Imperativo dos tabular-nums

A regra mais crítica para a tipografia financeira é o uso de **figuras tabulares** (tabular figures). Fontes tipográficas modernas frequentemente utilizam figuras proporcionais por padrão, onde o glifo do número "1" ocupa menos espaço horizontal do que o número "8". Em um parágrafo de texto, isso melhora a leitura; em uma tabela financeira, isso é desastroso. Figuras proporcionais fazem com que colunas de números não se alinhem verticalmente, impedindo que o usuário compare ordens de magnitude ao percorrer a lista visualmente.[7]

O Svelto deve impor, via **Tailwind CSS**, a classe tabular-nums (que aplica font-variant-numeric: tabular-nums) em todas as células de tabela que contêm valores numéricos, datas, CNPJs, NSUs ou IDs. Isso força a fonte a usar glifos de largura fixa para números, garantindo que o dígito das centenas de uma linha esteja perfeitamente alinhado com o dígito das centenas da linha abaixo. A escolha da família tipográfica também é vital; recomenda-se o uso de fontes *sans-serif* otimizadas para UI (como Inter, Roboto ou a system font stack) combinadas com uma fonte *monospace* para identificadores técnicos (NSUs, Hash de Transação), aplicando font-mono text-sm para distinguir visualmente dados técnicos de dados financeiros.[9]

3.2. Estratégias de Formatação Decimal e Monetária

A "Verdade Financeira" do Svelto opera com precisão de 4 casas decimais para auditoria de taxas (ex: MDR de 2,4900%), enquanto a "Verdade Bancária" opera com 2 casas decimais (ex: R$ 100,00). A interface deve respeitar essa distinção sem causar confusão.

- **Valores Monetários de Liquidação:** Devem ser sempre formatados com 2 casas decimais, alinhados à direita. O alinhamento à direita é inegociável para dados monetários, pois alinha as casas decimais e as vírgulas, facilitando a comparação visual de magnitudes (unidades, dezenas, centenas).[11]
- **Valores de Taxas e Cálculos Unitários:** Devem ser exibidos com até 4 casas decimais quando relevante para a auditoria. Ocultar as frações de centavos em taxas pode mascarar a origem de divergências de arredondamento que, acumuladas em milhares de transações, resultam em diferenças significativas no caixa. O Svelto deve adotar um

padrão de "precisão adaptativa" ou oferecer um *tooltip* que revela a precisão completa ao passar o mouse sobre um valor arredondado (hover:tooltip).[2]

- **Tratamento de Zeros e Nulos:** Valores zero devem ser formatados explicitamente (e.g., R$ 0,00 ou um traço - em cinza claro para reduzir ruído visual em tabelas densas), nunca deixados em branco. Isso confirma ao usuário que o dado existe e é zero, distinguindo-o de um dado ausente ou não carregado (null ou undefined), que deve ser representado por um estado de carregamento ou um placeholder específico ("--").[15]

3.3. Inputs de Alta Precisão e Máscaras

Para os formulários de entrada manual (ex: ajuste de conciliação ou configuração de taxas contratuais), a UX deve prevenir erros de inserção na fonte. Utilizar componentes de input que formatam automaticamente o valor conforme o usuário digita (máscaras de moeda) é essencial.

- **Feedback de Truncamento:** Se um usuário colar um valor com 6 casas decimais em um campo configurado para 4, a interface não deve apenas truncar silenciosamente; ela deve fornecer um feedback visual (um flash amarelo ou um ícone de info) indicando que o valor foi arredondado para se adequar à precisão do sistema.
- **Sanitização na Entrada:** Inputs devem bloquear caracteres não numéricos ao nível do evento onKeyDown, exceto os separadores decimais apropriados ao locale (pt-BR usa vírgula). O uso de bibliotecas como react-number-format integradas aos componentes do Shadcn UI pode padronizar esse comportamento em toda a aplicação.[16]

---

4. O Motor de Data Grid: Implementação com TanStack Table

A tabela de transações é o "chão de fábrica" do Svelto SaaS. Dada a exigência de processar e exibir milhares de transações (Milestone 2 - Conciliação Avançada), uma implementação ingênua com tabelas HTML padrão ou componentes React simples resultará em performance inaceitável. A arquitetura de frontend deve adotar o **TanStack Table (v8)** em modo *headless*, permitindo controle total sobre a renderização e a lógica, desacoplado dos estilos.[17]

4.1. Arquitetura de Paginação Server-Side e Virtualização

A premissa de "Big Data" financeiro exige que o frontend nunca tente carregar o dataset completo na memória do navegador. A paginação deve ser estritamente **Server-Side**.

- **Estado na URL:** O estado da paginação (pageIndex, pageSize, filters, sorting) deve ser sincronizado com a URL (Query Parameters). Isso permite que auditores compartilhem links diretos para "Página 5 das divergências de ontem" com colegas, facilitando a colaboração. No Next.js App Router, isso é implementado lendo searchParams no Server Component e passando-os para o hook da tabela ou para a query do banco de dados.[19]

- **Virtualização para Auditoria em Massa:** Em cenários onde o usuário precisa rolar por centenas de linhas para uma revisão visual rápida (padrão comum em auditoria), a paginação tradicional pode ser intrusiva. A implementação de **Virtualização** (usando @tanstack/react-virtual) permite renderizar apenas as linhas visíveis na viewport, mantendo o DOM leve mesmo se a "página" lógica contiver 500 ou 1000 registros. Isso é vital para manter a taxa de quadros (FPS) da interface suave durante a rolagem rápida.[12]

4.2. Cabeçalhos Fixos e Colunas Ancoradas (Sticky UI)

A perda de contexto é um inimigo da precisão. Ao rolar verticalmente por uma lista de 500 transações, o usuário não pode perder de vista o que cada coluna representa. Ao rolar horizontalmente para ver as 20+ colunas de metadados da "Quádrupla Verdade", o usuário não pode perder a referência de *qual* transação está olhando.

- **Sticky Headers:** O cabeçalho da tabela (<thead>) deve ser fixado no topo da viewport (sticky top-0 z-10 bg-white shadow-sm no Tailwind) para garantir que os rótulos das colunas estejam sempre visíveis.
- **Column Pinning:** A primeira coluna (contendo o ID/NSU da transação e o Checkbox de seleção) e a última coluna (Ações/Status) devem ser fixáveis (sticky left-0 e sticky right-0). O TanStack Table oferece suporte nativo a *Column Pinning*, calculando os offsets necessários via JavaScript, mas a implementação visual deve garantir que essas colunas flutuantes tenham uma sombra sutil ou borda para indicar que estão "acima" do conteúdo rolável, mantendo a orientação espacial do usuário.[1]

4.3. Ações em Lote e Feedback de Progresso (Optimistic UI)

A conciliação manual eficiente ocorre em lotes. O gestor filtra "Todas as vendas Visa de ontem" e aplica uma ação de "Conciliar em Massa" ou "Baixar no ERP".

- **Seleção Tri-State:** O checkbox mestre no cabeçalho deve refletir com precisão três estados: Não Selecionado, Selecionado Totalmente (todos da página/view) e Indeterminado (alguns selecionados). A representação visual do estado "Indeterminado" (geralmente um traço horizontal - em vez de um check ✓) é um padrão de UI crítico para evitar a falsa impressão de que todos os itens foram capturados.[23]
- **Feedback Otimista com Reversão:** Para ações em lote (ex: Baixar 50 títulos), a interface não deve bloquear e esperar a resposta sequencial do servidor para cada item. Utilize o padrão de **Optimistic UI**: marque visualmente as linhas como "Processando" ou "Conciliado (Provisório)" instantaneamente. Se a Server Action falhar para um item específico (ex: erro de API do Omie), o sistema deve reverter automaticamente o estado visual daquele item para "Erro" e exibir um *Toast* explicativo, sem perturbar o sucesso dos outros 49 itens. Isso cria uma sensação de performance e fluidez, essencial para a produtividade do usuário.[24]

---

5. Fluxos de Conciliação e a Interface do 'Matchmaker'

O motor de conciliação do Svelto ("NSU Hunter") é o cérebro da operação, mas a interface é onde o usuário valida e confia nesse cérebro. A UI deve categorizar e apresentar as sugestões de conciliação com base no nível de confiança algorítmica, exigindo diferentes níveis de interação humana.

5.1. Visualizando a Confiança do Algoritmo

A interface deve distinguir visualmente entre um "Hard Match" (Vínculo Determinístico) e um "Fuzzy Match" (Sugestão Heurística).

- **Hard Match (100% Confiança):** Quando o NSU (Número Sequencial Único) do Gateway coincide perfeitamente com o do ERP, o sistema tem certeza. A UI deve exibir um ícone de "Corrente" ou "Link" sólido e verde (text-green-600). Essas linhas podem ser pré-selecionadas para aprovação em lote ou até conciliadas automaticamente, dependendo da configuração do usuário.
- **Fuzzy Match (Sugestão):** Quando o vínculo é baseado em heurísticas fracas (mesmo valor, mesma data aproximada), a UI deve tratar isso com cautela. O ícone deve ser diferente (talvez um sinal de interrogação ou um link tracejado em amarelo/laranja) e a linha deve apresentar um status de SUGGESTED.
- **Interface de Revisão ("The Inspector"):** Para sugestões Fuzzy, deve haver um botão de ação "Revisar Match". Ao clicar, um modal ou painel lateral se abre, colocando os dados do Gateway à esquerda e o candidato do ERP à direita. Campos que coincidem (ex: Valor) devem ser destacados em verde; campos que diferem (ex: Data D+0 vs D+1) devem ser destacados em amarelo. Botões claros de "Confirmar Vínculo" e "Rejeitar/Buscar Outro" devem estar presentes. Esse fluxo de "Humano no Loop" é essencial para treinar a confiança do usuário no sistema.[2]

5.2. Resolução de Ambiguidade e Conflitos

Uma regra de negócio crítica do Svelto [2] dita que, se o sistema encontrar múltiplos candidatos possíveis para uma única transação (ex: duas vendas de R$ 50,00 no mesmo dia), ele não deve "adivinhar". A UI deve refletir essa ambiguidade.

- **Estado AMBIGUOUS:** A linha deve ser marcada com um status de atenção (AMBIGUOUS ou NEED_ATTENTION).
- **Interface de Seleção de Candidatos:** Ao interagir com essa linha, o usuário deve ver a transação do Gateway em destaque e uma lista de "Candidatos ERP Encontrados". O uso de *Radio Buttons* permite que o usuário selecione explicitamente qual título do ERP corresponde àquela venda. A lista de candidatos deve ser inteligentemente filtrada e ordenada pela probabilidade (ex: priorizando a conta bancária correta e a data mais próxima), reduzindo o esforço de busca do usuário.[2]

5.3. O "Botão Vermelho": UX de Baixa Financeira

A ação de escrever no ERP (liquidar um título) é uma operação de escrita crítica. A UI deve proteger o usuário contra erros acidentais ou duplicações.

- **Idempotência na Interface:** O botão de "Baixar no ERP" deve ser imediatamente desabilitado (disabled={pending}) ao ser clicado, prevenindo cliques duplos que poderiam disparar requisições concorrentes. O uso do hook useFormStatus do React/Next.js permite gerenciar esse estado de "pending" visualmente, substituindo o texto do botão por um *spinner* de carregamento.[27]
- **Feedback de Bloqueio:** Se o sistema detectar (via verificação prévia) que um título já está liquidado no ERP, a ação de baixa deve ser bloqueada ou transformada em um "Sucesso Lógico" informativo. Em vez de um erro vermelho assustador, a UI deve mostrar um aviso amarelo ou azul: "Título já estava baixado no ERP (Sincronizado)", orientando o usuário sem causar alarme desnecessário.[2]

---

6. Design System de Estados Financeiros e Acessibilidade

A taxonomia de estados no Svelto é complexa e vital para a operação. Um sistema de cores semântico e acessível é obrigatório para garantir que auditores possam identificar riscos rapidamente.

6.1. Paleta de Cores Semântica e Acessível

O uso de cores deve ir além do tradicional "Verde para bom, Vermelho para ruim", especialmente considerando a prevalência de daltonismo (deficiência de visão de cores vermelho-verde) na população masculina (que compõe uma parte significativa da força de trabalho financeira). Confiar apenas na cor para transmitir estado é uma falha de acessibilidade (WCAG).

- **Estratégia de Dupla Codificação:** Todo status deve ser comunicado por **Cor + Texto + Ícone**. Por exemplo, um estado de CHARGEBACK não deve ser apenas um badge vermelho. Deve incluir um ícone de alerta (ShieldAlert do Lucide React) e o texto claro.
- **Paleta Adaptada (Colorblind-Safe):** Evite a combinação pura de Vermelho vs. Verde para estados críticos. Utilize Vermelho/Laranja para erros/riscos e Azul/Turquesa (Teal ou Emerald no Tailwind) para sucesso. O azul é geralmente seguro para a maioria dos tipos de daltonismo.
    - **CONCILIATED:** bg-emerald-50 text-emerald-700 border-emerald-200 + Ícone CheckCircle2.
    - **CHARGEBACK:** bg-rose-50 text-rose-700 border-rose-200 + Ícone ShieldAlert.
    - **PENDING:** bg-slate-100 text-slate-700 border-slate-200 + Ícone Clock.
    - **DIVERGENT:** bg-amber-50 text-amber-700 border-amber-200 + Ícone

AlertTriangle.

- **Alto Contraste:** Assegure que o texto do badge tenha contraste suficiente (WCAG AA ou AAA) contra o fundo do badge. O Tailwind fornece escalas de cores (ex: 700 sobre 50 ou 100) que geralmente atendem a esses critérios, mas a validação manual é necessária.[30]

6.2. Modo Escuro (Dark Mode) em Ambientes Financeiros

Profissionais de finanças frequentemente trabalham longas horas em ambientes com múltiplas telas. O suporte a **Modo Escuro** não é um luxo, mas um requisito de ergonomia visual.

- **Implementação Tailwind:** Utilize o modificador dark: para ajustar cores. Badges que são bg-red-50 no modo claro devem transitar para fundos mais escuros e translúcidos no modo noturno (ex: dark:bg-red-900/30 text-red-400) para evitar o brilho excessivo que causa fadiga ocular.
- **Bordas Sutis:** Em modo escuro, o contraste de fundos é menor. O uso de bordas sutis (border border-white/10) nos componentes ajuda a definir limites visuais sem aumentar o brilho da tela.[33]

---

7. Dashboard, Visualização de Dados e Linhagem

O Dashboard é a ferramenta de diagnóstico do gestor. Ele não deve apenas mostrar "o que aconteceu", mas "o que requer atenção" e "de onde vieram os dados" (Linhagem).

7.1. KPIs Acionáveis e Cards de Resumo

No topo do dashboard, utilize cards para métricas que exigem ação imediata, não apenas vaidade.

- **Conciliação Pendente:** Número e Valor de transações MATCHED aguardando confirmação. Ação sugerida: "Resolver Agora".
- **Risco de Chargeback:** Valor total bloqueado em disputas. Este card deve usar cor de alerta para destacar a urgência financeira.
- **Divergência de Taxas (ROI):** Valor acumulado de taxas cobradas a maior pelo gateway. Isso demonstra o valor do produto Svelto imediatamente para o usuário.

7.2. Visualização de Linhagem de Dados (Data Lineage UI)

Para cumprir a promessa de "Auditoria Forense", o sistema deve oferecer uma visualização da **Linhagem dos Dados**. O usuário precisa confiar na origem do dado.

- **Diagrama de Fluxo:** Para uma transação selecionada, ofereça uma visualização (talvez um modal ou uma aba "Linhagem") que mostre o fluxo do dado: *Ícone Maquininha (Origem) -> Processamento Gateway (Taxas aplicadas) -> Svelto (Ingestão e Auditoria) -> ERP (Escrita Contábil) -> Banco (Conciliação Final)*.

- **Rastreabilidade:** Cada etapa desse fluxo deve mostrar o timestamp exato e o ID do evento, provando que o dado não foi "inventado" pelo sistema, mas sim transformado a partir de uma fonte confiável. Isso é crucial para auditorias externas.[34]

7.3. Feedback de Carregamento (Skeleton Screens)

Gráficos financeiros e tabelas grandes são pesados para calcular. Nunca utilize uma tela branca ou um spinner solitário enquanto os dados carregam. Utilize **Skeleton Screens** (telas de esqueleto).

- **Padrão Visual:** Renderize "barras cinzas pulsantes" (animate-pulse bg-slate-200) que mimetizam a estrutura da tabela ou a forma do gráfico (barras, linhas).
- **Benefício UX:** Isso reduz a percepção de lentidão e, mais importante, evita o *Cumulative Layout Shift* (CLS), onde a interface "pula" quando os dados chegam, o que é frustrante e desorientador para o usuário.[37]

---

8. Tratamento de Exceções e Ciclo de Vida de Chargebacks

O "Caminho Feliz" (venda aprovada e conciliada) é fácil. A robustez da UI do Svelto é testada nas exceções, especialmente no ciclo de vida de **Chargebacks**.

8.1. UI de Gestão de Chargebacks

O chargeback não é um estado estático; é um processo com prazos. A UI deve refletir isso.

- **Alertas de Prazo:** Transações em chargeback devem exibir claramente o prazo para envio de defesa (ex: "Responder até 15/01").
- **Status de Disputa:** Acompanhe visualmente o progresso: *Notificação -> Em Mediação -> Defesa Enviada -> Ganho/Perdido*. Utilize ícones de status progressivos.
- **Separação Contábil:** Visualmente, separe o valor bloqueado do saldo disponível. Em dashboards de fluxo de caixa, mostre o impacto do chargeback como uma "Provisão de Perda" ou "Bloqueio Judicial", garantindo que o gestor não conte com dinheiro que não tem.[40]

8.2. Feedback de Erros de Integração

Em sistemas distribuídos, a API do Mercado Pago pode cair ou o Omie pode dar timeout. A UI deve ser resiliente.

- **Toasts para Erros Transientes:** Se uma sincronização falhar por timeout, mostre um *Toast* (notificação flutuante) com opção de "Tentar Novamente" (Retry). Não bloqueie a tela inteira.
- **Indicadores de ERROR_SYNC:** Se uma transação específica falhou na sincronização, marque-a com um badge ERROR_SYNC e ofereça um *tooltip* com a mensagem técnica

do erro (ex: "Erro 500: Gateway indisponível"). Isso permite que o usuário técnico ou suporte diagnostique o problema sem acessar logs de servidor.[42]

---

9. Diretrizes Técnicas para Frontend (Next.js + Tailwind)

A implementação técnica deve seguir padrões que garantam performance, manutenibilidade e segurança dos dados.

9.1. Server Components e Data Fetching

Utilize a arquitetura de **React Server Components (RSC)** do Next.js (App Router).

- **Segurança:** Busque dados sensíveis (tokens, chaves de API) e execute a lógica de banco de dados (Prisma) exclusivamente nos Server Components (page.tsx). Nunca exponha essa lógica ou chaves ao cliente.
- **Performance:** Passe os dados serializados para os Client Components (Tabelas, Gráficos). Isso reduz drasticamente o tamanho do bundle JavaScript enviado ao navegador, melhorando o *First Contentful Paint* (FCP) e a responsividade da aplicação em dispositivos mais lentos.[43]

9.2. Server Actions para Mutações Seguras

Para ações de escrita (Baixar no ERP, Conciliar), utilize **Server Actions**.

- **Segurança de Tipo:** Server Actions oferecem tipagem end-to-end integrada, garantindo que os dados enviados pelo formulário correspondam ao esperado pelo backend.
- **Revalidação de Cache:** Após uma ação de sucesso (ex: conciliação realizada), utilize revalidatePath ou revalidateTag para atualizar automaticamente os dados na tela do usuário sem a necessidade de um *refresh* manual, mantendo a "Quádrupla Verdade" sempre sincronizada.[27]

9.3. Otimização de Layout e Navegação

Utilize o arquivo layout.tsx do Next.js para persistir a estrutura de navegação (Sidebar, Header).

- **Persistência de Estado:** O estado do menu lateral (expandido/contraído) deve ser salvo em Cookies ou LocalStorage para evitar o efeito de *flicker* (layout shift) durante a navegação entre páginas. A estabilidade visual é fundamental para a percepção de qualidade em software profissional.[45]

---

10. Conclusão: A Interface como Agente de

Conformidade

A interface do Svelto SaaS não é uma camada passiva de apresentação; ela atua como um **Agente de Conformidade Digital** (Digital Compliance Officer). Ao proibir a baixa sem vínculo bancário correto, ao gritar visualmente divergências de centavos e ao impor fluxos de revisão para ambiguidades, a UI protege a integridade financeira da empresa.

A adesão rigorosa a este manual — desde a escolha da fonte tabular até a implementação de Server Actions idempotentes — garante que o frontend do Svelto entregue a promessa de "Engenharia da Verdade". O resultado é um sistema onde a estética serve à ética contábil, e onde a experiência do usuário é definida pela tranquilidade de saber que os números na tela são, inequivocamente, reais.

Seção Detalhada: Referência de Implementação de Componentes

11. Estrutura do DataTable (TanStack + Tailwind)

A implementação da tabela deve ser modular para manutenibilidade.

TypeScript

```
// app/transactions/columns.tsx
"use client"

import { ColumnDef } from "@tanstack/react-table"
import { Transaction } from "@/types/schema" // Tipagem Prisma
import { StatusBadge } from "@/components/ui/status-badge"
import { formatCurrency } from "@/lib/utils"

export const columns: ColumnDef<Transaction> ="
      />
    ),
    cell: ({ row }) => (
      <Checkbox
        checked={row.getIsSelected()}
```

```
      onCheckedChange={(value) => row.toggleSelected(!!value)}
      aria-label="Selecionar linha"
      className="translate-y-[2px]"
    />
  ),
  enableSorting: false,
  enableHiding: false,
},
{
  accessorKey: "dateEvent",
  header: "Data Venda",
  cell: ({ row }) => {
    // Formatação de data com timezone correto é CRÍTICA
    // Exibe data e hora para precisão operacional
    return <div className="font-medium text-slate-700 whitespace-nowrap tabular-nums">
      {formatDate(row.getValue("dateEvent"), "dd/MM/yyyy HH:mm")}
    </div>
  },
},
{
  accessorKey: "gatewayId",
  header: "NSU / ID",
  cell: ({ row }) => (
    // Fonte monoespaçada para identificadores técnicos
    <div className="font-mono text-xs text-slate-500 truncate max-w-[120px]
tabular-nums" title={row.getValue("gatewayId")}>
      {row.getValue("gatewayId")}
    </div>
  ),
},
{
  accessorKey: "amountGross",
  header: () => <div className="text-right">Valor Bruto</div>,
  cell: ({ row }) => {
    const amount = parseFloat(row.getValue("amountGross"))
    // Alinhamento à direita e tabular-nums para valores monetários
    return <div className="text-right font-mono tabular-nums font-medium text-slate-900">
      {formatCurrency(amount)}
    </div>
  },
},
{
  accessorKey: "status",
```

```
    header: "Status",
    cell: ({ row }) => {
      // Badge com lógica semântica de cores e ícones
      return <StatusBadge status={row.getValue("status")} />
    },
  },
  {
    id: "actions",
    cell: ({ row }) => <DataTableRowActions row={row} />,
  },
]
```

12. Glossário Visual de Ícones (Consistência)

Mantenha a consistência semântica nos ícones (recomendação: lucide-react) para facilitar a leitura rápida.

- **Ações:**
  - Editar: Pencil
  - Baixar/Liquidar: ArrowDownToLine ou Stamp (Metáfora de carimbo contábil)
  - Investigar/Detalhes: Search ou FileSearch
  - Estornar: RotateCcw
- **Entidades:**
  - Transação: CreditCard
  - Conta Bancária: Landmark
  - Cliente: User
  - ERP: Database
- **Status e Alertas:**
  - Sucesso/Conciliado: CheckCircle2
  - Aviso/Divergente: AlertTriangle
  - Erro/Falha: XCircle
  - Chargeback: ShieldAlert
  - Pendente: Clock

13. Considerações Finais de Implementação

A prioridade zero é a **Integridade dos Dados**. Se a UI mostrar que um título está "Conciliado", ele deve estar matematicamente e sistemicamente conciliado nas quatro camadas da verdade. Qualquer otimização de "Optimistic UI" deve ser tratada com extrema cautela e deve ser capaz de se auto-corrigir (reverter) agressivamente em caso de falha no backend, comunicando o erro de forma clara ao usuário. O usuário financeiro pode perdoar uma interface utilitária, mas jamais perdoará uma interface que minta sobre o estado do seu dinheiro.

Fim do Relatório Técnico - Svelto Architect

Works cited

1. Design better data tables by Andrew Coyle, accessed January 19, 2026, https://www.andrewcoyle.com/blog/design-better-data-tables
2. docs_Svelto_SaaS_Business_Rules_v1.0.md.pdf
3. Data Table Design UX Patterns & Best Practices - Pencil & Paper, accessed January 19, 2026, https://www.pencilandpaper.io/articles/ux-pattern-analysis-enterprise-data-tables
4. Visualizing data in user interfaces by Andrew Coyle, accessed January 19, 2026, https://www.andrewcoyle.com/blog/visualizing-data-in-user-interfaces
5. Data Table Design: The Trick in Reconciliation | by Xiao Luo - Medium, accessed January 19, 2026, https://medium.com/@xiaoluo712/data-table-design-the-trick-in-reconciliation-ba529b344726
6. Behind the Scenes: Designing Our New UI - Modern Treasury, accessed January 19, 2026, https://www.moderntreasury.com/journal/behind-the-scenes-designing-our-new-ui
7. Tailwind CSS Font Variant Numeric - Kombai, accessed January 19, 2026, https://kombai.com/tailwind/font-variant-numeric/
8. font-variant-numeric - Typography - Tailwind CSS, accessed January 19, 2026, https://tailwindcss.com/docs/font-variant-numeric
9. Formatting Numbers - SAP, accessed January 19, 2026, https://www.sap.com/design-system/fiori-design-web/v1-136/foundations/best-practices/ui-elements/formatting/formatting-numbers
10. Tailwind CSS tutorial #20: Font Variant Numeric - DEV Community, accessed January 19, 2026, https://dev.to/fromshubhi/tailwind-css-tutorial-20-font-variant-numeric-1656
11. How to format tables with monetary data? - User Experience Stack Exchange, accessed January 19, 2026, https://ux.stackexchange.com/questions/51126/how-to-format-tables-with-monetary-data
12. Designing effective data table UI – best practices and tips - Justinmind, accessed January 19, 2026, https://www.justinmind.com/ui-design/data-table
13. The UX of Decimals | Javier Lo, accessed January 19, 2026, https://javierlo.com/blog/ux-of-decimals
14. How should 4 decimals places behave, being simple yet powerful - Stack Overflow, accessed January 19, 2026, https://stackoverflow.com/questions/2638147/how-should-4-decimals-places-behave-being-simple-yet-powerful
15. Format a number or currency field - Microsoft Support, accessed January 19,

2026, https://support.microsoft.com/en-us/office/format-a-number-or-currency-field-e48f2312-67f0-4921-aca0-15d36b7f9c3b
16. What is best practice for currency padding/rounding when entering values in a text field?, accessed January 19, 2026, https://ux.stackexchange.com/questions/60522/what-is-best-practice-for-currency-padding-rounding-when-entering-values-in-a-te
17. TanStack Table, accessed January 19, 2026, https://tanstack.com/table/v8
18. I Built an Enterprise-Grade Data Table for Next.js - Now Available via Shadcn Registry! - DEV Community, accessed January 19, 2026, https://dev.to/jacksonkasi/i-built-an-enterprise-grade-data-table-for-nextjs-now-available-via-shadcn-registry-406h
19. The Next.js Table Tango: Mastering Dynamic Data Tables with Server-Side Performance & Client-Side Fluidity | by Divyansh Sharma | Medium, accessed January 19, 2026, https://medium.com/@divyanshsharma0631/the-next-js-table-tango-mastering-dynamic-data-tables-with-server-side-performance-client-side-a71ee0ec2c63
20. TanStack Table: Backend Pagination, Filter & Sort with TanStack Query + TanStack Router - YouTube, accessed January 19, 2026, https://www.youtube.com/watch?v=F4zshDlnsJY
21. How to optimize TanStack Table (React Table) for rendering 1 million rows? : r/reactjs - Reddit, accessed January 19, 2026, https://www.reddit.com/r/reactjs/comments/1pk0ipl/how_to_optimize_tanstack_table_react_table_for/
22. Improve performance and memory use for large datasets by mleibman-db · Pull Request #5927 · TanStack/table - GitHub, accessed January 19, 2026, https://github.com/TanStack/table/pull/5927
23. Check box for manual reconciliation - Quicken Community, accessed January 19, 2026, https://community.quicken.com/discussion/7955511/check-box-for-manual-reconciliation
24. Understanding optimistic UI and React's useOptimistic Hook - LogRocket Blog, accessed January 19, 2026, https://blog.logrocket.com/understanding-optimistic-ui-react-useoptimistic-hook/
25. useOptimistic for Optimistic UI in Next.js Server Actions (+ Revalidate) - YouTube, accessed January 19, 2026, https://www.youtube.com/watch?v=PPOw-sDeoNw
26. Walking through the Manual Matching Process Using a Sample Reconciliation, accessed January 19, 2026, https://docs.oracle.com/en/cloud/saas/account-reconcile-cloud/raarc/reconcile_trans_match_manual_match_example.html
27. 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
28. Getting Started: Updating Data - Next.js, accessed January 19, 2026,

https://nextjs.org/docs/app/getting-started/updating-data
29. How to prevent form submission with enter key when a server action is pending?, accessed January 19, 2026, https://stackoverflow.com/questions/77902231/how-to-prevent-form-submission-with-enter-key-when-a-server-action-is-pending
30. Guidelines color blind friendly figures | Netherlands Cancer Institute, accessed January 19, 2026, https://www.nki.nl/about-us/responsible-research/guidelines-color-blind-friendly-figures
31. Accessible Color Palette Generator | WCAG Compliant - Venngage, accessed January 19, 2026, https://venngage.com/tools/accessible-color-palette-generator
32. Tailwind CSS Badge, Label, Chip | Free Preline UI Components, accessed January 19, 2026, https://preline.co/docs/badge.html
33. color-scheme - Interactivity - Tailwind CSS, accessed January 19, 2026, https://tailwindcss.com/docs/color-scheme
34. 12 Financial Dashboard Examples & Templates - Qlik, accessed January 19, 2026, https://www.qlik.com/us/dashboard-examples/financial-dashboards
35. Data Lineage - How to design with lots of related data - Underbelly, accessed January 19, 2026, https://www.underbelly.is/writing-about/bigeye-data-lineage
36. Data Lineage in 2025: Types, Techniques, Use Cases & Examples - Dagster, accessed January 19, 2026, https://dagster.io/learn/data-lineage
37. How to use React MUI Skeleton to build a loading placeholder for dashboard - Kombai, accessed January 19, 2026, https://kombai.com/mui/skeleton/
38. Skeleton Screens vs. Progress Bars vs. Spinners - YouTube, accessed January 19, 2026, https://www.youtube.com/watch?v=4GWqJEfzvmg
39. Implementing Skeleton Screens In React - Smashing Magazine, accessed January 19, 2026, https://www.smashingmagazine.com/2020/04/skeleton-screens-react/
40. First Chargeback | Payments Developer Portal, accessed January 19, 2026, https://developer.payments.jpmorgan.com/docs/commerce/dispute-management/first-chargeback
41. Explanation of the Chargeback Process & Flow - Kount, accessed January 19, 2026, https://kount.com/blog/explanation-chargeback-process-flow
42. 13 UI Design Patterns You Need to Know for Modern Interfaces - Designership, accessed January 19, 2026, https://www.thedesignership.com/blog/13-ui-design-patterns-you-need-to-know-for-modern-interfaces
43. Getting Started: Server and Client Components - Next.js, accessed January 19, 2026, https://nextjs.org/docs/app/getting-started/server-and-client-components
44. Getting Started: Fetching Data - Next.js, accessed January 19, 2026, https://nextjs.org/docs/app/getting-started/fetching-data
45. Creating Layouts and Pages - App Router - Next.js, accessed January 19, 2026, https://nextjs.org/learn/dashboard-app/creating-layouts-and-pages