Isadora Belmont

# Sistema de Gestão Financeira
# Especificação Complementar

**Versão 1.0**

# Histórico da Revisão

| Data | Versão | Descrição | Autor |
|---|---|---|---|
| 09/03/2025 | 1.0 | Criação do documento inicial | Lucas Rezende de Sales |
| 04/04/2025 | 2.0 | Atualização de escopo | Lucas Rezende de Sales |
| | | | |
| | | | |

# Índice

# Especificação Complementar

## 1. Introdução

### 1.1 Objetivo

Este documento visa complementar a especificação do sistema, detalhando requisitos não funcionais como segurança, desempenho, confiabilidade, suporte e restrições tecnológicas.

### 1.2 Escopo

O sistema será utilizado exclusivamente pela equipe da Isabela Belmont, auxiliando no gerenciamento financeiro interno. Ele não será comercializado externamente e será acessível apenas para usuários autorizados.

### 1.3 Definições, Acrônimos e Abreviações

**Fluxo de Caixa**: Controle financeiro das movimentações de entrada e saída.

**Dashboard**: Painel interativo com gráficos e relatórios financeiros.

**CRUD**: Create, Read, Update, Delete – operações básicas de um sistema de gestão de dados.

### 1.4 Referências

IEEE Std 830-1998 – Recommended Practice for Software Requirements Specifications.

Documentação oficial do Next.js – https://nextjs.org/docs.

Documentação do Prisma ORM – https://www.prisma.io/docs.

### 1.5 Visão Geral

Este documento contém os requisitos complementares do sistema, detalhando aspectos técnicos, segurança, desempenho e suporte.

## 2. Funcionalidade

### 2.1 Cadastro de Despesas e Receitas

O sistema deve permitir inserção, edição e exclusão de transações financeiras. As transações serão categorizadas por tipo (receita/despesa) e método de pagamento (Pix, Boleto, Cartão).

### 2.2 Dashboard Financeiro

O painel deve fornecer gráficos interativos, saldo atualizado e relatórios personalizados. Permitir filtros por data, categoria e método de pagamento.

### 2.3 Gestão de Faturas e Remessas

O sistema oferece o gerenciamento de faturas de contas a pagar e a receber, permitindo o acompanhamento de status e valores totais de cada fatura. As remessas financeiras também podem ser registradas, incluindo informações como data de envio, quantidade de títulos e valor total da remessa.

### 2.4 Gestão de Contas Bancárias

O sistema permite a gestão de contas bancárias, com cada conta podendo ter um saldo, um responsável e estar ativa ou inativa. As transações financeiras são associadas a essas contas, facilitando a reconciliação dos fluxos de caixa.

### 2.5 Gestão de Usuários

O sistema permite o gerenciamento de usuários, com controle de acesso baseado em funções (admin, usuário, etc.). Cada usuário tem informações de login e autenticação, e o sistema possibilita a criação, modificação e exclusão de contas de usuários.

### 2.6 Relatórios Personalizados

O sistema oferece a criação de relatórios financeiros personalizados, com base em filtros como período de tempo, categoria de transação, conta bancária e status de fatura. Esses relatórios ajudam os usuários a analisar suas finanças de forma detalhada e tomar decisões mais informadas.

### 2.7 Fluxo de Caixa

A funcionalidade de fluxo de caixa permite o controle de todas as entradas e saídas financeiras de cada usuário e conta bancária. Cada transação é registrada com uma descrição, valor, data e tipo (receita ou despesa), facilitando a gestão do fluxo de caixa.

### 2.8 Gestão de Faturas e Status de Pagamento

O sistema permite o monitoramento de faturas a pagar e a receber, com a possibilidade de acompanhar seu status (pago, pendente, cancelado) e as datas de vencimento, pagamento e recebimento.

## 3. Utilidade

### 3.1 Facilidade de Uso

A interface deve ser intuitiva e acessível para usuários sem conhecimento técnico avançado.

### 3.2 Acessibilidade:

O sistema deve ser compatível com navegadores modernos e dispositivos móveis.

## 4. Confiabilidade

### 4.1 Disponibilidade

O sistema deve garantir uma disponibilidade mínima de 99,5%, permitindo no máximo 3,6 horas de inatividade não planejada por mês.

O período de manutenção preventiva será agendado em horários de menor uso (entre 00h e 06h, horário de Brasília) para evitar impactos operacionais.

Caso o sistema entre em modo de degradação devido a alta carga ou falha parcial, funcionalidades críticas (como cadastro de transações e consulta de saldo) continuarão operacionais.

### 4.2 MTBF (Mean Time Between Failures)

O sistema deve apresentar um MTBF mínimo de 2.000 horas (aproximadamente 83 dias), garantindo baixa incidência de falhas operacionais.

As falhas devem ser registradas e analisadas para a implementação de medidas corretivas que aumentem o tempo médio entre falhas.

### 4.3 MTTR (Mean Time To Repair)

Em caso de falhas críticas, o sistema deve ser restaurado em um prazo máximo de 30 minutos.

Falhas menores devem ser resolvidas em um período máximo de 2 horas.

O sistema contará com logs de erro estruturados e alertas automáticos, permitindo rápida identificação e correção de problemas.

### 4.4 Exatidão e Precisão das Operações

Todas as operações financeiras devem ser registradas com precisão de até duas casas decimais para valores monetários, seguindo o padrão BRL (R$).

O cálculo do saldo financeiro deve ser exato e livre de arredondamentos imprecisos, garantindo que a soma de receitas e despesas sempre resulte no valor correto.

As datas e horários das transações devem ser armazenadas em UTC para evitar discrepâncias entre fusos horários.

### 4.5 Taxa Máxima de Erros ou Defeitos

O sistema deve manter uma taxa de erro inferior a 0,5% por 1.000 linhas de código (KLOC).

Falhas críticas (exemplo: perda de dados financeiros) não podem ocorrer sob nenhuma circunstância.

Pequenos erros visuais ou operacionais devem ser corrigidos no prazo máximo de 7 dias úteis após identificação.

### 4.6 Classificação de Erros e Defeitos

| Tipo de Erro | Impacto | Tempo Máximo de Resolução |
| --- | --- | --- |
| **Erro Crítico** | Perda de dados financeiros, falha na autenticação de usuários, falha completa do sistema. | **30 minutos** |
| **Erro Significativo** | Problemas na exibição de informações financeiras, falhas em relatórios, lentidão excessiva. | **2 horas** |
| **Erro Menor** | Problemas visuais na interface, pequenos bugs sem impacto financeiro. | **Até 7 dias úteis** |

## 5. Desempenho

### 5.1 Tempo de Resposta

O tempo médio de resposta para consultas ao banco de dados deve ser inferior a 500ms.

O tempo máximo de resposta para qualquer requisição não deve ultrapassar 2 segundos, exceto em operações complexas como geração de relatórios extensos, onde o tempo máximo permitido será 5 segundos.

O carregamento do dashboard principal deve ocorrer em menos de 2 segundos para exibir os gráficos e métricas financeiras iniciais.

A exibição de detalhes individuais de transações deve ter resposta em menos de 1 segundo.

### 5.2 Rendimento do Processamento

O sistema deve ser capaz de processar pelo menos 100 transações financeiras por segundo sem degradação perceptível no desempenho.

A API deve suportar até 200 requisições simultâneas sem impactar a usabilidade.

O tempo de inserção de uma nova transação deve ser inferior a 1 segundo.

Relatórios financeiros podem ser gerados em lote, otimizando a carga no servidor para evitar bloqueios de processamento.

### 5.3 Capacidade e Escalabilidade

O banco de dados deve ser dimensionado para armazenar um mínimo de 1 milhão de registros de transações sem degradação no desempenho.

O sistema deve ser capaz de escalar horizontalmente para atender a um aumento de 50% na carga de transações financeiras sem necessidade de reformulação estrutural.

### 5.4 Modos de Degradação

Caso o sistema atinja limites críticos de carga, ele deve operar em um modo de degradação controlada, adotando as seguintes estratégias:

Priorização de Requisições → Durante picos de uso, as requisições essenciais (como consultas de saldo e registro de transações) terão prioridade sobre operações menos críticas (como exportação de relatórios e geração de gráficos).

Cache Inteligente → Para reduzir a sobrecarga de consultas repetitivas, o sistema armazenará temporariamente os dados mais acessados, reduzindo a necessidade de processamento em tempo real.

Fila Assíncrona para Relatórios → Geração de relatórios extensos será movida para um processamento assíncrono, onde o usuário será notificado quando o relatório estiver pronto.

Desativação Temporária de Funcionalidades Não Essenciais → Em casos extremos, recursos como atualizações automáticas de gráficos e exibição de estatísticas detalhadas podem ser temporariamente desativados para preservar a estabilidade.

### 5.5 Utilização de Recursos

**Memória**: O sistema deve utilizar menos de 500MB de RAM por instância ativa, exceto em operações de alto processamento como geração de relatórios.

**Disco**: O armazenamento em banco de dados deve ser otimizado para evitar crescimento desnecessário, com estratégias de arquivamento de dados antigos para reduzir a carga do banco transacional.

**Comunicações**: Todas as requisições entre cliente e servidor devem ser otimizadas via compressão Gzip para minimizar o consumo de banda.

**Banco de Dados**: O PostgreSQL será configurado para suportar alta concorrência e otimizar consultas por meio de índices e estratégias de cache.

## 6. Suportabilidade

### 6.1 Padrões de Codificação e Convenções de Nomenclatura

O sistema seguirá as diretrizes de Clean Code e SOLID para garantir um código legível, modular e de fácil manutenção.

As convenções de nomenclatura seguirão os padrões:

Variáveis e Funções → camelCase (exemploDeVariavel).

Classes e Componentes → PascalCase (ExemploDeComponente).

Constantes → UPPER_CASE (EXEMPLO_DE_CONSTANTE).

Rotas de API → RESTful, utilizando snake_case (/api/obter_transacoes).

O código-fonte será escrito em TypeScript, reduzindo erros de tipagem e aumentando a robustez do sistema.

Todos os commits no repositório seguirão o padrão Conventional Commits, garantindo clareza no histórico de alterações.

### 6.2 Bibliotecas de Classe e Dependências

**Frontend**:

Next.js para o desenvolvimento de interfaces web.

ShadCN/UI e Tailwind CSS para estilização e componentes reutilizáveis.

Recharts para visualização de dados gráficos.

**Backend**:

Node.js + Next.js API Routes para processamento de lógica de negócio.

Prisma ORM para manipulação eficiente do banco de dados.

Zod para validação de dados no backend.

jsonwebtoken (JWT) para autenticação segura.

**Banco de Dados:**

PostgreSQL como banco relacional principal.

NeonDB como provedor em nuvem.

Redis (opcional) para cache de dados frequentemente acessados.

### 6.3 Acesso e Manutenção do Sistema

O código-fonte será armazenado em um repositório Git privado, utilizando GitHub ou GitLab para versionamento e controle de alterações.

O sistema será desenvolvido seguindo a estratégia de branching model Git Flow, separando:

**Main** → Apenas para versões estáveis.

**Develop** → Para funcionalidades em andamento.

**Feature Branches** → Para novas implementações antes de serem mescladas.

**Log de Erros:**

Logs serão registrados com Winston no backend para análise de falhas.

Erros críticos serão reportados em tempo real via webhook para canais internos da equipe de suporte.

**Monitoramento:**

Implementação de Sentry ou LogRocket para capturar e reportar erros em produção.

Monitoramento do servidor utilizando Grafana e Prometheus para análise de métricas.

### 6.4 Estratégias de Atualização e Suporte

**Deploy contínuo (CI/CD):**

O sistema será integrado com pipelines de GitHub Actions ou GitLab CI/CD para automação de testes e deploy.

**Backups e Recuperação:**

O banco de dados terá backups automáticos diários armazenados por 30 dias.

Caso haja falha no sistema, será possível restaurar uma versão anterior em menos de 5 minutos.

**Ambientes de Desenvolvimento e Produção:**

Será mantido um ambiente de staging para testes antes da publicação.

O ambiente de produção será isolado e protegido contra alterações diretas.

**Documentação**:

O código será documentado utilizando Rup template para facilitar a leitura e manutenção.

Um manual técnico será criado para orientar futuras manutenções no sistema.

## 7. Restrições de Design

### 7.1 Linguagens de Software

**Frontend**: Desenvolvido com TypeScript e Next.js para garantir melhor manutenibilidade, segurança e tipagem estática.

**Backend**: Implementado em Node.js utilizando Next.js API Routes, permitindo melhor integração entre front e back-end.

**Banco de Dados**: Utilização do PostgreSQL devido à sua robustez, escalabilidade e suporte para transações ACID.

**Consultas ao Banco**: Gerenciadas por Prisma ORM, reduzindo complexidade e melhorando a integração com o banco de dados.

### 7.2 Arquitetura e Padrões de Desenvolvimento

O sistema adota uma arquitetura modular, separando responsabilidades entre frontend, backend e banco de dados.

O padrão RESTful API será seguido para comunicação entre front e back-end, garantindo eficiência e escalabilidade.

O backend será stateless, ou seja, não manterá informações de sessão do usuário no servidor, utilizando JWT para autenticação.

O banco de dados utilizará migrations do Prisma para versionamento de schema e controle de alterações.

Utilização do padrão MVC (Model-View-Controller) para modularizar a lógica de negócio, organizando a aplicação de maneira escalável.

### 7.3 Ferramentas de Desenvolvimento e Tecnologias Obrigatórias

| Categoria | Ferramenta/Tecnologia | Motivo da Escolha |
|---|---|---|
| Frontend | Next.js + TypeScript | Melhor desempenho, SEO e suporte a SSR/SSG. |
| UI/UX | Tailwind CSS + ShadCN/UI | Desenvolvimento ágil, design responsivo e consistência visual. |
| Backend | Node.js + Next.js API Routes | Melhor integração entre front e back-end, escalabilidade. |
| Banco de Dados | PostgreSQL (NeonDB) | Suporte robusto a transações, consultas eficientes e escalabilidade. |
| ORM | Prisma ORM | Facilidade de manipulação de dados e integração segura. |

| | | |
| --- | --- | --- |
| Segurança | JWT + Bcrypt | Proteção de credenciais e autenticação segura. |
| Versionamento | Git + GitHub | Controle de versão e colaboração eficiente. |
| CI/CD | GitHub Actions | Automação de testes e deploys contínuos. |

### 7.4 Restrições de Arquitetura e Design

O sistema não permitirá múltiplos usuários simultâneos por conta, sendo um sistema de uso individual e interno para a Isabela Belmont.

Não será permitido o armazenamento de senhas em texto plano. Todas as senhas serão criptografadas com Bcrypt antes de serem salvas no banco de dados.

O design da interface seguirá diretrizes de UI/UX modernas, garantindo acessibilidade e compatibilidade com telas de diferentes tamanhos.

Gráficos e relatórios serão atualizados de forma assíncrona, evitando bloqueios na interface.

Todas as transações financeiras serão armazenadas com timestamp e logs de auditoria, permitindo rastreamento de operações.

### 7.5 Restrições de Infraestrutura e Ambiente de Execução

O sistema será 100% baseado na web, sem necessidade de instalação local.

O servidor deve operar em um ambiente Linux-based (Ubuntu ou Debian) para otimização de performance e segurança.

O NeonDB será o provedor de banco de dados, garantindo escalabilidade e suporte a backups automáticos.

Todos os logs críticos serão armazenados utilizando Winston para monitoramento de erros e falhas no sistema.

O sistema será containerizado utilizando Docker para facilitar o deploy e escalabilidade futura.

### 7.6 Restrições de Componentes Comprados e Licenciamento

Todas as bibliotecas utilizadas devem ser open-source ou possuir licenciamento gratuito para uso comercial.

Nenhum serviço externo pago será utilizado para armazenamento de dados financeiros.

O sistema não poderá utilizar tecnologias proprietárias que impeçam sua modificação futura, garantindo total controle da equipe interna da Isabela Belmont.

## 8. Documentação do Usuário On-line e Requisitos do Sistema de Ajuda

O sistema terá um manual interno para treinamento da equipe.

Será disponibilizada uma seção de ajuda dentro da plataforma.

## 9. Componentes Comprados

Não há dependência de componentes pagos externos nesta fase do projeto.

## 10. Interfaces

### 10.1 Interfaces com o Usuário

A interface com o usuário será baseada em um design responsivo e intuitivo, garantindo acessibilidade em

diferentes dispositivos.

**Características da Interface**

**Frontend:** Desenvolvido em Next.js com Tailwind CSS e ShadCN/UI.

**Modo Escuro/Claro:** Suporte a temas adaptáveis.

**Acessibilidade:** Conformidade com padrões WCAG (Web Content Accessibility Guidelines).

**Dashboard interativo:**

Exibição de gráficos dinâmicos.

Listagem de transações com filtros e paginação.

Visualização detalhada de receitas e despesas.

**Formulários de entrada de dados:**

Cadastro de novas transações.

Edição e remoção de registros.

**Navegação:**

Layout responsivo adaptado para desktop e mobile.

Atalhos de teclado para operações rápidas.

### 10.2 Interfaces de Hardware

O sistema não exige hardware específico, sendo projetado para rodar em qualquer dispositivo com um navegador moderno. No entanto, algumas considerações de compatibilidade são:

**Dispositivos Suportados:**

Computadores e notebooks com Windows, macOS e Linux.

Tablets e smartphones (Android/iOS).

**Requisitos Mínimos:**

Resolução mínima recomendada: 1366x768.

Conexão com a internet para acesso contínuo.

**Dispositivos de Entrada:**

Teclado e mouse/touchpad (para desktop).

Tela sensível ao toque (para dispositivos móveis).

### 10.3 Interfaces de Software

O sistema interage diretamente com diversas APIs e bibliotecas externas para garantir seu funcionamento eficiente.

| Componente | Descrição | Biblioteca/API |
|---|---|---|
| **Banco de Dados** | Armazenamento de transações e usuários | PostgreSQL (NeonDB) |
| **ORM** | Abstração para manipulação do banco de dados | Prisma ORM |
| **Autenticação** | Geração e verificação de tokens de login | JSON Web Token (JWT) |
| **Criptografia de Senhas** | Proteção de credenciais do usuário | Bcrypt |
| **Gráficos Interativos** | Exibição de relatórios financeiros | Recharts |
| **Validação de Dados** | Garantia de integridade nas requisições | Zod |
| **Cache de Dados** | Otimização de consultas frequentes | Redis (opcional) |

### 10.4 Interfaces de Comunicações

O sistema utiliza protocolos de comunicação seguros para garantir a integridade e privacidade dos dados.

**Protocolo de Transporte:**

Todas as comunicações serão realizadas via HTTPS (TLS 1.2+) para evitar interceptação de dados.

Requisições da API serão feitas via RESTful JSON, garantindo compatibilidade e eficiência.

**Portas Utilizadas:**

**Frontend**: Porta 3000 (Next.js) durante o desenvolvimento.

**Backend**: Porta 5000 no ambiente de produção.

**Banco de Dados:** PostgreSQL rodando na porta 5432.

**Rede e Segurança:**

O acesso ao banco será permitido apenas via IP autorizado, protegendo contra acessos externos.

Logs de requisições serão armazenados para auditoria e detecção de possíveis falhas.

## 11. Requisitos de Licença

O sistema será de uso exclusivo da Isabela Belmont, sem permissão para distribuição externa.

## 12. Observações Legais, sobre Direitos Autorais e Outras Observações

Todas as informações financeiras armazenadas são confidenciais e acessíveis apenas pela equipe autorizada.

Os dados serão protegidos em conformidade com a LGPD (Lei Geral de Proteção de Dados).

## 13. Padrões Aplicáveis

**Padrões de segurança de dados:** LGPD, OWASP, melhores práticas de autenticação JWT.

**Padrões de UI/UX:** Material Design, Web Content Accessibility Guidelines (WCAG).