

GUSTAVO JOÃO CHAVES



Ziro: Zere desperdícios, maximize ganhos.

ENGENHARIA DE SOFTWARE
22/05/2025

Resumo

A gestão financeira é um dos pilares fundamentais para a sustentabilidade e crescimento de pequenas e médias empresas. No entanto, muitas enfrentam dificuldades em organizar seus lançamentos, acompanhar despesas e interpretar dados financeiros de forma eficiente. Nesse contexto, soluções tecnológicas acessíveis e inteligentes tornam-se essenciais para apoiar decisões estratégicas e otimizar o uso dos recursos.

Neste documento é apresentada a especificação de uma aplicação web chamada **Ziro**, desenvolvida para simplificar a gestão financeira empresarial. A aplicação permite o registro e importação de lançamentos, categorização de despesas e receitas, e disponibiliza gráficos analíticos interativos. Utilizando recursos de inteligência artificial, o Ziro identifica padrões de comportamento financeiro, sugere oportunidades de economia e emite alertas sobre gastos atípicos, tornando-se uma ferramenta prática e estratégica para o público-alvo.

1. Introdução

A gestão financeira eficiente é um fator-chave para a sustentabilidade e o crescimento de pequenas e médias empresas. No entanto, muitas delas enfrentam desafios recorrentes como a desorganização dos lançamentos, a categorização manual e trabalhosa de despesas e receitas, e a escassez de informações claras para embasar decisões estratégicas. Essas limitações podem levar ao desperdício de recursos, à alocação ineficiente do capital e a decisões baseadas em suposições, comprometendo a competitividade dessas organizações.

Apesar da existência de sistemas financeiros no mercado, muitos deles são complexos, custosos ou não adaptados à realidade de pequenos empreendedores, que frequentemente carecem de suporte técnico ou tempo para configurações extensas. Além disso, há uma lacuna significativa na utilização de inteligência analítica e preditiva voltada à tomada de decisão simplificada nesses contextos.

Com o objetivo de enfrentar esses obstáculos, este projeto propõe o Ziro, uma aplicação web desenvolvida para apoiar a gestão financeira empresarial de forma prática e acessível. A aplicação permite que gestores importem ou registrem lançamentos financeiros, categorizem-nos manualmente com agilidade e visualizem os dados por meio de gráficos interativos e relatórios intuitivos. Como diferencial, o Ziro utiliza inteligência artificial para analisar padrões de comportamento financeiro, identificar anomalias e sugerir ajustes preventivos — funcionando como um apoio à tomada de decisão baseada em dados.

Espera-se que os recursos analíticos e os alertas personalizados do Ziro contribuam para uma maior consciência sobre o uso dos recursos financeiros, apoiando uma gestão mais estratégica e eficiente. A aplicação será desenvolvida com foco em usabilidade, escalabilidade e possibilidade de integração futura, utilizando tecnologias consolidadas, como Spring Boot e Vue.js, aliadas a boas práticas de engenharia de software voltadas à experiência do usuário e à confiabilidade dos dados.

Estudos preliminares indicam que ferramentas de gestão financeira com recursos analíticos podem ajudar empresas a reduzir desperdícios em até 10%. A validação dessa hipótese será explorada por meio de avaliações práticas, comparando indicadores financeiros antes e depois da adoção da aplicação ao longo de um período específico. Este trabalho, portanto, busca não apenas entregar uma solução funcional, mas também contribuir para o fortalecimento do ecossistema empreendedor por meio da tecnologia aplicada.

2. Descrição do Projeto

Tema do Projeto

Desenvolvimento de uma aplicação web voltada à gestão financeira de pequenas e médias empresas, com funcionalidades para registro de lançamentos, visualização de dados e apoio à análise de desempenho financeiro.

Problemas a Resolver

- Dificuldade na categorização de despesas e receitas.
- Inexistência de recursos que ajudem a prever e evitar gastos excessivos. O sistema utilizará inteligência artificial para identificar tendências de aumento de despesas e emitir alertas preventivos.
- Necessidade de visualização clara e estratégica dos resultados financeiros da empresa.
- Ausência de alertas personalizados baseados em metas financeiras e anomalias de comportamento, como variações incomuns em determinados tipos de gastos. A inteligência artificial permitirá detectar comportamentos atípicos e notificar o usuário proativamente.

Limitações

- O sistema não realizará integração com sistemas bancários ou ERPs externos nesta primeira versão.
- Transações financeiras não serão executadas dentro da plataforma, sendo ela exclusivamente de controle e análise.

- A inteligência artificial será limitada à sugestão de ajustes e identificação de padrões com base no histórico de lançamentos.
- O sistema será focado em pequenas e médias empresas, não incluindo funcionalidades complexas exigidas por grandes corporações ou contabilidade avançada.

3. Especificação Técnica

Esta seção descreve de forma detalhada os aspectos técnicos da proposta, incluindo os requisitos de software, decisões de design, tecnologias utilizadas e ferramentas que serão aplicadas ao longo do desenvolvimento do sistema.

3.1. Requisitos de Software

A seguir, são apresentados os requisitos necessários para o funcionamento do sistema inteligente de gestão financeira empresarial. Eles foram classificados em **Requisitos Funcionais (RF)** e **Requisitos Não Funcionais (RNF)**, e numerados para facilitar a rastreabilidade ao longo do projeto.

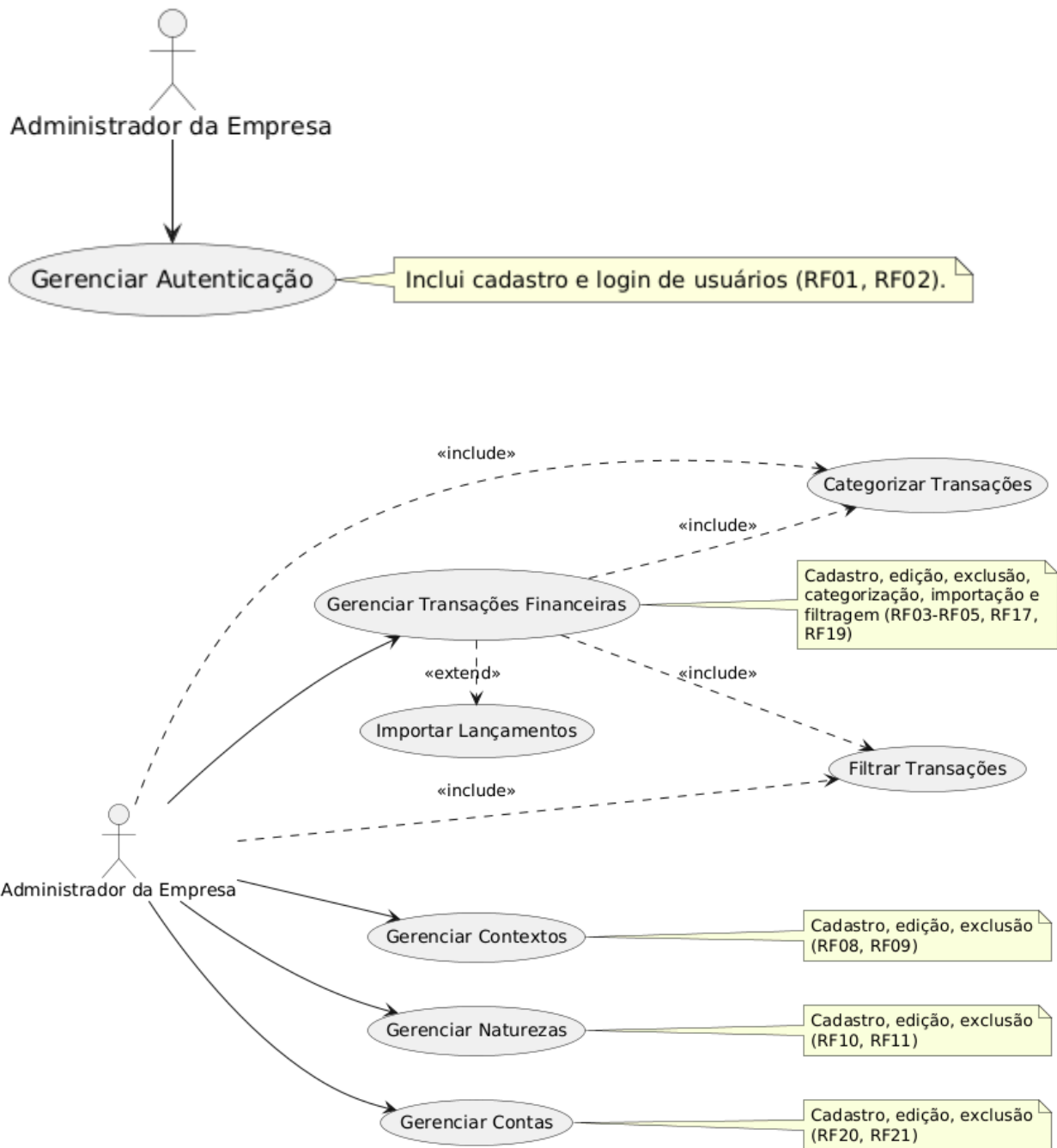
Requisitos Funcionais (RF):

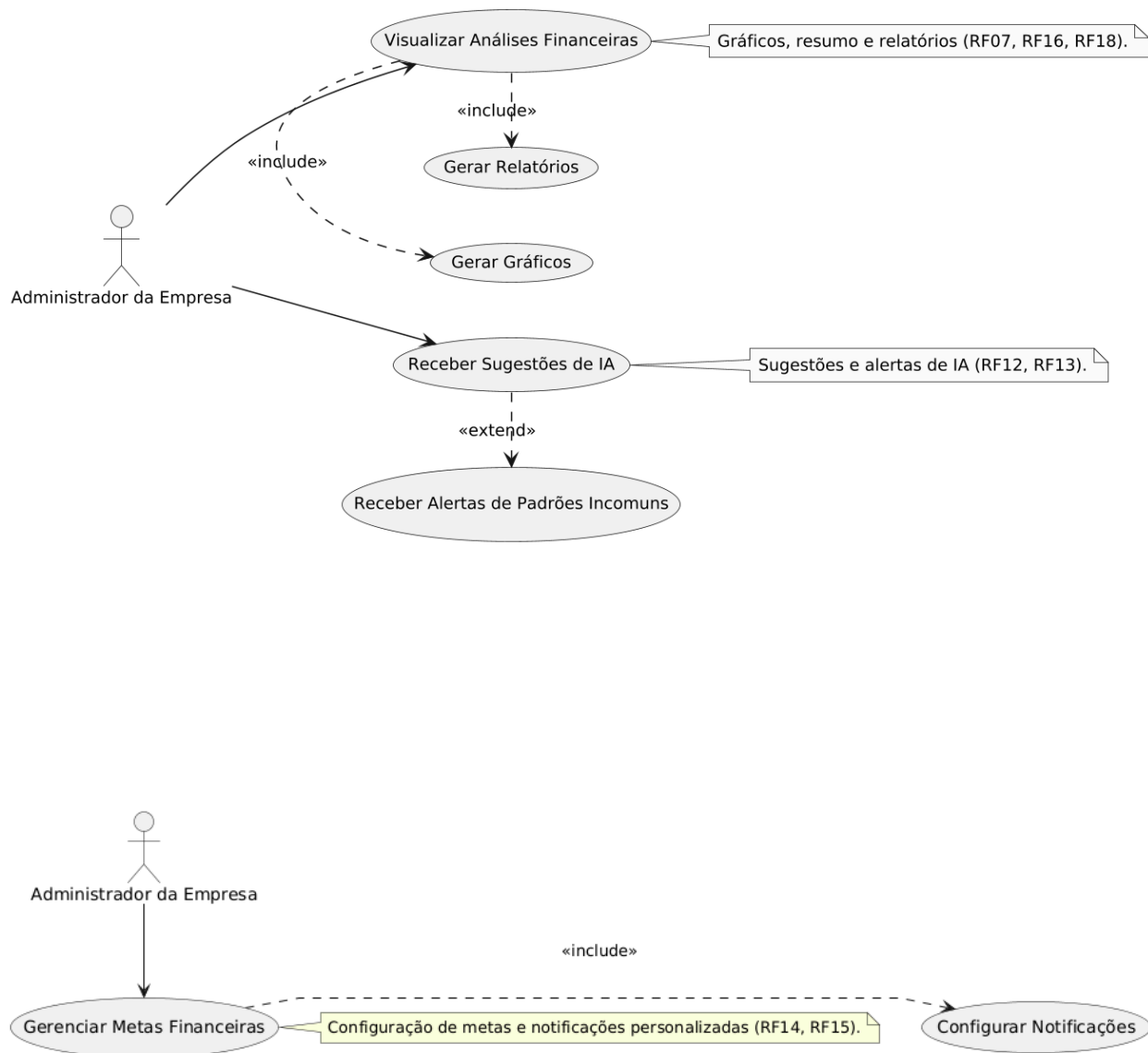
- **RF01:** O sistema deve permitir o **cadastro de usuários** com dados como nome, e-mail e senha.
- **RF02:** O sistema deve permitir a **autenticação de usuários** com verificação de credenciais.
- **RF03:** O sistema deve permitir o **cadastro manual de lançamentos financeiros**, com campos como data, valor, natureza e descrição.
- **RF04:** O sistema deve permitir a **importação de lançamentos financeiros** a partir de arquivos nos formatos CSV ou Excel.
- **RF05:** O sistema deve permitir a **edição e exclusão de lançamentos** cadastrados.
- **RF07:** O sistema deve permitir a **visualização de gráficos interativos** que apresentam a distribuição de **receitas e despesas**, permitindo filtros por **período, contexto e natureza**.
- **RF08:** O sistema deve permitir o **cadastro manual de contextos**, com campos como descrição e observação.
- **RF09:** O sistema deve permitir a **edição e exclusão de contextos** cadastrados.
- **RF10:** O sistema deve permitir o **cadastro manual de naturezas**, com campos como descrição e observação.
- **RF11:** O sistema deve permitir a **edição e exclusão de naturezas** cadastrados.

- **RF12:** O sistema deve **identificar e alertar sobre padrões incomuns** nos lançamentos financeiros (por exemplo, gastos acima da média).
- **RF13:** O sistema deve permitir a **configuração de metas financeiras** mensais ou anuais.
- **RF14:** O sistema deve enviar **notificações e alertas personalizados**, com base nos objetivos definidos pelo usuário.
- **RF15:** O sistema deve permitir a **visualização de um painel com resumo financeiro**, destacando receitas, despesas, saldo e desempenho.
- **RF16:** O sistema deve permitir ao usuário categorizar lançamentos financeiros por natureza, facilitando a organização e análise de seus gastos e receitas.
- **RF17:** O sistema deve exibir relatórios financeiros agrupados por natureza, permitindo ao usuário entender o impacto de cada natureza em seu orçamento.
- **RF18:** O sistema deve permitir ao usuário associar múltiplos lançamentos a uma mesma natureza de forma prática, inclusive por meio de ações em lote.
- **RF19:** O sistema deve permitir ao usuário filtrar os lançamentos financeiros por natureza em todas as visualizações de lista, relatório e gráfico.
- **RF20:** O sistema deve permitir o **cadastro manual de contas**, com campos como descrição e observação.
- **RF21:** O sistema deve permitir a **edição e exclusão de contas** cadastrados.

Requisitos Não Funcionais (RNF):

- **RNF01:** O sistema deve possuir **interface responsiva**, compatível com navegadores modernos e dispositivos móveis.
- **RNF02:** O sistema deve garantir a **segurança dos dados do usuário**, com criptografia de senhas e conexões HTTPS.
- **RNF03:** O sistema deve apresentar **tempo de resposta inferior a 2 segundos** em ações comuns (ex.: carregamento de dados e exibição de gráficos).
- **RNF04:** O sistema deve estar disponível com **uptime de no mínimo 99%** mensalmente.
- **RNF05:** O sistema deve estar preparado para **suportar múltiplos usuários simultâneos** sem degradação perceptível de desempenho.
- **RNF06:** O sistema deve permitir **backup periódico** das informações financeiras do usuário.
- **RNF07:** O sistema deve estar **documentado**, tanto em termos de uso quanto de arquitetura técnica, para facilitar manutenção e evolução.
- **RNF08:** O sistema deve seguir **boas práticas de acessibilidade**, como contraste de cores adequado e suporte a leitores de tela.
- **RNF09:** O sistema deve utilizar **inteligência artificial** para sugerir ajustes financeiros com base no comportamento de gastos.





3.2. Considerações de Design

Durante a fase de concepção da solução, foram consideradas algumas abordagens arquiteturais, como **Arquitetura Monolítica Tradicional**, **Microserviços** e **MVC (Model-View-Controller)**.

Apesar dos benefícios de escalabilidade da arquitetura baseada em microserviços, optou-se pelo padrão **MVC** por ser mais adequado à realidade atual do projeto, dada sua **simplicidade, organização, baixo overhead e fácil manutenção**. Esse padrão favorece uma **separação clara de responsabilidades** entre os componentes da aplicação, facilitando a evolução do sistema e a testabilidade do código.

A escolha do MVC também foi motivada pela compatibilidade com frameworks amplamente utilizados (como Spring Boot no backend e Vue.js no frontend), permitindo agilidade no desenvolvimento e maior produtividade.

Visão Inicial da Arquitetura:

A arquitetura inicial do sistema segue uma abordagem tradicional MVC, com os seguintes componentes principais:

- **Model (Modelo):** Responsável pela representação dos dados, regras de negócio e acesso ao banco de dados. Inclui entidades como Usuario, Lancamento, Natureza, Contexto, Meta e repositórios de dados.
- **View (Visão):** Interface gráfica do usuário, responsável por apresentar os dados e capturar interações. Será implementada como uma SPA (Single Page Application) utilizando Vue.js.
- **Controller (Controlador):** Camada intermediária entre a View e o Model. Gerencia as requisições do usuário, orquestra regras de negócio e retorna as respostas adequadas.

Padrões de Arquitetura Utilizados:

- **MVC (Model-View-Controller):** Principal padrão adotado para organização da aplicação.
- **DTO (Data Transfer Object):** Utilizado para transportar dados entre as camadas, reduzindo acoplamento.
- **Repository Pattern:** Utilizado na camada de persistência para abstrair o acesso aos dados.
- **Service Layer:** Camada de serviços entre os Controllers e os Repositórios, contendo as regras de negócio da aplicação.

Modelo C4

Diagrama de Contexto

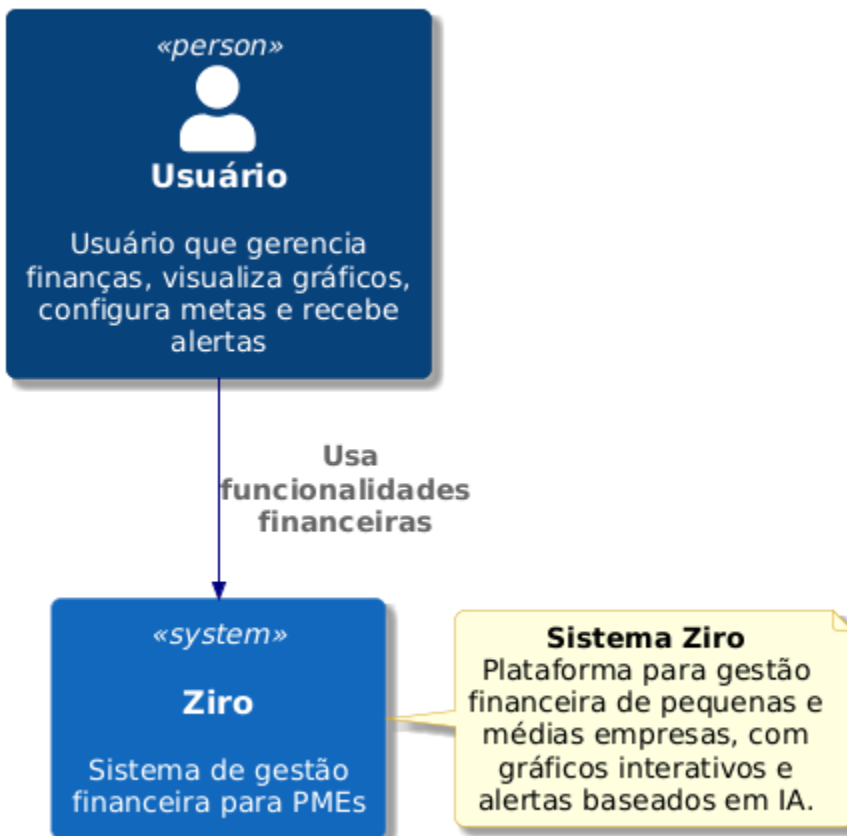


Diagrama de Contexto do Sistema Ziro - Desenvolvido para PMEs

Diagrama de Container

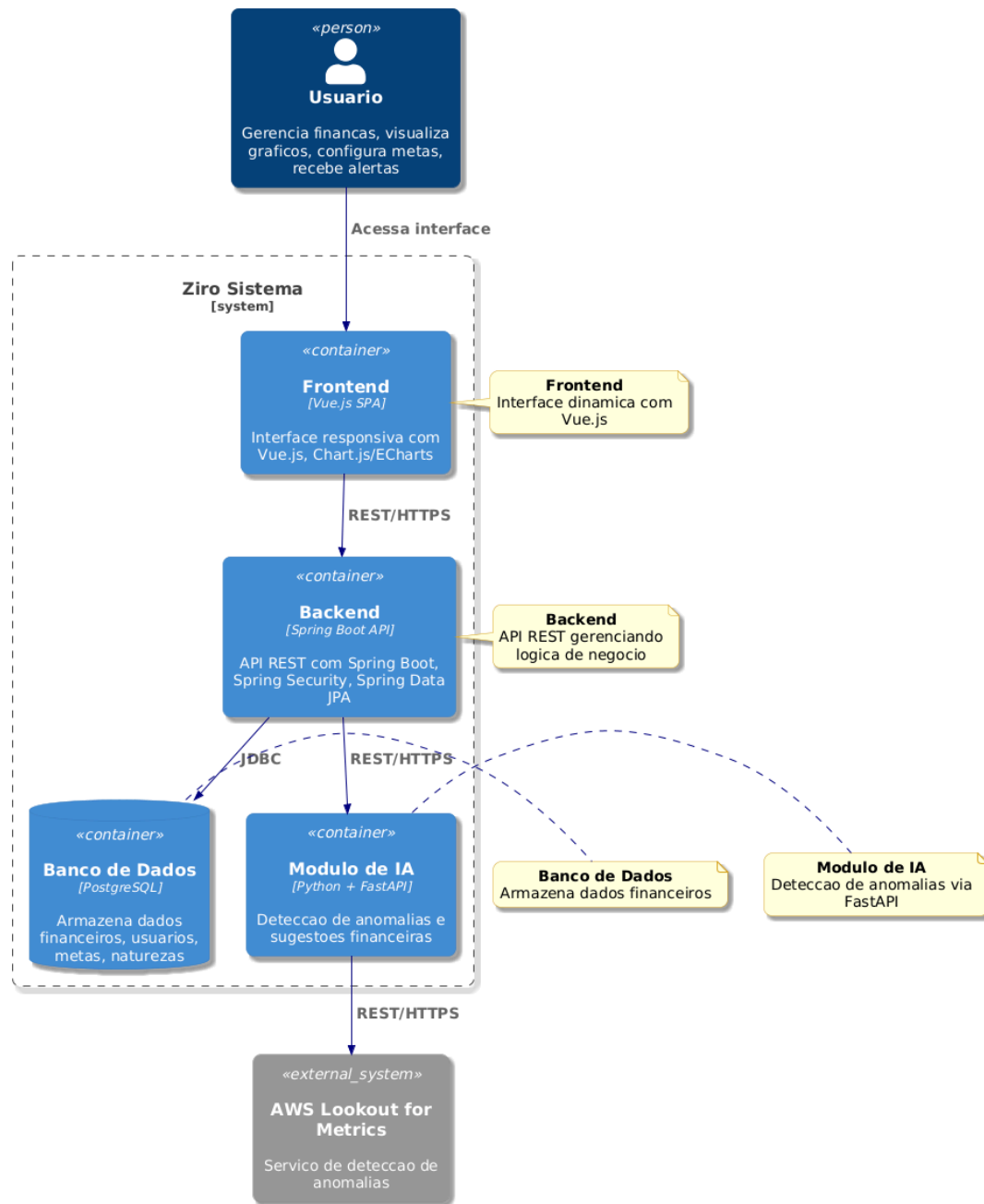


Diagrama de Entidades

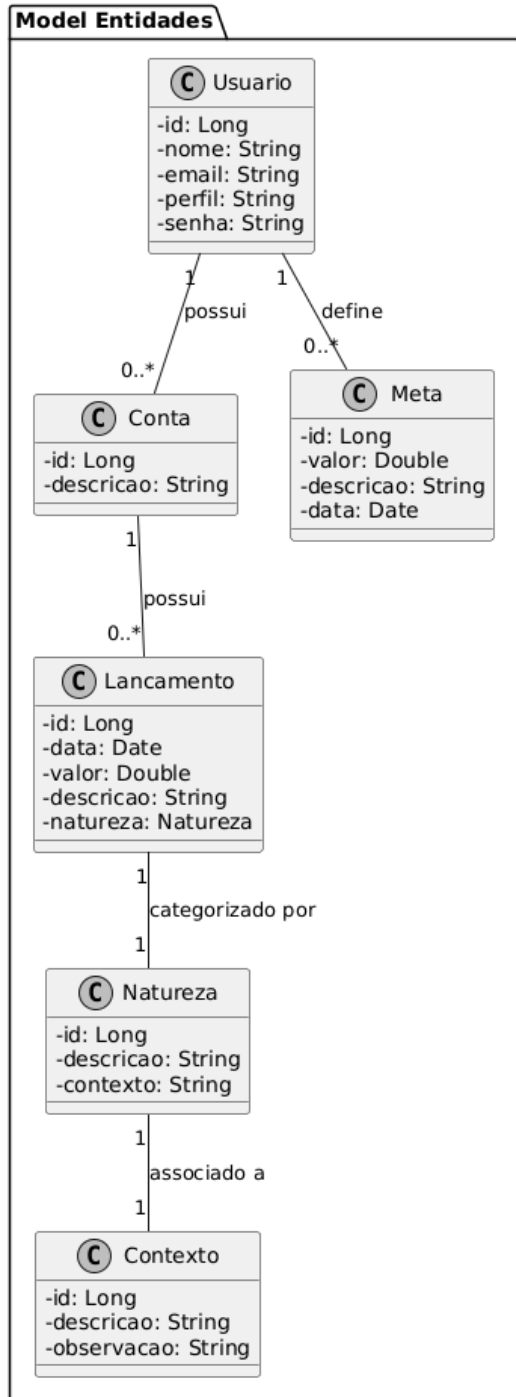
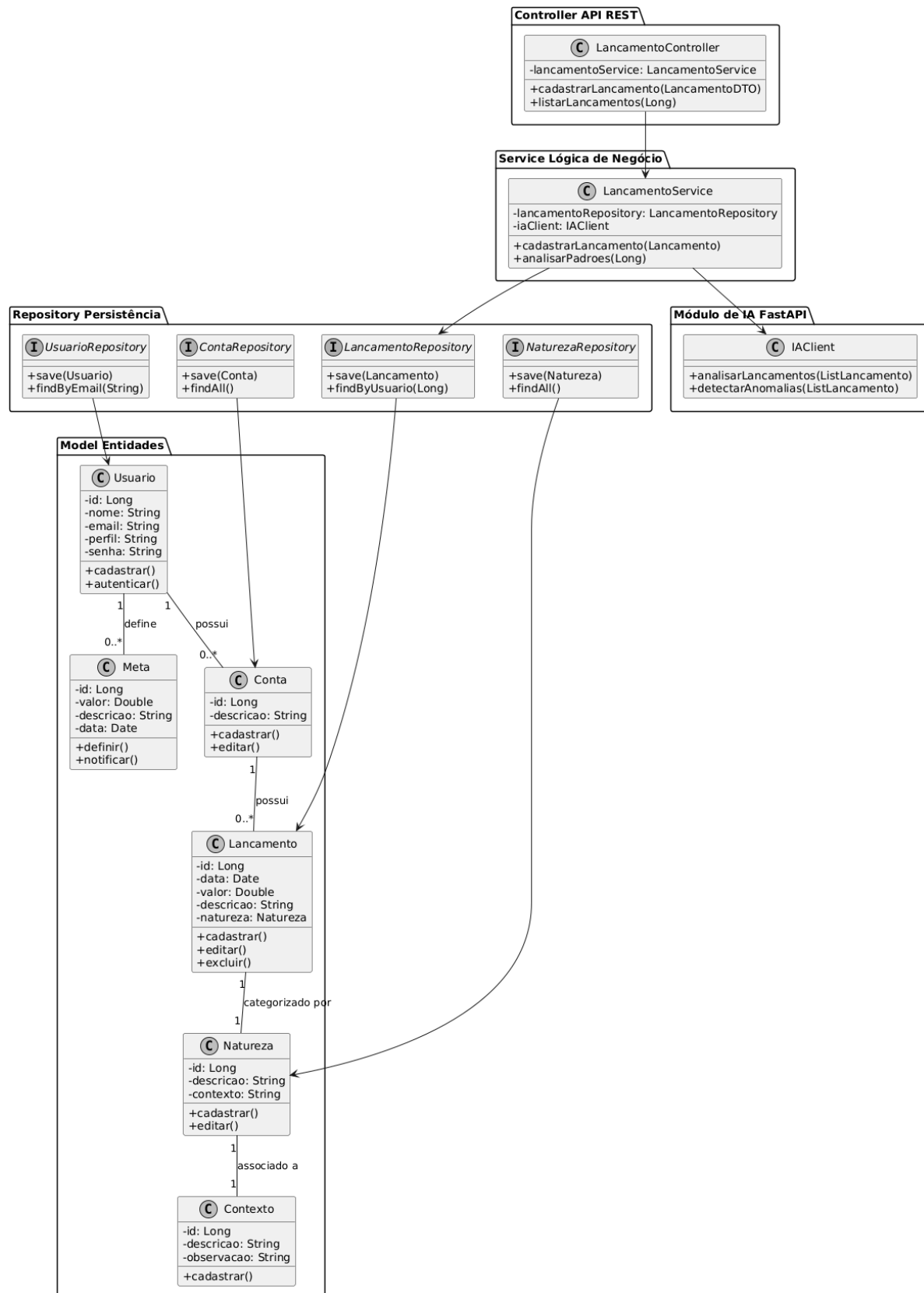


Diagrama de Código



3.3. Stack Tecnológica

A escolha das linguagens de programação levou em consideração fatores como produtividade, robustez, suporte da comunidade e compatibilidade com ferramentas modernas de desenvolvimento.

Linguagens de Programação:

A seleção das linguagens considerou a familiaridade, a robustez das tecnologias e a facilidade de manutenção da aplicação:

- **Java (Backend):**
 - Linguagem consolidada no mercado corporativo, com forte suporte à construção de APIs RESTful robustas utilizando o ecossistema Spring.
- **TypeScript (Frontend):**
 - Utilizado para trazer tipagem estática ao JavaScript, promovendo maior confiabilidade, legibilidade e produtividade no desenvolvimento de interfaces ricas com Vue.js.
- **SQL (PostgreSQL):**
 - Linguagem padrão para interação com bancos de dados relacionais. O PostgreSQL foi escolhido por sua performance, extensibilidade e confiabilidade.
- **Python (Inteligência Artificial):**
 - Linguagem escolhida para o desenvolvimento do microserviço de Inteligência Artificial, devido à sua ampla adoção na área de ciência de dados, vasto ecossistema de bibliotecas para análise de dados e machine learning, e facilidade de integração com outros sistemas via APIs.

Frameworks e Bibliotecas:

Backend:

- **Spring Boot:** Para criação da API REST, com configuração simplificada e integração nativa com diversas bibliotecas.
- **Spring Security:** Para autenticação e autorização.
- **Spring Data JPA / Hibernate:** Para abstração do acesso a dados e persistência via ORM.

Frontend:

- **Vue.js 3:** Framework JavaScript progressivo para construção de interfaces dinâmicas e reativas.
- **Vue Router:** Para controle de rotas e navegação entre páginas.

- **Chart.js ou ECharts:** Para exibição de gráficos financeiros de forma interativa e visual.
- **Tailwind CSS:** Para criação de estilos responsivos com classes utilitárias de forma ágil e padronizada.

Inteligência Artificial:

- **AWS Lookout for Metrics:** Serviço **gerenciado de detecção de anomalias** baseado em machine learning, que automatiza a identificação de padrões anormais em séries temporais financeiras, **sem necessidade de construir modelos manualmente**. Ideal para detectar oscilações incomuns em receitas, despesas e fluxo de caixa.
- **FastAPI:** Para desenvolvimento da API REST que expõe os serviços de IA, permitindo comunicação eficiente entre o microserviço Python e o backend Java.

Outras Bibliotecas:

- **Axios:** Para realizar requisições HTTP entre o frontend e o backend.

Ferramentas de Desenvolvimento e Gestão de Projeto

- **IDE e Editores de Código:**
 - **IntelliJ IDEA (backend)** e **Visual Studio Code (frontend)** para desenvolvimento com suporte a plugins, depuração e produtividade.
- **Controle de Versão:**
 - **Git + GitHub:** Para versionamento, controle de branches e colaboração entre membros da equipe via pull requests.
- **Gerenciamento de Projeto:**
 - **Jira:** Ferramenta adotada para acompanhamento de tarefas, organização de sprints e controle de backlog.
- **Integração Contínua e Entrega Contínua (CI/CD):**
 - **GitHub Actions ou Jenkins:** Para automação de build, testes e deploy em diferentes ambientes.
 - **Docker:** Para empacotamento da aplicação em contêineres, garantindo portabilidade e consistência entre os ambientes.
- **Padronização e Qualidade de Código:**
 - **ESLint e Prettier (frontend)** para padronização de estilo e boas práticas.
 - **Checkstyle (backend)** para análise de padrões no código Java.
 - **SonarQube (opcional):** Para análise contínua de qualidade de código e detecção de vulnerabilidades.
- **Testes:**
 - **JUnit e Mockito (backend)** para testes unitários e de integração.

- **Vitest ou Jest / Cypress (frontend):** Para testes unitários e testes end-to-end, respectivamente.

Outras Considerações

- **Banco de Dados:**
 - **PostgreSQL:** Utilizado como banco de dados relacional principal, suportando transações complexas, relacionamentos, e integridade dos dados.
- **Infraestrutura e Deploy:**
 - Inicialmente, a infraestrutura pode ser provisionada em servidores físicos ou cloud, com potencial migração para serviços como AWS, GCP ou Azure no futuro.
 - Utilização de containers Docker para facilitar deploy, testes e replicação de ambiente.

3.4. Considerações de Segurança

A segurança da aplicação é uma prioridade desde a fase de planejamento. Serão adotadas medidas preventivas e corretivas para proteger os dados dos usuários, garantir a integridade do sistema e mitigar riscos relacionados a ataques cibernéticos.

Autenticação e Autorização

- **Criptografia de Senhas:**

As senhas dos usuários serão armazenadas de forma segura utilizando algoritmos de hash como **BCrypt** com salt, impedindo a recuperação direta da senha original mesmo em caso de vazamento.
- **Autenticação com Tokens (JWT):**

O sistema utilizará **JSON Web Tokens** para autenticação e autorização, protegendo rotas e recursos com base no nível de acesso do usuário.
- **Expiração de Sessão:**

Tokens terão tempo de expiração e poderão ser revogados a qualquer momento para prevenir uso indevido.

Validação e Sanitização de Dados

- **Validação no Backend:**

Todas as entradas do usuário serão validadas no backend para garantir integridade, tipo e tamanho dos dados recebidos.
- **Proteção contra Injeção de SQL:**

A utilização do **Spring Data JPA** e de prepared statements mitiga o risco de injeções SQL.

- **Sanitização de Entrada (XSS):**

Entradas do usuário que serão exibidas no frontend serão devidamente tratadas para evitar **Cross-Site Scripting (XSS)**.

Comunicação Segura

- **HTTPS:**

Toda a comunicação entre cliente e servidor será feita via **HTTPS**, utilizando certificados válidos para criptografar os dados trafegados.

- **CORS (Cross-Origin Resource Sharing):**

A política de CORS será configurada de forma restritiva, permitindo apenas domínios autorizados a se comunicar com a API.

Controle de Acesso e Permissões

- **Perfis e Papéis de Usuário:**

O sistema contará com níveis de permissão para controlar o acesso a funcionalidades sensíveis ou administrativas.

- **Filtro de Recursos Sensíveis:**

Mesmo usuários autenticados terão acesso restrito apenas aos dados que lhes pertencem.

Proteção contra Ataques Comuns

- **CSRF (Cross-Site Request Forgery):**

O uso de tokens CSRF ou a estruturação segura da API com métodos REST mitiga riscos desse tipo de ataque.

- **Brute Force:**

Será implementado um mecanismo de **limitação de tentativas de login**, como bloqueio temporário após múltiplas tentativas falhas.

- **Rate Limiting:**

A API terá controle de taxa de requisições por IP, protegendo contra ataques de negação de serviço (DoS).

Logs e Monitoramento

- **Registro de Atividades Sensíveis:**

Ações como login, alteração de senha, exclusão de dados e importações serão registradas com data, hora e identificação do usuário.

- **Monitoramento de Erros e Acessos Incomuns:**

Erros críticos e acessos fora do padrão serão monitorados com alertas para a equipe técnica.

Conformidade com a LGPD

- **Consentimento e Política de Privacidade:**

O sistema solicitará o aceite da política de uso e privacidade, conforme exigido pela **Lei Geral de Proteção de Dados (LGPD)**.

- **Direito ao Esquecimento:**

O usuário poderá solicitar a remoção de seus dados pessoais conforme previsto na legislação.

4. Próximos Passos

Com a definição da arquitetura, requisitos e tecnologias, o projeto entra agora na fase de implementação. A seguir estão descritos os próximos passos para o início do desenvolvimento tanto do backend quanto do frontend da aplicação.

Início do Desenvolvimento do Backend

- **Estruturação Inicial do Projeto Spring Boot**

- Criar um novo projeto Spring Boot utilizando o Spring Initializr.
- Incluir as dependências básicas: Spring Web, Spring Data JPA, Spring Security, PostgreSQL Driver, Lombok, Validation, JWT.
- Configurar o application.properties (ou application.yml) com dados do banco, porta do servidor e configurações de segurança.

- **Modelagem e Criação das Entidades**

- Criar as entidades: Usuario, Lancamento, Natureza e Contexto.
- Implementar os repositórios (Repository Pattern) com Spring Data JPA.

- **Camada de Serviço e Lógica de Negócio**

- Criar a Service Layer com validações de regras de negócio.
- Implementar lógica para cadastro, edição, exclusão e listagem dos registros.

- **Autenticação e Segurança**

- Configurar autenticação com JWT.
- Implementar endpoints para login e registro de usuário.
- Garantir proteção das rotas com Spring Security e roles de acesso.


- **Testes**

- Criar testes unitários com JUnit e Mockito para validar os serviços e repositórios.

Início do Desenvolvimento do Frontend

- **Estruturação Inicial do Projeto Vue.js**
 - Criar um projeto com Vue CLI ou Vite.
- **Layout Base e Navegação**
 - Criar o layout padrão da aplicação (barra lateral, cabeçalho, áreas principais).
- **Integração Inicial com Backend**
 - Implementar tela de login e integração com endpoint de autenticação.
 - Armazenar o token JWT no localStorage ou sessionStorage.
 - Criar interceptors no Axios para adicionar o token nas
- **Protótipos de Telas**
 - Começar com a tela de Dashboard mostrando um resumo financeiro com dados mockados.
 - Desenvolver as interfaces de cadastro e listagem de lançamentos, contextos e naturezas.
 - Usar Chart.js ou ECharts para protótipos de gráficos.
- **Integração Contínua e Versionamento**
 - Criar repositório no GitHub com branches definidas (main, dev, etc.).
 - Configurar GitHub Actions (ou Jenkins) para execução de testes e build automático.
 - Definir convenções de commit e política de pull request.
- **Planejamento de Sprints**
 - Dividir as entregas por sprints de 1 ou 2 semanas.
 - Priorizar funcionalidades principais:
 - Login e autenticação.
 - Cadastro de lançamentos.
 - Visualização de gráficos.

5. Referências

- **Treasy** | Curso de Modelagem Financeira e Mapas de Indicadores | <https://youtu.be/k61KbjjiLJ8?si=CWziLAbXrDITUuDb>
- **Treasy** | Curso completo de Modelagem Financeira e Indicadores de Desempenho na prática |  Curso completo de Modelagem Financeira e Indicadores de Desempenho ...

