Logotipo

Descrição gerada automaticamente

**CENTRO UNIVERSITÁRIO DE BRASÍLIA**

**Pós Graduação – Desenvolvimento FullStack**

**THAÍS MALDONADO KONISHI**

**VINÍCIUS DE SOUSA SANTANA**

**Sistema de Investimentos ‘Deus Ex Finance’**

BRASÍLIA – DF

2024

Sumário

[1. Introdução 3](#_Toc175400974)

[2. Visão Geral 4](#_Toc175400975)

[3. Requisitos do Produto 5](#_Toc175400976)

[4. Definições, Acrônimos e Abreviações 6](#_Toc175400977)

[5. Configuração do Sistema 6](#_Toc175400978)

[6. Arquitetura do Sistema 7](#_Toc175400979)

[7. Detalhamento dos Componentes 9](#_Toc175400980)

[8. Documentação da API 10](#_Toc175400981)

[9. Conclusão 11](#_Toc175400982)

# 1. Introdução

Este sistema foi desenvolvido com o objetivo de proporcionar uma maneira mais eficiente e acessível para o controle de investimentos, similar a plataformas como Investidor10 e Fundamentei. Com uma interface intuitiva e funcionalidades robustas, o sistema permite que os usuários gerenciem seus ativos de investimento e acompanhem as cotações desses ativos ao longo do tempo. Em decorrência do curto tempo para desenvolvimento do projeto, o escopo inicial para este trabalho foi extremamente reduzido com o objetivo de haver tempo hábil para o desenvolvimento e ainda assim permitir a aplicação dos conceitos estudados durante o curso.

# 2. Visão Geral

Finalidade:

A aplicação ‘Deus Ex Finance’ tem por objetivo permitir uma maior flexibilidade e agilidade na hora de gerir investimentos. Este sistema foi desenvolvido com o objetivo de proporcionar uma maneira mais eficiente e acessível para o controle de investimentos, similar a plataformas como Investidor10 e Fundamentei. Com uma interface intuitiva e funcionalidades robustas, o sistema permite que os usuários gerenciem seus ativos de investimento e acompanhem as cotações desses ativos ao longo do tempo.

Escopo:

ALTERAR ESCOPO

Visão Geral do Produto:

- Frontend: Desenvolvido em Vue.js, utilizando Vuetify para a interface de usuário, `axios` para comunicação com a api backend, e `chart.js` para exibição de gráficos de vendas.

- Backend: Implementado em Java utilizando Spring Boot, com controladores (controllers) para gerenciar operações relacionadas a produtos, vendas e funcionários.

- Banco de Dados: Utiliza MySQL para armazenar dados de produtos, vendas, funcionários, entre outros.

# 3. Requisitos do Produto

Requisitos Funcionais

- Autenticação e Autorização: O sistema deve permitir que usuários e administradores se autentiquem, com acesso a funcionalidades restritas conforme o nível de permissão.

- Gerenciamento de Produtos: Deve ser possível criar, ler, atualizar e excluir produtos.

- Registro e Exibição de Vendas: O sistema deve registrar cada venda realizada e permitir a visualização de um histórico de vendas.

- Visualização de Gráficos de Vendas: Deve apresentar gráficos de vendas para análise de desempenho.

- Controle de Estoque: O sistema deve gerenciar o estoque de produtos, impedindo vendas quando o estoque estiver esgotado.

Requisitos Não Funcionais

- Segurança: Dados dos usuários e informações sensíveis devem ser protegidos contra acessos não autorizados.

- Usabilidade: A interface do usuário deve ser intuitiva e fácil de usar, proporcionando uma boa experiência para todos os tipos de usuários.

# 4. Definições, Acrônimos e Abreviações

- CRUD: Create, Read, Update, Delete - operações básicas de gerenciamento de dados.

- API: Application Programming Interface - conjunto de rotas e métodos para interação entre frontend e backend.

- MySQL: Sistema de gerenciamento de banco de dados relacional.

# 5. Configuração do Sistema

Git: https://github.com/megatroll01/ProjetoFinal.git

Dependências e Instalação

- Frontend: As dependências do frontend estão listadas no arquivo `package.json`. Algumas principais incluem:

- `vue`: Framework JavaScript progressivo.

- `vuetify`: Biblioteca de componentes UI para Vue.js.

- `axios`: Cliente HTTP para requisições ao backend.

- `chart.js`: Biblioteca de gráficos.

- Backend: As dependências do backend são gerenciadas pelo `pom.xml`. Principais dependências incluem:

- `spring-boot-starter-web`: Framework para desenvolvimento de aplicações web em Java.

- `spring-boot-starter-data-jpa`: Integração de JPA para persistência de dados.

- `mysql-connector-java`: Driver JDBC para conexão com MySQL.

Configuração do Banco de Dados

- Banco de Dados MySQL: O banco de dados MySQL armazena todas as informações essenciais, como produtos, vendas e funcionários. As configurações de conexão estão definidas no arquivo `application.properties` do Spring Boot, incluindo a URL de conexão, usuário, senha e outras propriedades específicas.

# 

# 6. Arquitetura do Sistema

Frontend (Vue.js)

- Componentes Modulares: O frontend é dividido em componentes Vue, permitindo uma organização modular do código.

- Vuetify: Utilizado para o design responsivo e consistente da interface de usuário.

- Integração com API: O frontend se comunica com o backend através de `axios`, enviando e recebendo dados em formato JSON.

Backend (Spring Boot)

- Arquitetura em Camadas: O backend segue uma arquitetura em camadas, com Controllers para a interface de aplicação, Services para a lógica de negócios e Repositories para a interação com o banco de dados.

- Segurança: Implementação de autenticação e autorização utilizando Spring Security.

- Persistência de Dados: Utilização do Spring Data JPA para mapeamento objeto-relacional (ORM).

Banco de Dados (MySQL)

- Estrutura de Tabelas: O banco de dados é composto por tabelas que armazenam informações sobre produtos, vendas, funcionários, entre outros.

- Relacionamentos: Relacionamentos entre as tabelas são mapeados utilizando JPA, como `OneToMany` para relacionar vendas a produtos.

Interface gráfica do usuário

Descrição gerada automaticamente

# 7. Detalhamento dos Componentes

Controladores (Controllers)

- ProdutoController: Responsável por gerenciar as operações CRUD para produtos.

- VendaController: Gerencia as operações relacionadas a vendas e exibição de gráficos de desempenho.

- FuncionarioController: Cuida do gerenciamento dos funcionários, incluindo registro, edição e remoção.

- GraficoController: Responsável por consultar dados para os graficos no front.

- AuthController: Responsável pela autenticação de usuários

Modelos (Models)

- ProdutoObj: Modelo que representa um produto no sistema, mapeado para a tabela `products` no MySQL.

- VendaObj: Modelo que representa uma venda, incluindo detalhes dos produtos vendidos e a data da transação.

- FuncionarioObj: Modelo que representa um funcionário, armazenando informações como nome, cargo e credenciais de acesso.

Serviços (Services)

- ProdutoService: Contém a lógica de negócios para operações com produtos.

- VendaService: Encapsula a lógica de registro de vendas e geração de relatórios.

- FuncionarioService: Gerencia a lógica associada ao cadastro, edição e autenticação de funcionários.

# 8. Documentação da API

Endpoints Principais

Autenticação

* POST /api/auth/usuario: Autentica um usuário e retorna um objeto FuncionarioObj com os detalhes do usuário autenticado.

Funcionários (/api/usuarios)

* GET /api/usuarios: Retorna a lista de todos os funcionários.
* GET /api/usuarios/{id}: Retorna os detalhes de um funcionário específico pelo ID.
* POST /api/usuarios: Cria um novo funcionário.
* PUT /api/usuarios/{id}: Atualiza as informações de um funcionário existente pelo ID.
* DELETE /api/usuarios/{id}: Remove um funcionário pelo ID.

Gráficos de Vendas (/api/grafico)

* GET /api/grafico/funcionario/{login}: Retorna os dados de vendas do funcionário logado e a venda geral, agrupados por mês.

Produtos (/api/produtos)

* GET /api/produtos: Retorna a lista de todos os produtos.
* GET /api/produtos/{id}: Retorna os detalhes de um produto específico pelo ID.
* POST /api/produtos: Cria um novo produto.
* PUT /api/produtos/{id}: Atualiza as informações de um produto existente pelo ID.
* DELETE /api/produtos/{id}: Remove um produto pelo ID.

Vendas (/api/vendas)

* GET /api/vendas: Retorna o histórico de todas as vendas.
* GET /api/vendas/{id}: Retorna os detalhes de uma venda específica pelo ID.
* POST /api/vendas: Registra uma nova venda e atualiza o estoque dos produtos.
* PUT /api/vendas/{id}: Atualiza uma venda existente pelo ID.
* DELETE /api/vendas/{id}: Remove uma venda pelo ID.
* GET /api/vendas/relatorio: Gera um relatório de vendas dentro de um intervalo de datas especificado (startDate e endDate).

# 9. Conclusão

Este documento fornece uma visão abrangente do sistema de e-commerce 'Magnetic', desde sua configuração e arquitetura até os componentes detalhados e a documentação da API. Ele serve como uma referência central para desenvolvedores, administradores e qualquer outra pessoa envolvida na manutenção ou evolução do sistema.