Gestão de Recursos Humanos

Disciplina Técnicas de Programação 1

Grupo: 6

Eduardo Rocha Biagini – 242032273 – dudurbiagini@gmail.com
Lucas Centurion Netto – 242003781 – lcenturionnetto@gmail.com
Filipe Araújo Lopes Grillo – 202023541 – lipe.grillo007@gmail.com
João Vitor Lopes Rocha – 242014041 – jvitorlocha55@gmail.com

Setembro 2025

Resumo

Este relatório apresenta o desenvolvimento de um software voltado para a gestão de recursos humanos, cuja necessidade surge da demanda das empresas em organizar cadastros de colaboradores, otimizar processos internos de RH e melhorar o atendimento das demandas administrativas. A aplicação busca oferecer maior eficiência na administração de informações de funcionários, reduzindo falhas manuais e agilizando o acesso a dados relevantes. O objetivo principal do trabalho é criar uma solução capaz de centralizar e automatizar as rotinas de gestão de pessoas, como cadastro de colaboradores, controle de cargos e salários, acompanhamento de férias e geração de relatórios, atendendo às necessidades identificadas e proporcionando uma ferramenta prática de apoio ao setor de RH.

1 Introdução

Com o avanço da tecnologia e a crescente complexidade dos processos organizacionais, torna-se cada vez mais necessário o uso de ferramentas digitais que auxiliem na gestão eficiente de recursos humanos. A administração de informações relacionadas aos colaboradores, como dados cadastrais, cargos, salários e benefícios, exige precisão, agilidade e organização. Nesse contexto, o desenvolvimento de sistemas específicos para o setor de RH se apresenta como uma solução estratégica para atender às demandas operacionais e administrativas das empresas. Este relatório aborda a criação de um software voltado para a gestão de recursos humanos, destacando sua importância, funcionalidades e os benefícios esperados com sua implementação.

1.1 Contexto do Projeto

A gestão de RH é uma área essencial dentro das organizações, responsável por administrar informações e processos relacionados aos colaboradores. Com o crescimento das empresas e o aumento das demandas administrativas, torna-se cada vez mais complicado manter a organização, a precisão e a agilidade no controle de informações quando se utiliza apenas processos manuais ou ferramentas genéricas. Neste caso, surge a necessidade de desenvolver uma solução tecnológica que centralize e automatize as rotinas do setor de RH, permitindo maior controle, segurança e eficiência nos processos internos. O projeto visa atender a essa demanda, oferecendo uma aplicação prática e funcional para facilitar o gerenciamento de pessoas dentro das organizações.

1.2 Objetivos de Desenvolvimento

Objetivo Geral: Desenvolver um software de gestão de recursos humanos que permita um controle de dados dos colaboradores e que otimize processos administrativos relacionados a área de RH.
Objetivos Específicos:

- Criar uma interface intuitiva para cadastro e consulta dos colaboradores.
- Implementar funcionalidades para controle.

- Automatizar a geração de relatórios administrativos e gerenciais.
- Reduzir erros manuais de dados.
- Melhorar o atendimento a demanda.

1.3 Requisitos Funcionais

Administração/Gestão:

- Cadastrar, editar, excluir e listar contas de usuário do sistema (administradores, gestores, recrutadores, funcionários).
- Restringir funcionalidades de acordo com o perfil de acesso.
- Permitir pesquisa para todos os usuários por múltiplos critérios combinados.
- Validar dados de usuários (CPF, email).
- Gerar relatórios consolidados de gestão.

Candidatura:

- Cadastrar, editar, excluir e listar candidatos.
- Validar CPF no cadastro de candidatos.
- Associar candidatos a vagas (candidatura).
- Registrar candidaturas com status (pendente, em análise, aprovado, reprovado).
- Consultar candidatos por múltiplos critérios.

Recrutamento:

- Cadastrar, editar, excluir e listar vagas.
- Permitir filtrar vagas por múltiplos critérios.
- Agendar entrevistas vinculadas a uma candidatura.
- Permitir a solicitação do recrutador para contratação de candidatos aprovados.
- Permitir a autorização do gestor para a contratação.
- Cadastrar funcionários aprovados no recrutamento.

Financeiro:

- Configurar regras salariais adicionais.
- Calcular automaticamente o salário do funcionário com base no regime.
- Gerar a folha de pagamento mensal com todos os funcionários ativos.
- Exportar relatórios da folha em formato visual (tela) e arquivo (PDF/CSV).
- Filtrar funcionários por múltiplos critérios.

1.4 Requisitos não Funcionais

O sistema deve conter os seguintes requisitos não funcionais:

- Ter interface amigável para interação (JavaFX).
- Oferecer usabilidade adequada (botões e menus claros, feedback ao usuário).
- Exibir mensagens claras para erros de regra de negócio, usando exceções personalizadas.
- Ser implementado de forma orientada a objetos, garantindo modularidade.
- A modelagem deve priorizar herança e composição para reaproveitamento de código.
- Respeitar o princípio de encapsulamento.
- Adotar boas práticas de POO, como responsabilidade única e baixo acoplamento.
- Empregar pelo menos dois padrões de projeto.
- Ser implementado em camadas bem definidas.
- Armazenar dados em arquivos.
- Ser executável em qualquer sistema operacional com Java instalado.
- Garantir autenticação obrigatória via login e senha.

1.5 Regras de negócio

Administração/Gestão:

- Perfis de acesso são hierárquicos: administrador (acesso total), gestor, recrutador e funcionário.
- Um usuário pode ter um ou mais perfis.
- Apenas administradores podem gerenciar contas de outros usuários.
- Senhas devem respeitar políticas mínimas de segurança.

Candidatura:

- Candidatos podem se candidatar a várias vagas, mas não à mesma vaga mais de uma vez.
- Toda candidatura deve estar vinculada a uma vaga existente e possuir um status.
- O status da candidatura é atualizado pelo Recrutador.

Recrutamento:

- Somente Gestores podem criar vagas e atribuir recrutadores.
- O recrutador gerencia apenas as vagas sob sua responsabilidade.
- Nenhum candidato pode ser contratado sem ao menos uma entrevista registrada.
- Apenas candidatos aprovados podem ser contratados, mediante autorização de um Gestor.

Financeiro:

- A folha de pagamento deve incluir apenas funcionários ativos.
- O salário deve ser calculado conforme o regime de contratação.
- Apenas Administradores podem alterar regras salariais.

1.6 Estrutura do Relatório

As próximas seções deste relatório detalham a análise e o desenvolvimento do software de gestão de recursos humanos implementado.

- Na **Seção 2**, é apresentada a solução proposta, abordando a modelagem e a arquitetura do sistema. Serão detalhados o design, a descrição das classes que estruturam o sistema, a implementação da lógica de negócio e os mecanismos de persistência de dados.
- Na **Seção 3**, são discutidos os resultados e a análise do software finalizado. Esta seção demonstra as funcionalidades implementadas através de telas do sistema, descreve as tecnologias utilizadas e detalha como técnicas de programação orientada a objetos foram aplicadas para garantir a qualidade da solução.
- Por fim, a **Seção 4** apresenta a conclusão, sintetizando a relevância do sistema de RH desenvolvido e seu impacto no contexto proposto, além de apontar possíveis melhorias e evoluções futuras.

2 Solução Proposta

2.1 Estrutura Inicial e Divisão de Tarefas

A estrutura do projeto foi organizada para separar o código-fonte da lógica de negócio dos arquivos de interface do usuário. As classes Java foram alocadas no diretório `src/main/java/grupo/trabalho`, enquanto os arquivos FXML foram mantidos em `src/main/java/resources/grupo/trabalho`.

Para organizar o desenvolvimento do sistema, a equipe dividiu as tarefas com base em módulos funcionais, garantindo que cada integrante tivesse responsabilidade clara sobre um conjunto de funcionalidades. A distribuição foi definida da seguinte forma:

- **Eduardo Rocha Biagini - Administração e Gestão:** Responsável pela gestão de contas de usuário, supervisão de recrutadores e contratações, e criação de relatórios de gestão.
  - *Classes de Domínio:* Usuario, Administrador, Gestor.
  - *Telas:* Login, Administrar Usuários, Relatórios Gestão.
- **Filipe Araújo Lopes Grillo - Candidatura:** Responsável pela gestão de candidaturas.
  - *Classes de Domínio:* Pessoa, Candidato, Candidatura.
  - *Telas:* Cadastro de Candidato, Candidatura à Vaga, Status da Candidatura.
- **Lucas Centurion Netto - Recrutamento:** Responsável pela gestão de vagas e contratações.
  - *Classes de Domínio:* Recrutador, Vaga, Entrevista, Contratação.
  - *Telas:* Menu do Recrutamento, Cadastro de Candidatos, Realizar Candidaturas, Marcar Entrevista, Solicitar Contratações, Consultar Contratações.
- **João Vitor Lopes Rocha - Financeiro:** Responsável pelo controle de regras salariais, folha de pagamento e relatórios financeiros.
  - *Classes de Domínio:* Funcionario, RegraSalario, FolhaPagamento.
  - *Telas:* Menu do Financeiro, Cadastro de Funcionário, Configurar Regras Salariais, Gerar Folha de Pagamento, Relatório Financeiro, Contracheques.

2.2 Design e Modelagem

O design do sistema foi orientado a objetos, buscando representar as entidades do domínio de RH de forma clara. Para a segunda fase de modelagem, cada membro da equipe ficou responsável por detalhar o diagrama de classes do módulo sob sua responsabilidade. Os diagramas foram programados utilizando PlantUML e exportados como imagens gráficas, conforme apresentado a seguir.

2.2.1 Diagramas de Classes

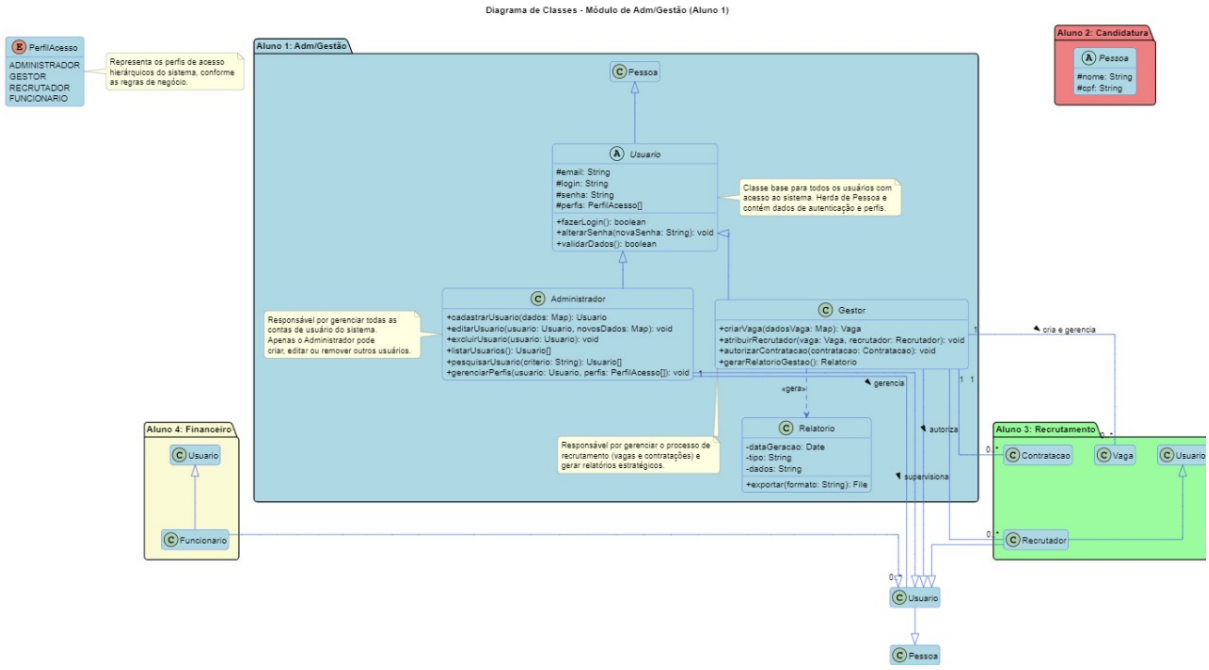

Figura 1: Diagrama de Classes do Módulo de Administração/Gestão.

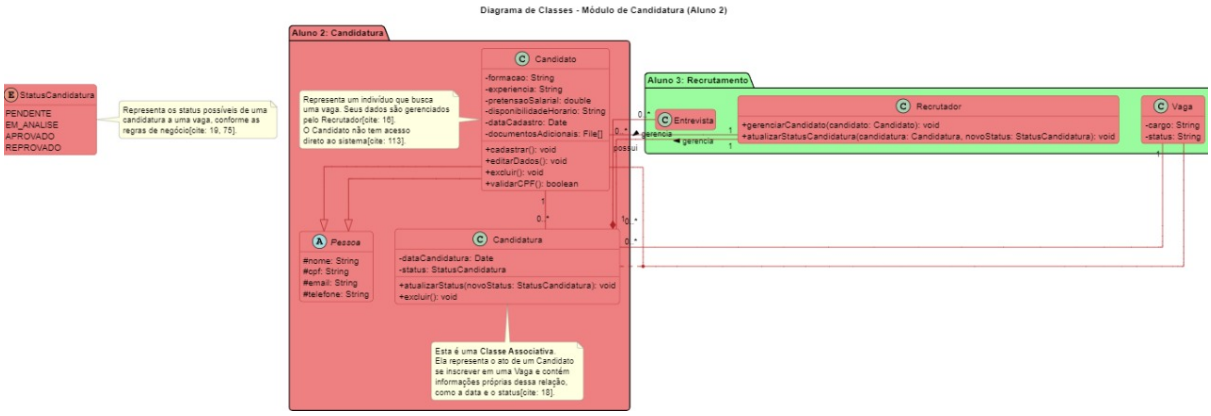

Figura 2: Diagrama de Classes do Módulo de Candidatura.

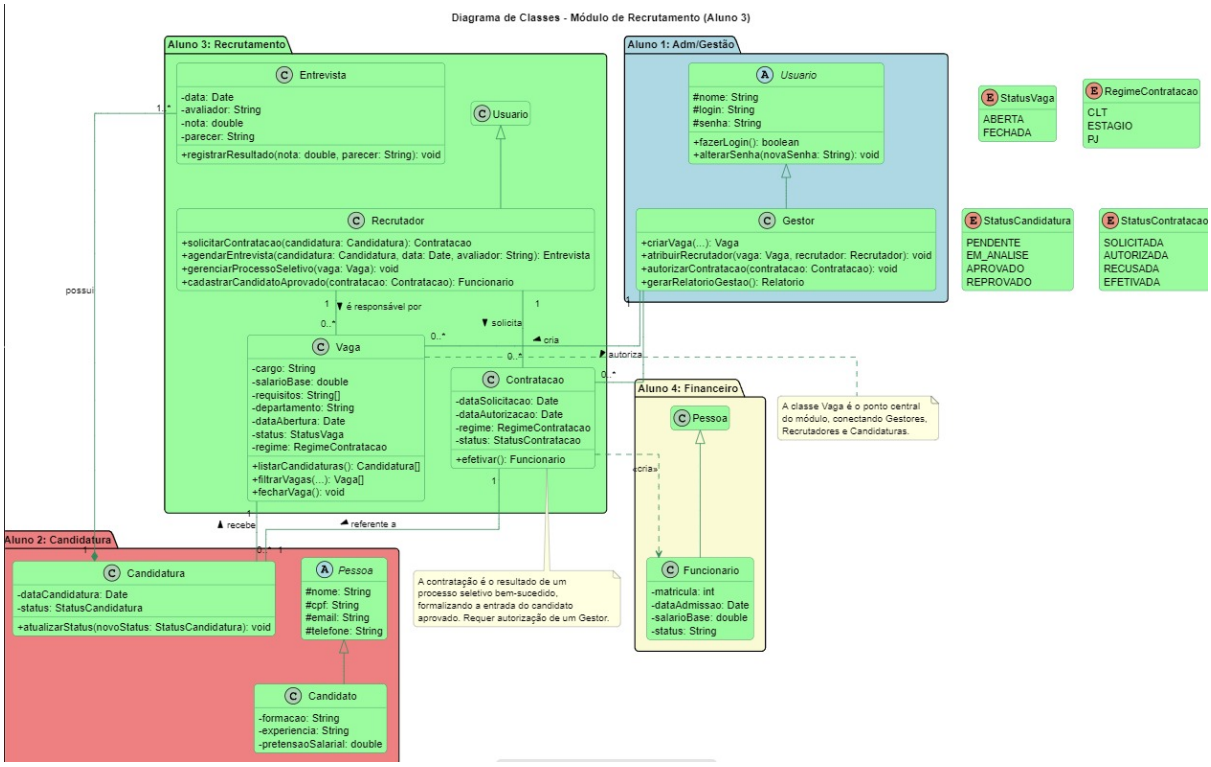

Figura 3: Diagrama de Classes do Módulo de Recrutamento.

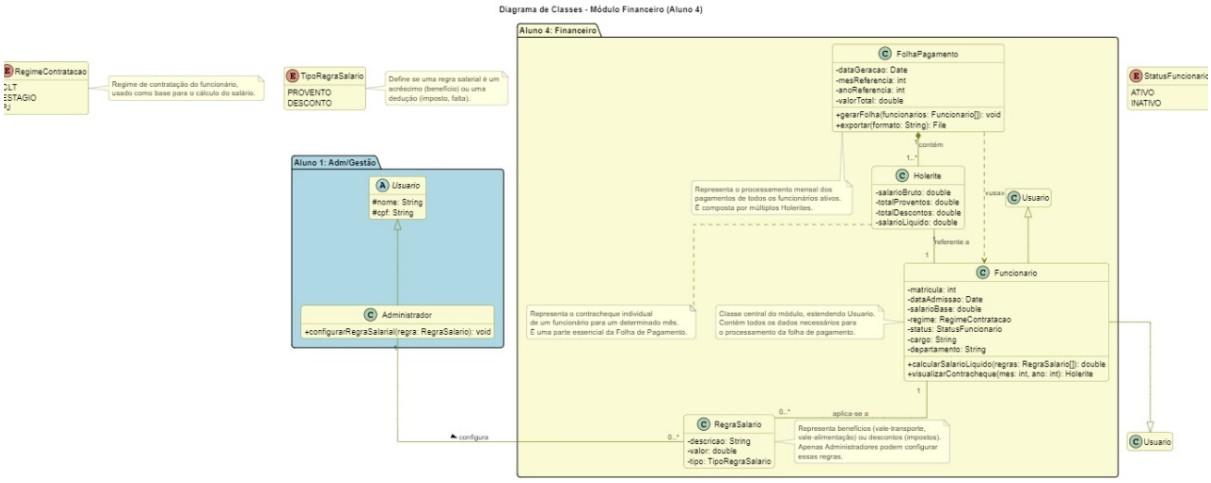

Figura 4: Diagrama de Classes do Módulo Financeiro.

2.2.2 Descrição das Classes

O sistema foi modularizado em diversas classes com responsabilidades bem definidas, seguindo o padrão Model-View-Controller (MVC) para a aplicação e representando as principais entidades do domínio.

Classes da Aplicação:

- **HelloApplication:** Classe principal que inicia a aplicação JavaFX.
- **HelloController:** Controla a tela de login.
- **MainController:** Gerencia o menu principal e a navegação entre módulos.
- **AdmController, FinanceiroController, CandidaturaController e RecrutamentoController:** Controladores das principais subtelas do projeto.

Principais Classes de Domínio:

- **Pessoa:** Superclasse que armazena dados comuns como nome e CPF, utilizada como base para outras classes.
- **Usuario:** Representa todos os usuários com acesso ao sistema, gerenciando autenticação e perfis. Herda de Pessoa.
- **Candidato:** Representa um indivíduo que busca uma vaga, armazenando seu currículo e informações pertinentes. Herda de Pessoa.
- **Candidatura:** Classe associativa que representa o ato de um candidato se inscrever em uma vaga, contendo data e status.
- **Vaga:** Descreve uma oportunidade de emprego, com cargo, requisitos e status (aberta/fechada).
- **Funcionario:** Classe central do módulo financeiro, contendo todos os dados de um colaborador contratado. Herda de Pessoa.
- **FolhaPagamento:** Agrega os dados de pagamento de todos os funcionários ativos em um determinado período.

2.2.3 Diagramas de Caso de Uso

A seguir, são apresentados os diagramas de caso de uso, que modelam as interações dos principais atores do sistema com suas respectivas funcionalidades, também projetados em UML.

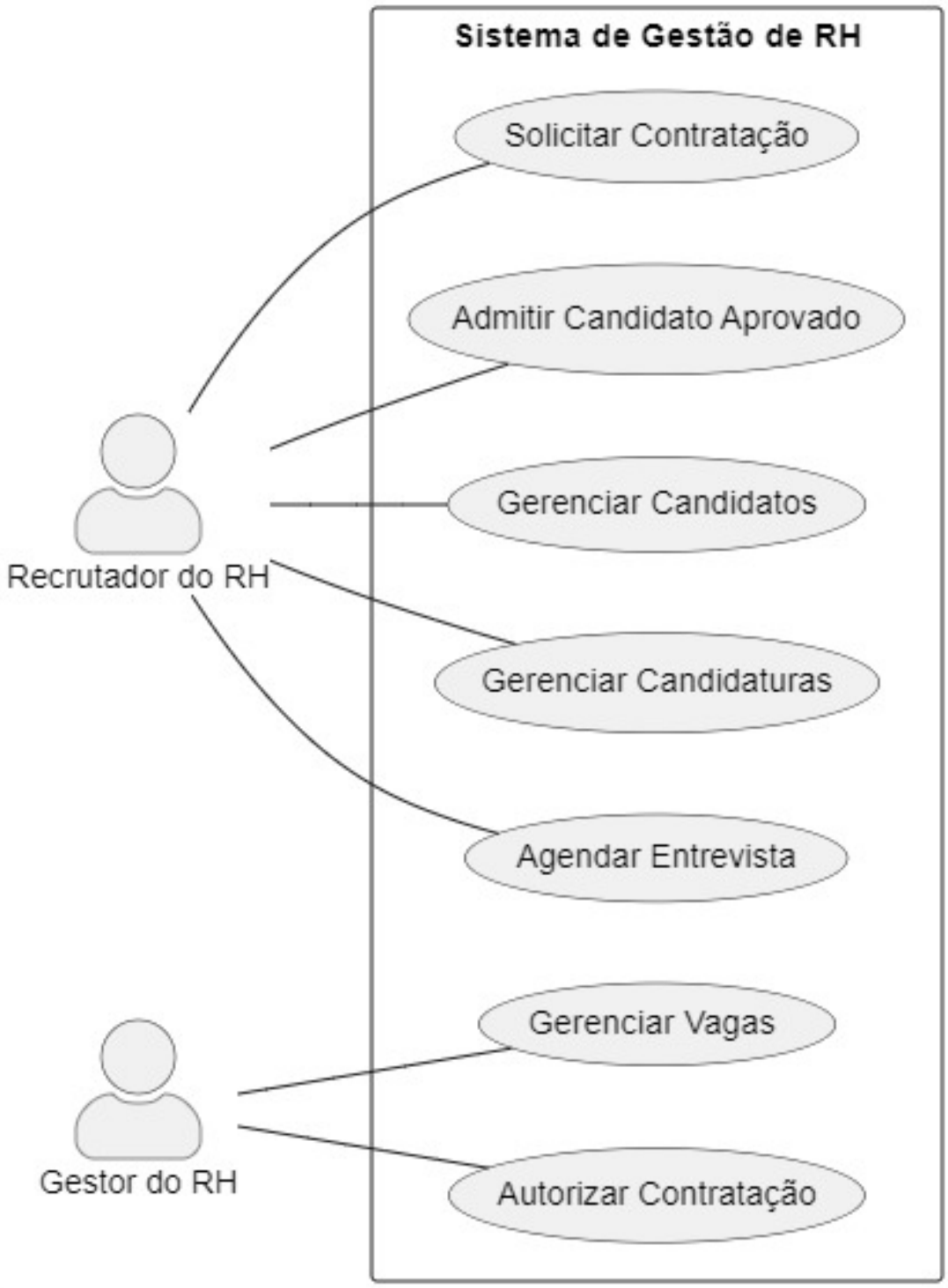

Figura 5: Diagrama de Caso de Uso dos atores Administrador e Gestor do RH.

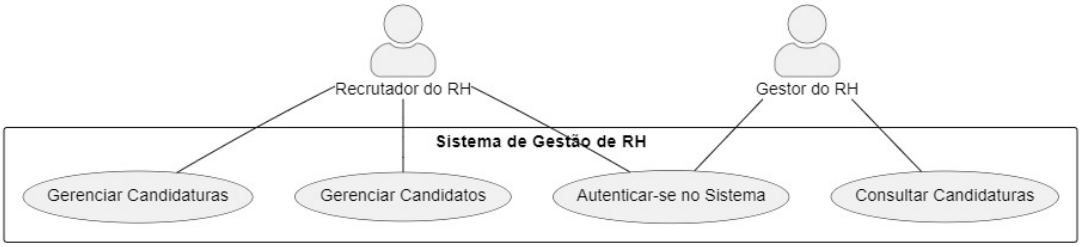

Figura 6: Diagrama de Caso de Uso dos atores Recrutador e Gestor, referente ao módulo de Candidatura.

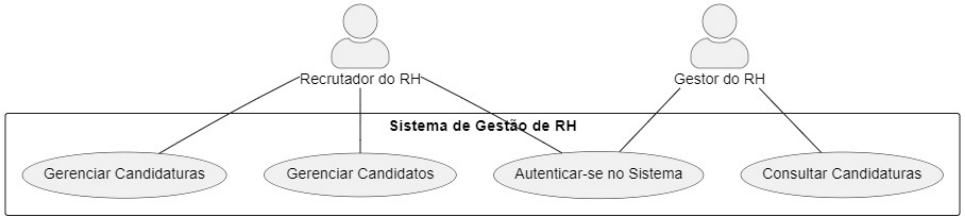

Figura 7: Diagrama de Caso de Uso dos atores Recrutador e Gestor, referente ao módulo de Recrutamento.

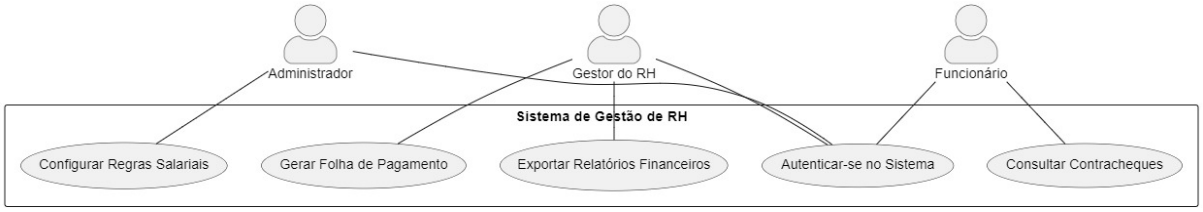

Figura 8: Diagrama de Caso de Uso dos atores Administrador, Gestor e Funcionário, referente ao módulo Financeiro.

2.3 Implementação da Lógica do Software

A lógica de negócio, detalhada na Seção 1.5, foi implementada através de métodos de validação e processamento dentro das classes controladoras. Regras como "Não é permitido cadastrar dois colaboradores com o mesmo CPF"ou "Apenas candidatos aprovados podem ser contratados"são traduzidas para estruturas condicionais (‘if-else‘) e consultas às coleções de dados antes de permitir uma operação. A lógica de perfis de acesso (Administrador, Gestor, etc.) é implementada nos controladores, habilitando ou desabilitando componentes da interface com base no perfil do usuário logado.

No estado atual do protótipo, os mecanismos de persistência são simulados em tempo de execução. Em vez de um banco de dados ou arquivos físicos, o sistema utiliza coleções da API do Java (como ‘ArrayList‘ e ‘HashMap‘) para armazenar os objetos de domínio (ex: ‘Funcionario‘, ‘Candidato‘, ‘Usuario‘) em memória. Isso permite que a aplicação realize operações de CRUD (Criar, Ler, Editar, Excluir) durante uma sessão de uso, facilitando o desenvolvimento e teste ágil da lógica de negócio, desacoplando-a da implementação da persistência definitiva, que será feita em arquivos, conforme o requisito não funcional.

O tratamento de exceções é um pilar para a robustez do sistema. Ele é implementado de duas formas principais: primeiro, no tratamento de exceções nativas do Java, como ‘IOException‘ (vista na Seção 3.3) ao carregar arquivos FXML, garantindo que falhas de leitura de interface sejam tratadas. Segundo, através da validação de entradas do usuário, onde blocos ‘try-catch‘ são usados para prevenir erros de conversão (ex: ‘NumberFormatException‘ ao tentar converter um texto de salário para ‘double‘). Erros de regras de negócio (ex: CPF inválido) são tratados com alertas (‘Alert‘) do JavaFX, fornecendo feedback claro ao usuário e garantindo a confiabilidade das operações.

2.4 Integração e navegabilidade

A integração entre a interface gráfica (View) e a lógica de negócio (Controller) é o pilar da arquitetura JavaFX adotada. A conexão é estabelecida de duas formas principais: o 'FXMLLoader' é responsável por "inflar"o arquivo FXML (a View) e, ao mesmo tempo, instanciar sua classe controladora associada. Em seguida, a anotação '@FXML' realiza a injeção de dependência, conectando os componentes visuais definidos no FXML (como 'Button' e 'TextField') a atributos Java dentro da classe controladora. Isso permite que a lógica de negócio, encapsulada em métodos, seja acionada diretamente por eventos da interface (ex: 'onAction' de um botão).

A navegabilidade entre os diferentes módulos é gerenciada pelos controladores. O 'MainController' (Tela de Menu) atua como um hub central. Cada botão no menu (ex: 'Financeiro', 'Candidatura') aciona um método de navegação dedicado, como o 'goToCandidatura()' (exemplificado na Seção 3.3). Este método encapsula a lógica de fechar a janela atual ('prevStage.close()') e carregar o FXML e o controlador do novo módulo, criando uma nova janela ('Stage'). Essa abordagem modular garante que cada funcionalidade permaneça desacoplada, melhorando a manutenibilidade, e assegura uma transição de telas fluida, contribuindo para a usabilidade do sistema.

3 Resultados e Discussão

Nesta seção, é apresentada uma análise detalhada do software desenvolvido até o momento. O foco é discutir os resultados alcançados em termos de implementação e design, em contraponto aos objetivos propostos na Seção 1. As subseções a seguir detalharão as tecnologias que viabilizaram o projeto (3.1), uma demonstração visual das funcionalidades implementadas (3.2), uma discussão aprofundada sobre as técnicas de programação orientada a objetos aplicadas (3.3), os desafios superados (3.4) e uma análise crítica da qualidade da solução atual (3.5).

3.1 Tecnologia Utilizadas

- **Framework:** JavaFX para a construção da interface gráfica.
- **IDE:** IntelliJ IDEA.
- **Ferramenta de UI:** Scene Builder para projetar os arquivos FXML.
- **Controle de Versão:** GitHub.

3.2 Demonstração da Solução

A seguir, são apresentadas as principais telas do protótipo funcional, demonstrando o fluxo de navegação do sistema de RH.

Figura 9: Tela inicial de Login do sistema.

Figura 10: Menu Principal, ponto de acesso para os módulos do sistema.

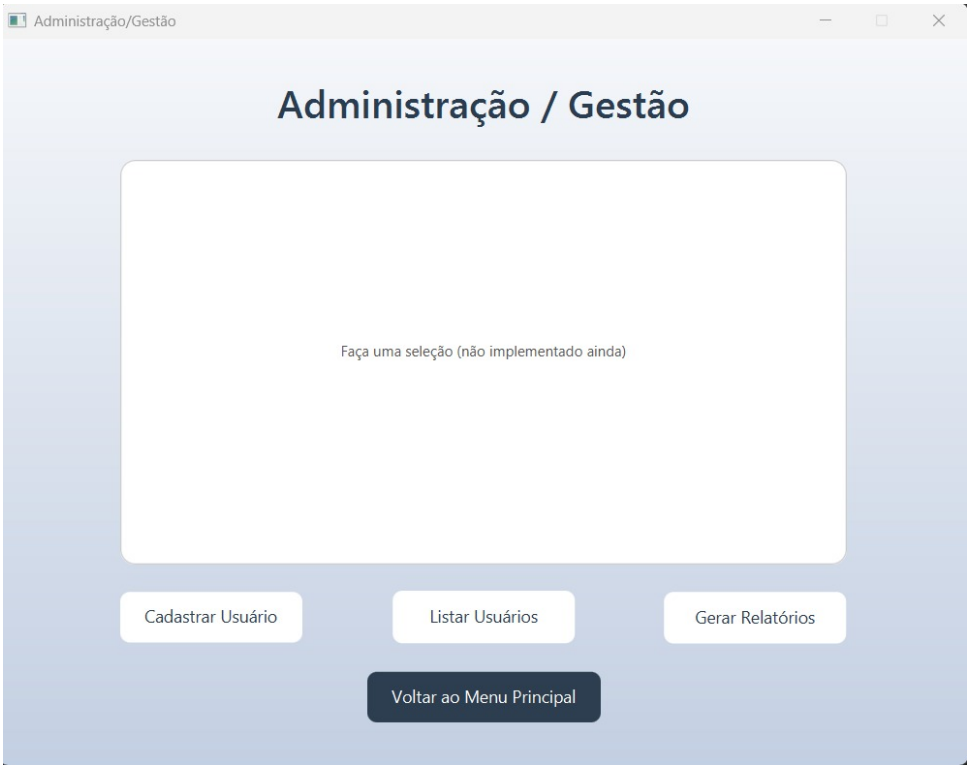

Figura 11: Módulo de Administração / Gestão.

Figura 12: Módulo de Candidatura.

Figura 13: Módulo de Recrutamento.

Figura 14: Módulo Financeiro.

3.3 Técnicas de programação Aplicadas

A arquitetura do sistema foi fundamentada em boas práticas de design e nos pilares da Programação Orientada a Objetos (POO). A **Modularização** é a principal estratégia de organização do projeto, evidente na divisão de responsabilidades (Seção 2.1), onde cada módulo (Administração, Financeiro, etc.) possui seus próprios controladores e classes de domínio. Isso promove baixo acoplamento e alta coesão. A adoção do padrão Model-View-Controller (MVC) com JavaFX reforça essa modularidade, separando a interface (View, nos arquivos FXML) da lógica de controle (Controller).

O **Encapsulamento** foi rigorosamente aplicado em todas as classes de domínio, como 'Funcionario' e 'Vaga', onde os atributos são privados e acessados apenas por métodos públicos (getters e setters), protegendo a integridade dos dados. A **Herança** foi uma técnica central para a **Abstração** do domínio e o reaproveitamento de código. Como visto nos diagramas de classes (Seção 2.2.1), a superclasse

‘Pessoa‘ define atributos comuns (nome, cpf), que são herdados pelas subclasses ‘Usuario‘, ‘Candidato‘ e ‘Funcionario‘. Isso não apenas evita duplicação, mas também permite o uso de **Polimorfismo**, onde uma coleção do tipo ‘List¡Pessoa¿‘ poderia armazenar objetos de qualquer uma de suas subclasses.

```java
package grupo.trabalho;

import java.time.LocalDate;
import java.util.List;

public class Funcionario extends Usuario {  GG
    private int matricula;  2 usages
    private LocalDate dataAdmissao;  2 usages
    private double salarioBase;  3 usages
    private RegimeContratacao regime;  2 usages
    private StatusFuncionario status;  3 usages
    private String cargo;  2 usages
    private String departamento;  2 usages

    public Funcionario(String nome, String cpf, int matricula, LocalDate dataAdmissao,  1 usage  GG
                       double salarioBase, RegimeContratacao regime,
                       StatusFuncionario status, String cargo, String departamento) {
        super(nome, cpf);
        this.matricula = matricula;
        this.dataAdmissao = dataAdmissao;
        this.salarioBase = salarioBase;
        this.regime = regime;
        this.status = status;
        this.cargo = cargo;
```

Figura 15: Exemplo da relação de herança no módulo Financeiro, onde `Funcionario` herda da superclasse `Usuario` (que por sua vez herda de `Pessoa`).

O **Tratamento de Exceções** é utilizado para garantir a robustez da aplicação. Funções que lidam com entrada e saída, como o carregamento de arquivos FXML (Figura 16), declaram ‘throws IOException‘, forçando o método chamador a lidar com possíveis falhas. O código também utiliza blocos ‘try-catch‘ para validação de entradas (ex: conversão de texto para números) e para aplicar regras de negócio (ex: tentativa de cadastrar um CPF duplicado). A Figura 17 ilustra o mecanismo de injeção de dependência do JavaFX, uma boa prática de design que desacopla a definição da interface de sua implementação, facilitando a manutenção.

```java
@FXML  Eduardo Rocha
private void goToCandidatura() throws IOException {
    Stage prevStage = (Stage) candidaturaButton.getScene().getWindow();
    prevStage.close();

    Stage candidaturaStage = new Stage();
    FXMLLoader candidaturaFXMLLoader = new FXMLLoader(getClass().getResource("/grupo/trabalho/candidatura-view.fxml"));
    Parent candidaturaRoot = candidaturaFXMLLoader.load();

    CandidaturaController candidaturaController = candidaturaFXMLLoader.getController();
    candidaturaController.setMainController(this);

    Scene scene = new Scene(candidaturaRoot);
    candidaturaStage.setTitle("Candidatura");
    candidaturaStage.setScene(scene);
    candidaturaStage.setResizable(false);
    candidaturaStage.show();
```

Figura 16: Exemplo de método para navegação entre telas, demonstrando modularização (carregamento de um FXML) e tratamento de exceções (`throws IOException`).

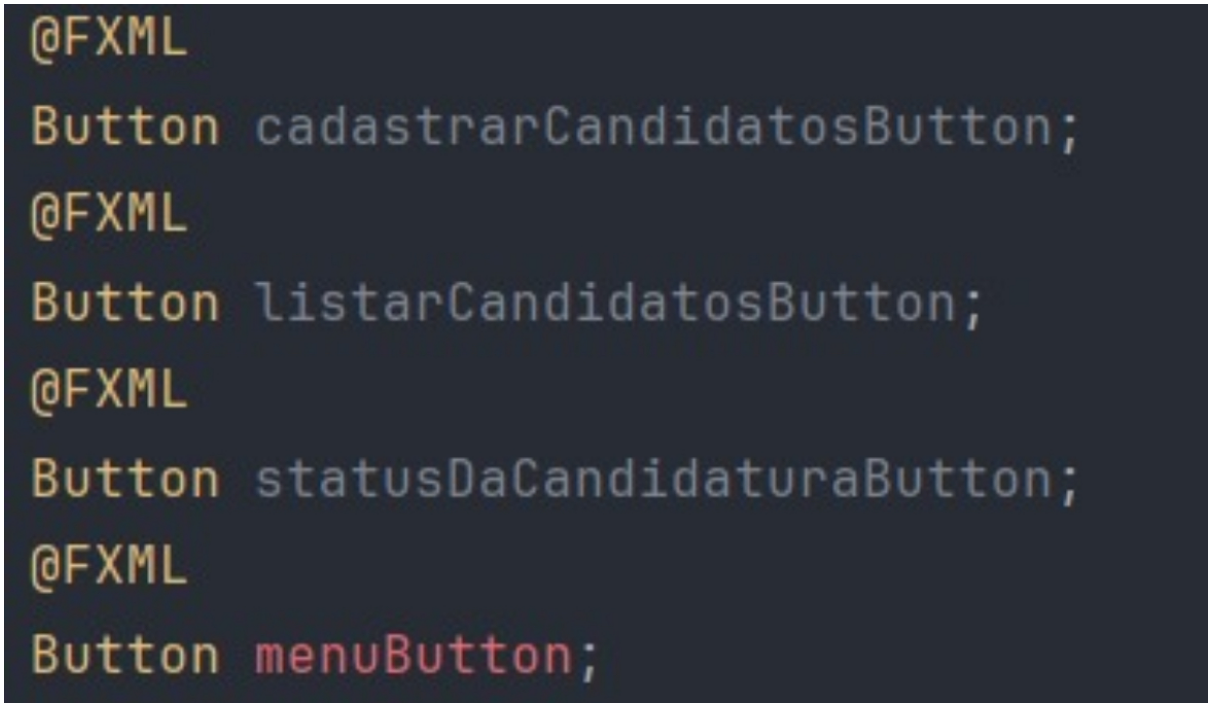

Figura 17: Vinculação de componentes FXML via anotação `@FXML`, exemplificando o desacoplamento entre a View e o Controller.

3.4 Desafios Encontrados e Soluções Adotadas

Durante o desenvolvimento, diversos desafios foram encontrados. A configuração inicial do ambiente JavaFX e SceneBuilder exigiu tempo para garantir a correta integração. O principal desafio técnico foi implementar a **transição entre telas** de forma coesa e que permitisse a passagem de informações entre módulos quando necessário. A solução adotada foi criar uma lógica de navegação centralizada nos controladores, utilizando o gerenciamento do ciclo de vida das janelas (‘Stage‘) do JavaFX.

Outro desafio foi a implementação da **persistência de dados**. Optou-se por utilizar arquivos Java para armazenar e recuperar as informações do sistema. Embora funcional, essa abordagem apresentou desafios relacionados à manipulação de arquivos e à garantia de integridade dos dados, especialmente com múltiplos usuários ou operações concorrentes (embora a concorrência não seja o foco atual). A **integração dos módulos** desenvolvidos individualmente também demandou coordenação para garantir que as interfaces e trocas de dados entre eles funcionassem corretamente, respeitando as dependências modeladas.

3.5 Análise Crítica da Qualidade da Solução

Avaliando a solução atual, identificamos pontos fortes e limitações importantes. Do lado positivo, o projeto está bem **modularizado**, seguindo a divisão de responsabilidades estabelecida e os princípios de baixo acoplamento e alta coesão, facilitados pelo uso do padrão MVC e JavaFX. A **navegação** entre as telas implementadas está fluindo naturalmente, proporcionando uma experiência de usuário básica e funcional dentro do escopo desenvolvido. A aplicação das técnicas de POO, como herança e encapsulamento, estabeleceu uma base de código organizada e com potencial de reuso.

No entanto, a principal limitação reside na **segurança**, que é praticamente inexistente neste estágio. A persistência de dados em arquivos Java, embora cumpra o requisito funcional, não oferece nenhuma proteção. Os arquivos podem ser facilmente acessados e modificados externamente, comprometendo a confidencialidade e integridade das informações de RH. Falta implementar mecanismos de **criptografia** para proteger esses dados. A autenticação atual é básica e não há controle robusto de permissões por perfil. A confiabilidade geral ainda é baixa devido à falta de tratamento exaustivo de erros e à ausência de um mecanismo de persistência mais robusto e seguro. Apesar disso, a estrutura estabelecida é um bom ponto de partida.

4 Conclusão e Evolução Futura

Este trabalho resultou no desenvolvimento de um protótipo funcional de um sistema de gestão de RH, com interfaces gráficas básicas e lógica de negócio inicial implementada, atendendo aos requisitos parciais da disciplina. A modelagem orientada a objetos e a arquitetura modular estabeleceram uma base sólida para futuras evoluções.

Como próximos passos e evoluções futuras, a equipe priorizará os seguintes pontos:

- **Segurança da Persistência:** Implementar mecanismos de **criptografia** para proteger os dados armazenados nos arquivos Java, garantindo maior confidencialidade e integridade das informações de RH. Avaliar alternativas mais seguras de persistência, se necessário.
- **Segurança de Acesso:** Desenvolver um sistema de autenticação mais robusto e implementar o controle de acesso baseado nos perfis de usuário definidos (Administrador, Gestor, etc.).
- **Padronização e Estética:** Padronizar o layout e o design das janelas para garantir uma identidade visual coesa e realizar melhorias estéticas na interface gráfica.
- **Finalização de Funcionalidades:** Completar a implementação de todos os requisitos funcionais pendentes, detalhados na Seção 1.3, para cada módulo.
- **Robustez:** Melhorar o tratamento de erros e exceções em toda a aplicação para aumentar a confiabilidade e fornecer feedback mais claro ao usuário em caso de falhas.

Referências

[1] DEITEL, Paul; DEITEL, Harvey. *Java: Como Programar.* 10ª ed. São Paulo: Pearson Prentice Hall, 2016.

[2] MARTIN, Robert C. *Código Limpo: Habilidades Práticas do Agile Software.* Rio de Janeiro: Alta Books, 2009.

[3] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos.* Porto Alegre: Bookman, 2000.

[4] ORACLE. *Getting Started with JavaFX.* Documentação Oficial. Disponível em: `https://openjfx.io/openjfx-docs/`. Acessado em set. 2025.

[5] BLOCH, Joshua. *Java Efetivo.* 3ª ed. Rio de Janeiro: Alta Books, 2018.