

Disciplina Técnicas de Programação 1

Humanize - Sistema de Gestão de Recursos Humanos

Grupo 7 – 2025-2

Davi Lopes Brito – 242023425 – davilopesbrito64@gmail.com

Ricardo Rian da Silva Melo – 242003861 – rianricardo836@gmail.com

Samara da Conceição Gomes – 242014097 – samaracgomiiis@gmail.com

Valquíria dos Santos Machado – 242003807 – valquiria.masan@gmail.com

Profa. Roberta Barbosa Oliveira

Resumo

O presente trabalho apresenta o desenvolvimento do sistema Humanize, uma aplicação voltada à gestão integrada de recursos humanos, projetada para otimizar os processos administrativos, de recrutamento e financeiros dentro de uma organização. A necessidade do software surgiu a partir da observação de dificuldades comuns no gerenciamento de informações de candidatos, vagas, colaboradores e regras salariais, que muitas vezes são realizadas de forma manual e descentralizada. Nesse contexto, o Humanize propõe uma solução capaz de centralizar cadastros, automatizar tarefas e melhorar a comunicação entre os setores, tornando o fluxo de trabalho mais ágil e eficiente.

O objetivo principal do projeto é desenvolver uma plataforma modular que permita gerenciar todo o ciclo de vida do colaborador — desde o recrutamento até a gestão financeira — promovendo integração entre os diferentes setores da empresa. Para isso, o sistema foi estruturado em módulos interdependentes, cada um com funcionalidades específicas e voltadas para suas respectivas demandas.

Durante o desenvolvimento, foram aplicados princípios de programação orientada a objetos, utilizando conceitos de classes, herança, polimorfismo e encapsulamento, além de práticas de modularização e reutilização de código. Também foram implementadas técnicas de tratamento de exceções e persistência de dados por meio de arquivos CSV, garantindo maior confiabilidade e facilidade na manutenção. As classes de repositório seguem o padrão Singleton, assegurando consistência nas operações de leitura e escrita.

O projeto foi desenvolvido na linguagem Java, com interface construída em JavaFX, e conta com uma arquitetura em camadas dividida entre Model, Repository e Controller, facilitando a organização e evolução do sistema. Como resultado, o Humanize oferece uma solução intuitiva, funcional e de fácil integração, proporcionando usabilidade, segurança e eficiência no gerenciamento de informações de recursos humanos.

Conclui-se que o sistema Humanize contribui significativamente para a modernização e digitalização dos processos administrativos, reduzindo o retrabalho, minimizando erros e otimizando o tempo de execução das atividades, consolidando-se como uma ferramenta relevante para o contexto corporativo e educacional em que foi desenvolvido.

1 Introdução

1.1 Contexto e Objetivos do Projeto

Como contexto, fomos inspirados a criar um sistema de RH.

Esta iniciativa surge como resposta a um problema comum nas empresas: a dependência de planilhas dispersas, documentos físicos e sistemas desconexos para gerenciar processos críticos de RH. Essa fragmentação gera dificuldades como acesso não centralizado às informações, duplicação de esforços e falta de padronização, impactando a eficiência do departamento. Cada tela foi projetada para atender a uma funcionalidade específica do RH, como administração e gestão, candidatura, recrutamento e financeiro, com a divisão de tarefas entre os membros do grupo, onde cada aluno ficou responsável por um tópico. Paralelamente, implementamos os controllers para gerenciar a lógica de navegação e a comunicação entre as diferentes partes da aplicação, garantindo um fluxo intuitivo e funcional entre as telas.

1.2 Requisitos Funcionais

Para iniciar a listagem e desenvolvimento dos requisitos do sistema, os dividimos em módulos. São requisitos funcionais encontrados do sistema:

1.2.1 Módulo de Cadastros

- Cadastro de Funcionários, Usuários e Candidatos: Permite o registro completo de todos os agentes envolvidos no sistema, com campos específicos para cada perfil;
- Cadastro de Usuários Administradores: Gerencia perfis de administração com diferentes níveis de acesso.

1.2.2 Módulo de Recrutamento e Seleção

- Candidatura a Vaga: Interface para candidatos se inscreverem em oportunidades disponíveis;
- Status de Candidatura: Acompanhamento em tempo real do andamento do processo seletivo;
- Marcar Entrevista: Agendamento e gestão de etapas avaliativas;
- Menu de Recrutamento: Navegação centralizada para todas as funcionalidades de seleção;
- Solicitar Contratação: Fluxo para formalização da admissão de candidatos aprovados.

1.2.3 Módulo Financeiro

- Contracheque e Folha de Pagamento: (Geração e consulta de demonstrativos de remuneração);
- Regras Salariais: Configuração de parâmetros remuneratórios;
- Relatório Financeiro: Análises e extrato de dados financeiros do departamento.

1.2.4 Módulo Administrativo

- Configurações administrativas: Personalização de parâmetros e comportamentos do sistema;
- Consultar Contratação: Acesso rápido ao histórico e detalhes de admissões;
- Relatório do administrador: Painéis gerenciais com métricas e indicadores de RH;
- Tela Principal do administrador: Dashboard unificado para gestão administrativa.

1.3 Requisitos não Funcionais

O sistema, além dos requisitos funcionais mencionados acima, terá suas características de qualidade para o sistema funcionar, como:

- Interface amigável e responsiva, desenvolvida em JavaFX;
- Usabilidade adequada, com botões claros, feedback ao usuário e alertas de exclusão ou edição;
- Mensagens de erro, pelas especificações do uso;
- Autenticação obrigatória por login e senha.

1.4 Regras de negócio

Outros aspectos que impactam diretamente no funcionamento do sistema são as condições, restrições e normas do site. Essas regras refletem no desenvolvimento do ambiente em que o software será utilizado, e algumas delas são:

- Perfis de acesso hierárquicos (Administrador, Gestor, Recrutador, Funcionário);
- Candidatos podem se candidatar a várias vagas, mas apenas uma vez por vaga;
- Cada candidatura deve ter um status: Pendente, Em análise, Aprovado ou Reprovado;
- Apenas candidaturas pendentes podem ser excluídas;
- Somente candidatos aprovados e entrevistados podem ser contratados;
- Apenas gestores podem criar vagas e autorizar contratações.
- A folha de pagamento deve incluir apenas funcionários ativos;
- Prestadores de serviço devem estar associados a contratos válidos e categorias específicas.

Serão utilizados, principalmente, tratamento de exceções para cumprir as regras de negócio previstas.

1.5 Estrutura do Relatório

As próximas seções do relatório detalham a análise e o desenvolvimento do software implementado. Na Seção 2 são apresentadas a modelagem, a estrutura, os módulos, a arquitetura e as funcionalidades desenvolvidas, descrevendo como cada etapa do projeto contribuiu para a construção do software. Na Seção 3, discute-se a análise das funcionalidades desenvolvidas, destacando resultados alcançados, integração entre os módulos, confiabilidade, usabilidade e outras características do software, além de eventuais limitações ou desafios encontrados durante o desenvolvimento. Por fim, a Na Seção 4 sintetiza a relevância do trabalho desenvolvido, avaliando seu impacto no contexto aplicado e apontando possíveis melhorias e extensões para evoluções futuras.

Nos relatórios parciais, devem ser incluídas apenas as partes que já foram desenvolvidas até o momento. Cada entrega deve corresponder a uma etapa específica da seção de Solução Proposta: Estrutura Inicial, Design e Modelagem, Implementação da Lógica do Software e Integração e Navegabilidade. Nessas versões, é obrigatório incluir o Resumo e a Introdução, enquanto a Solução Proposta deve refletir a etapa de desenvolvimento atual, detalhando funcionalidades, diagramas e técnicas de programação aplicadas. Alterações ou melhorias em relação a versões anteriores devem ser documentadas na versão atual.

No relatório final, o documento deve ser revisado e atualizado, mantendo apenas o conteúdo referente ao que foi de fato desenvolvido, contemplando todas as implementações, funcionalidades, diagramas, módulos, técnicas de programação aplicadas e integração final. Além do Resumo, da Introdução e da Solução Proposta, o relatório final deve incluir as seções Resultados e Discussão, e Conclusão e Evolução Futura, para refletir a solução completa desenvolvida pelo grupo e sua relevância para o domínio aplicado.

2 Solução Proposta

2.1 Estrutura Inicial

Finalmente, iniciando o projeto, definimos algumas ferramentas para auxiliar na organização, versionamento e padronização do projeto, são elas:

- **Git e Github:** guarda o repositório com o projeto, bem como o torna acessível para quaisquer membros através do Git, que também auxilia no versionamento e controle do código com diversas pessoas na equipe;
- **Jetbrains IntelliJ:** foi a IDE escolhida para o desenvolvimento do sistema, por recomendação da professora e também pela sua facilidade de uso;
- **Maven:** é uma ferramenta de gestão de dependências em projetos Java que simplifica e padroniza o processo de desenvolvimento, principalmente com a utilização do JavaFX, que é bem exigente quanto às dependências;
- **Arquitetura MVC:** é um padrão de design que separa uma aplicação em três partes interligadas: o Modelo (que lida com os dados e a lógica de negócios), a Visão (a interface gráfica que o utilizador vê e com a qual interage) e o Controlador (que atua

como intermediário, processando as requisições do utilizador e coordenando as interações entre o Modelo e a Visão)

- **Gitflow:** é um modelo de desenvolvimento de software que utiliza um conjunto de estratégias de ramificação (branches) no Git para organizar o ciclo de vida de um projeto, especialmente para aqueles com lançamentos agendados.

Através dessas ferramentas, inicializamos o repositório no Github e modelamos de acordo com as ferramentas acima, acesse o projeto em: <https://github.com/davilb64/projeto-tp1>.

O grupo foi dividido da seguinte forma:

- Administração e Gestão: Davi Lopes;
- Candidatura: Valquiria Machado;
- Recrutamento: Ricardo Rian;
- Financeiro: Samara Gomes.

Para além disso, o grupo, nessa primeira etapa, focou no desenvolvimento da interface gráfica e no sistema de navegação para o sistema de recursos humanos. A construção das telas foi realizada utilizando o **SceneBuilder**, software "arrasta e solta" que permite a edição de arquivos *.FXML* de forma gráfica, permitindo uma criação visual e organizada de cada interface.

Foram criadas as seguintes telas (disponíveis no Github. Caminho: <https://github.com/davilb64/projeto-tp1/blob/c75af91dc5671013a23e288b1407b1b662df4c3f/src/main/resources/view>), cada membro em sua respectiva área de desenvolvimento. Telas descritas em 3.2. Cada qual com seu "Controller" respectivo para manipulação básica e interações entre telas.

2.2 Design e Modelagem

Na seção de Design e Modelagem, é elaborado o Diagrama de Casos de Uso, que apresenta as principais interações entre os usuários e o sistema. Além disso, é desenvolvida a representação conceitual do sistema por meio de diagramas de classes, estruturando seus principais elementos e relacionamentos.

2.3 Diagrama de Casos de Uso

O diagrama de caso de uso resume os detalhes dos agentes do sistema

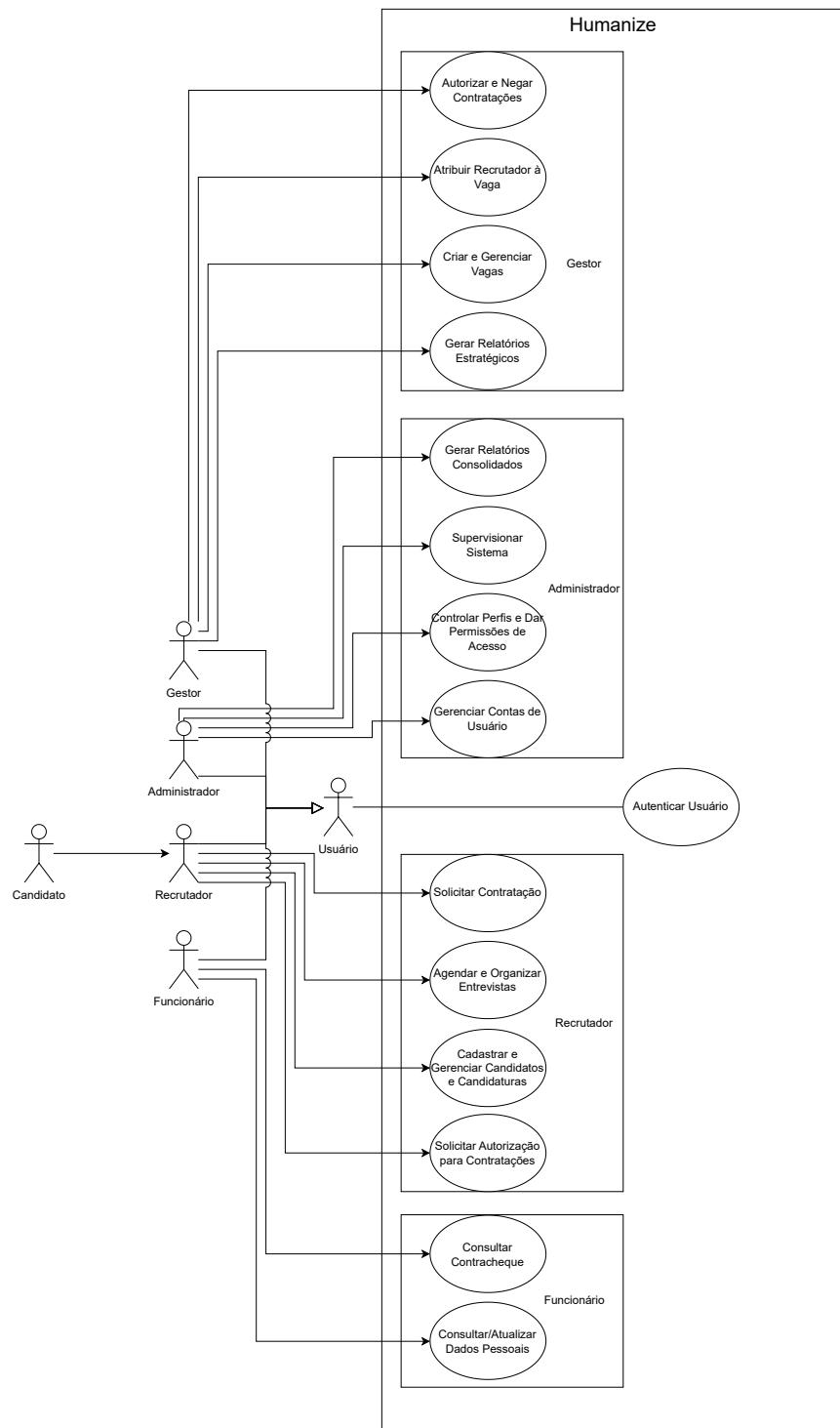


Figura 1: Diagrama de Casos de Uso

Descrição dos Agentes

- **Usuário:**

Generalização de agentes que possuem acesso ao sistema.

- **Gestor:**

É responsável por criar e gerenciar vagas de emprego, atribuir recrutadores responsáveis às vagas, avaliar os pedidos de contratação submetidos pelo recrutadores e gerar relatórios estratégicos para apoio à tomada de decisão (por exemplo: contratação por vaga, folha de pagamento por regime de trabalho).

- **Administrador:**

É responsável por gerenciar contas de usuários (criar, editar, excluir), controlar perfis (Gestores, Funcionários, Recrutadores) e permissões de acesso, supervisionar as operações gerais do sistema.

- **Recrutador:**

É responsável por cadastrar e gerenciar candidatos e suas candidaturas, registrar e organizar entrevistas (data e nota), atualizar o status do processo seletivo, solicitar autorização para contratações de candidatos aprovados, realizar a admissão dos candidatos.

- **Funcionário:**

É responsável por consultar e atualizar algumas informações pessoais (como endereço, telefone e senha de acesso), visualizar contracheques, benefícios e histórico financeiro individual.

- **Candidato:**

Não tem acesso direto ao sistema. Seus dados são registrados pelo Recrutador e o acompanhamento do processo seletivo deve ocorrer por meio de canais externos (como e-mail, telefone ou portal da empresa).

Com o diagrama de casos de uso em mãos, podemos ter uma noção melhor de quais classes devem ser criadas com quais intenções, levando em conta, principalmente, as ações de nossos agentes.

2.3.1 Diagrama de Classes

Coloque seu diagrama na pasta “Diagramas” e substitua pelo nome do seu arquivo.

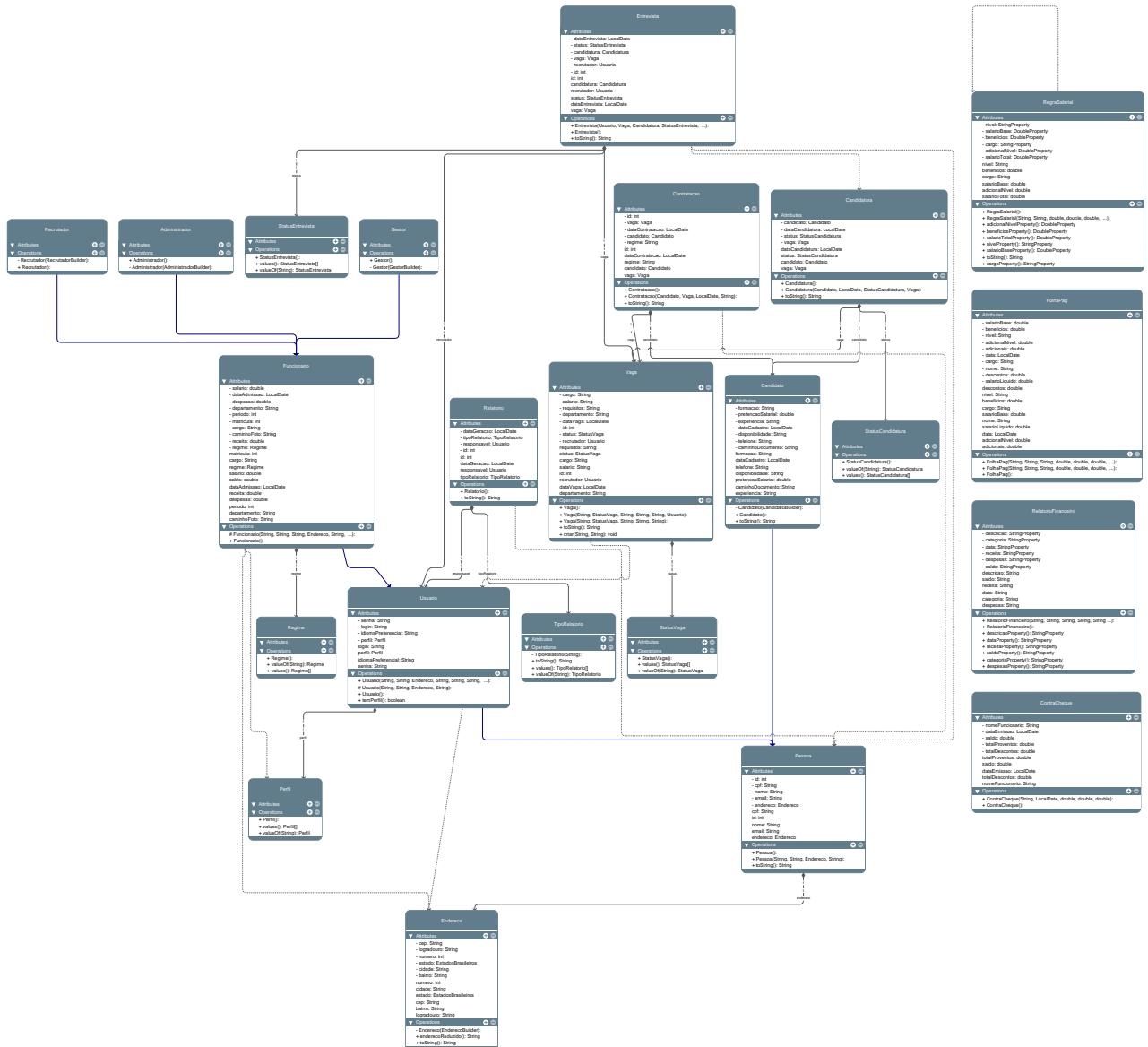


Figura 2: Diagrama de Classes de Domínio

Descrição das Classes

Descreva as principais classes do seu software, destacando suas responsabilidades, atributos e métodos. Exemplo:

- Pacote Model:

– Classe Abstrata Pessoa:

A classe Pessoa é uma classe abstrata que serve como modelo base para outras entidades do sistema. Ela contém os principais atributos de identificação de uma pessoa — id, nome, CPF, email e endereço — além de métodos de acesso (getters e setters) para manipular esses dados.

* **Classe Abstrata Usuario extends Pessoa:**

A classe Usuario, transforma *Pessoa* em um usuário do sistema, ou seja, alguém que terá login e senha de acesso ao sistema.

* **Classe Funcionario extends Usuario:**

A classe Funcionario representa uma especialização da classe abstrata Usuario, sendo responsável por modelar as informações específicas dos colaboradores do sistema. Ela armazena dados administrativos e financeiros relevantes, como matrícula, data de admissão, período, receita, despesas, salário, cargo, regime de trabalho, departamento e o caminho da foto do funcionário. Além dos métodos de acesso (getters e setters) para manipulação desses atributos, a classe também disponibiliza um método para calcular o saldo financeiro, considerando receitas, salário e despesas. A classe faz uso do padrão de projeto Builder, por meio da classe interna estática FuncionarioBuilder, que permite a criação de objetos Funcionario de forma mais legível e flexível, facilitando a inicialização de múltiplos atributos sem sobrecarregar o construtor principal. As classes que herdam de *Funcionario*, não possuem atributos próprios, porém, possuem diferentes níveis de acesso às funcionalidades. *Funcionario* tem acesso mínimo.

* **Classe Administrador extends Funcionario:**

A classe *Administrador* é a que mais possui acessos, podendo acessar *TODOS* os módulos.

* **Classe Gestor extends Funcionario:**

A classe *Gestor* possui grande número acessos, podendo acessar diversos módulos, com foco em módulos para criação de vagas e contratação de funcionários.

* **Classe Recrutador extends Funcionario:**

A classe *Recrutador* tem acesso limitado a entrevistas e criação de candidatos, possui baixo nível de acessos.

– **Classe Candidato extends Pessoa:**

A classe Candidato representa uma pessoa que poderá ser associada a uma vaga por meio da candidatura, entretanto, o candidato não tem acesso ao sistema, sendo ele cadastrado pelo recrutador, que também realiza as candidaturas. Ela herda de Pessoa e adiciona informações específicas como formação, experiência, pretensão salarial, entre outros. Ela também utiliza o padrão Builder, permitindo criar objetos de forma organizada e flexível.

• **Classe Candidatura:**

A classe Candidatura representa a inscrição de um candidato em uma vaga. Ela relaciona um Candidato, uma Vaga, o status da candidatura (como pendente, em análise, aprovado ou reprovado) e a data da candidatura. Serve como elo entre candidatos e vagas, permitindo o controle e atualização do progresso de cada processo seletivo.

• **Classe ContraCheque:**

A classe contracheque representa o documento financeiro de um funcionário, contendo as informações de proventos, descontos, saldo, data de emissão e salario bruto. Este

documento é super importante para a transparência das relações trabalhistas. Ela herda informações de folha de pagamento e relatorio financeiro.

- **Classe Contratacao:**

A classe Contratacao é responsável por representar o processo de admissão de um candidato em uma vaga disponível no sistema. Ela associa diretamente as entidades Candidato e Vaga, registrando informações essenciais sobre a contratação, como a data em que ocorreu e o regime de trabalho adotado (por exemplo, CLT, estágio ou PJ).

Cada instância da classe possui um identificador único gerado automaticamente e métodos de acesso (getters e setters) que permitem manipular seus atributos de forma controlada. Além disso, o método `toString()` foi sobrescrito para fornecer uma descrição textual resumida da contratação, incluindo o nome do candidato, o regime e o cargo correspondente à vaga associada.

- **Classe Endereco:**

A classe *Endereco* foi criada com o intuito de facilitar o armazenamento de um endereço, pois um endereço possui diversos atributos (CEP, rua, cidade), assim, podemos acessar todo esse endereço através de uma chamada no objeto, armazená-lo numa String poderia gerar alguns conflitos de formatação.

- **Classe Entrevista:**

A classe Entrevista representa o agendamento e o registro das entrevistas realizadas entre candidatos e recrutadores durante o processo seletivo. Ela estabelece a relação entre as entidades Usuario (que atua como recrutador), Vaga e Candidatura, armazenando informações fundamentais como a data da entrevista, o status atual (por meio do enumerador StatusEntrevista) e o relatório elaborado após a avaliação do candidato.

Cada objeto Entrevista possui um identificador único gerado automaticamente e métodos de acesso que permitem consultar e modificar seus atributos de forma controlada. O método `toString()` foi sobrescrito para exibir uma identificação simplificada da entrevista, destacando o nome do candidato associado.

- **Classe FolhaPag:**

A classe Folha de pagamento é responsável por armazenar e gerenciar as informações gerais dos funcionários relacionadas à folha de pagamento. Nela são utilizados como ferramenta de busca os dados como nome, cargo, nível e período de referência, além de adicionarmos valores correspondentes a adicionais (como horas extras) e descontos (como atrasos e multas). A partir desses dados, é possível compor a tabela de folha de pagamento dos colaboradores, exibindo de forma organizada o nome, cargo, salário bruto, descontos aplicados e o total líquido a ser recebido.

- **Classe RegraSalarial:**

A classe regra salarial é responsável por definir os parâmetros e critérios utilizados no cálculo dos salários dos funcionários. Ela herda informações referentes ao funcionário (como cargo, nome) foi acrescentado tambem dados como o salário base e o adicional

por nível. Essa classe desempenha um papel fundamental dentro do sistema, pois serve como base para as demais classes do módulo financeiro, garantindo a padronização e a consistência dos cálculos salariais.

- **Classe Relatorio:**

A classe *Relatorio* é responsável por armazenar os metadados de um relatório gerado no sistema. Ela não armazena o relatório em si, mas sim as informações sobre ele, como seu id de identificação, o tipoRelatorio, a dataGeracao e o Usuario que executou a geração. Essa classe é essencial para manter um histórico de todos os relatórios emitidos, permitindo que eles sejam listados e consultados posteriormente.

- **Classe RelatorioFinanceiro:**

A classe relatorio financeiro é responsável por representar e gerar os registros de transações financeiras do sistema. Ela permite o acompanhamento detalhado de todas as movimentações, armazenando informações organizadas por data, descrição, receita, despesas, saldo e categoria. Por meio dessas informações, é possível realizar o controle de entradas e saídas, analisar o desempenho financeiro e identificar padrões de gastos e receitas.

- **Classe Vaga:**

A classe Vaga representa as oportunidades de emprego disponíveis no sistema, armazenando informações essenciais sobre cada cargo ofertado. Ela contém atributos como o título do cargo, salário, requisitos, departamento responsável, data de abertura e o status atual da vaga (definido pelo enumerador StatusVaga). Além disso, a classe mantém uma associação direta com um objeto Usuario, que atua como o recrutador responsável pela vaga.

Cada instância possui um identificador único gerado automaticamente e fornece métodos de acesso (getters e setters) que permitem manipular seus dados de forma segura. O método `toString()` foi sobrescrito para retornar uma descrição textual detalhada da vaga, incluindo seu identificador, cargo, status, salário e requisitos.

- **Classe Enumerativa Perfil:**

Define os perfis permitidos para usuários (ADMINISTRADOR, GESTOR, RECRUTADOR e FUNCIONARIO).

- **Classe Enumerativa Regime:**

Define os regimes permitidos para contratação (CLT, PJ e ESTAGIARIO).

- **Classe Enumerativa StatusCandidatura:**

Define o status da candidatura como sendo "pendente", "em análise", "aprovado" e "reprovado".

- **Classe Enumerativa StatusEntrevista:**

A classe StatusEntrevista é uma enumeração que define os possíveis estados de uma entrevista dentro do processo seletivo. Ela contém três valores constantes — Pendente, Aprovado e Reprovado — que representam, respectivamente, uma entrevista ainda não

concluída, uma entrevista em que o candidato foi aprovado e uma entrevista cujo resultado foi negativo.

- **Classe Enumerativa StatusVaga:**

A classe StatusVaga é uma enumeração que representa os possíveis estados de uma vaga de emprego no sistema. Ela possui dois valores constantes — ABERTA e FECHADA — que indicam, respectivamente, se a vaga está disponível para candidaturas ou se já foi encerrada.

- **Classe Enumerativa TipoRelatorio:**

Define o tipo de relatório gerado (LISTA_USUARIOS, CONTRACHEQUE_GERAL, FINANCEIRO_GERAL).

- **Pacote Exceptions**

Este pacote armazena todas as exceções customizadas do sistema. Elas são usadas para sinalizar erros específicos de regras de negócio, permitindo que as camadas de serviço e controle (Controllers) capturem e tratem esses erros de forma adequada, geralmente exibindo um alerta ao usuário.

- **Classe CpfInvalidoException**

É uma exceção lançada pelo *ValidaCpf* ou serviços relacionados quando o CPF fornecido por um usuário durante um cadastro ou atualização não atende aos padrões de formato ou falha no cálculo dos dígitos verificadores.

- **Classe EmailInvalidoException**

É uma exceção lançada pelo *ValidaEmail* quando o formato do email inserido em um formulário não corresponde a um padrão de email válido, garantindo a integridade do dado.

- **Classe LoginException**

Uma exceção genérica utilizada pelo *LoginService* para sinalizar qualquer tipo de falha durante o processo de autenticação. Ela serve como uma superclasse para erros mais específicos como *SenhaIncorrectaException* e *UsuarioNaoEncontradoException*.

- **Classe SenhaIncorrectaException extends LoginException**

É uma exceção específica lançada pelo *LoginService* quando o usuário é encontrado no sistema, mas a senha fornecida por ele não corresponde à senha armazenada no repositório.

- **Classe SenhaInvalidaException**

É uma exceção lançada pelo *ValidaSenha* durante o cadastro ou alteração de senha, indicando que a senha fornecida não cumpre os requisitos mínimos de segurança (ex; número mínimo de caracteres, presença de letras e números).

- **Classe UsuarioNaoEncontradoException extends LoginException**

É uma exceção lançada pelo *LoginService* ou *UsuarioRepository* quando uma tentativa de login é feita com um email ou CPF que não existe na base de dados (ou seja, o usuário não está cadastrado).

– Classe `ValidacaoException extends LoginException`

Uma exceção genérica usada para sinalizar falhas de validação de regras de negócio que não se enquadram nas outras exceções. Por exemplo, ela pode ser usada para impedir que um candidato se candidate duas vezes à mesma vaga.

– Pacote `Repository`:

* Classe Abstrata `BaseRepository`:

Uma classe ”mãe” para as demais *Repositories*. Precisou ser criada pois o sistema de arquivos utilizados durante o desenvolvimento não é realizável após fazer o empacotamento do *.jar* e *.exe*, por conta das permissões de escrita. Essa classe define uma pasta na *Home* do usuário onde os arquivos de persistência serão salvos, e reescritos.

· Classe `CandidatoRepository extends BaseRepository`:

A classe `CandidatoRepository` segue o padrão Singleton, garantindo uma única instância em toda a aplicação. Ela mantém uma lista de candidatos em memória e realiza operações de CRUD (criar, ler, atualizar e remover) com persistência em um arquivo `candidatos.csv`.

· Classe `CandidaturaRepository extends BaseRepository`:

A classe `CandidaturaRepository` é responsável por armazenar as candidaturas dos candidatos às vagas. Ela utiliza o padrão Singleton, garantindo uma única instância ativa em todo o sistema.

· Classe `ContrachequeRepository extends BaseRepository`:

A classe `contrachequeRepository` é responsável por guardar e buscar os contracheques dos funcionários, ela usa o arquivo `contracheque.csv`, tendo duas funções principais, a de buscar contracheques e a de salvar novos contracheques

· Classe `ContratacaoRepository extends BaseRepository`:

Ela é responsável por gerenciar todas as contratações registradas no sistema, armazenando-as em memória e garantindo a persistência dos dados no arquivo `contratacoes.csv`.

A classe executa operações de CRUD (criação, leitura, atualização e exclusão) sobre objetos do tipo `Contratacao`, mantendo a consistência entre os dados da aplicação e o arquivo de armazenamento.

Durante o carregamento, o repositório realiza o parse das linhas do arquivo CSV para reconstruir objetos `Contratacao`, incluindo suas relações com `Candidato` e `Vaga`.

Além disso, a classe contém métodos específicos para gerar identificadores automáticos (`getProximoId()`), persistir alterações (`persistirAlteracoesNoCSV()`), e formatar os dados para escrita no arquivo.

Em caso de ausência do arquivo de persistência, o repositório realiza a cópia de um arquivo padrão a partir dos recursos da aplicação, garantindo o correto funcionamento do sistema mesmo em primeira execução.

· Classe `EntrevistaRepository extends BaseRepository`:

Sua principal responsabilidade é gerenciar o ciclo de vida das entrevistas — armazenando, recuperando, atualizando e removendo registros — com persistência dos dados no arquivo `entrevistas.csv`.

Ela mantém uma lista de objetos `Entrevista` em memória, permitindo acesso rápido às informações e sincronização constante com o arquivo CSV.

Durante a inicialização, o repositório realiza a leitura do arquivo e reconstrói as entrevistas, associando corretamente seus objetos relacionados, como `Candidato`, `Vaga`, `Recrutador` e `Candidatura`.

Além das operações básicas de CRUD, a classe oferece métodos utilitários que facilitam consultas específicas, como:

Buscar entrevistas marcadas para o dia atual, filtrar candidatos aprovados em entrevistas, localizar entrevistas por nome do candidato, obter vagas vinculadas a candidatos aprovados.

A escrita e leitura dos dados são realizadas com controle de integridade, tratando exceções de parsing e normalizando textos de enums (como `StatusEntrevista` e `StatusCandidatura`).

O método `persistirAlteracoesNoCSV()` garante que toda modificação em memória seja imediatamente refletida no arquivo, mantendo a persistência consistente.

- **Classe `FolhaPagRepository` extends `BaseRepository`:**

Este repositório, seguindo o padrão Singleton, gerencia a persistência dos objetos `FolhaPag` no arquivo `folhapagamento.csv`. É responsável por carregar as folhas de pagamento para a memória na inicialização e persistir quaisquer novas folhas geradas, garantindo que o histórico de pagamentos seja mantido.

- **Classe `RelatorioFinanceiroRepository` extends `BaseRepository`:**

A classe Relatorio financeiro é responsável por guardar e buscar as transações financeiras da empresa, ela utiliza o arquivo `relatorio_financeiro.csv` para armazenar os dados, essa classe tem três funções principais : criar o arquivo caso não exista ainda, salvar todas as transações e as carregar.

- **Classe `RelatorioRepository` extends `BaseRepository`:**

Este repositório gerencia a persistência dos metadados dos relatórios (objetos `Relatorio`) no arquivo `relatorios.csv`. Ele salva um registro de cada relatório gerado, incluindo o tipo, a data e o responsável, permitindo que a tela de "Relatórios" liste um histórico de todas as exportações.

- **Classe `SalarioRepository` extends `BaseRepository`:**

A classe SalarioRepository é responsável por guardar e buscar as regras salariais dos cargos, utilizando o arquivo `regras_salariais.csv` para armazenar os dados, Esta classe tem duas funções principais, salvar uma nova regra ou atualizar caso exista e carregar as regras salariais salvias. Caso o arquivo não exista, ela cria um automaticamente.

- **Classe `UsuarioRepository` extends `BaseRepository`:**

Um dos repositórios centrais do sistema, o `UsuarioRepository` gerencia a persistência de todos os tipos de `Usuario` (Administrador, Gestor, etc.) no

arquivo *usuarios.csv*. Ele lida com o polimorfismo no (de)serialização, salvando e carregando os atributos específicos de cada perfil de usuário. É fundamental para o *LoginService*, fornecendo métodos para buscar usuários por email ou CPF.

- **Classe VagaRepository extends BaseRepository:**

Sua principal responsabilidade é gerenciar o ciclo de vida das vagas — criação, listagem, atualização e exclusão — com persistência dos dados em um arquivo CSV.

Ao ser inicializada, a classe carrega automaticamente todas as vagas existentes para a memória, garantindo que as operações sejam rápidas e consistentes.

Os objetos Vaga são reconstruídos a partir do CSV, preservando informações como cargo, salário, status, requisitos, departamento, data da vaga e dados do recrutador associado.

Além das operações básicas de CRUD, a classe fornece métodos utilitários para recuperar todas as vagas cadastradas, listar somente as vagas com status ABERTA, filtrar vagas abertas por recrutador específico, obter a quantidade total de vagas, listar todos os cargos disponíveis.

A escrita no arquivo é controlada pelo método *persistirAlteracoesNoCSV()*, que garante que toda modificação na lista em memória seja imediatamente refletida no arquivo CSV, mantendo a persistência sincronizada.

Durante a leitura (*carregarVagaDoCSV()*), a classe realiza tratamento de exceções e parsing seguro, prevenindo falhas causadas por inconsistências nos dados.

O método *getProximoId()* é responsável por gerar automaticamente o identificador incremental das vagas, assegurando unicidade.

A remoção e atualização de registros são seguidas de persistência imediata, garantindo que o estado do arquivo esteja sempre alinhado ao da memória.

- **Pacote Service**

Este pacote implementa a camada de serviço, que contém a lógica de negócios centralizada. Ele atua como um intermediário entre os *Controllers* (que não devem conter regras de negócio) e os *Repositories* (que apenas persistem dados).

- *** Pacote Formatters**

Este pacote contém as implementações do padrão de design Strategy para a formatação de relatórios. Ele desacopla a lógica de *geração* de dados da lógica de *formatação* do arquivo final (PDF ou CSV).

- **Interface IReportFormatter**

Define o contrato (método ‘formatar’) que todas as classes de formatação devem implementar. Ela recebe um objeto *ReportData* e é responsável por convertê-lo para um formato específico.

- **Classe CsvFormatter implements IReportFormatter**

Implementação concreta da interface que transforma os dados contidos no *ReportData* (cabeçalhos e linhas) em uma *String* formatada com ponto e

vírgula, pronta para ser salva em um arquivo .csv.

- **Classe PdfFormatter implements IReportFormatter**

Implementação que utiliza uma biblioteca externa (como iText ou OpenPDF) para pegar os dados do *ReportData* e construir um documento PDF, formatando tabelas, títulos e parágrafos.

- * **Pacote Relatorios**

Contém as classes responsáveis por *coletar e preparar* os dados para cada tipo de relatório.

- **Interface IGeradorRelatorio**

Define o contrato (método ‘gerarDados’) para as classes que sabem como buscar e processar dados de um relatório específico.

- **Classe RelatorioContracheque implements IGeradorRelatorio**

Implementação responsável por buscar os dados de um funcionário e seus cálculos financeiros no *ContrachequeRepository* para montar os dados de um contracheque.

- **Classe RelatorioHistoricoFinanceiro implements IGeradorRelatorio**

Implementação que consulta o *RelatorioFinanceiroRepository* para agrupar todas as transações (receitas e despesas) e gerar os dados para o relatório financeiro geral.

- **Classe RelatorioListaUsuarios implements IGeradorRelatorio**

Implementação que busca todos os usuários no *UsuarioRepository* e formata os dados (nome, email, perfil) para serem exibidos no relatório de usuários.

- * **Classe ReportData**

Uma classe de transporte de dados simples. Ela armazena os dados gerados por um *IGeradorRelatorio* (ex. cabeçalhos e uma lista de linhas) de forma genérica, para que possam ser passados para qualquer *IReportFormatter*.

- * **Pacote Validacoes**

Pacote que agrupa classes utilitárias estáticas responsáveis pela validação de dados de entrada, como formulários de cadastro.

- **Classe ValidaCpf**

Fornece métodos estáticos para verificar se uma *String* de CPF é válida, aplicando o algoritmo de cálculo dos dígitos verificadores. Lança *CpfInvalidoException* se a validação falhar.

- **Classe ValidaEmail**

Utiliza uma expressão regular (Regex) para verificar se uma *String* corresponde a um formato de email válido. Lança *EmailInvalidoException* se o formato for inválido.

- **Classe ValidaSenha**

Verifica se uma senha atende aos requisitos de segurança definidos (ex. comprimento mínimo, combinação de caracteres). Lança *SenhaInvalidaException* se os critérios não forem atendidos.

- * **Classe LoginService**

Orquestra a lógica de autenticação. Recebe o login e a senha do *LoginController*, usa o *UsuarioRepository* para buscar o usuário e, em caso de sucesso, inicializa a *UserSession*. É responsável por lançar as exceções de login.

- **Pacote Util**

Pacote que contém classes utilitárias diversas, que oferecem funcionalidades de apoio reutilizáveis por toda a aplicação, mas que não se encaixam em outras categorias de serviço ou modelo.

- **Classe final UserSession**

Uma classe Singleton (e final, não podendo ser herdada) que armazena o objeto *Usuario* do usuário atualmente logado. Ela permite que qualquer *Controller* acesse os dados do usuário (como nome, perfil, permissões) de forma centralizada.

- **Classe EnderecoViaCep**

Classe utilitária que se conecta à API externa ”ViaCEP”. Ela possui um método que, ao receber um CEP, faz uma requisição HTTP, processa o JSON de resposta e retorna um objeto *Endereco* com os campos (rua, bairro, cidade) preenchidos.

- **Classe ScreenController**

Classe central para o gerenciamento da navegação entre as telas (Scenes) do JavaFX. Ela é responsável por carregar os arquivos FXML, injetar dependências (como a própria instância do *ScreenController*) nos *Controllers* das telas e realizar a troca de cenas na janela principal.

- **Classe enumerativa EstadosBrasileiros**

Um ‘enum’ que lista os 27 estados brasileiros (sigla e nome por extenso). É utilizado para popular componentes de interface como *ComboBox* em formulários de endereço, garantindo a padronização dos dados.

2.4 Implementação da Lógica do Software

A transformação da modelagem em código funcional deve ser abordada, detalhando como as regras de negócio, os mecanismos de persistência e o tratamento de exceções foram implementados. As regras de negócio implementadas devem ser explicadas, mostrando como o software aplica as normas e restrições do domínio. Para os mecanismos de persistência, é necessário detalhar como os dados são armazenados e recuperados, como por meio de arquivos, listas, coleções ou outros recursos de persistência utilizados no projeto. Para o tratamento de exceções, devem ser apresentados exemplos de situações previstas e como o sistema lida com elas, como entradas inválidas, arquivos não encontrados ou operações inválidas, garantindo robustez e confiabilidade.

2.5 Integração e navegabilidade

A integração entre a interface gráfica e a lógica de negócio no sistema foi realizada utilizando o padrão de arquitetura MVC (Model–View–Controller), que separa as responsabilidades

de cada camada. As telas desenvolvidas em FXML (View), por meio do JavaFX, estão conectadas aos controladores (Controller), que controlam a interação do usuário e usam os métodos de modelo e repositório (Model e Repository). Essa estrutura garante modularidade, reutilização de código e facilidade de manutenção. Por exemplo, a tela Cadastro de Candidato está vinculada à classe CadastroDeCandidatoController, responsável por coletar as informações inseridas pelo usuário e validá-las antes de criar ou atualizar objetos da classe Candidato. Esses dados são enviados para o CandidatoRepository, que realiza a persistência em arquivo CSV, garantindo o armazenamento e recuperação das informações para usos futuros de forma eficiente.

A naveabilidade entre telas é feita por meio dos controladores, que carregam diferentes arquivos FXML conforme as ações do usuário. Isso permite alternar entre telas sem perda de contexto, limitando, por exemplo, o acesso dos usuários às telas. Esse raciocínio segue as regras de negócio do sistema, em que, por exemplo, restringe o gestor a somente visualizar as candidaturas em análise e pendentes, diferentemente do recrutador, que tem acesso aos candidatos, ao cadastro de candidato, a associação de candidaturas e a todas estas, independente do status. Além disso, os botões da interface podem alterar sua visibilidade e interatividade de acordo com o estado atual do sistema (por exemplo, ocultando botões de edição em modo de visualização).

O sistema utiliza também em sua estrutura os repositórios, que servem para abstrair e centralizar o acesso aos dados especificados em cada repositório, como um banco de dados, fornecendo uma interface para operações como salvar, consultar, atualizar e excluir dados de maneira organizada. Assim, no sistema, é possível manter a coerência na interação entre os módulos por meio do uso de repositórios únicos (Singletons), o que garante que todas as telas acessem os mesmos dados atualizados em memória e nos arquivos persistidos. Isso também simplifica a integração entre partes diferentes do sistema, como a associação entre candidatos, vagas e candidaturas.

Em termos de usabilidade, foram aplicados elementos visuais de feedback, como alertas informativos e de confirmação, exibidos através da classe Alert do JavaFX. Esses recursos alertam o usuário sobre o sucesso ou falha das operações realizadas. Além disso, funcionalidades como o filtro de busca nas tabelas tornam a navegação mais ágil, permitindo localizar, por exemplo, candidatos ou candidaturas por nome, cargo ou status rapidamente. A atualização das tabelas é feita de maneira automática através de listas observáveis (ObservableList), garantindo que qualquer modificação nos dados seja imediatamente refletida na interface.

Assim, a integração entre interface, controle e dados ocorre de forma fluida, proporcionando uma navegação fácil e conectada, seguindo as regras do sistema, utilizando os princípios de modularidade e reutilização de código.

3 Resultados e Discussão

Nesta seção, deve ser apresentada a análise do software desenvolvido, destacando as funcionalidades implementadas, a integração entre os módulos e a eficácia da solução frente aos

objetivos propostos.

3.1 Tecnologia Utilizadas

Para além das tecnologias de versionamento e padrão de projeto citados em 2, temos também algumas que foram adotadas conforme o andamento do projeto, são elas:

- **Java 17:** Optamos pela versão 17 da linguagem de programação Java, pois ela se mostra mais estável ao trabalhar com o JavaFX, tecnologia utilizada para a criação da interface gráfica;
- **JavaFX:** A biblioteca *JavaFX* foi escolhida por conta do seu poder em produzir interfaces agradáveis ao usuário de forma prática (através do SceneBuilder), além de contar com suporte para *CSS* (Cascading Style Sheets), o que ajudou na estilização;
- **SceneBuilder:** Foi utilizado para o desenvolvimento das interfaces gráficas, através dessa ferramenta, pudemos realizar edições de forma rápida e fácil (interface "arrasta e solta");
- **CSS:** Foi utilizado para estilizar as telas criadas, permitindo também a criação de padrões de design de forma simples;
- **PokeAPI:** Para a geração de imagens aleatórias na criação de usuário, foi utilizada a [PokeAPI](#), plataforma que fornece informações de *Pokemons* através de requisições (no caso, utilizamos os Sprites).

3.2 Demonstração da Solução

A seguir, uma demonstração da versão final aplicação *Humanize*:

3.2.1 Modulo Inicial

A Tela de Login é o ponto de partida do sistema.

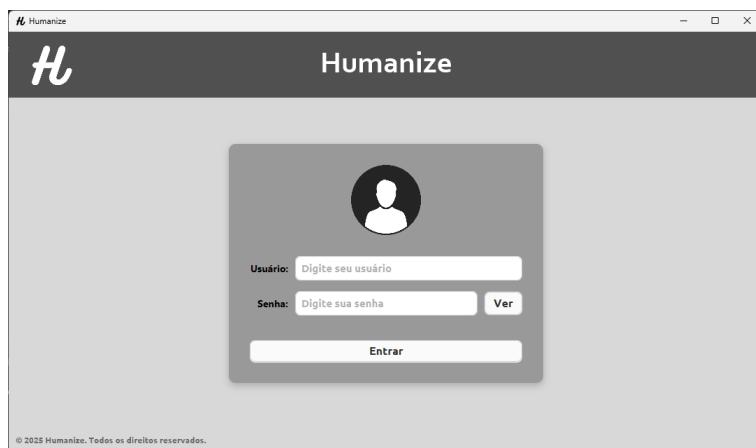


Figura 3: Tela Login

A Tela Login permite ao usuário acessar sua área de trabalho a partir de um login e uma senha. A existência desses dados é verificada no *Repository* de Usuário para liberar o acesso.



Figura 4: Tela Configurações

A tela configurações existe dentro de todos os módulos, ou seja, qualquer usuário do sistema possui acesso à ela, onde pode alterar sua senha pessoal e trocar o idioma do sistema.

3.2.2 Módulo Funcionário

O módulo funcionário é o mais geral, todos os funcionários tem acesso às telas disponíveis nesse módulo.

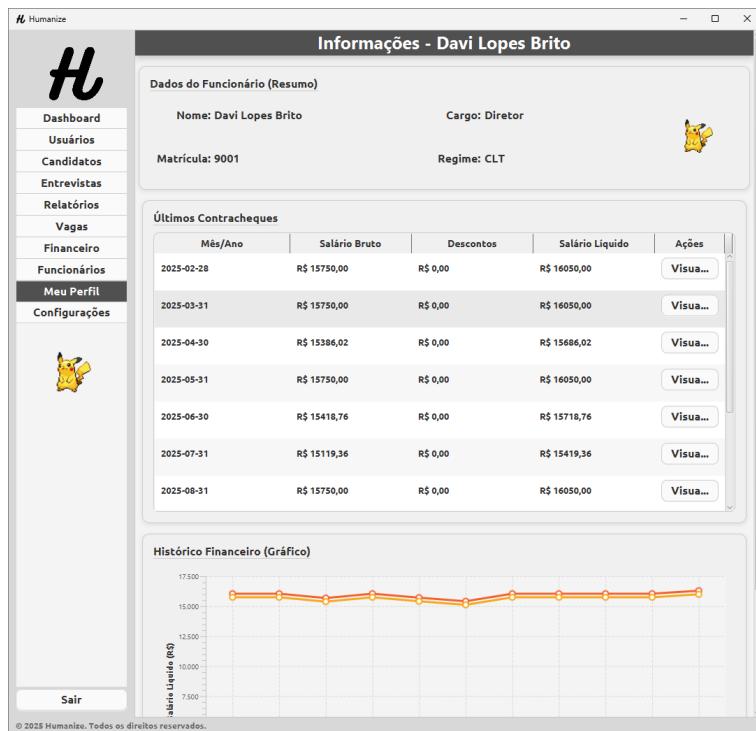


Figura 5: Tela Meu Perfil

A tela ”Meu Perfil” possui um resumo geral do usuário atual (usa a classe *UserSession*). São exibidos um histórico financeiro dos últimos recebimentos, além também de exibir os últimos contracheques pessoais:



Figura 6: Modal dos Detalhes do Contracheque

O modal é exibido quando, na tela ”Meu Perfil”, a opção ”Visualizar” em um dos contracheques é clicada.

3.2.3 Módulo Recrutador

O módulo recrutador possui apenas ferramentas a serem utilizadas pelo recrutador (*cadastrar candidato, atribuir candidato e realizar entrevista*).

Figura 7: Tela de Gestão de Candidatos

A tela ”Gestão de Candidatos” lista todos os candidatos do sistema e permite filtrá-los, além disso, é possível visualizar, editar, excluir ou criar um novo usuário a partir daqui.

Candidato

Nome Completo: Nome completo do candidato

CPF: Somente números

E-mail: email@example.com

Telefone: (XX) XXXXX-XXXX

Formação: Ex: Graduação em..., Curso Técnico...

Disponibilidade: Ex: Integral, Meio Período, Remoto

Pretensão Salari... R\$

Experiência: Descreva brevemente sua experiência relevante

Salvar

Figura 8: Tela de Cadastro de Candidato

A tela ”Cadastro de Candidatos” abre um formulário para inserção de informações da pessoa a ser cadastrada como candidato, após salvar, o candidato é guardado no sistema.

Candidatos

Candidatura à Vaga

Vagas Disponíveis

- Limpador de calha - Limpeza - R\$ 2
- Jogador de Lol - e-sports - R\$ 10.000,00
- treinador de la - T.J - R\$ 2567
- Faxineiro - Limpeza - R\$ 3000
- Advogado - Jurídico - R\$ 10000
- Uxo - lixo - R\$ 100
- Designer UI/UX - Design - R\$ 6000
- Analista de Marketing Digital - Marketing - R\$ 7500
- Gerente de Projetos - Projetos - R\$ 12000

Candidatos

- Junior - Junior@outlook.com
- Davizera - davi@gmail.com
- Vinicius Ruela - ruela.urb@gmail.com
- Ana Silva - ana.silva@hotmail.com
- Lucas Mendes - lucas.mendes@yahoo.com
- Mariana Costa - mari.costa@gmail.com
- Pedro Henrique - pedro.h@gmail.com

Cancelar **Associar Candidatura**

Figura 9: Tela Candidatura A Vagas

A tela ”Candidatura A Vaga” permite que o recrutador atribua uma vaga que a ele pertence a um dos candidatos cadastrados no sistema.

Status da Candidatura

Candidato	Cargo	Status
Davizera	Advogado	Pendente
Vinicius Ruela	treinador de ia	Em Análise
Vinicius Ruela	Jogador de LoL	Pendente
Junior	Jogador de LoL	Aprovado
Ana Silva	Analista de Marketing Digital	Em Análise
Lucas Mendes	Gerente de Projetos	Pendente
Mariana Costa	Designer UI/UX	Em Análise

© 2025 Humanize. Todos os direitos reservados.

Figura 10: Tela Status da Candidatura

A tela ”Status da Candidatura” permite que o recrutador altere ou exclua rapidamente o status de uma das candidaturas.

Alterar Status da Candidatura

Candidato: **Davizera**

Vaga: **Advogado**

Status: **Pendente**

Salvar

Figura 11: Tela Alterar Status da Candidatura

A tela ”Alterar Status da Candidatura” é o formulário que é aberto ao editar uma das candidaturas.



Figura 12: Tela Gestão e Entrevistas

A tela ”Gestão de Entrevistas” apresenta uma visão geral das entrevistas do recrutador atual, nela é possível visualizar entrevistas marcadas, relatar uma entrevista, editar, ou excluir.



Figura 13: Tela Marcar Entrevista

A tela ”Marcar Entrevista” gera uma nova entrevista, ela se limita apenas às vagas correspondentes do usuário logado.

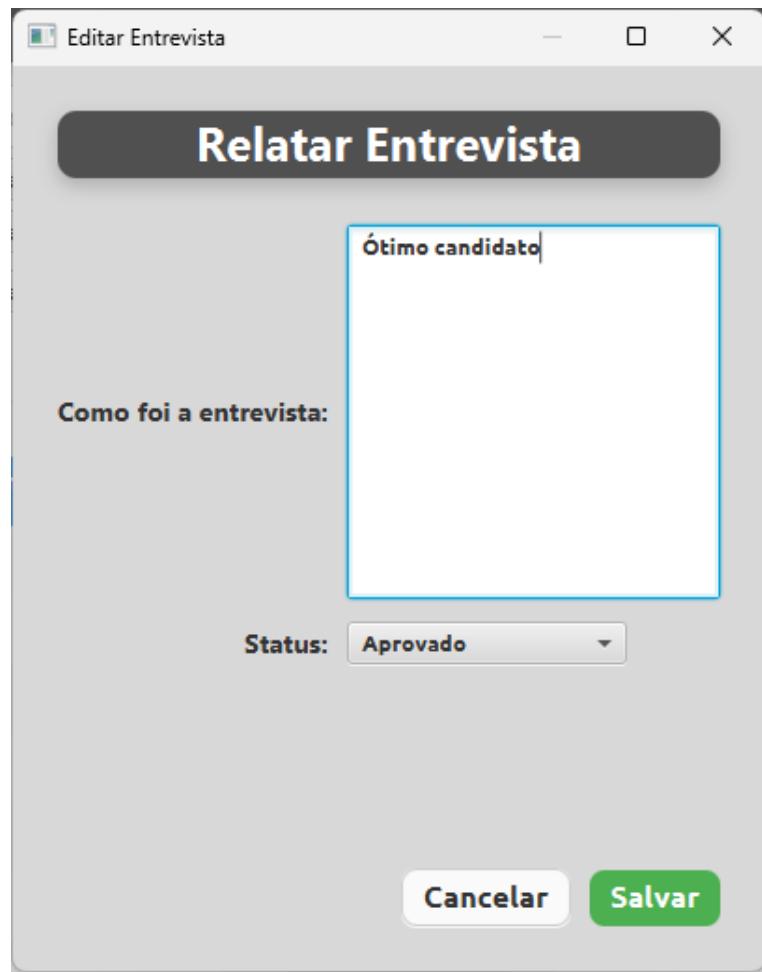


Figura 14: Tela Relatar Entrevista

A tela ”Relatar Entrevista” é para uso durante a entrevista, ela possui um campo grande de texto para o relato, além de alterar o status da entrevista ao final dela.

3.2.4 Módulo Financeiro

Um módulo com as ferramentas financeiras, apenas gestores e administradores possuem acesso a essas ferramentas.

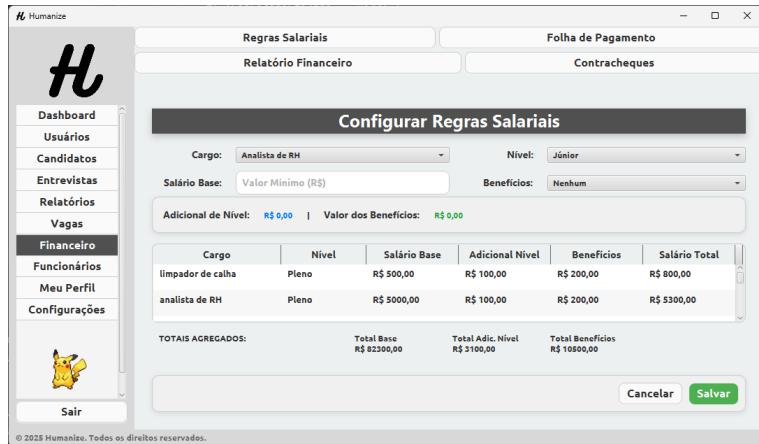


Figura 15: Tela Regras Salariais

A tela ”Regras Salariais” serve para a listagem e criação de novas regras salariais de forma rápida, em uma tela só.

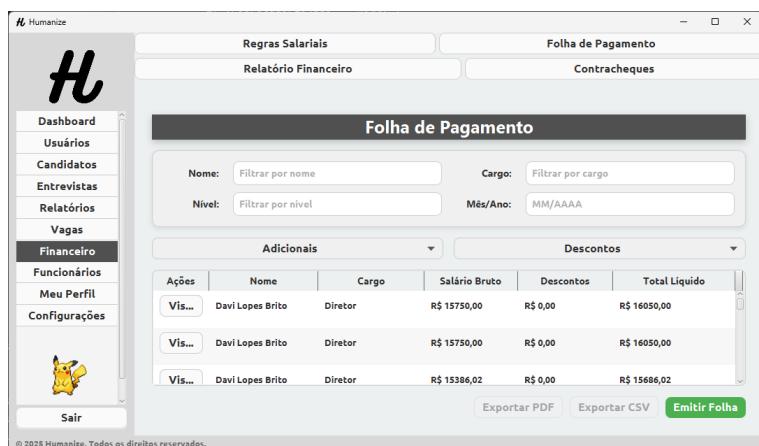


Figura 16: Tela Folha de Pagamento

A tela ”Folha de Pagamento” serve para a geração de folhas de pagamento, visualização e exportação das folhas de pagamento, centralizados em uma só tela.

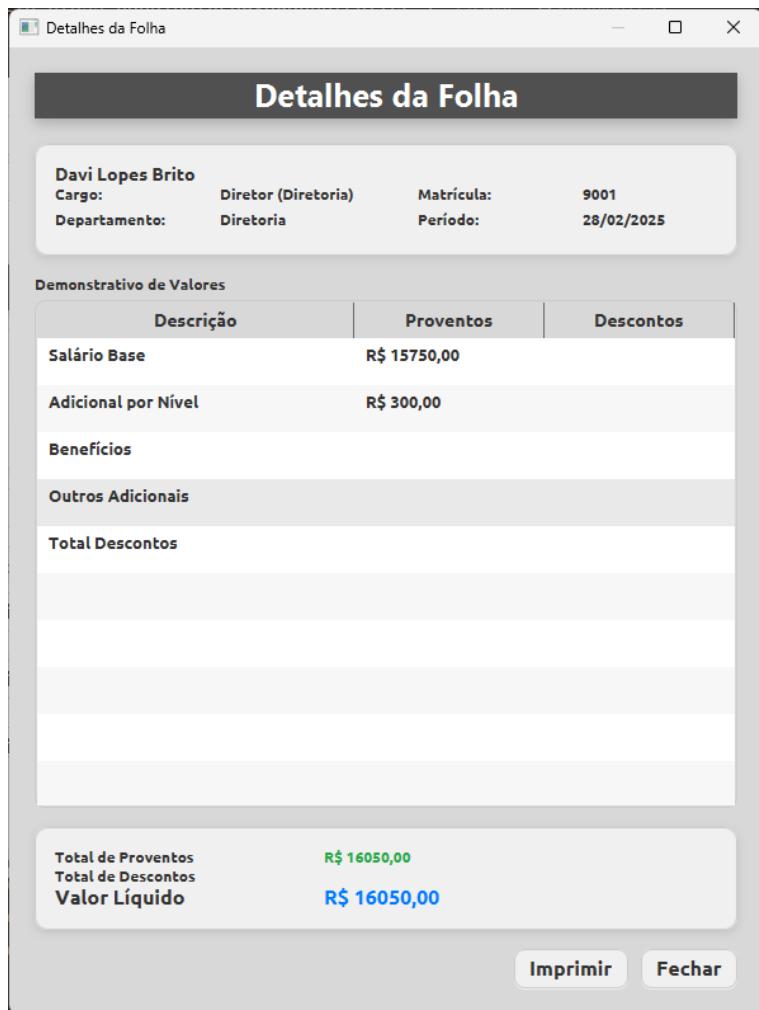


Figura 17: Tela Detalhes da Folha de Pagamento

A tela "Detalhes da Folha de Pagamento" surge ao "Visualizar" uma das folhas de pagamento, ela detalha aquela folha.



Folha de Pagamento - Davi Lopes Brito (28/02/2025)

DETALHE	PROVENTOS	DESCONTOS
Nome do Funcionário	Davi Lopes Brito	
Cargo	Diretor	
Salário Base	R\$ 15750,00	
Adicional Nível	R\$ 300,00	
Benefícios	R\$ 0,00	
Outros Adicionais/Horas	R\$ 0,00	
TOTAL BRUTO	R\$ 16050,00	
TOTAL DESCONTOS		R\$ 0,00
TOTAL LÍQUIDO	R\$ 16050,00	

Gerado em: 11/11/2025 | © 2025 Humanize. Todos os direitos reservados.

Figura 18: Relatório Folha de Pagamento

Ao imprimir a folha, ou exportar em PDF, é gerado o relatório ”Folha de Pagamento” é o relatório da folha de pagamento individual.



Figura 19: Tela Relatório Financeiro

A tela ”Relatório Financeiro” lista todas as transações realizadas pela empresa, além de possuir a opção de adicionar novas despesas ou receitas.



Relatório Financeiro Geral (Fluxo de Caixa)

DATA	DESCRIÇÃO	CATEGORIA	RECEITA	DESPESA
01/10/2025	Saldo Inicial	Inicial	R\$ 50.000,00	
01/10/2025	Venda de Serviços (Cliente A)	Vendas	R\$ 15.200,00	
10/10/2025	Pagamento de Aluguel (Escritório)	Despesas Fixas		R\$ 3.500,00
15/10/2025	Payroll - Davi Lopes Brito (Diretor)	Folha de Pagamento		R\$ 16.050,00
15/10/2025	Payroll - Bruno Silva (Analista de RH)	Folha de Pagamento		R\$ 4.700,00
20/10/2025	Receita de Investimentos	Investimentos	R\$ 500,00	
25/10/2025	Despesa com Marketing Digital	Marketing		R\$ 1.200,00
30/10/2025	Venda de Serviços (Cliente B)	Vendas	R\$ 22.800,00	
01/11/2025	Compra de Software Licença Anual	Despesas Fixas		R\$ 950,00
02/11/2025	Reembolso de Despesas (Gestor Teste)	Reembolso		R\$ 150,00
03/11/2025	Receita de Consultoria (Cliente C)	Vendas	R\$ 8.900,00	
05/11/2025	Pagamento de Energia Elétrica	Despesas Fixas		R\$ 420,50
06/11/2025	Payroll - Valquiria Machado (Diretora TI)	Folha de Pagamento		R\$ 15.100,00
07/11/2025	Venda de Novo Contrato (Cliente)	Vendas	R\$	

Figura 20: Relatório Financeiro Geral

Ao fazer a exportação do PDF, o documento ”Relatório Financeiro Geral” é gerado.

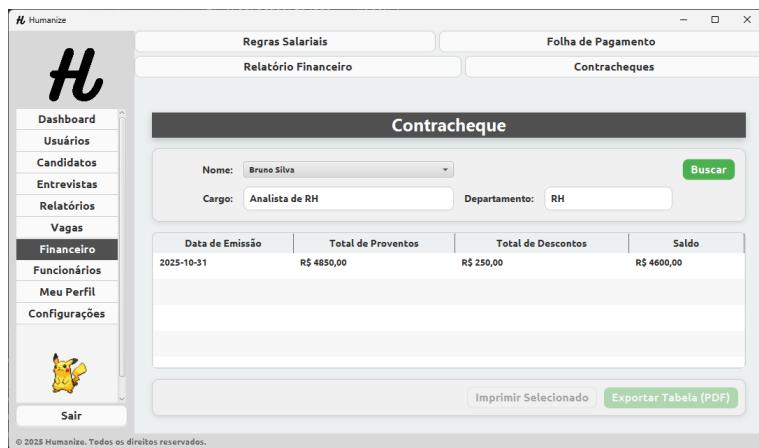


Figura 21: Tela Contracheque

A tela ”Contracheque” permite a busca e exportação do último contracheque de um usuário específico.



Contracheque - 31/10/2025

ITEM	VALOR
Nome	Admnistrador Souza
Data de Referência	31/10/2025
---	---
Outros Adicionais	R\$ 31230,15
Total Descontos	R\$ 2500,00
---	---
Salário Líquido	R\$ 28730,15

Gerado em: 12/11/2025 | © 2025 Humanize. Todos os direitos reservados.

Figura 22: Relatório Contracheque

O ”Relatório Contracheque” é gerado ao buscar um contracheque e exportá-lo.

3.2.5 Módulo Gestor

Possui acesso parcial ao sistema completo, bem similar ao Administrador com algumas restrições.

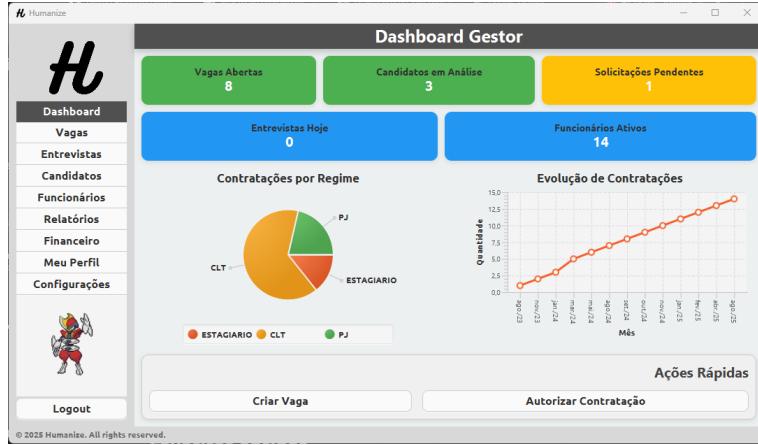


Figura 23: Tela Dashboard do Gestor

A tela ”Dashboard do Gestor” possui um grande resumo logo na entrada do sistema, são elas:

- Cards com informações rápidas sobre: Vagas abertas, candidatos em análise, solicitações (de contratação) pendentes, entrevistas hoje e funcionários ativos;
- Gráficos gerais: Contratações por regime e evolução das contratações conforme os meses do ano.
- Ações rápidas: Atalhos para facilidades: Criar Vaga e Autorizar Contratações.

The screenshot shows the 'Gestão de Vagas' (Job Management) screen. The left sidebar has a 'Vagas' section selected. The main area is titled 'Gestão de Vagas' and contains a table of job listings:

ID	Cargo	Departamento	Salário	Status	Requisitos
1	Limpador de calha	Limpeza	2	ABERTA	ser lindo
2	Jogador de LoL	e-sports	10.000,00	ABERTA	Ser Nerd
3	treinador de ia	T.I	2567	ABERTA	falar ingles
4	Faxineiro	Limpeza	3000	ABERTA	Experiencia comprovada
5	Advogado	Jurídico	10000	ABERTA	Ser prepotente
6	lixo	Lixo	100	ABERTA	lixo

At the top right, there are buttons for 'Cadastrar Nova Vaga', 'Editar', and 'Excluir'. There are also filter inputs for 'ID', 'Cargo', 'Status', and 'Salário' with a 'Filtrar' button.

Figura 24: Tela Gestão de Vagas

A tela ”Gestão de Vagas” lista todas as vagas, além de possuir a possibilidade e exclusão e criação de vagas.

Cadastro de Vaga

ID: 10

Cargo: Nome cargo

Salário: R\$ 9.999,99

Requisitos: Requisitos da Vaga

Status: ABERTA

Departamento: Nome do departamento

Recrutador: (dropdown menu)

Cancelar **Salvar**

Figura 25: Tela Cadastro de Vagas

A tela ”Cadastro de Vaga” é um formulário que é aberto ao tentar criar uma nova vaga, a partir daqui são criadas vagas e vinculadas aos recrutadores.

Status da Candidatura

Candidato	Cargo	Status
Vinicius Ruela	treinador de la	Em Análise
Junior	Jogador de LoL	Aprovado
Ana Silva	Analista de Marketing Digital	Em Análise
Mariana Costa	Designer UI/UX	Em Análise

Figura 26: Tela Candidatos do Gestor

A tela ”Candidatos” no módulo gestor exibe apenas os candidatos sob análise e os aprovados.

ID	Nome	Cargo	Perfil	Salário
5	Rec Teste	Recrutador Pleno	RECRUTADOR	6000.0
6	Gestor Teste	Gerente de Contas	GESTOR	9000.0
8	Recrutador Silva	Recrutador Sénior	RECRUTADOR	6500.0
9	Bruno Silva	Analista de RH	FUNCIONARIO	4500.0
10	Junior	Jogador de LoL	FUNCIONARIO	0.0
11	Teste	Limador de calha	FUNCIONARIO	0.0

Figura 27: Tela Gestão de Funcionários (Contratação)

A tela ”Gestão de Candidatos” lista todos os funcionários (tirando *Administradores*) para o gestor, além disso, a partir dessa tela é possível contratar candidatos aprovados.



Figura 28: Tela Aprovar e Cadastrar Funcionário

A tela ”Cadastro de Funcionário”é o ponto de virada do candidato para um funcionário efetivado, é possível selecionar o candidato aprovado e realizar sua contratação através da criação de um usuário.

ID	Tipo de Relatório:	Data de Geração	Responsável
1	LISTA_USUARIOS	2025-11-11	Davi Lopes Brito

Figura 29: Tela Relatórios

A tela ”Relatórios” lista os últimos relatórios gerados no sistema, permitindo também: visualizá-los, editá-los, excluí-los, criar novo relatório, ou exportar em PDF ou CSV.

Figura 30: Tela Gerar Relatório

A tela ”Gerar Relatório” serve para a geração de um novo relatório, aqui possuímos três deles:

- **Lista de Usuários:**



Lista Completa de Usuários

ID	NOME	EMAIL	LOGIN	PERFIL
2	Davi Lopes Brito	davilopesbrito64@gmail.com	davi	ADMINISTRADOR
3	Valquiria dos Santos Machado	valquiria.masan@gmail.com	valkiki	ADMINISTRADOR
4	Ricardo Rian da Silva Melo	rianricardo836@gmail.com	RianRSM	ADMINISTRADOR
5	Rec Teste	akaka@gmail.com	recrutador	RECRUTADOR
6	Gestor Teste	teste@gmail.com	gestor	GESTOR
7	Samara Costa	samara@gmail.com	Samara	ADMINISTRADOR
8	Recrutador Silva	rec@gmail.com	rec	RECRUTADOR
9	Bruno Silva	bruno.silva@humanize.com	funcionario	FUNCIONARIO
10	Junior		null	FUNCIONARIO
11	Teste		null	FUNCIONARIO
12	Administrador	adm@gmail.com	administrador	ADMINISTRADOR

Gerado em: 11/11/2025 | © 2025 Humanize. Todos os direitos reservados.

Figura 31: Relatório de Usuários

Relatório de todos os usuários do sistema.

- **Contracheque Geral:**

Relatório Geral de Folhas de Pagamento

NOME	DATA DE REFERÊNCIA	CARGO	SALÁRIO BASE	ADICIONAIS	DESCONTOS	SALÁRIO LÍQUIDO
Admnistrador Souza	2025-02-28	Analista de Sistemas	R\$ 4000,00	R\$ 43751,00	R\$ 248,35	R\$ 47502,65
Admnistrador Souza	2025-03-31	Analista de Sistemas	R\$ 4000,00	R\$ 9533,60	R\$ 354,88	R\$ 13178,72
Admnistrador Souza	2025-04-30	Analista de Sistemas	R\$ 4000,00	R\$ 28316,66	R\$ 56,93	R\$ 32259,73
Admnistrador Souza	2025-05-31	Analista de Sistemas	R\$ 4066,05	R\$ 31026,17	R\$ 313,44	R\$ 34778,79
Admnistrador Souza	2025-06-30	Analista de Sistemas	R\$ 4000,00	R\$ 31311,25	R\$ 337,27	R\$ 34973,98
Admnistrador Souza	2025-07-31	Analista de Sistemas	R\$ 4000,00	R\$ 22209,73	R\$ 172,59	R\$ 26037,14
Admnistrador Souza	2025-08-31	Analista de Sistemas	R\$ 3807,57	R\$ 11438,57	R\$ 313,55	R\$ 14932,60
Admnistrador Souza	2025-09-30	Analista de Sistemas	R\$ 4000,00	R\$ 24279,25	R\$ 142,64	R\$ 28136,61
Admnistrador Souza	2025-10-31	Analista de Sistemas	R\$ 4000,00	R\$ 27325,12	R\$ 94,97	R\$ 31230,15
Admnistrador Souza	2025-11-30	Analista de Sistemas	R\$ 4000,00	R\$ 24357,95	R\$ 207,13	R\$ 28150,82
Admnistrador Souza	2025-12-31	Analista de Sistemas	R\$ 4000,00	R\$ 25300,00	R\$ 200,00	R\$ 29100,00
Bruno Silva	2025-02-28	Analista de RH	R\$ 4500,00	R\$ 681,12	R\$ 57,39	R\$ 5123,74

Figura 32: Contracheque Geral

Relatório de todos os contracheques do sistema.

- **Financeiro Geral:**

Imagen em 3.2.4

Relatório financeiro geral.

3.2.6 Módulo Administrador

O módulo administrador possui *TODOS* os acessos do sistema



Figura 33: Tela Gestão de Usuários

A tela ”Gestão de Usuários” é a única tela **exclusiva** do administrador, ela exibe todos os usuários do sistema, permitindo a edição, exclusão e criação de novos usuários.

ID:	13
Perfil:	Selecionar perfil
Nome:	Nome completo do usuário
CPF:	Somente números
Endereço:	Nenhum endereço selecionado
E-mail:	exemplo@email.com
Login:	Nome de usuário para login
Senha:	Senha

Figura 34: Tela Cadastro de Usuários

A tela ”Cadastro de Usuários” permite a criação de um novo usuário no sistema.

3.3 Técnicas de programação Aplicadas

Durante o projeto, diversas técnicas de programação foram aplicadas, detalhadas a seguir:

- **Programação Orientada à Objetos:**

A Programação Orientada à Objetos foi a maior base do projeto realizado. A abstração de agentes e soluções em objetos, classes e métodos foi crucial para a realização plena de nossos planos. A seguir, detalhamos como usamos cada um dos pilares da POO e outros conceitos relacionados:

- **Polimorfismo:**

O polimorfismo permitiu que objetos de classes diferentes fossem tratados de maneira uniforme através de uma interface ou superclasse comum, tornando o código mais flexível e extensível.

Exemplo: Utilizamos polimorfismo no nosso sistema de geração de relatórios. Criamos uma interface *IGeradorRelatorio* com um método *gerar()*. Classes concretas como *RelatorioContracheque* e *RelatorioListaUsuarios* implementam essa interface. Isso permite que o *RelatorioService* chame o método *gerar()* em qualquer objeto do tipo *IGeradorRelatorio*, sem precisar saber qual tipo específico de relatório está sendo criado.

- **Herança:**

A herança foi utilizada para modelar a hierarquia de *extends* em nosso domínio, promovendo o reuso de código e estabelecendo um contrato comum entre as classes.

Exemplo: A principal aplicação da herança está na nossa estrutura de usuários. Definimos uma classe abstrata *Pessoa* (com nome, cpf). A classe *Usuario* herda de *Pessoa* (adicionando login, senha, perfil). A classe *Funcionario* herda de *Usuario* (adicionando atributos de funcionario: salario, periodo). Por fim, classes *Administrador*, *Gestor*, *Recrutador* herdaram de *Funcionario*, cada uma especializando o comportamento, mas todas compartilhando a base comum de *Pessoa*, *Usuario* e *Funcionario*.

- **Encapsulamento:**

O encapsulamento foi aplicado para proteger a integridade dos dados e ocultar a complexidade interna das classes. Todos os atributos de nossas classes de modelo (como *Vaga*, *Candidato*) foram declarados como *private*.

Exemplo: O acesso e a modificação desses atributos são controlados exclusivamente por métodos públicos (getters e setters). Na classe *Vaga*, por exemplo, o salário não pode ser alterado diretamente; ele deve ser modificado através do método *setSalario(double salario)*, onde poderíamos adicionar lógicas de validação (como garantir que o salário não seja negativo).

- **Abstração:**

A abstração foi usada para definir ”contratos” e esconder detalhes complexos de implementação, focando no que uma classe faz, em vez do como ela faz.

Exemplo: A nossa classe *BaseRepository* é um exemplo claro. Ela define um método concreto *getArquivoDePersistencia(String nomeArquivo)* que centraliza e abstrai toda a lógica de encontrar ou criar a pasta *.humanize-app-data* na home do usuário. Os repositórios concretos, como *VagaRepository* e *UsuarioRepository*, herdam dela e simplesmente usam esse método, sem precisar saber os detalhes da manipulação de *File* ou *System.getProperty("user.home")*.

– Design e Modelagem do Domínio POO:

Mais do que apenas usar os pilares da POO, foi crucial modelar o domínio do problema. O processo de design envolveu traduzir os requisitos do "Humanize" (como "Gestor aprova Vaga" ou "Recrutador agenda Entrevista") em classes, atributos e relacionamentos (como 'Vaga', 'Candidato', 'Entrevista', 'Contratacao') que representam fielmente o sistema de RH.

* Padrões de Design (Design Patterns)

Para resolver problemas comuns de design, aplicamos Padrões de Design (Design Patterns) consolidados, que funcionam como soluções testadas e aprovadas pela comunidade de desenvolvimento. <https://refactoring.guru/>

· Padrão Builder:

O padrão Builder foi usado para simplificar a criação de objetos complexos, especialmente aqueles com muitos atributos ou campos opcionais, tornando o código mais legível.

Exemplo: As classes *Funcionario* e *Candidato* possuem muitos atributos. Em vez de um construtor com 10 parâmetros, utilizamos um *Builder* interno (*FuncionarioBuilder*). Isso permite criar um objeto de forma fácil: `new Funcionario.FuncionarioBuilder().setNome("Joao").setCargo("Dev").build();`

· Padrão Singleton:

O padrão Singleton foi aplicado para garantir que existisse apenas uma única instância de classes que gerenciam um recurso compartilhado, como o acesso aos nossos arquivos de dados.

Exemplo: Todos os nossos repositórios (*VagaRepository*, *UsuarioRepository*, etc.) são Singletons. Isso assegura que todas as partes do sistema (diferentes controllers e telas) acessem a **mesma** lista de vagas e usuários em memória, evitando inconsistência de dados. O acesso é feito estaticamente: *VagaRepository.getInstance().getTodasVagas();*

• Tratamento de Exceções:

O tratamento de exceções foi fundamental para criar um software robusto. Utilizamos blocos *try-catch* para lidar com erros esperados (como falhas de I/O) e criamos exceções personalizadas para regras de negócio.

Exemplo: O uso de exceções customizadas como *LoginException* ou *SenhaIncorretaException* no *LoginService*, permitindo que o *LoginController* capture essa exceção específica e exiba uma mensagem de erro amigável ao usuário.

3.4 Desafios Encontrados e Soluções Adotadas

Inicialmente, tivemos uma dificuldade conceitual em entender a diferença entre folha de pagamento e contracheque. No primeiro relatório, confiante de que havia compreendido, acabei trocando algumas informações. Só depois percebi que uma poderia complementar a outra, o que foi um aprendizado importante.

Na parte técnica, enfrentamos desafios com o Scene Builder durante a criação das telas. Essas dificuldades, no entanto, foram facilmente resolvidas com a ajuda de um colega mais experiente. Na programação, os problemas foram menores, geralmente relacionados a tentar soluções complexas para problemas simples. Com o decorrer das aulas e a troca de conhecimentos, conseguimos sanar essas dúvidas.

Um desafio geral foi a gestão do tempo, não por falta de organização da equipe, mas devido à alta demanda de outras disciplinas, que exigiam tanto quanto este projeto.

3.5 Análise Crítica da Qualidade da Solução

3.5.1 Pontos Positivos e Fortalezas do Grupo

A comunicação entre os participantes foi excelente. Todos estavam sempre dispostos a responder dúvidas e se prontificavam para ajudar uns aos outros. Gostaria de destacar especialmente o estudante Davi, que foi um líder essencial para o grupo. Sua paciência e orientação nos guiaram durante todas as etapas, levando-nos além em termos de criatividade e interesse.

3.5.2 Sugestões de Melhoria (Autoavaliação)

Em uma avaliação geral, acreditamos que nós poderíamos ter sido mais ousados. Ficamos limitados aos exemplos da professora mais do que pretendíamos, e não nos permitimos pensar muito além, com mais criatividade e ousadia. Entretanto, em alguns tópicos, conseguimos implementar ideias extras, como a mudança de idioma do sistema, e inclusão de foto de usuário no perfil.

4 Conclusão e Evolução Futura

Para muitos de nós, foi a primeira vez participando de um projeto dessa magnitude. Foi uma forma extremamente interessante de vivenciar na prática como funciona um trabalho do gênero. Melhoramos muito nossa comunicação interna e a capacidade de explicar nossos próprios códigos e ideias. Essas habilidades, com certeza, serão de grande valia para projetos futuros.

Referências

- [1] Renato Adorno. Padrões de Commits (Commit Patterns). <https://dev.to/renatoadorno/padroes-de-commits-commit-patterns-41co>, 2020. Artigo na plataforma Dev.to.

- [2] Atlassian. Fluxo de trabalho Gitflow. <https://www.atlassian.com/br/git/tutorials/comparing-workflows/gitflow-workflow>, 2025. Tutorial de Git.
- [3] Rafaella Ballerini. Git e github para iniciantes - o guia completo. https://www.youtube.com/watch?v=Uszj_kODGsg, 2021. Vídeo do Canal Rafaella Ballerini.
- [4] Thomás da Costa. Maven: o básico para trabalhar com projetos java. <http://www.youtube.com/watch?v=J9y5Spv8WsE>, 2023. Vídeo em Curso do Prof. Thomás da Costa.
- [5] FXDocs. Documentação HTML5. <https://fxdocs.github.io/docs/html5/>, 2025. Página Web de Documentação.
- [6] Gustavo Guanabara. Curso de java para iniciantes. https://www.youtube.com/playlist?list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR, 2015. Curso em vídeo.
- [7] Robert C. (Uncle Bob) Martin. *Código Limpoo: Habilidades Práticas do Agile Software*. Alta Books, 1^a edition, 2016.
- [8] Nome do Canal ou Criador (Ajustar). Título Exato da Playlist (Ajustar). https://www.youtube.com/playlist?list=PLcoYAcR89n-qb07YAVj5S0alABLis_QVU, 2025. Playlist no YouTube.
- [9] Oracle. Documentação Oficial da Plataforma Java. <https://docs.oracle.com/en/java/>, 2025. Página Web de Documentação.
- [10] PokéAPI. PokéAPI: The RESTful Pokémon API. <https://pokeapi.co/>, 2025. Interface de Programação de Aplicações (API).
- [11] Refactoring.Guru. Refactoring.Guru. <https://refactoring.guru/>, 2025. Site Web.
- [12] Rocketseat. O que é git flow? entendendo a diferença entre branchs. <https://www.youtube.com/watch?v=-PGHFTlEF0E>, 2019. Vídeo do Canal Rocketseat.