Pró-Reitoria Acadêmica
Escola de Educação, Tecnologia e Comunicação
Curso de Bacharelado em Engenharia de Software
Trabalho de Disciplina de Teste de Software

SISTEMA DE GERENCIAMENTO DE FUNCIONÁRIOS

Autores:
Gustavo Brian Maciel Rodrigues Portela
John Robert Campos de Lima Monteiro
Izabelle Ferreira da Silva
Gustavo Camargo da Silva

Orientador: Prof. Zoe Roberto Magalhaes Junior

Brasília - DF
2025

John Robert Campos de Lima Monteiro

Izabelle Ferreira da Silva

Gustavo Camargo da Silva

SISTEMA DE GERENCIAMENTO DE FUNCIONÁRIOS

Documento apresentado ao Curso de graduação de Bacharelado em Engenharia de Software da Universidade Católica de Brasília, como requisito parcial para obtenção da aprovação na disciplina de Teste de Software.

Orientador: Profª. Zoe Roberto

Brasília
2025

Sumário

1 INTRODUÇÃO

O desenvolvimento de software não se limita à criação de funcionalidades, mas também à verificação da qualidade do produto através de testes. Nosso objetivo é desenvolver um software destinado à execução de testes, assegurando que ele cumpra os requisitos esperados e opere adequadamente em vários cenários.

Portanto, criaremos um plano de testes, no qual especificaremos seu propósito no projeto, a abrangência da aplicação, as ferramentas empregadas, as táticas de teste implementadas. Assim, nosso objetivo é garantir que o software cumpra os critérios de qualidade esperados e reduza possíveis erros antes de sua implementação.

1.1 DIAGNÓSTICO DA EMPRESA / TEMA

Muitas pequenas e médias empresas enfrentam dificuldades no gerenciamento eficaz de informações relacionadas aos seus funcionários. Processos manuais ou sistemas desatualizados resultam em falhas de registro, perda de dados, baixa produtividade e dificuldades na tomada de decisão. Dessa forma, a adoção de um sistema confiável e que facilita processos se torna essencial para modernizar e garantir a integridade das informações dos funcionários.

2 OBJETIVOS

Este capítulo detalha os **objetivos** que guiaram o desenvolvimento e a validação de um **sistema de gestão de funcionários**. Nosso foco foi criar uma solução robusta e eficiente para o controle de dados cadastrais, garantindo operações de **inserção, edição, visualização e exclusão de registros** de forma integrada e segura.

2.1 OBJETIVO GERAL

Desenvolver e testar um sistema de gestão de funcionários que permita o controle eficiente de dados cadastrais, integrando funcionalidades de inserção, edição, visualização e exclusão de registros, assegurando qualidade, usabilidade e desempenho.

2.2 OBJETIVOS ESPECÍFICOS

- Criar uma aplicação utilizando Java (Spring), JavaScript (React) e MySQL com integração via APIs RESTful;
- Estabelecer um plano de testes que garanta a conformidade dos requisitos funcionais e não funcionais;
- Aplicar estratégias de teste que reduzam falhas e garantam confiabilidade ao sistema antes da implementação;
- Fornecer uma interface acessível e intuitiva para equipes de RH e gestores;
- Automatizar o processo de gerenciamento de dados dos funcionários, minimizando erros humanos e aumentando a produtividade.

3 PROPOSTA DO SISTEMA

A seguir será apresentada a proposta do sistema, visando detalhar os principais pontos a serem seguidos.

3.1 DESCRIÇÃO DO SISTEMA PROPOSTO

O sistema proposto tem como objetivo facilitar a gestão de dados de funcionários em pequenas e médias empresas. Permitirá o cadastro, edição, visualização e exclusão de registros, garantindo segurança e integridade das informações.

A aplicação irá facilitar processos manuais, oferecendo uma solução moderna, acessível e fácil de usar. Utilizará tecnologias como Java (Spring), React e MySQL, com integração via APIs RESTful, proporcionando um ambiente eficiente e confiável para as equipes de RH, gestores e administradores de funcionários.

3.2 RESULTADOS ESPERADOS

- Redução de erros manuais no gerenciamento de dados;
- Ganho de tempo e aumento de produtividade;
- Maior confiabilidade e integridade das informações;
- Interface simples e intuitiva para facilitar o uso;
- Apoio à tomada de decisão com dados precisos e atualizados.

4 FERRAMENTAS UTILIZADAS

4.1 LINGUAGENS/FRAMEWORKS

O sistema foi desenvolvido com Java, Maven e Spring no back-end, React com JavaScript no front-end e MySQL como banco de dados.

4.2 DEPENDÊNCIAS

- **spring-boot-starter-data-jpa**
- **spring-boot-starter-validation**
- **spring-boot-starter-web**
- **mysql-connector-j**
- **spring-boot-starter-test**
- **spring-boot-starter-actuator**
- **spring-boot-starter-security**
- **io.jsonwebtoken:jjwt-api**
- **io.jsonwebtoken:jjwt-impl**

- **io.jsonwebtoken:jjwt-jackson**

5 VISÃO

5.1 INTRODUÇÃO

Este documento de visão define as necessidades de alto nível e os recursos-chave do Sistema de Gestão de Funcionários. Ele serve como uma referência fundamental para todas as partes interessadas, delineando o propósito do sistema, seus objetivos, o escopo do projeto, os usuários-alvo, a tecnologia subjacente e os critérios de qualidade esperados para o desenvolvimento e testes da solução.

5.2 FINALIDADE E OPORTUNIDADE DE NEGÓCIOS

FINALIDADE

Prover uma visão clara e compartilhada para o desenvolvimento, testes e implantação de um sistema de gestão de dados de funcionários, estabelecendo as expectativas e direcionando os esforços de toda a equipe.

OPORTUNIDADE DE NEGÓCIOS

Muitas Pequenas e Médias Empresas (PMEs) enfrentam ineficiências na gestão de dados de seus colaboradores devido a processos manuais ou sistemas desatualizados. Isso resulta em perda de dados, baixa produtividade e erros operacionais. Este projeto visa oferecer uma solução tecnológica integrada e intuitiva que promova maior eficiência operacional e reduza falhas humanas, posicionando a organização como fornecedora de soluções confiáveis e acessíveis para o mercado de PMEs.

5.3 PROBLEMA E SOLUÇÃO PROPOSTA

DESCRIÇÃO DO PROBLEMA

O gerenciamento manual ou ineficaz de informações de funcionários afeta diretamente equipes de Recursos Humanos, gestores e administradores de PMEs. O impacto dessa deficiência se manifesta em:

- **Perda de Dados e Inconsistência:** Risco de informações importantes serem perdidas ou apresentarem inconsistências.
- **Baixa Produtividade:** Processos trabalhosos e demorados que consomem tempo valioso das equipes.

- **Erros Operacionais:** Maior propensão a falhas humanas em tarefas rotineiras de cadastro e atualização.
- **Dificuldade na Tomada de Decisões:** Falta de acesso rápido e confiável a dados para análises estratégicas.

SOLUÇÃO PROPOSTA

A implementação de um sistema web automatizado, com interface amigável e segura, que permita o controle eficiente de dados cadastrais e operacionais dos funcionários. A solução focará em funcionalidades CRUD (Criar, Ler, Atualizar, Deletar), garantindo confiabilidade, desempenho e facilidade de uso para otimizar o controle de dados, reduzir erros manuais e aumentar a produtividade.

5.4 ESCOPO DO PRODUTO

O Sistema de Gestão de Funcionários será uma aplicação web autônoma e modular, acessível via navegadores modernos (Google Chrome, Firefox, Microsoft Edge), com foco em segurança (conexão HTTPS). Ele será composto por um **backend robusto** e uma **interface de usuário intuitiva**.

5.5 FUNCIONALIDADES PRINCIPAIS

- **Autenticação de Usuários:** Permite login seguro de usuários (administradores, RH) com validação de credenciais e geração de token JWT.
- **Gestão de Perfis de Acesso:** Implementação de controle de acesso baseado em papéis (RBAC - Role-Based Access Control) com ROLE_ADMIN_ROOT e ROLE_ADMIN para garantir a segurança e a segregação de privilégios.
- **Cadastro de Funcionários:** Capacidade de registrar novos funcionários com nome, cargo, matrícula e status (ativo/inativo).
- **Visualização de Funcionários:** Listagem completa de todos os funcionários e busca por ID.
- **Edição de Dados de Funcionários:** Atualização das informações de funcionários existentes.
- **Exclusão de Funcionários:** Remoção de registros de funcionários.

5.6 ESCOPO DE TESTES

O projeto inclui um plano de testes abrangente para assegurar a conformidade do sistema com os requisitos funcionais (RF) e não funcionais (RNF). Serão aplicadas estratégias de testes de unidade, integração, sistema, segurança e aceitação para garantir a confiabilidade do sistema antes da implantação.

5.7 USUÁRIOS-ALVO E ENVOLVIDOS

O sistema foi concebido para atender às necessidades específicas dos seguintes perfis:

USUÁRIOS PRINCIPAIS

- **Equipe de RH:** Profissionais encarregados de registrar, atualizar e consultar os dados dos funcionários. Necessitam de uma interface simples, eficiente e segura.
- **Gestores Operacionais:** Supervisores que consultam informações para acompanhar suas equipes e tomar decisões gerenciais.
- **Administradores de Sistema:** Responsáveis pelo controle de acesso dos usuários, suporte técnico e manutenção do sistema.

ENVOLVIDOS NO PROJETO

- **Equipe de Desenvolvimento:** Responsável pela implementação técnica, qualidade do código, execução de testes e manutenção.
- **Alta Gestão:** Interessados nos resultados estratégicos, aprovação de recursos e avaliação do impacto no negócio.

5.8 RESTRIÇÕES E SUPOSIÇÕES

- **Ambiente de Operação:** O sistema será acessado via navegadores web atualizados (Chrome, Firefox, Edge) e exigirá conexão estável à internet. A compatibilidade com dispositivos móveis (tablets, smartphones) será via interface responsiva.
- **Infraestrutura do Cliente:** As empresas usuárias devem possuir infraestrutura básica para uso de sistemas web (computadores, rede).
- **Colaboração:** A colaboração dos usuários finais (RH, gestores) é fundamental para o levantamento de requisitos detalhados e validação durante os testes de aceitação.

- **Integração:** A capacidade de integração com outros sistemas (ERPs, folha de pagamento, ponto eletrônico) dependerá da disponibilidade de APIs compatíveis por parte dos sistemas de terceiros.
- **Manutenção:** Atualizações e manutenções futuras do sistema serão responsabilidade da equipe de desenvolvimento ou fornecedor contratado.
- **Segurança da Informação:** A segurança será garantida por práticas como criptografia de senhas (BCrypt), autenticação e autorização via JWT, e práticas de backup regulares.

5.9 ARQUITETURA E TECNOLOGIAS

O sistema será desenvolvido com uma arquitetura modular, seguindo o padrão RESTful, para garantir escalabilidade e manutenibilidade.

O sistema será estruturado em pacotes, seguindo a arquitetura em camadas com as seguintes divisões principais: Service, DTO, Controller, Repository e Model.

5.10 COMPONENTES PRINCIPAIS

- **Frontend (UI):** Desenvolvido com React, provendo uma interface web responsiva e intuitiva para os usuários.
- **Backend (API):** Desenvolvido em Java utilizando o framework Spring Boot, responsável pela lógica de negócio, segurança e persistência de dados.
- **Banco de Dados:** MySQL, para armazenamento relacional dos dados de funcionários e usuários.
- **APIs RESTful:** Ponto de comunicação entre o frontend, backend e, potencialmente, outros sistemas externos.

5.11 TECNOLOGIAS E FERRAMENTAS CHAVE

- **Linguagem de Programação:** Java (JDK 17) para o Backend.
- **Frameworks Backend:** Spring Boot, Spring Security, Spring Data JPA, Hibernate Validator.
- **Ferramentas de Banco de Dados:** MySQL Server 8.0.33, MySQL Driver (JDBC).
- **Ferramentas de Teste:**

- **Unitários e Integração (Backend):** JUnit 5, Mockito, Spring Boot Test.
- **API Testing:** Postman.
- **Automação Web (UI):** Selenium WebDriver (Robot Framework).
- **Performance/Carga:** Apache JMeter.

5.12 ANÁLISE DE QUALIDADE E RISCOS

A qualidade do produto será avaliada conforme as características da ISO 25010, com foco em:

- **Funcionalidade:** Adequação, completude e corretude dos recursos (CRUD, autenticação).
- **Confiabilidade:** Maturidade e tolerância a falhas (tratamento de exceções, validação de entrada).
- **Usabilidade:** Facilidade de aprendizado e operabilidade (interfaces intuitivas, mensagens de erro claras).
- **Eficiência de Desempenho:** Tempos de resposta adequados e utilização otimizada de recursos.
- **Segurança:** Confidencialidade e integridade dos dados (criptografia, JWT, controle de acesso).
- **Manutenibilidade:** Modularidade, reusabilidade, analisabilidade e modificabilidade do código.
- **Compatibilidade e Portabilidade:** Interoperabilidade via API RESTful e adaptabilidade a diferentes ambientes de implantação.

5.13 PRINCIPAIS RISCOS IDENTIFICADOS E MITIGAÇÃO

- **R-001: Vulnerabilidades de Segurança:** Mitigado por JWT, criptografia BCrypt, validação rigorosa de entrada e uso de JPA para prevenir injeção SQL.
- **R-002: Falhas de Desempenho:** Mitigado por otimização de consultas e testes de carga/estresse.
- **R-003: Inconsistência de Dados:** Mitigado por transações de banco de dados e validação de dados em todas as camadas.

- **R-006: Defeitos em Integrações:** Mitigado por testes de integração robustos e tratamento de exceções nas APIs.

5.14 CUSTOS E PREÇOS

O modelo de comercialização do Sistema de Gestão de Funcionários será baseado em licenciamento por assinatura (SaaS), com planos mensais ou anuais que variarão conforme o porte da empresa e a quantidade de usuários ativos.

CUSTOS PREVISTOS

- **Infraestrutura de Nuvem:** Custo com servidores, banco de dados e armazenamento (se hospedado em nuvem).
- **Suporte e Manutenção:** Inclui correções de bugs, atualizações periódicas e suporte técnico.
- **Customizações sob Demanda:** Cobrança adicional para alterações específicas solicitadas pelo cliente.
- **Integrações Externas:** Possível cobrança adicional dependendo da complexidade das integrações com sistemas terceirizados.

6 ANÁLISE DE NEGÓCIO

Neste capítulo será descrito, através de diagramas e especificações, o processo do negócio em que o *software* em questão será inserido, sendo estes o diagrama do modelo de caso de uso de negócio, diagrama do modelo de classes do negócio, e, por fim, o diagrama de atividades.

6.1 REGRAS DE NEGÓCIO

Número	Nome	Descrição	Setor
RN1	Unicidade do Nome de Usuário	Um nome de usuário deve ser único no sistema.	Módulo de Autenticação e Autorização - Registro de Usuário
RN2	Rejeição de Registro por Nome de Usuário Existente	Se um nome de usuário já existir, o registro deve ser rejeitado.	Módulo de Autenticação e Autorização - Registro de Usuário
RN3	Tamanho Mínimo da Senha	A senha deve ter no mínimo 6 caracteres.	Módulo de Autenticação e Autorização - Registro de Usuário
RN4	Atribuição da Role ROLE_ADMIN_ROOT ao Primeiro Usuário	A criação do primeiro usuário no sistema automaticamente atribui a ele a role ROLE_ADMIN_ROOT.	Módulo de Autenticação e Autorização - Registro de Usuário
RN5	Restrição na Criação de Novos ROLE_ADMIN_ROOT	Após o primeiro ROLE_ADMIN_ROOT ser criado, novos usuários só podem ser registrados como ROLE_ADMIN (e não como ROLE_ADMIN_ROOT) por um usuário já autenticado com a role ROLE_ADMIN_ROOT.	Módulo de Autenticação e Autorização - Registro de Usuário
RN6	Geração de Token JWT no Registro	Ao registrar um usuário, um token JWT válido deve ser gerado e retornado.	Módulo de Autenticação e Autorização - Registro de Usuário
RN7	Credenciais Necessárias para Login	O login requer um nome de usuário e uma senha.	Módulo de Autenticação e Autorização - Login de Usuário
RN8	Validação de Credenciais no Login	As credenciais fornecidas (nome de usuário e senha) devem	Módulo de Autenticação e

Número	Nome	Descrição	Setor
		corresponder a um usuário existente no sistema.	Autorização - Login de Usuário
RN9	Rejeição de Login com Credenciais Inválidas	Em caso de credenciais inválidas (usuário não encontrado ou senha incorreta), o login deve ser rejeitado com um status de não autorizado.	Módulo de Autenticação e Autorização - Login de Usuário
RN10	Geração de Token JWT no Login	Em caso de sucesso, um token JWT válido deve ser gerado e retornado.	Módulo de Autenticação e Autorização - Login de Usuário
RN11	Permissões da ROLE_ADMIN_ROOT	ROLE_ADMIN_ROOT: Possui permissões máximas no sistema, incluindo a capacidade de criar outros usuários com a role ROLE_ADMIN.	Módulo de Autenticação e Autorização - Autorização Baseada em Papéis (Roles)
RN12	Permissões da ROLE_ADMIN	ROLE_ADMIN: Possui permissões administrativas gerais (presumivelmente para gerenciar funcionários, etc.), mas não pode criar outros ROLE_ADMIN_ROOT.	Módulo de Autenticação e Autorização - Autorização Baseada em Papéis (Roles)
RN13	Requisição de Autenticação	Todas as requisições, exceto /auth/register e /auth/login, requerem autenticação (um token JWT válido).	Módulo de Autenticação e Autorização - Autorização Baseada em Papéis (Roles)
RN14	Criptografia de Senhas	As senhas devem ser criptografadas antes de serem armazenadas no banco de dados (usando BCryptPasswordEncoder).	Módulo de Autenticação e Autorização - Gerenciamento de Senhas
RN15	Validação de Campos em Branco nos DTOs de Autenticação	Tanto o nome de usuário quanto a senha nos DTOs de LoginRequest e RegisterRequest não podem ser em branco.	Módulo de Autenticação e Autorização - Validação de Entrada (Authentication DTOs)
RN16	Campos Obrigatórios para Criação de Funcionário	Ao criar um funcionário, os campos nome, cargo, matricula e ativo são esperados.	Módulo de Gerenciamento de Funcionários - Criação de Funcionário

Número	Nome	Descrição	Setor
RN17	Geração Automática de ID para Funcionário	Um novo funcionário é criado com um ID gerado automaticamente.	Módulo de Gerenciamento de Funcionários - Criação de Funcionário
RN18	Autenticação e Autorização para Criação de Funcionário	A criação de funcionários provavelmente requer que o usuário esteja autenticado e tenha uma role permitida (ex: ROLE_ADMIN ou ROLE_ADMIN_ROOT).	Módulo de Gerenciamento de Funcionários - Criação de Funcionário
RN19	Funcionalidade de Listagem de Funcionários	É possível listar todos os funcionários registrados no sistema.	Módulo de Gerenciamento de Funcionários - Listagem de Funcionários
RN20	Autenticação para Listagem de Funcionários	A listagem de funcionários requer autenticação.	Módulo de Gerenciamento de Funcionários - Listagem de Funcionários
RN21	Funcionalidade de Busca de Funcionário por ID	É possível buscar um funcionário específico utilizando seu ID único.	Módulo de Gerenciamento de Funcionários - Busca de Funcionário por ID
RN22	Tratamento de Funcionário Não Encontrado na Busca	Se o funcionário não for encontrado, a resposta deve indicar que não foi encontrado.	Módulo de Gerenciamento de Funcionários - Busca de Funcionário por ID
RN23	Autenticação para Busca de Funcionário por ID	A busca por ID requer autenticação.	Módulo de Gerenciamento de Funcionários - Busca de Funcionário por ID
RN24	Funcionalidade de Atualização de Funcionário	É possível atualizar os dados de um funcionário existente através de seu ID.	Módulo de Gerenciamento de Funcionários -

Número	Nome	Descrição	Setor
			Atualização de Funcionário
RN25	Tratamento de Funcionário Não Encontrado na Atualização	Se o funcionário a ser atualizado não for encontrado, a operação deve falhar (retornar "não encontrado").	Módulo de Gerenciamento de Funcionários - Atualização de Funcionário
RN26	Autenticação para Atualização de Funcionário	A atualização de funcionários requer autenticação.	Módulo de Gerenciamento de Funcionários - Atualização de Funcionário
RN27	Funcionalidade de Exclusão de Funcionário	É possível deletar um funcionário do sistema utilizando seu ID.	Módulo de Gerenciamento de Funcionários - Exclusão de Funcionário
RN28	Autenticação para Exclusão de Funcionário	A exclusão de funcionários requer autenticação.	Módulo de Gerenciamento de Funcionários - Exclusão de Funcionário

7 ANÁLISE DE REQUISITOS

Nesta seção, serão detalhados os **requisitos funcionais** do Sistema de Gerenciamento de Funcionários. Estes requisitos descrevem as funcionalidades específicas que o sistema deve executar para atender às necessidades dos usuários e aos objetivos de negócio.

7.1 REQUISITOS FUNCIONAIS

Nº	Requisito Funcional	Origem (RN)	
RF001	O sistema deve permitir que usuários registrados (administradores) façam login.	RN7, RN8, RN9, RN10	
RF002	O sistema deve permitir o cadastro de novos funcionários com nome, cargo, matrícula e status.	RN16, RN17, RN18	
RF003	O sistema deve permitir que usuários autorizados visualizem uma lista de funcionários.	RN19, RN20	
RF004	O sistema deve permitir a atualização dos dados de funcionários existentes.	RN24, RN25, RN26	
RF005	O sistema deve permitir que funcionários sejam removidos do cadastro.	RN27, RN28	
RF006		O sistema deve permitir o registro de novos usuários com papéis específicos.	RN1, RN2, RN3, RN4, RN5, RN6, RN11, RN12, RN13, RN14, RN15

7.2 REQUISITOS NÃO-FUNCIONAIS

Nesta seção, serão detalhados os **requisitos não funcionais** do Sistema de Gerenciamento de Funcionários. Estes requisitos definem os critérios de qualidade e as restrições operacionais que o sistema deve satisfazer para garantir sua eficácia e adequação, indo além das funcionalidades explícitas.

Nº	Requisito Não Funcional	Origem (Descrição no Texto)	
RNF001		O sistema deve garantir a segurança dos dados e do	RN: 1.1 - Token JWT, Criptografia de Senhas

	acesso, utilizando JWT, papéis de acesso e senhas criptografadas.	(BCrypt), Autorização por Papéis
RNF002	A interface deve ser intuitiva, consistente e de fácil navegação.	Declaração geral de usabilidade no item "Requisitos Não Funcionais"
RNF003	O sistema deve operar de forma confiável, sem falhas inesperadas e com tratamento adequado de erros.	Menção a tratamento de erro em "não encontrado", e falhas de comunicação com o BD, validação, etc.
RNF004	O desempenho deve ser eficiente, com respostas rápidas e suporte a múltiplas requisições simultâneas.	Requisito explícito em "Eficiência de Desempenho"
RNF005	O código deve ser modular, legível e bem documentado para facilitar manutenção e evolução.	Requisito explícito em "Manutenibilidade"
RNF006	O sistema deve ser compatível com os principais navegadores e com a versão definida do MySQL.	Requisito explícito em "Compatibilidade"

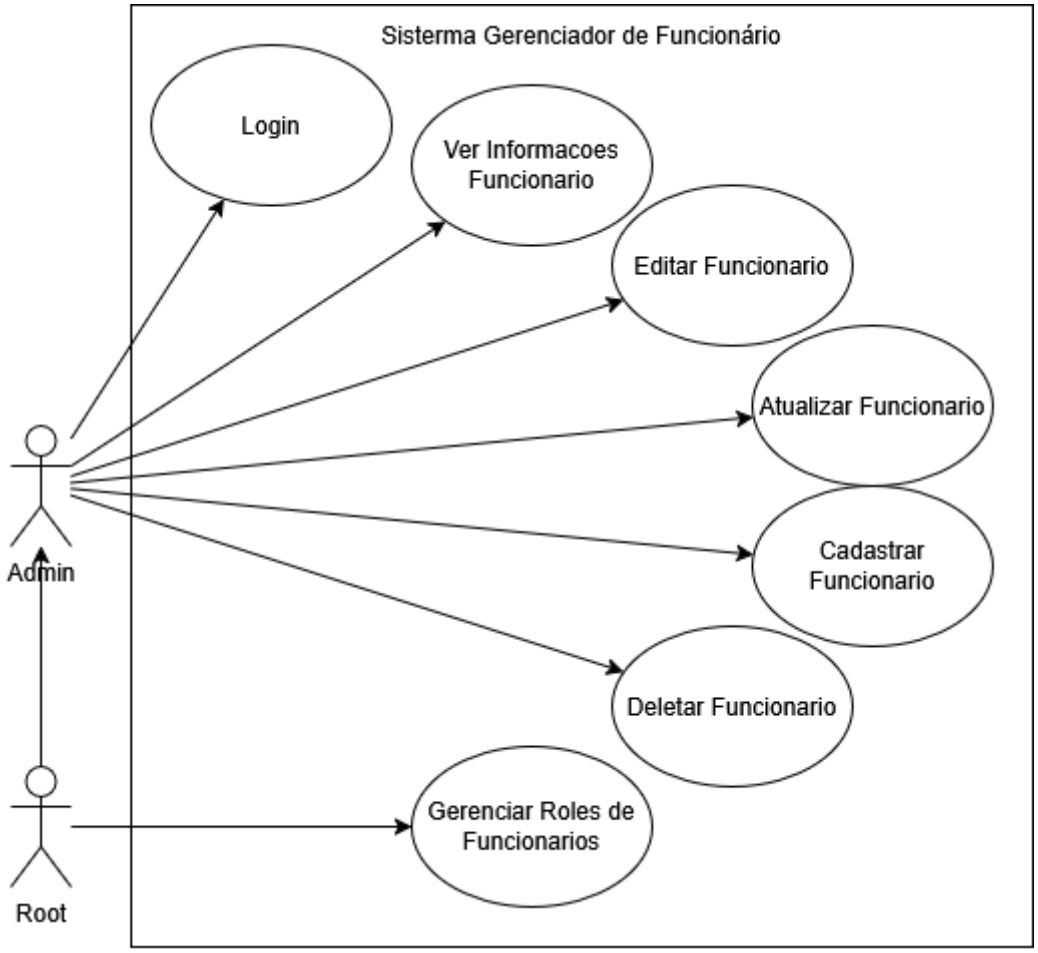
7.3 DIAGRAMA DE CASOS DE USO DA SOLUÇÃO

Nesta seção serão definidos os modelos de casos de uso. Primeiramente será mostrada uma visão geral dos casos de uso que definem as funcionalidades do sistema, com seus respectivos atores.

VISÃO GERAL DOS CASOS DE USO E ATORES

A Figura 1 a seguir será apresentado o Diagrama de Casos de Uso de *Software* com a visão de cada ator do sistema, root e admin, abrangendo assim todas as funcionalidades previstas para a implementação.

Figura 1 - Diagrama de Casos de Uso de Software.



8 HISTÓRIAS DE USUÁRIO

HU-001 REGISTRO DE NOVOS USUÁRIOS

Campo	Conteúdo
Título	Registro de Novos Usuários do Sistema
Descrição	O sistema deve permitir que novos usuários (administradores) sejam registrados, para que possam acessar e gerenciar o sistema.
Critérios de Aceitação	1. O nome de usuário (username) deve ser único no sistema (RN1). 2. Se um nome de usuário já existir, o registro deve ser rejeitado (RN2). 3. A senha deve ter no mínimo 6 caracteres (RN3). 4. A criação do primeiro usuário no sistema deve atribuir a ele a role ROLE_ADMIN_ROOT automaticamente (RN4). 5. Após o primeiro ROLE_ADMIN_ROOT ser criado, novos usuários só podem ser registrados como ROLE_ADMIN (e não como ROLE_ADMIN_ROOT) por um usuário já autenticado com a role ROLE_ADMIN_ROOT (RN5). 6. Um token JWT válido deve ser gerado e retornado ao registrar um usuário com sucesso (RN6).

Campo	Conteúdo
	7. As senhas devem ser criptografadas (usando BCrypt) antes de serem armazenadas (RN14). 8. O nome de usuário e a senha nos DTOs de registro não podem ser em branco (RN15).
Regras de Negócio	1. RN1: Unicidade do Nome de Usuário - Um nome de usuário deve ser único no sistema. 2. RN2: Rejeição de Registro por Nome de Usuário Existente - Se um nome de usuário já existir, o registro deve ser rejeitado. 3. RN3: Tamanho Mínimo da Senha - A senha deve ter no mínimo 6 caracteres. 4. RN4: Atribuição da Role ROLE_ADMIN_ROOT ao Primeiro Usuário - A criação do primeiro usuário no sistema automaticamente atribui a ele a role ROLE_ADMIN_ROOT. 5. RN5: Restrição na Criação de Novos ROLE_ADMIN_ROOT - Após o primeiro ROLE_ADMIN_ROOT ser criado, novos usuários só podem ser registrados como ROLE_ADMIN (e não como ROLE_ADMIN_ROOT) por um usuário já autenticado com a role ROLE_ADMIN_ROOT. 6. RN6: Geração de Token JWT no Registro - Ao registrar um usuário, um token JWT válido deve ser gerado e retornado. 7. RN11: Permissões da ROLE_ADMIN_ROOT - Possui permissões máximas no sistema, incluindo a capacidade de criar outros usuários com a role ROLE_ADMIN. 8. RN12: Permissões da ROLE_ADMIN - Possui permissões administrativas gerais (presumivelmente para gerenciar funcionários, etc.), mas não pode criar outros ROLE_ADMIN_ROOT. 9. RN14: Criptografia de Senhas - As senhas devem ser criptografadas antes de serem armazenadas no banco de dados (usando BCryptPasswordEncoder). 10. RN15: Validação de Campos em Branco nos DTOs de Autenticação - Tanto o nome de usuário quanto a senha nos DTOs de RegisterRequest não podem ser em branco.
User Story	Como Administrador de Sistema, eu quero registrar novos usuários com papéis (ROLE_ADMIN), para que eu possa controlar quem tem acesso.
Cenários BDD	Cenário 1: Registro do primeiro usuário (ROLE_ADMIN_ROOT)

Campo	Conteúdo
	• Dado que não há nenhum usuário cadastrado no sistema, • Quando eu envio uma requisição POST para /auth/register com username: "adminRoot", password: "senhaSegura123", • Então o usuário "adminRoot" deve ser cadastrado com a role ROLE_ADMIN_ROOT, e um token JWT deve ser retornado. Cenário 2: Registro de novo usuário (ROLE_ADMIN) por um ROLE_ADMIN_ROOT • Dado que um ROLE_ADMIN_ROOT está autenticado, • Quando o ROLE_ADMIN_ROOT envia uma requisição POST para /auth/register com username: "novoAdmin", password: "senhaNova123", • Então o usuário "novoAdmin" deve ser cadastrado com a role ROLE_ADMIN, e um token JWT deve ser retornado. Cenário 3: Tentativa de registro com nome de usuário existente • Dado que o usuário "usuarioExistente" já está cadastrado, • Quando eu tento registrar um novo usuário com username: "usuarioExistente", • Então o registro deve ser rejeitado com uma mensagem indicando que o nome de usuário já existe. Cenário 4: Tentativa de registro com senha muito curta • Dado que o sistema está em execução, • Quando eu tento registrar um novo usuário com username: "usuarioTeste" e password: "abc", • Então o registro deve ser rejeitado com uma mensagem indicando que a senha é muito curta.
Prioridade	Essencial para o gerenciamento e segurança do sistema.
Tarefas Técnicas	1. Criar endpoint de registro (/auth/register) no Backend (Spring Boot). 2. Implementar validação de unicidade do nome de usuário. 3. Criptografar senhas usando BCryptPasswordEncoder. 4. Desenvolver lógica para atribuição de roles (ROLE_ADMIN_ROOT, ROLE_ADMIN) baseada nas regras de negócio (primeiro usuário, permissões do criador). 5. Gerar e retornar token JWT após o registro bem-sucedido. 6. Implementar validação para campos em branco no DTO de registro.

Campo	Conteúdo
Dependências	Módulo de Autenticação e Autorização, Banco de Dados (MySQL) para armazenamento de usuários e roles, Spring Security para controle de acesso.
Definição de Pronto	• Código do endpoint de registro de usuário implementado e revisado. • Testes unitários e de integração (com Postman) para todas as regras de negócio de registro passando. • Cenários BDD de registro automatizados e passando. • Documentação da API de registro de usuário atualizada.

HU-002 LOGIN DE USUÁRIOS

Campo	Conteúdo
Título	Login de Usuário
Descrição	Permitir que usuários registrados (administradores) façam login de forma segura no sistema, validando suas credenciais de "username" e "password" e gerando um token JWT, para que possam acessar as funcionalidades de gestão de funcionários.
Critérios de Aceitação	1. Usuários com credenciais válidas (username e password) devem conseguir fazer login com sucesso e receber um token JWT (RF001, RN7, RN8, RN10). 1. Em caso de credenciais inválidas (username não encontrado ou password incorreto), o login deve ser rejeitado com um status de não autorizado (RN9). 2. Tanto o nome de usuário quanto a senha não podem ser em branco (RN15).
Regras de Negócio	• RN7: Credenciais Necessárias para Login - O login requer um nome de usuário e uma senha. • RN8: Validação de Credenciais no Login - As credenciais fornecidas (nome de usuário e senha) devem corresponder a um usuário existente no sistema. • RN9: Rejeição de Login com Credenciais Inválidas - Em caso de credenciais inválidas (usuário não encontrado ou senha incorreta), o login deve ser rejeitado com um status de não autorizado. • RN10: Geração de Token JWT no Login - Em caso de sucesso, um token JWT válido deve ser gerado e retornado. • RN15: Validação de Campos em Branco nos DTOs de Autenticação - Tanto o nome de usuário quanto a senha nos DTOs de LoginRequest não podem ser em branco. • RNF001: O sistema deve garantir a segurança dos dados e do acesso, utilizando JWT, papéis de acesso e senhas criptografadas.

Campo	Conteúdo
User Story	Como Usuário (Administrador), eu quero fazer login no sistema com minhas credenciais, para que eu possa acessar as funcionalidades de gestão de funcionários de forma segura.
Cenários BDD	Cenário 1: Login com credenciais válidas • Dado que o sistema está em execução e um usuário (username: "usuario_existente", password: "senhaCorreta123") está previamente cadastrado no banco de dados, • Quando eu envio uma requisição POST para {{baseURL}}/auth/login com username: "usuario_existente" e password: "senhaCorreta123", • Então o Status da Resposta deve ser 200 OK, e o Corpo da Resposta deve conter um token JWT válido, conforme AuthResponse.java. Cenário 2: Login com nome de usuário inexistente • Dado que o sistema está em execução, • Quando eu tento fazer login com username: "usuario_inexistente" e password: "qualquerSenha", • Então o login deve ser rejeitado com um status de não autorizado. Cenário 3: Login com senha incorreta • Dado que o sistema está em execução e o usuário "usuario_existente" está cadastrado, • Quando eu tento fazer login com username: "usuario_existente" e password: "senhaIncorreta", • Então o login deve ser rejeitado com um status de não autorizado. Cenário 4: Login com campos em branco • Dado que o sistema está em execução, • Quando eu tento fazer login com username: "" e password: "senha", • Então o login deve ser rejeitado com um erro de validação (status de requisição inválida).
Prioridade	Essencial para o acesso inicial e segurança do sistema
Tarefas Técnicas	• Implementar o endpoint /auth/login no Backend (Spring Boot). • Integrar com o módulo de segurança (Spring Security) para validação de credenciais.

Campo	Conteúdo
	• Consultar o banco de dados para verificar a existência do usuário e a correspondência da senha criptografada. • Gerar e assinar o token JWT após autenticação bem-sucedida. • Configurar a infraestrutura para aceitar e validar tokens JWT em requisições subsequentes. • Implementar tratamento de exceções para credenciais inválidas e campos em branco.
Dependências	• Módulo de Autenticação e Autorização. • Banco de Dados (MySQL) para armazenamento de usuários e senhas criptografadas. • Biblioteca de JWT (ex: jjwt). • Spring Security.
Definição de Pronto	1. Código do endpoint de login implementado, testado unitariamente e revisado. 2. Testes de integração (com Postman) cobrindo todos os cenários de sucesso e falha de login passando. 3. Cenários BDD "Login com credenciais válidas", "Login com nome de usuário inexistente", "Login com senha incorreta" e "Login com campos em branco" automatizados e passando. 4. Documentação da API de login atualizada e clara.

HU-003 CADASTRO DE FUNCIONÁRIOS

Campo	Conteúdo
Título	Cadastro de Novo Funcionário
Descrição	Permitir o registro de novos funcionários no sistema, coletando informações essenciais como nome, cargo, matrícula e status (ativo/inativo), para manter a base de dados de colaboradores atualizada e acessível.
Critérios de Aceitação	• O sistema deve permitir o cadastro de funcionários com nome, cargo, matrícula e status (ativo/inativo) (conforme RF002, RN16). • Um ID deve ser gerado automaticamente para cada novo funcionário (conforme RN17). • A criação de funcionários requer que o usuário esteja autenticado e tenha uma role permitida (ex: ROLE_ADMIN ou ROLE_ADMIN_ROOT) (conforme RN18). • Os campos nome, cargo, matricula e ativo são esperados e não devem ser em branco.

Campo	Conteúdo
Regras de Negócio	• RN16: Ao criar um funcionário, os campos nome, cargo, matricula e ativo são esperados. • RN17: Um novo funcionário é criado com um ID gerado automaticamente. • RN18: A criação de funcionários provavelmente requer que o usuário esteja autenticado e tenha uma role permitida (ex: ROLE_ADMIN ou ROLE_ADMIN_ROOT).
User Story	Como Equipe de RH ou Administrador, eu quero registrar novos funcionários com seus dados essenciais, para que eu possa manter o registro de todos os colaboradores atualizado no sistema.
Cenários BDD	Cenário 1: Cadastro de funcionário com dados válidos • Dado que eu estou autenticado como um usuário com permissão de ROLE_ADMIN, • Quando eu envio uma requisição POST para o endpoint de criação de funcionários com nome: "João Silva", cargo: "Desenvolvedor", matricula: "12345", status: "ativo", • Então o funcionário "João Silva" deve ser criado com sucesso e um ID único deve ser gerado e retornado. Cenário 2: Tentativa de cadastro de funcionário sem autenticação • Dado que eu não estou autenticado no sistema, • Quando eu tento enviar uma requisição POST para cadastrar um novo funcionário, • Então a requisição deve ser rejeitada com um status de erro de autenticação (não autorizado). Cenário 3: Tentativa de cadastro de funcionário com campos obrigatórios em branco • Dado que eu estou autenticado como um usuário com permissão de ROLE_ADMIN, • Quando eu envio uma requisição POST para o endpoint de criação de funcionários com um campo obrigatório (ex: nome) em branco, • Então o sistema deve rejeitar o cadastro com uma mensagem de erro indicando o campo inválido.
Prioridade	Essencial para a funcionalidade principal do sistema de gerenciamento de funcionários.

Campo	Conteúdo
Tarefas Técnicas	• Criar endpoint POST /funcionarios no Backend. • Implementar a lógica de negócio para persistir os dados do novo funcionário no banco de dados (MySQL). • Gerar o ID do funcionário automaticamente antes da persistência. • Garantir a validação dos campos nome, cargo, matrícula e ativo. • Implementar o controle de acesso (RBAC) para garantir que apenas usuários autorizados possam criar funcionários.
Dependências	• Módulo de Gerenciamento de Funcionários (Backend). • Banco de Dados MySQL para armazenamento de dados de funcionários. • Módulo de Autenticação e Autorização para verificação de permissões.
Definição de Pronto	• Código do endpoint de criação de funcionário implementado e revisado. • Testes unitários para a lógica de negócio de criação de funcionário passando. • Testes de integração (com Postman) para o endpoint de criação de funcionário passando. • Cenários BDD de cadastro de funcionário automatizados e passando. • Documentação da API de criação de funcionário atualizada.

HU-004 VISUALIZAÇÃO E BUSCA DE FUNCIONÁRIOS

Campo	Conteúdo
Título	Visualização e Busca de Funcionários.
Descrição	Oferecer a capacidade de listar todos os funcionários registrados no sistema e permitir a busca por um funcionário específico utilizando seu ID único, otimizando a consulta de informações para usuários autorizados.
Critérios de Aceitação	• O sistema deve exibir uma lista completa de todos os funcionários (RF003, RN19). • Deve ser possível buscar um funcionário específico pelo seu ID (RN21). • Se um funcionário não for encontrado pela busca de ID, a resposta deve indicar que não foi encontrado (RN22). • A listagem e busca de funcionários requerem autenticação (RN20, RN23).
Regras de Negócio	• RN19: Funcionalidade de Listagem de Funcionários - É possível listar todos os funcionários registrados no sistema. • RN20: Autenticação para Listagem de Funcionários - A listagem de funcionários requer autenticação.

Campo	Conteúdo
	• RN21: Funcionalidade de Busca de Funcionário por ID - É possível buscar um funcionário específico utilizando seu ID único. • RN22: Tratamento de Funcionário Não Encontrado na Busca - Se o funcionário não for encontrado, a resposta deve indicar que não foi encontrado. • RN23: Autenticação para Busca de Funcionário por ID - A busca por ID requer autenticação.
User Story	Como Equipe de RH ou Gestor Operacional, eu quero visualizar a lista completa de funcionários e buscar por um funcionário específico pelo ID, para que eu possa consultar rapidamente as informações dos colaboradores.
Cenários BDD	Cenário 1: Listar todos os funcionários com sucesso • Dado que eu estou autenticado no sistema e existem funcionários cadastrados, • Quando eu envio uma requisição GET para o endpoint de listagem de funcionários, • Então o sistema deve retornar uma lista completa de todos os funcionários cadastrados. Cenário 2: Buscar funcionário por ID existente • Dado que eu estou autenticado no sistema e o funcionário com ID "1" existe, • Quando eu envio uma requisição GET para /funcionarios/1, • Então o sistema deve retornar os dados completos do funcionário com ID "1". Cenário 3: Buscar funcionário por ID inexistente • Dado que eu estou autenticado no sistema, • Quando eu envio uma requisição GET para /funcionarios/ID_INEXISTENTE, • Então o sistema deve retornar uma mensagem indicando que o funcionário não foi encontrado.
Prioridade	Essencial para a consulta e gestão diária de funcionários.
Tarefas Técnicas	• Criar endpoints GET /funcionarios (para listar todos) e GET /funcionarios/{id} (para buscar por ID) no Backend.

Campo	Conteúdo
	• Implementar a lógica de recuperação de dados do banco de dados (MySQL). • Garantir tratamento adequado para casos em que o funcionário não é encontrado. • Aplicar o controle de acesso (autenticação) para ambas as operações.
Dependências	• Módulo de Gerenciamento de Funcionários (Backend). • Banco de Dados MySQL. • Módulo de Autenticação e Autorização para validação de acesso.
Definição de Pronto	1. Código dos endpoints de visualização e busca de funcionários implementado e revisado. 2. Testes unitários e de integração (com Postman) para listar e buscar funcionários passando. 3. Cenários BDD de visualização e busca automatizados e passando. 4. Documentação da API de visualização e busca de funcionário atualizada.

HU-005 ATUALIZAÇÃO DE DADOS DE FUNCIONÁRIOS

Campo	Conteúdo
Título	Atualização de Dados de Funcionários.
Descrição	Habilitar a atualização dos dados de um funcionário existente por meio de seu ID, permitindo a correção ou modificação de informações (nome, cargo, matrícula, status) conforme a necessidade.
Critérios de Aceitação	• Deve ser possível atualizar os dados de um funcionário existente através de seu ID (RF004, RN24). • Se o funcionário a ser atualizado não for encontrado, a operação deve falhar e retornar uma mensagem de "não encontrado" (RN25). • A atualização de funcionários requer autenticação (RN26).
Regras de Negócio	• RN24: Funcionalidade de Atualização de Funcionário - É possível atualizar os dados de um funcionário existente através de seu ID. • RN25: Tratamento de Funcionário Não Encontrado na Atualização - Se o funcionário a ser atualizado não for encontrado, a operação deve falhar (retornar "não encontrado"). • RN26: Autenticação para Atualização de Funcionário - A atualização de funcionários requer autenticação.
User Story	Como Equipe de RH ou Administrador, eu quero atualizar os dados de um funcionário existente, para que eu possa corrigir ou modificar suas informações conforme necessário.

Campo	Conteúdo
Cenários BDD	Cenário 1: Atualizar dados de funcionário existente com sucesso • Dado que eu estou autenticado no sistema e o funcionário com ID "FUNC002" existe com nome: "Maria", cargo: "Analista", • Quando eu envio uma requisição PUT para /funcionarios/FUNC002 com nome: "Maria Silva", cargo: "Analista Sênior", • Então os dados do funcionário com ID "FUNC002" devem ser atualizados para "Maria Silva" e "Analista Sênior" no sistema. Cenário 2: Tentar atualizar funcionário inexistente • Dado que eu estou autenticado no sistema, • Quando eu envio uma requisição PUT para /funcionarios/ID_INEXISTENTE com dados para atualização, • Então o sistema deve retornar uma mensagem de erro indicando que o funcionário não foi encontrado. Cenário 3: Tentar atualizar funcionário sem autenticação • Dado que eu não estou autenticado no sistema, • Quando eu tento enviar uma requisição PUT para atualizar um funcionário, • Então a requisição deve ser rejeitada por falta de autenticação (não autorizado).
Prioridade	Importante para a manutenção da base de dados de funcionários.
Tarefas Técnicas	• Criar endpoint PUT /funcionarios/{id} no Backend. • Implementar a lógica para localizar o funcionário pelo ID e persistir as atualizações no banco de dados. • Garantir tratamento adequado para casos em que o funcionário a ser atualizado não é encontrado. • Aplicar o controle de acesso (autenticação) para a operação.
Dependências	• Módulo de Gerenciamento de Funcionários (Backend). • Banco de Dados MySQL. • Módulo de Autenticação e Autorização para validação de acesso.
Definição de Pronto	• Código do endpoint de atualização de funcionário implementado e revisado. • Testes unitários e de integração (com Postman) para atualização de funcionário passando. • Cenários BDD de atualização automatizados e passando.

Campo	Conteúdo
	• Documentação da API de atualização de funcionário atualizada.

HU-006 EXCLUSÃO DE FUNCIONÁRIO

Campo	Conteúdo
Título	Exclusão de Funcionário.
Descrição	Possibilitar a remoção de um funcionário do sistema utilizando seu ID único, visando manter a base de dados limpa e atualizada, removendo registros de ex-colaboradores.
Critérios de Aceitação	• Deve ser possível deletar um funcionário do sistema utilizando seu ID (RF005, RN27). • A exclusão de funcionários requer autenticação (RN28). • Se o funcionário a ser excluído não for encontrado, a operação deve indicar que não foi encontrado.
Regras de Negócio	• RN27: Funcionalidade de Exclusão de Funcionário - É possível deletar um funcionário do sistema utilizando seu ID. • RN28: Autenticação para Exclusão de Funcionário - A exclusão de funcionários requer autenticação.
User Story	Como Administrador, eu quero remover um funcionário do cadastro do sistema, para que eu possa manter a base de dados limpa e atualizada, removendo registros de ex-colaboradores.
Cenários BDD	Cenário 1: Excluir funcionário existente com sucesso • Dado que eu estou autenticado como um usuário com permissão de ROLE_ADMIN_ROOT ou ROLE_ADMIN, e o funcionário com ID "FUNC003" existe no sistema, • Quando eu envio uma requisição DELETE para /funcionarios/FUNC003, • Então o funcionário com ID "FUNC003" deve ser removido do sistema com sucesso e o sistema deve retornar um status de sucesso. Cenário 2: Tentar excluir funcionário inexistente • Dado que eu estou autenticado no sistema, • Quando eu envio uma requisição DELETE para /funcionarios/ID_INEXISTENTE, • Então o sistema deve retornar uma mensagem de erro indicando que o funcionário não foi encontrado. Cenário 3: Tentar excluir funcionário sem autenticação • Dado que eu não estou autenticado no sistema,

Campo	Conteúdo
	• Quando eu tento enviar uma requisição DELETE para excluir um funcionário, • Então a requisição deve ser rejeitada por falta de autenticação (não autorizado).
Prioridade	Importante para a gestão do ciclo de vida dos funcionários no sistema.
Tarefas Técnicas	• Criar endpoint DELETE /funcionarios/{id} no Backend. • Implementar a lógica para remover o registro do funcionário do banco de dados com base no ID. • Garantir tratamento adequado para casos em que o funcionário a ser excluído não é encontrado. • Aplicar o controle de acesso (autenticação) para a operação de exclusão.
Dependências	• Módulo de Gerenciamento de Funcionários. • Banco de Dados MySQL. • Módulo de Autenticação e Autorização para validação de acesso.
Definição de Pronto	• Código do endpoint de exclusão de funcionário implementado e revisado. • Testes unitários e de integração (com Postman) para exclusão de funcionário passando. • Cenários BDD de exclusão automatizados e passando. • Documentação da API de exclusão de funcionário atualizada.

9 MODELAGEM DO SISTEMA

9.1 METODOLOGIA DE DESIGN DE CLASSES

A modelagem das classes do sistema foi guiada pela metodologia orientada a objetos (OO), buscando a criação de um design modular, coeso e de baixo acoplamento. A definição das classes partiu da análise dos requisitos funcionais e não funcionais levantados, identificando as principais entidades do domínio e suas responsabilidades. Foram aplicados princípios de design OO, como encapsulamento e polimorfismo, para garantir a flexibilidade e a manutenibilidade do código. As entidades foram mapeadas para classes do backend (Java), que representam os objetos de negócio e interagem com o banco de dados. No frontend (React), os componentes foram estruturados para refletir a interface do usuário e consumir os serviços da API.

9.2 PRINCIPAIS CLASSES E RELACIONAMENTOS

A seguir, são apresentadas as principais classes identificadas na modelagem do sistema e seus relacionamentos, que refletem diretamente os requisitos levantados.

Figura 2 - DIAGRAMA DE CLASSE DO MÓDULO DE AUTENTICAÇÃO

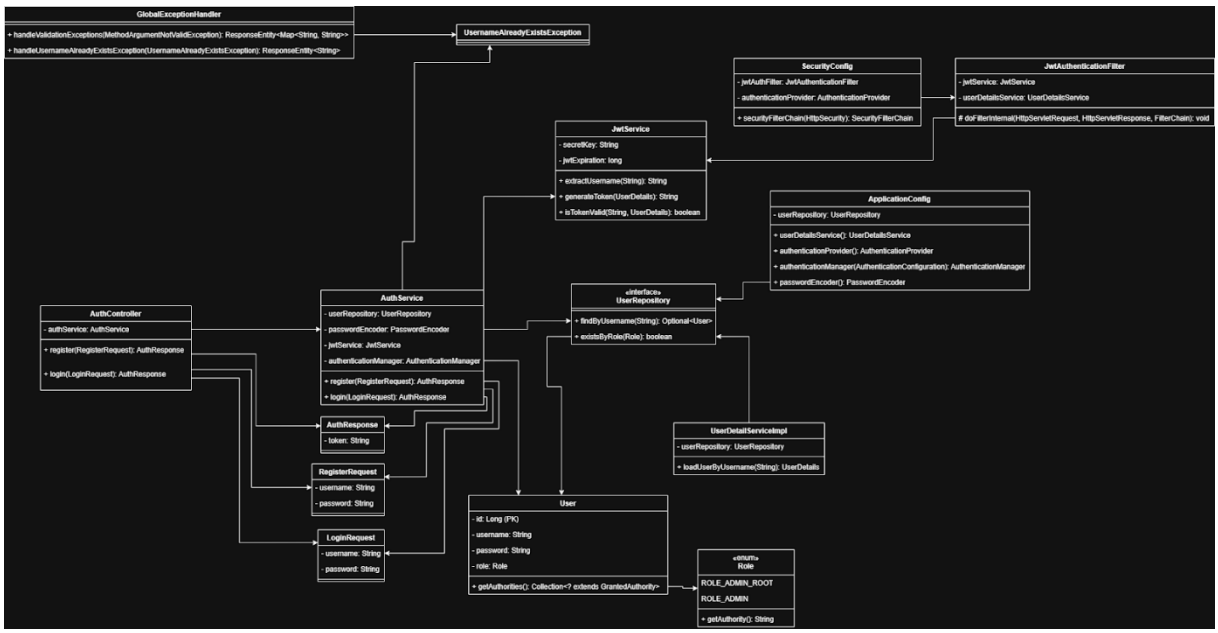
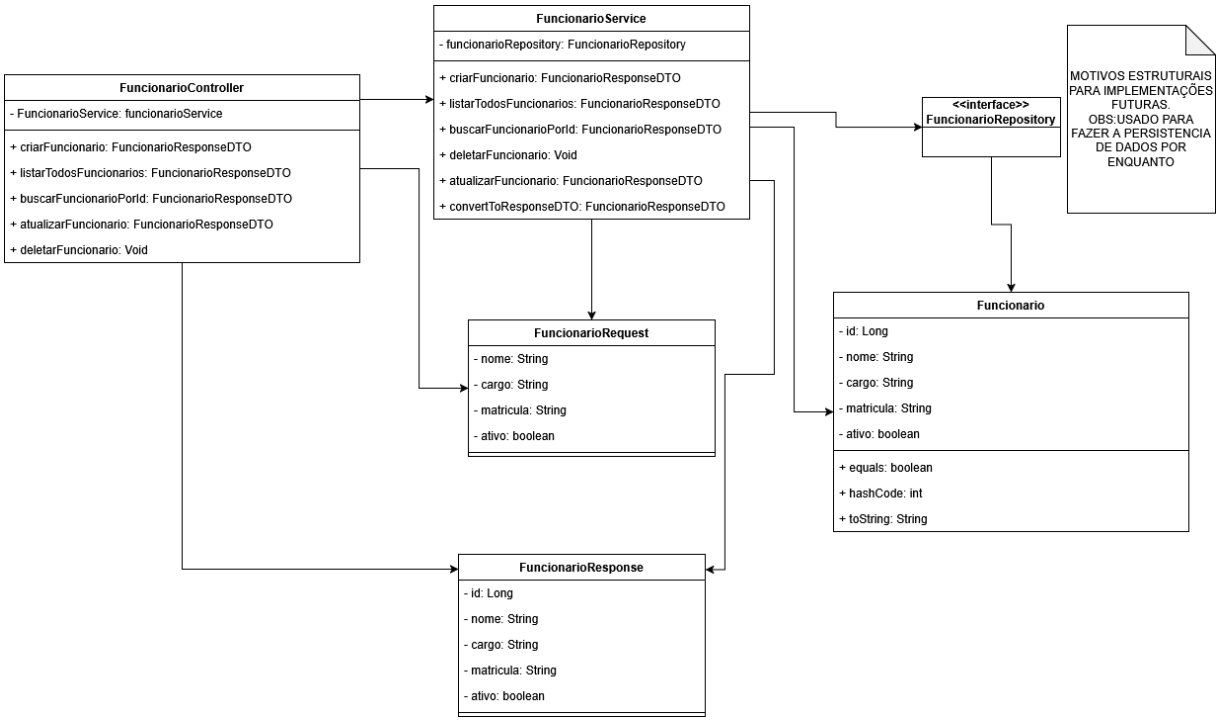


Figura 3 - DIAGRAMA DE CLASSE DO MÓDULO QUE REPRESENTA O PRODUTO



DESCRIÇÃO DAS CLASSES

9.2.1.1 ENTIDADES DE DOMÍNIO

- **User(br.ucb.teste.software.gerenciadorfuncionarios.auth.entities.User)** Essa entidade é o coração do gerenciamento de usuários, armazenando ID, username, senha (criptografada) e o papel (role) do usuário, como **ROLE_ADMIN_ROOT** ou **ROLE_ADMIN**. Ao implementar **UserDetails** do **Spring Security**, ela é crucial para:
    - **RF001 (Registro de Usuário):** Persiste os dados de registro e garante a unicidade do username.
    - **RF002 (Login de Usuário):** Armazena as credenciais para a validação do login.
    - **RF003 (Controle de Acesso Baseado em Papéis):** Seu atributo role define as permissões, controlando o que o usuário pode e não pode fazer no sistema.
- **Funcionario(br.ucb.teste.software.gerenciadorfuncionarios.funcionario.entity.Funcionario)** Representa os funcionários no sistema, contendo ID, nome, cargo, matrícula e status ativo. É a entidade base para as operações de gerenciamento de funcionários, atendendo a:
    - **RF004 (Criação de Funcionário):** Encapsula os dados de um novo funcionário.
    - **RF005 (Listagem de Funcionários):** Os objetos dessa entidade são exibidos nas listagens.
    - **RF006 (Busca de Funcionário por ID):** É o alvo da busca.
    - **RF007 (Atualização de Dados de Funcionário):** Representa os dados a serem modificados.
    - **RF008 (Exclusão de Funcionário):** A entidade que será removida do sistema.

9.2.1.2 CAMADA DE SERVIÇOS

- **AuthService(br.ucb.teste.software.gerenciadorfuncionarios.auth.services.AuthService)** Esta classe encapsula toda a lógica de negócio para autenticação (login) e registro de usuários. Ela se comunica com UserRepository para persistência e JwtService para a criação de tokens. Suas responsabilidades incluem:
    - **RF001 (Registro de Usuário):** Gerencia a unicidade do username, criptografia de senhas, atribuição de roles (incluindo a lógica para ROLE_ADMIN_ROOT inicial e restrições) e a geração do JWT após o registro.
    - **RF002 (Login de Usuário):** Valida credenciais, lida com falhas de login e gera o JWT em caso de sucesso.

- **JwtService(br.ucb.teste.software.gerenciadorfuncionarios.auth.services.JwtService)** Serviço especializado na geração, extração e validação de JSON Web Tokens (JWTs). É fundamental para a segurança, pois:
    - **RF001 (Registro de Usuário) e RF002 (Login de Usuário):** Gera o JWT que é retornado ao usuário após o sucesso das operações.
    - **RF003 (Controle de Acesso Baseado em Papéis):** É essencial para validar a autenticação de todas as requisições protegidas, garantindo acesso apenas a usuários com um token válido.
- **FuncionarioService(br.ucb.teste.software.gerenciadorfuncionarios.funcionario.service.FuncionarioService)** Esta camada de serviço gerencia as regras de negócio para todas as operações CRUD (Criar, Ler, Atualizar, Deletar) de Funcionario. Ela interage diretamente com FuncionarioRepository e é responsável por:
    - **RF004 (Criação de Funcionário):** Implementa a lógica para persistir novos funcionários e a geração automática de IDs.
    - **RF005 (Listagem de Funcionários):** Recupera todos os funcionários.
    - **RF006 (Busca de Funcionário por ID):** Executa a busca e trata casos de "não encontrado".
    - **RF007 (Atualização de Dados de Funcionário):** Gerencia a atualização dos dados do funcionário, incluindo a verificação de existência.
    - **RF008 (Exclusão de Funcionário):** Implementa a remoção de funcionários do banco de dados.

9.2.1.3 CAMADA DE CONTROLADORES

- **AuthController(br.ucb.teste.software.gerenciadorfuncionarios.auth.controllers.AuthController)** Este é o controlador REST que expõe os endpoints /auth/register e /auth/login, servindo como a porta de entrada para as funcionalidades de autenticação.
    - **RF001 (Registro de Usuário):** É o endpoint para registrar novos usuários.
    - **RF002 (Login de Usuário):** É o endpoint para o processo de login.
- **FuncionarioController(br.ucb.teste.software.gerenciadorfuncionarios.funcionario.controller.FuncionarioController)** Este controlador REST expõe os endpoints para todas as operações de gerenciamento de funcionários (ex: /api/funcionarios)**.**
    - **RF004 (Criação de Funcionário):** Endpoint para criar funcionários.
    - **RF005 (Listagem de Funcionários):** Endpoint para listar funcionários.
    - **RF006 (Busca de Funcionário por ID):** Endpoint para buscar por ID.
    - **RF007 (Atualização de Dados de Funcionário):** Endpoint para atualizar dados.
    - **RF008 (Exclusão de Funcionário):** Endpoint para excluir funcionários.

9.2.1.4 CAMADA DE REPOSITÓRIOS

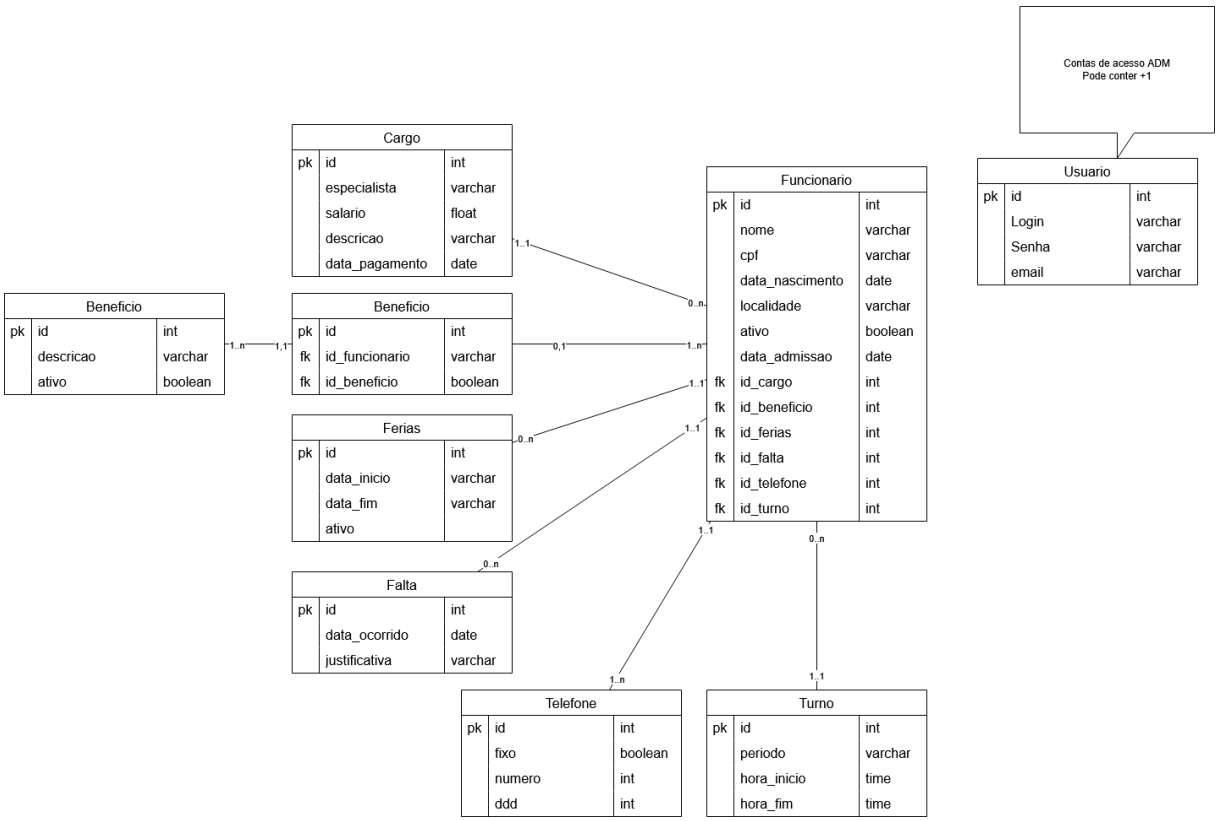
- **UserRepository(br.ucb.teste.software.gerenciadorfuncionarios.auth.repositories.UserRepository)** Interface de acesso a dados para a entidade User, estendendo JpaRepository. Ela permite operações de persistência diretamente no banco de dados.
    - **RF001 (Registro de Usuário):** Usado para salvar novos usuários e verificar a unicidade.
    - **RF002 (Login de Usuário):** Utilizado para buscar usuários por username durante o login.
- **FuncionarioRepository(br.ucb.teste.software.gerenciadorfuncionarios.funcionario.repository.FuncionarioRepository)** Interface de acesso a dados para a entidade Funcionario, estendendo JpaRepository. É responsável pelas operações de persistência dos dados de funcionários.
    - **RF004 (Criação de Funcionário):** Persiste novos registros.
    - **RF005 (Listagem de Funcionários):** Recupera listas de funcionários.
    - **RF006 (Busca de Funcionário por ID):** Busca um funcionário por ID.
    - **RF007 (Atualização de Dados de Funcionário):** Atualiza dados de funcionários existentes.
    - **RF008 (Exclusão de Funcionário):** Remove registros de funcionários.

9.2.1.5 OBJETOS DE TRANFERÊRENCIA DE DADOS

- **DTOs (LoginRequest, RegisterRequest, AuthResponse, FuncionarioRequestDTO, FuncionarioResponseDTO)** São objetos que definem a estrutura dos dados trocados entre o frontend e o backend, garantindo validação e tipagem corretas. Eles são cruciais para:
- **RF001 (Registro de Usuário):** RegisterRequest (entrada) e AuthResponse (saída com JWT).
- **RF002 (Login de Usuário):** LoginRequest (entrada) e AuthResponse (saída com JWT).
- **RF004 (Criação de Funcionário):** FuncionarioRequestDTO (entrada).
- **RF005 (Listagem de Funcionários) e RF006 (Busca de Funcionário por ID):** FuncionarioResponseDTO (saída).
- **RF007 (Atualização de Dados de Funcionário):** FuncionarioRequestDTO (entrada para atualização).

10 MODELO LÓGICO DO SISTEMA

Figura 4 - Modelo Lógico



11 ANÁLISE DE QUALIDADE E RISCOS

11.1 ANÁLISE DA QUALIDADE DO PRODUTO (ISO 25010)

A qualidade do Sistema de Gerenciamento de Funcionários foi avaliada com base nas características da norma ISO 25010, que define um modelo de qualidade de produto de software.

a) Funcionalidade:

- **Adequação Funcional:** O sistema atende aos requisitos funcionais (RF001 a RF006), permitindo o gerenciamento completo de funcionários (CRUD) e a autenticação/registro de usuários. As operações de CRUD são implementadas conforme especificado, com tratamento para credenciais inválidas e dados de entrada incorretos.
- **Completude Funcional:** As principais funcionalidades para o gerenciamento de funcionários e autenticação de usuários estão presentes e operacionais. Não foram identificadas lacunas críticas no escopo definido.
- **Corretude Funcional:** Os testes indicam que as funções executam os resultados esperados. Por exemplo, o login com credenciais válidas resulta em acesso autorizado (RF001) e o cadastro de um funcionário persiste os dados corretamente (RF002). As validações de entrada e o tratamento de erros (ex: 401 Unauthorized, 400 Bad Request) funcionam conforme esperado, demonstrando que as saídas estão corretas para diferentes cenários.
- **Apropriabilidade Funcional:** As funcionalidades implementadas são relevantes e úteis para o público-alvo (equipes de RH e gestores), simplificando a organização das informações de funcionários.

b) Confiabilidade:

- **Maturidade:** O sistema demonstra estabilidade nas operações testadas, com poucos erros ou falhas inesperadas durante a execução dos casos de teste. Os mecanismos de validação e tratamento de exceções contribuem para a robustez.
- **Tolerância a Falhas:** O sistema possui mecanismos de tratamento de erros, como validação de dados de entrada (@Valid em DTOs, GlobalExceptionHandler) e respostas de erro HTTP apropriadas para cenários como credenciais inválidas (401 Unauthorized), usuário já existente (409 Conflict) ou dados malformados (400 Bad Request). Erros internos do servidor são capturados e retornam um 500 Internal Server Error, protegendo a exposição de detalhes sensíveis da implementação.
- **Recuperabilidade:** Embora não haja um mecanismo de recuperação de desastres robusto implementado no escopo do projeto (como replicação de banco de dados ou sistemas de *failover*), a persistência de dados em MySQL e o uso de transações garantem a integridade das operações CRUD. A aplicação é *stateless* (devido ao uso de JWT), o que facilita a recuperação em caso de falha de uma instância do backend.

c) Usabilidade:

- **Reconhecimento da Adequação:** A estrutura dos endpoints RESTful e os DTOs são claros e seguem padrões conhecidos, o que facilita a compreensão e a integração com o frontend. Embora a interface do usuário não tenha sido fornecida para análise, a arquitetura de backend suporta uma interface intuitiva (RNF002).
- **Aprendizagem:** A clareza das APIs e a separação de responsabilidades no código-fonte do backend contribuem para a facilidade de aprendizado e compreensão do sistema por novos desenvolvedores.
- **Operabilidade:** As operações são diretas e mapeadas para métodos HTTP padrão (POST para criar, GET para ler, PUT para atualizar, DELETE para excluir), tornando a interação com a API previsível.
- **Proteção contra Erros de Usuário:** O backend inclui validações de entrada (@NotBlank, @Size nos DTOs de autenticação e registro) e um GlobalExceptionHandler que retorna mensagens de erro claras para guiar o usuário ou o frontend em caso de entradas inválidas.

d) Eficiência de Desempenho:

- **Comportamento em Tempo:** O sistema responde em tempo hábil para as operações de CRUD. Foram realizados testes de carga e estresse e as respostas das APIs foram observadas como rápidas em um ambiente de desenvolvimento local.
- **Utilização de Recursos:** O consumo de CPU e memória é considerado adequado para o escopo da aplicação, que lida com operações de gerenciamento de dados de funcionários. Não foram identificados gargalos de desempenho em cenários de uso típico.

e) Segurança:

- **Confidencialidade:** O sistema protege os dados sensíveis dos funcionários e credenciais de usuários através da criptografia de senhas utilizando BCrypt (BCryptPasswordEncoder em ApplicationConfig). A autenticação baseada em JWT garante que as sessões são seguras e sem estado. As APIs são protegidas por filtros de segurança (JwtAuthenticationFilter) que validam o token JWT antes de conceder acesso aos recursos (RNF001).
- **Integridade:** As operações de atualização e exclusão de funcionários são controladas por autenticação e autorização (implicitamente via Spring Security com JWT), garantindo que apenas usuários autorizados possam modificar os dados. As validações de entrada contribuem para a integridade dos dados antes da persistência.
- **Não Repudiação:** Não há um mecanismo de não repudiação explícito (como logs de auditoria detalhados de todas as ações do usuário) implementado como foco primário neste estágio do projeto.
- **Responsabilidade:** Similar à não repudiação, a rastreabilidade direta de quem realizou cada ação não é um requisito primário neste momento. A responsabilidade é implícita através da autenticação do usuário que realiza a requisição.

- **Autenticidade:** A autenticação de usuário através do endpoint /auth/login verifica a identidade do usuário e, se as credenciais forem válidas, concede um token JWT que serve como prova de autenticidade para requisições subsequentes.

f) Manutenibilidade:

- **Modularidade:** A arquitetura em camadas (*controllers*, *services*, *repositories*) e a separação clara de responsabilidades (autenticação vs. gerenciamento de funcionários) promovem alta modularidade. A configuração de segurança (SecurityConfig) e o tratamento global de exceções (GlobalExceptionHandler) são bons exemplos de módulos bem definidos (RNF005).
- **Reusabilidade:** Componentes como JwtService e UserRepository são projetados para serem reutilizáveis dentro do contexto de segurança e acesso a dados, respectivamente. DTOs também promovem a reusabilidade de estruturas de dados.
- **Analisabilidade:** O código é estruturado de forma lógica, com nomes de classes e métodos claros. A padronização do projeto Spring Boot facilita a análise para identificação de defeitos ou novas funcionalidades.
- **Modificabilidade:** Pequenas alterações em requisitos (ex: adicionar um novo campo ao funcionário) podem ser incorporadas com impacto limitado, principalmente através da modificação dos DTOs, entidade e serviço correspondente.
- **Testabilidade:** A separação de responsabilidades e o uso de injeção de dependências facilitam a escrita de testes de unidade e integração, permitindo que os componentes sejam testados de forma isolada (ex: AuthService pode ser testado com *mocks* para UserRepository e JwtService).

g) Compatibilidade:

- **Coexistência:** Não é um requisito explícito para o sistema coexistir com outros sistemas na mesma infraestrutura neste momento.
- **Interoperabilidade:** O sistema é acessível através de uma API RESTful, o que permite fácil interoperabilidade e futura integração com outros sistemas ou frontends via HTTP (RNF006). A configuração CORS (CorsConfig.java) demonstra a preocupação com a interoperabilidade entre diferentes origens do frontend.

h) Portabilidade:

- **Adaptabilidade:** O backend é construído com Spring Boot e Java, o que o torna altamente adaptável a diferentes ambientes de implantação (servidores Linux, Windows, contêineres Docker, plataformas de nuvem como AWS, Azure, Google Cloud) desde que tenham uma JVM instalada. O MySQL é um banco de dados amplamente suportado.
- **Instalabilidade:** O processo de implantação do backend Spring Boot é relativamente direto, geralmente envolvendo a execução de um arquivo JAR empacotado. O frontend React pode ser empacotado para servir arquivos estáticos.

- **Substitutibilidade:** A arquitetura desacoplada permitiria, em teoria, substituir componentes (ex: trocar React por outro *framework* frontend, ou MySQL por PostgreSQL/MariaDB) com menor esforço, embora exigiria implementações de adaptadores para as novas tecnologias.

11.2 ANÁLISE DE RISCOS

A identificação e análise de riscos são componentes essenciais do planejamento de testes, pois permitem antecipar potenciais problemas que podem impactar a qualidade do software e definir estratégias de mitigação.

Abaixo estão os principais riscos identificados para o Sistema de Gerenciamento de Funcionários:

ID do Risco	Risco Identificado	Impacto Potencial	Probabilidade	Mitigação / Plano de Ação
R-001	Vulnerabilidades de Segurança: Falhas na autenticação (RF001) ou autorização, injeção SQL, exposição de dados sensíveis (RNF001).	Acesso não autorizado, vazamento de dados, comprometimento da integridade do sistema, multas regulatórias, danos à reputação.	Alta	Implementação de JWT para autenticação e autorização (configurado em SecurityConfig), criptografia de senhas com BCrypt (ApplicationConfig), validação rigorosa de entrada de dados no backend (LoginRequest, RegisterRequest com @NotBlank, @Size, e tratamento via GlobalExceptionHandler), uso de JPA/Hibernate (minimizando riscos de injeção SQL direta), testes de segurança específicos (ex: login com credenciais inválidas, campos vazios), revisão de código por pares.
R-002	Falhas de Desempenho: Lentidão nas respostas da API, sobrecarga do banco de dados (RNF004).	Má experiência do usuário, baixa produtividade, incapacidade de lidar com volume de dados esperado.	Média	Otimização de consultas SQL (implícita pelo uso de Spring Data JPA), uso de índices no banco de dados para campos de busca comuns (ex: username em User), testes de carga e estresse em cenários críticos (ex: listagem de muitos funcionários, múltiplas requisições simultâneas - se

				ferramentas como JMeter forem usadas, mencionar aqui). A arquitetura RESTful e stateless (JWT) contribui para escalabilidade.
R-003	Inconsistência de Dados: Dados corrompidos ou inconsistentes devido a falhas em operações de CRUD (RF002, RF004, RF005).	Relatórios errados, perda de informações críticas, erros em processos de RH.	Média	Implementação de transações em nível de banco de dados (gerenciado automaticamente pelo Spring Data JPA), validação de dados em todas as camadas (frontend, backend com DTOs e jakarta.validation.constraints), testes de integração e sistema para verificar integridade dos dados após operações CRUD (ex: verificar se um funcionário cadastrado pode ser listado e atualizado corretamente).
R-004	Problemas de Usabilidade: Interface complexa, difícil de usar ou com erros de UX (RNF002).	Baixa adesão dos usuários, aumento no tempo de treinamento, insatisfação dos stakeholders.	Baixa	Design centrado no usuário (no frontend), testes de usabilidade (mesmo que informais), feedback de usuários durante o desenvolvimento, revisão da interface por pares. A clareza da API RESTful no backend facilita a construção de uma interface intuitiva.
R-005	Incompatibilidade: Problemas de renderização em diferentes navegadores ou versões do MySQL (RNF006).	Sistema não funcional para alguns usuários ou ambientes.	Baixa	Testes em múltiplos navegadores (para o frontend), uso de bibliotecas compatíveis (ex: polyfills para JavaScript no React), verificação da compatibilidade da versão do MySQL (MySQL Server 8.0.33), e CorsConfig para garantir comunicação entre diferentes origens do frontend.

R-006	Defeitos em Integrações: Falhas na comunicação entre frontend e backend (RNF003).	Funcionalidades parciais ou inoperantes.	Média	Testes de integração robustos cobrindo interações entre as camadas (ex: testar login no frontend, receber token e usá-lo para acessar recursos protegidos), uso de logs para monitorar a comunicação, tratamento de exceções nas APIs (GlobalExceptionHandler), configuração CORS (CorsConfig).
R-007	Falta de cobertura de testes: Testes insuficientes para garantir a qualidade total do software.	Defeitos não detectados, degradação da qualidade, retrabalho.	Média	Planejamento de testes abrangente cobrindo todos os requisitos, utilização de técnicas de cobertura funcional (caminhos de sucesso e falha, validação de entrada), revisão dos planos de teste por pares. Recomendado uso de JUnit para testes de unidade no backend para aumentar cobertura.
R-008	Ambiente de Teste Não Representativo: Discrepância entre o ambiente de desenvolvimento/teste e o ambiente de produção.	Testes que passam no ambiente de teste podem falhar em produção.	Baixa	Padronização do ambiente de teste para que seja o mais próximo possível do de produção (versões de SO, JDK, Node.js, MySQL Server 8.0.33, bibliotecas). Uso de Docker (se aplicável) para padronizar ainda mais o ambiente.

12 PLANEJAMENTO DE TESTES

12.1 Estratégia de Testes

A estratégia de testes adotada para o Sistema de Gerenciamento de Funcionários segue uma abordagem em camadas, visando a detecção precoce de defeitos e a garantia da qualidade em diferentes níveis do sistema. Foram planejados os seguintes tipos de testes:

- **Testes de Unidade:** Focados em validar componentes individuais do código (classes, métodos) de forma isolada. São executados pelos desenvolvedores para garantir que cada parte da aplicação funcione corretamente. No contexto Java/Spring, isso envolveria testes de métodos em AuthService, FuncionarioService, JwtService, e outros componentes.

- **Testes de Integração:** Destinam-se a verificar a comunicação e a interação entre diferentes módulos ou camadas do sistema (ex: backend com banco de dados através dos repositórios, controladores com serviços). Exemplos incluem testar se o AuthService consegue salvar um usuário no UserRepository e se o FuncionarioController interage corretamente com o FuncionarioService.
- **Testes de Sistema:** Validam o sistema como um todo, garantindo que todos os componentes integrados funcionem conforme os requisitos funcionais e não funcionais. Esses testes simulam cenários de uso real através da API RESTful (Postman), abrangendo o fluxo completo de autenticação e gerenciamento de funcionários.
- **Testes de Aceitação:** São realizados para verificar se o sistema atende às expectativas e requisitos de negócio dos usuários finais/*stakeholders*. Validam os requisitos funcionais do ponto de vista do cliente, muitas vezes com base em cenários de negócios.
- **Testes de Segurança:** Focados em identificar vulnerabilidades de segurança, como falhas de autenticação e autorização (relacionado a RNF001), tentativas de acesso a recursos não autorizados, e validação de entrada para prevenir ataques (ex: verificação de 401 Unauthorized para acessos sem token).

12.2 Ferramentas de Teste

As seguintes ferramentas foram utilizadas para o planejamento e execução dos testes:

- **Postman:** Ferramenta robusta para testar APIs RESTful, utilizada para criar e executar requisições HTTP (GET, POST, PUT, DELETE) para os endpoints do backend e validar as respostas (status codes, corpo da resposta, headers). Essencial para testes de integração e sistema.
- **JUnit 5 (Java):** Framework de testes de unidade para aplicações Java. Utilizado para escrever testes automatizados para classes e métodos individuais no backend (ex: lógica de negócio em serviços, funcionalidades do JwtService).
- **Mockito (Java):** Framework de *mock* para Java, frequentemente utilizado em conjunto com JUnit para simular o comportamento de dependências em testes de unidade, permitindo testar componentes isoladamente.
- **Spring Boot Test:** Módulo de testes do Spring Boot que fornece utilitários para escrever testes de integração para aplicações Spring, incluindo a capacidade de levantar um contexto Spring para testes de controlador e serviço.
- **Selenium WebDriver (Robot Framework):** Framework de automação de testes para aplicações web. Utilizado para simular interações reais do usuário em navegadores (como cliques, preenchimento de formulários, navegação), permitindo a validação de funcionalidades da interface gráfica. Combinado com o Robot Framework, proporciona uma abordagem de automação de testes de aceitação e *end-to-end* com sintaxe legível.

- **Apache JMeter:** Ferramenta *open-source* utilizada para testes de carga e desempenho em aplicações web e serviços. Permite simular múltiplas requisições simultâneas para avaliar a escalabilidade, o tempo de resposta e a estabilidade dos endpoints backend sob diferentes condições de estresse.

12.3 Ambiente de Teste

O ambiente de teste foi configurado para simular, o mais próximo possível, o ambiente de produção, garantindo que os resultados dos testes sejam representativos.

- **Sistema Operacional:** Windows 10 Pro / Linux Ubuntu 22.04 LTS
- **Máquina:** Laptop com processador Intel Core i7, 16GB RAM (Configuração típica de desenvolvimento).
- **Java Development Kit (JDK):** OpenJDK 17
- **Node.js:** v18.17.0 (Para o frontend React, se necessário para execução local, mas o foco aqui é o backend)
- **Gerenciador de Pacotes Frontend:** npm v9.6.7 (Para o frontend React, se necessário para execução local)
- **Banco de Dados:** MySQL Server 8.0.33
- **Servidor de Aplicação Backend:** Spring Boot Embedded Tomcat (O servidor web embarcado padrão do Spring Boot)
- **Ferramentas de Teste:** Postman v11.0.0, JUnit 5 v5.13.1, Mockito v5.18.0, JMeter v5.6.3, Selenium WebDriver v4.33.0.
- **Conectividade:** Ambiente de rede local, sem proxies ou VPNs que pudessem interferir na comunicação entre o cliente de teste (Postman) e o servidor de backend.
- **Dados de Teste:** Para os testes de sistema e aceitação, foi utilizada uma base de dados limpa, populada com dados de teste específicos (ex: usuários admin/admin, rh/rh, funcionários fictícios como "João da Silva", "Maria Oliveira").

12.4 Métricas de Teste

- **Cobertura Funcional:** A bateria de testes planejada foi projetada para cobrir todos os requisitos funcionais (RF001 a RF006) e os principais requisitos não funcionais (RNF001 - Segurança, RNF002 - Usabilidade, RNF003 - Confiabilidade, RNF004 - Eficiência de Desempenho) do sistema, garantindo que cada funcionalidade seja validada através de pelo menos um caso de teste, abrangendo cenários de sucesso e falha (ex: validação de entrada, credenciais inválidas).
- **Cobertura de Código:** Devido às limitações de tempo e ferramentas disponíveis neste escopo do projeto, uma análise formal de cobertura de código não foi realizada para apresentar métricas percentuais específicas. No entanto, a estratégia de testes unitários e de integração buscou maximizar a cobertura dos caminhos de execução mais críticos e complexos do sistema, focando nas lógicas de negócio e nos fluxos de autenticação.

A verificação manual do código e a escrita de testes para os caminhos principais foram priorizadas para assegurar a qualidade do software.

13 EVIDÊNCIAS DE TESTES

Esta seção descreve os casos de teste planejados e executados para validar o Sistema de Gerenciamento de Funcionários. Os testes estão categorizados por funcionalidade principal.

13.1 TESTES DE AUTENTICAÇÃO (RF001, RNF001)

CT-AUTH-001: LOGIN COM CREDENCIAIS VÁLIDAS

- **Tipo de Teste:** Teste de Sistema / Aceitação
- **Requisito(s) Associado(s):** RF001 - Autenticação de Usuário
- **Descrição:** Verificar se um usuário com credenciais válidas pode fazer login com sucesso no sistema e receber um token JWT.
- **Pré-condições:** Um usuário (usuario_existente, senhaCorreta123) deve estar previamente cadastrado no banco de dados.
- **Passos:**
    1. Enviar uma requisição POST para {{baseURL}}/auth/login.
    2. No corpo da requisição, incluir um JSON com username: "usuario_existente" e password: "senhaCorreta123".
- **Dados de Teste:**

JSON

```
{
  "username": "usuario_existente",
  "password": "senhaCorreta123"
}
```

- **Resultado Esperado:**
    - Status da Resposta: 200 OK
    - Corpo da Resposta: Um objeto JSON contendo um token JWT válido, conforme AuthResponse.java.
        - Exemplo: {"token": "eyJhbGciOiJIUzI1Ni..."}
- **Critério de Aprovação:** Status 200 OK e presença de um token JWT válido e não vazio no corpo da resposta.

CT-AUTH-002: LOGIN COM CREDENCIAIS INVÁLIDAS

- **Tipo de Teste:** Teste de Sistema / Aceitação

- **Requisito(s) Associado(s):** RF001 - Autenticação de Usuário, RNF001 - Segurança
- **Descrição:** Verificar se o sistema rejeita tentativas de login com credenciais inválidas (usuário não encontrado ou senha incorreta).
- **Pré-condições:** Nenhuma especial.
- **Passos:**
    1. Enviar uma requisição POST para {{baseURL}}/auth/login.
    2. No corpo da requisição, incluir um JSON com username: "usuario_inexistente" ou password: "senhaInvalida".
- **Dados de Teste:**

JSON

```
{
  "username": "usuario_inexistente",
  "password": "senhaInvalida"
}
```

- **Resultado Esperado:**
    - Status da Resposta: 401 Unauthorized
    - Corpo da Resposta: Vazio ou uma mensagem de erro indicando "Credenciais inválidas" ou similar (o AuthController atualmente retorna um corpo vazio para essa exceção, mas o GlobalExceptionHandler poderia ser estendido para mensagens mais descritivas, embora 401 seja o status correto).
- **Critério de Aprovação:** Status 401 Unauthorized e ausência de token JWT.

CT-AUTH-003: ERRO DE VALIDAÇÃO DE ENTRADA – LOGIN

- **Tipo de Teste:** Teste de Sistema
- **Requisito(s) Associado(s):** RNF003 - Confiabilidade (Tratamento de erros)
- **Descrição:** Verificar se o sistema lida corretamente com dados de entrada inválidos para o login (ex: campos obrigatórios ausentes ou vazios), retornando um erro de validação.
- **Pré-condições:** Nenhuma.
- **Passos:**
    1. Enviar uma requisição POST para {{baseURL}}/auth/login.
    2. No corpo da requisição, incluir um JSON com dados inválidos (ex: username ou password vazios, ou campos ausentes).

- **Dados de Teste:**

JSON

```
{
   "username": "", // Campo obrigatório vazio
   "password": "minhaSenha"
}
```

Ou:

JSON

```
{
   "username": "usuario_valido"
   // password ausente
}
```

- **Resultado Esperado:**
    - Status da Resposta: 400 Bad Request (conforme GlobalExceptionHandler.java para MethodArgumentNotValidException).
    - Corpo da Resposta: Um objeto JSON contendo detalhes dos erros de validação, indicando quais campos estão inválidos (ex: {"username": "O nome de usuário não pode estar em branco"}).
- **Critério de Aprovação:** Status 400 Bad Request e mensagem de erro clara e informativa detalhando os problemas de validação.

13.2 TESTES DE CADASTRO DE FUNCIONÁRIOS (RF002, RNF001, RNF003)

CT-EMP-001: CADASTRAR FUNCIONÁRIO COM DADOS VÁLIDOS

- **Tipo de Teste:** Teste de Sistema / Aceitação
- **Requisito(s) Associado(s):** RF002 - Cadastro de Funcionários
- **Descrição:** Verificar se é possível cadastrar um novo funcionário com todos os dados válidos e obrigatórios.
- **Pré-condições:** Usuário autenticado (token JWT válido obtido de CT-AUTH-001).
- **Passos:**
    1. Obter um token JWT válido através do login (CT-AUTH-001).
    2. Enviar uma requisição POST para {{baseURL}}/api/funcionarios com o token JWT no cabeçalho Authorization: Bearer <token_jwt>.

3. No corpo da requisição, incluir um JSON com os dados completos e válidos do novo funcionário.

- **Dados de Teste:**

JSON

```
{
  "nome": "Maria Silva",
  "cargo": "Analista de RH",
  "matricula": "RH001",
  "ativo": true
}
```

- **Resultado Esperado:**
  - Status da Resposta: 201 Created
  - Corpo da Resposta: Objeto JSON representando o funcionário recém-cadastrado, incluindo um ID gerado automaticamente.
    - Exemplo: {"id": 1, "nome": "Maria Silva", "cargo": "Analista de RH", "matricula": "RH001", "ativo": true}.
  - Verificação no BD: O funcionário deve ser persistido no banco de dados com os dados fornecidos e um ID único.
- **Critério de Aprovação:** Status 201 Created e dados do funcionário corretos (incluindo o ID) no corpo da resposta e persistidos no banco de dados.

CT-EMP-002: CADASTRAR FUNCIONÁRIO COM DADOS INVÁLIDOS/FALTANTES

- **Tipo de Teste:** Teste de Sistema
- **Requisito(s) Associado(s):** RNF003 - Confiabilidade (Validação de Entrada)
- **Descrição:** Verificar se o sistema rejeita o cadastro de funcionário com dados inválidos ou faltando, retornando um erro de validação.
- **Pré-condições:** Usuário autenticado (token JWT válido).
- **Passos:**
  1. Obter um token JWT válido.
  2. Enviar uma requisição POST para {{baseURL}}/api/funcionarios com o token JWT.
  3. No corpo da requisição, incluir um JSON com dados inválidos (ex: nome vazio, matricula faltando).

- **Dados de Teste:**

JSON

```
{
  "nome": "", // Campo 'nome' vazio
  "cargo": "Desenvolvedor",
  "ativo": true
  // 'matricula' faltando
}
```

- **Resultado Esperado:**
    - Status da Resposta: 400 Bad Request ou 422 Unprocessable Entity (dependendo da validação implementada. Para campos vazios ou ausentes, 400 Bad Request é comum com Spring's @Valid).
    - Corpo da Resposta: Mensagem de erro detalhando os campos inválidos ou ausentes.
- **Critério de Aprovação:** Status 400 ou 422 e mensagem de erro clara e informativa.

CT-EMP-003: CADASTRO DE FUNCIONÁRIO DUPLICADO (SE APLICÁVEL)

- **Tipo de Teste:** Teste de Sistema
- **Requisito(s) Associado(s):** RNF003 - Confiabilidade (Integridade de Dados)
- **Descrição:** Verificar se o sistema lida corretamente com tentativas de cadastrar um funcionário que já existe, caso haja uma regra de negócio para unicidade (ex: por matrícula).
- **Pré-condições:** Um funcionário com um identificador único (ex: matricula) já cadastrado. Usuário autenticado.
- **Passos:**
    1. Cadastrar um funcionário com uma matricula específica (ex: "RH001").
    2. Tentar cadastrar um novo funcionário com a *mesma* matricula ("RH001").
- **Dados de Teste:** Usar dados de um funcionário já existente, alterando apenas campos não-únicos (se houver) ou mantendo-os idênticos.
- **Resultado Esperado:**
    - Status da Resposta: 409 Conflict (se o backend verificar a unicidade explicitamente, como para UsernameAlreadyExistsException) ou 400 Bad Request (se for uma validação genérica de entrada).

    - Corpo da Resposta: Mensagem de erro indicando que a matrícula já existe ou que o funcionário é duplicado.
- **Critério de Aprovação:** Status 409 ou 400 e mensagem de erro apropriada para duplicação. (Atualmente, a entidade Funcionario não possui @Column(unique=true) para matricula, então essa validação precisaria ser adicionada no serviço para retornar 409).

13.3 TESTES DE VISUALIZAÇÃO DE FUNCIONÁRIOS (RF003, RNF004)

CT-EMP-004: LISTAR TODOS OS FUNCIONÁRIOS

- **Tipo de Teste:** Teste de Sistema / Aceitação
- **Requisito(s) Associado(s):** RF003 - Visualização de Funcionários
- **Descrição:** Verificar se o sistema lista corretamente todos os funcionários cadastrados.
- **Pré-condições:** Usuário autenticado. Mínimo de 3 funcionários previamente cadastrados no banco de dados.
- **Passos:**
    1. Obter um token JWT válido.
    2. Enviar uma requisição GET para {{baseURL}}/api/funcionarios com o token JWT no cabeçalho Authorization.
- **Dados de Teste:** Não aplicável (a requisição não exige corpo, espera-se a listagem de todos).
- **Resultado Esperado:**
    - Status da Resposta: 200 OK
    - Corpo da Resposta: Um array JSON contendo todos os funcionários cadastrados (conforme FuncionarioResponseDTO).
    - Verificação: O número de funcionários no array deve corresponder ao número de funcionários esperados no BD. Os dados de cada funcionário no array devem estar corretos.
- **Critério de Aprovação:** Status 200 OK e array de funcionários completo e correto.

CT-EMP-005: BUSCAR FUNCIONÁRIO POR ID

- **Tipo de Teste:** Teste de Sistema
- **Requisito(s) Associado(s):** RF003 - Visualização de Funcionários
- **Descrição:** Verificar se é possível buscar um funcionário específico utilizando seu ID e se os dados retornados estão corretos.

- **Pré-condições:** Usuário autenticado. Um funcionário com um ID conhecido (ex: ID 1) deve estar previamente cadastrado.
- **Passos:**
    1. Obter um token JWT válido.
    2. Enviar uma requisição GET para {{baseURL}}/api/funcionarios/{id_do_funcionario} (substituindo {id_do_funcionario} pelo ID real, ex: 1) com o token JWT no cabeçalho Authorization.
- **Dados de Teste:** ID de um funcionário existente (ex: 1).
- **Resultado Esperado:**
    - Status da Resposta: 200 OK
    - Corpo da Resposta: Objeto JSON do funcionário correspondente ao ID.
- **Critério de Aprovação:** Status 200 OK e dados do funcionário no corpo da resposta correspondem exatamente aos dados do funcionário com o ID especificado.

CT-EMP-006: BUSCAR FUNCIONÁRIO POR ID INEXISTENTE

- **Tipo de Teste:** Teste de Sistema
- **Requisito(s) Associado(s):** RNF003 - Confiabilidade
- **Descrição:** Verificar se o sistema lida corretamente com a busca por um ID de funcionário que não existe no banco de dados, retornando um erro apropriado.
- **Pré-condições:** Usuário autenticado.
- **Passos:**
    1. Obter um token JWT válido.
    2. Enviar uma requisição GET para {{baseURL}}/api/funcionarios/{id_inexistente} (ex: 99999, um ID que não existe).
- **Dados de Teste:** Um ID que sabidamente não existe no banco de dados (ex: 99999).
- **Resultado Esperado:**
    - Status da Resposta: 404 Not Found
    - Corpo da Resposta: Vazio ou uma mensagem de erro indicando que o recurso não foi encontrado.
- **Critério de Aprovação:** Status 404 Not Found.

14 LINK DO REPOSITÓRIO

Gustavo-Brian/UCB-TS-Trabalho: Trabalho acadêmico com o objetivo de viabilizar a execução e análise de testes de software.

15 CONCLUSÃO E LIÇÕES APRENDIDAS

Neste trabalho, tivemos a oportunidade de trabalhar em diversas etapas do desenvolvimento de software. Começamos com a documentação, onde elaboramos um documento de **visão** para ter uma ideia clara do projeto. Também definimos as **regras de negócio**, os **requisitos funcionais** e **não funcionais**, e as **histórias de usuário**, criando uma base sólida.

Além da documentação, nos aprofundamos na modelagem do sistema, utilizando **diagramas UML** como os de classe, caso de uso e entidade-relacionamento para visualizar a estrutura e o funcionamento do nosso projeto.

Na parte prática, focamos no desenvolvimento, construindo um **backend RESTful** robusto em **Java** com **Spring**. Também trabalhamos com o banco de dados **MySQL** para armazenar as informações de forma eficiente. Nossa aplicação foi projetada para rodar em **dois servidores diferentes**, onde o **frontend**, desenvolvido em **React**, consumia a **API** do **backend**, estabelecendo uma comunicação **cliente-servidor** eficaz.

Também nos dedicamos aos testes de software. Aplicamos diferentes ferramentas, como Selenium Robot para testes de interface, Postman para validação das APIs, JMeter para avaliar o desempenho e JUnit para testes unitários, assegurando que o sistema funcionasse conforme o esperado.

16 REFERÊNCIAS

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

Documentação do Spring Boot – https://spring.io/projects/spring-boot

Documentação do React – https://reactjs.org/

Guia do Hibernate Validator – https://hibernate.org/validator/

Documentação oficial do MySQL – https://dev.mysql.com/doc/