Sistema de Análise de Performance Organizacional (SaaS)

Gustavo William Larsen

Resumo

O presente trabalho propõe o desenvolvimento de um Sistema de Análise de Performance Organizacional baseado no modelo SaaS (Software as a Service), visando a otimização da administração de funcionários, equipes e desempenho organizacional. O sistema permitirá o controle hierárquico das equipes em uma estrutura de árvore, a gestão de desempenho individual e coletivo, a administração de salários e planos de carreira, bem como a geração de métricas de avaliação. Além disso, incluirá a funcionalidade de exportação da estrutura organizacional e a separação por filiais. O sistema contará com um módulo avançado de gestão de performance, onde será possível definir e acompanhar KPIs, avaliar desempenho em níveis individuais, de equipe e organizacional, além de gerar dashboards dinâmicos para análise de resultados.

Sumário

1 Introdução

1.1 Contexto

A gestão de recursos humanos em grandes empresas pode ser complexa, especialmente no que se refere ao controle de hierarquias, desempenho e crescimento profissional dos colaboradores. A tecnologia SaaS se tornará uma solução viável para empresas que buscam digitalizar e otimizar processos administrativos.

1.2 Justificativa

A criação de um sistema SaaS para análise de performance organizacional possibilita maior eficiência, acessibilidade e integração de dados organizacionais. Empresas de diferentes portes podem se beneficiar da flexibilidade e da centralização das informações em um único ambiente. Além disso, um sistema robusto de gestão de performance permite a otimização dos processos de avaliação de colaboradores, facilitando tomadas de decisão estratégicas baseadas em dados.

1.3 Objetivos

Objetivo Geral: Desenvolver um sistema SaaS que otimize o processo de análise de performance organizacional com base em KPIs personalizados, permitindo avaliações eficazes em múltiplos níveis (individual, equipe e empresa), para subsidiar decisões estratégicas e promover a evolução profissional dentro das organizações.

Objetivos Específicos:

- Estruturar a gestão de funcionários e equipes por meio de uma representação hierárquica em árvore;
- Implementar funcionalidades para avaliação de desempenho individual e coletivo, associadas à gestão salarial e evolução de carreira;
- Permitir a definição, acompanhamento e personalização de KPIs para cada colaborador, equipe e unidade organizacional;
- Desenvolver dashboards interativos com gráficos e relatórios dinâmicos para a análise da performance em tempo real;
- Implementar recursos de exportação da estrutura organizacional e separação por filiais para empresas com múltiplas unidades.

2 Descrição do Projeto

2.1 Tema do Projeto

O sistema proposto será um software em nuvem para gestão de recursos humanos, permitindo o gerenciamento de funcionários e equipes de maneira eficiente e estruturada.

2.2 Problemas a Resolver

- Falta de um sistema unificado para controle de desempenho e crescimento profissional;
- Permitir a extração de relatórios organizacionais;
- Ausência de um meio eficiente para administrar unidades organizacionais separadas;
- Falta de ferramentas para definir e monitorar KPIs;
- Dificuldade na geração de dashboards e relatórios de análise de performance.

2.3 Modelo de KPIs

Para garantir um sistema robusto e adaptável à realidade de diferentes organizações, a solução proposta permite a definição e o acompanhamento de KPIs de forma flexível, considerando:

2.3.1 Tipos distintos de avaliação:

- Quanto maior, melhor (ex.: bugs corrigidos, funcionalidades entregues, cobertura de testes, contribuição para documentação);
- Quanto menor, melhor (ex.: bugs encontrados pós entrega, velocidade para resolução de problemas);
- Binário (sim/não) (ex.: cursos relacionados a área, certificados).

2.3.2 Origem da população dos KPIs:

- Preenchidos manualmente por gestores (ex.: relacionados ao sistema, como bugs, horas dedicadas);
- Preenchidos pelos próprios colaboradores, sujeitos a aprovação (ex.: cursos realizados, certificados).

2.3.3 Aplicação hierárquica:

- KPIs individuais (por colaborador);
- KPIs de equipe (controlados pelo gestor);
- KPIs do gestor (pessoais através de feedbacks de liderados, e baseados na performance da equipe).

3 Especificações Técnicas

Esta seção apresenta os detalhes técnicos envolvidos no desenvolvimento do Sistema de Análise de Performance Organizacional. São descritos os requisitos funcionais e não funcionais do sistema, as decisões de design adotadas, a arquitetura proposta e a stack tecnológica utilizada. O objetivo é documentar de forma clara os fundamentos técnicos que orientarão a implementação da solução, garantindo alinhamento com os objetivos do projeto e viabilidade de desenvolvimento.

3.1 Requisitos de Software

Requisitos Funcionais (RF)

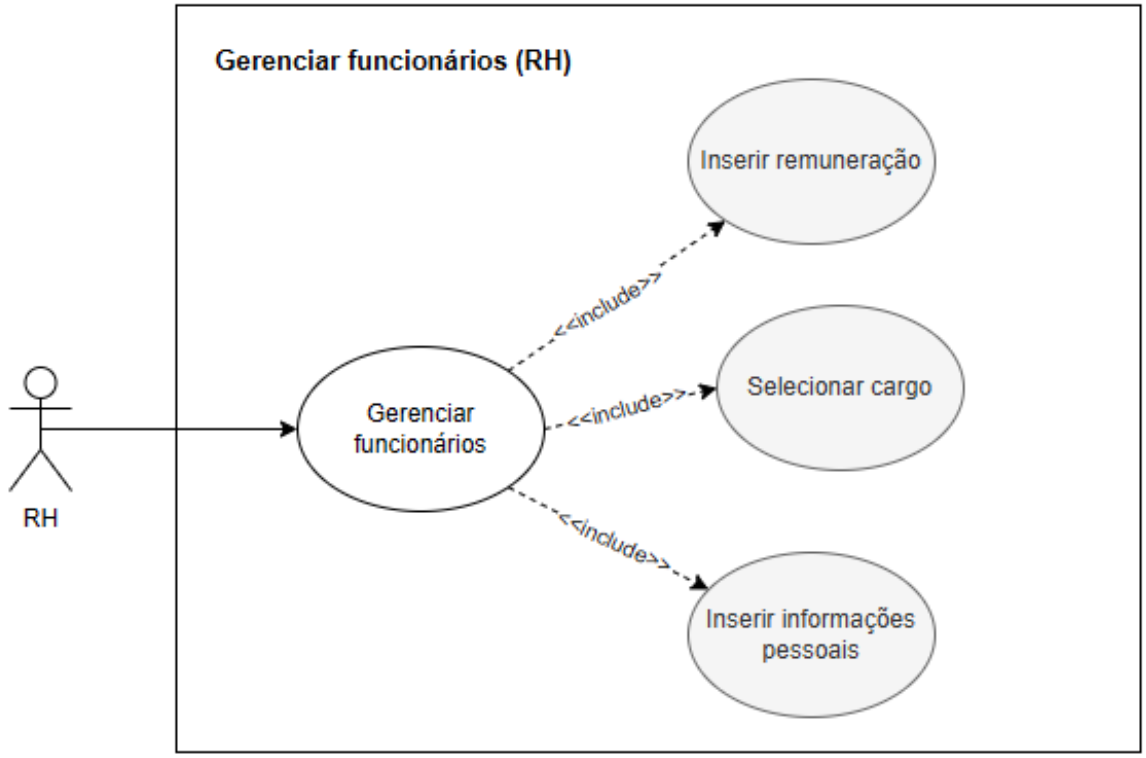
Código	Descrição
RF01	O sistema deve permitir a gestão de funcionários.
RF02	O sistema deve permitir a criação e gestão de equipes em estrutura hierárquica de árvore.
RF03	O sistema deve registrar avaliações de desempenho para funcionários e equipes.
RF04	O sistema deve permitir gerenciar planos de carreira dos colaboradores.
RF05	O sistema deve permitir a definição e acompanhamento de KPIs individuais, de equipe e organizacionais.
RF06	O sistema deve permitir a exportação da árvore hierárquica.
RF07	O sistema deve permitir gerenciar múltiplas filiais.
RF08	O sistema deve permitir a definição de KPIs personalizados para cada colaborador e equipe.
RF09	O sistema deve permitir a geração de dashboards dinâmicos para análise de desempenho.
RF10	O sistema deve permitir o cadastro de tipos de avaliação de KPIs (maior melhor, menor melhor, binário).
RF11	O sistema deve permitir a definição da origem do preenchimento dos KPIs (gestor, colaborador).
RF12	O sistema deve permitir a criação de workflow de aprovação para KPIs preenchidos por colaboradores.
RF13	O sistema deve permitir a definição KPIs específicos para gestores, tanto individuais quanto da equipe sob sua responsabilidade.
RF14	O sistema deve gerar uma interface para acompanhamento histórico dos KPIs por período.
RF16	O sistema deve permitir a avaliação do gestor por parte dos colaboradores.

Requisitos Não Funcionais (RNF)

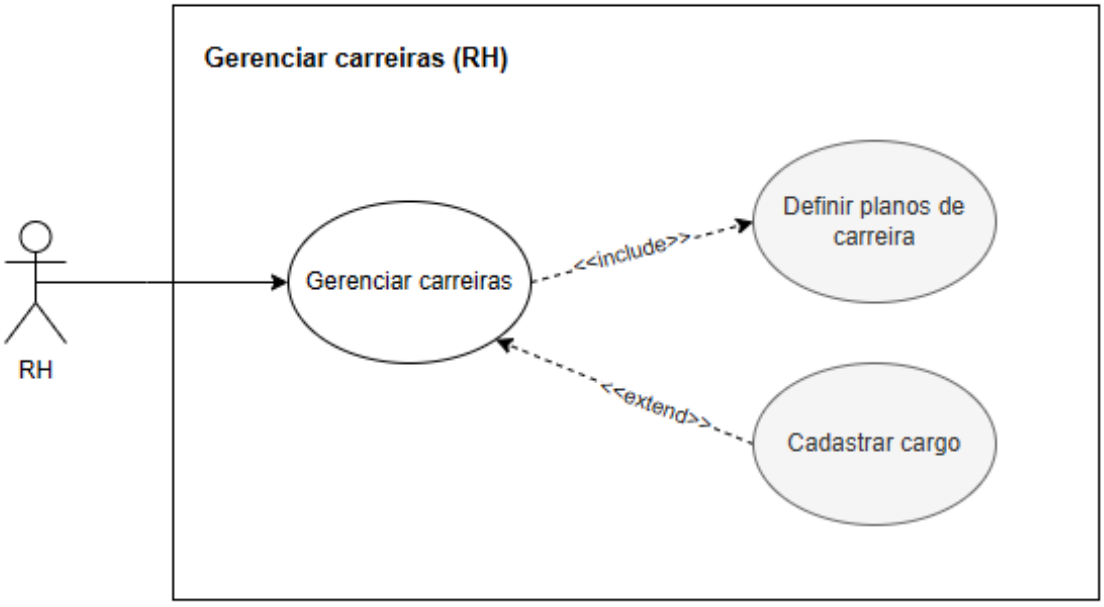
Código	Descrição
RNF01	O sistema deve ser disponibilizado como SaaS, acessível via navegador web sem necessidade de instalação local.
RNF02	O sistema deve ser responsivo e adaptável a diferentes dispositivos (desktop, tablet, smartphone).
RNF03	O sistema deve garantir a segurança dos dados com criptografia de senhas e comunicações (HTTPS).
RNF04	A plataforma deve ter alta disponibilidade (uptime superior a 99,5%).
RNF05	O sistema deve realizar backups automáticos do banco de dados ao menos uma vez ao dia.
RNF06	O sistema deve disponibilizar acesso via autenticação segura (login e senha, suporte a OAuth).
RNF07	O sistema deve ter capacidade de até 50 usuários simultâneos no sistema sem perda de desempenho.
RNF08	O sistema deve persistir todos os dados em um banco de dados.
RNF09	O sistema deve disponibilizar a visualização do dashboard com um tempo de resposta de até 1 segundo.
RNF10	O tempo de resposta do carregamento inicial da aplicação não deve ultrapassar 3 segundos.
RNF11	Dados sensíveis como salários e avaliações devem ser visíveis apenas a usuários com permissões exclusivas
RNF12	O tempo de resposta para requisições internas não devem ultrapassar 500ms com cache e 1 segundo sem cache.

3.2 Diagramas de Caso de Uso

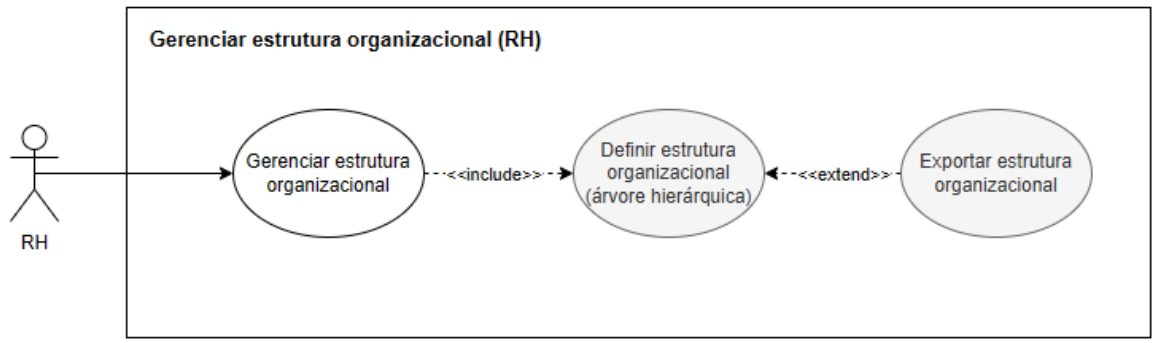
Gerenciar Funcionários (RH)



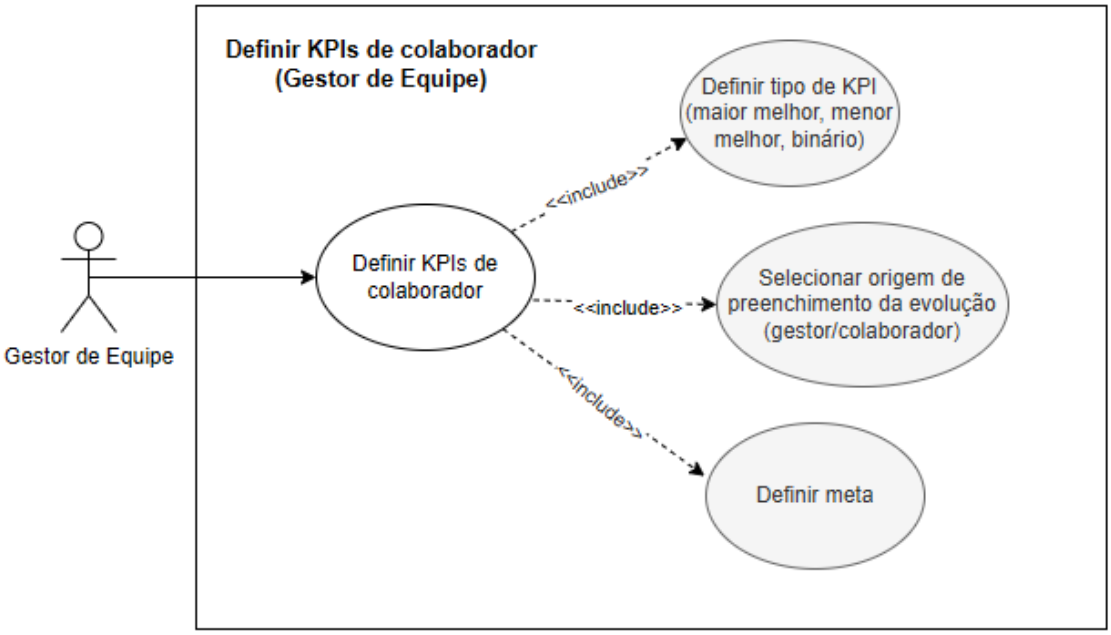
Gerenciar Carreiras (RH)



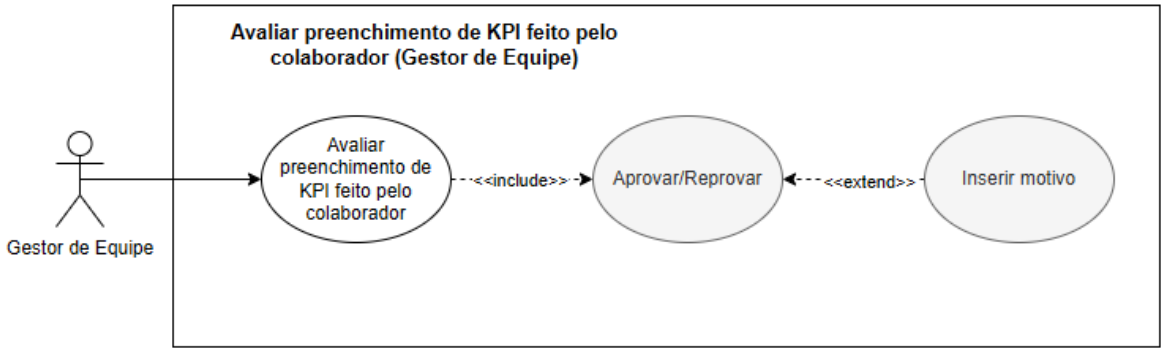
Gerenciar estrutura organizacional (RH)



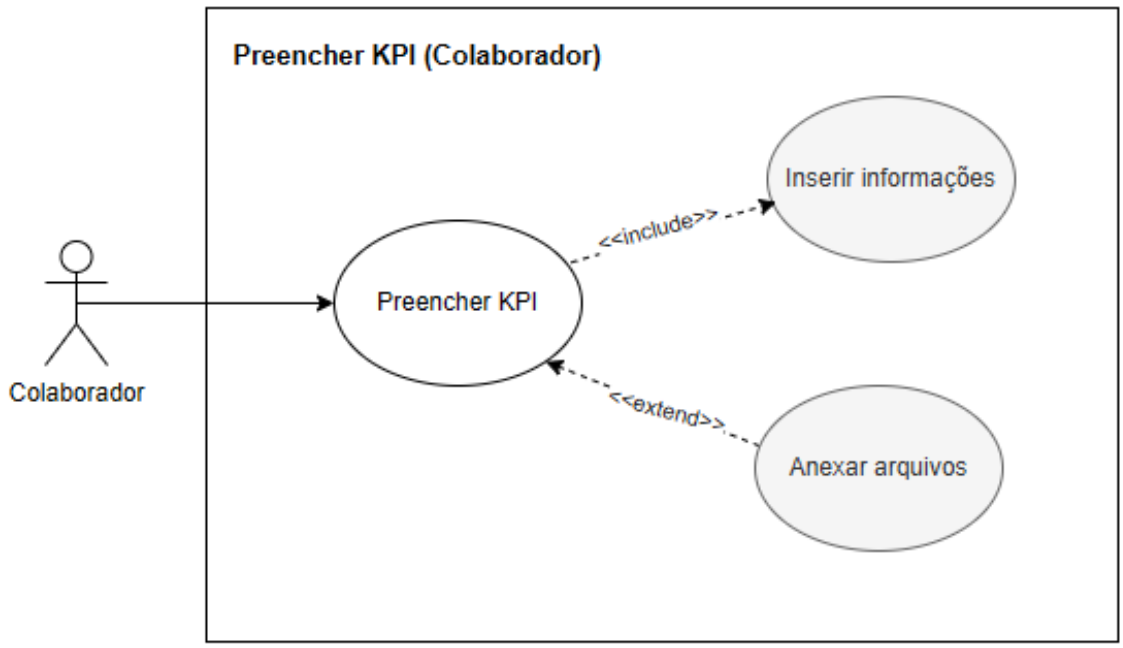
Definir KPIs de colaborador (Gestor de Equipe)



Avaliar preenchimento de KPI feito pelo colaborador (Gestor de Equipe)



Preencher KPI (Colaborador)



3.3 Considerações de Design

Discussão sobre as Escolhas de Design

O projeto adotará uma arquitetura monolítica, centralizando toda a aplicação em uma única base de código e banco de dados. Essa escolha foi feita visando a simplicidade de desenvolvimento, facilidade de integração inicial e manutenção de um sistema coeso.

Alternativas Consideradas:

- Microserviços: Embora escaláveis, microserviços foram descartados nesta fase pela complexidade adicional de deploy, orquestração e comunicação entre serviços.

Justificativas para as Decisões:

- Facilidade de Desenvolvimento: Trabalho em um único repositório, com integração contínua simples.
- Desempenho Inicial: Em ambientes com baixa quantidade de usuários e dados, o monolito apresenta boa performance.
- Custo de Deploy: Um servidor é suficiente para hospedar toda a aplicação (frontend + backend).

Visão Inicial da Arquitetura

Componentes principais:

- Frontend: Desenvolvido em React com TypeScript. A estrutura do projeto é organizada em pastas como pages (páginas da aplicação), components (componentes reutilizáveis), hooks (hooks customizados para consumo de APIs e regras de negócio), auth (fluxo de autenticação), lib (integrações e utilitários externos) e utils (funções de apoio), além da pasta test para testes de interface. A navegação é controlada pelo React Router, enquanto o consumo do backend em NestJS é feito por meio de chamadas HTTP encapsuladas em hooks, garantindo separação de responsabilidades, reutilização de código e uma experiência de uso fluida.
- Backend: As API Routes do NestJS servirão como endpoints RESTful para as funcionalidades de RH, incluindo cadastro de funcionários, avaliação de desempenho, gestão de KPIs e estrutura organizacional. Containers serão utilizados com Docker, permitindo reinício automático em caso de falhas. Os endpoints seguirão convenções REST, como:
  - GET /colaboradores/:id/kpis: retorna KPIs de um colaborador.
  - POST /kpis/:kpiId/avaliacao: permite que o gestor avalie um KPI.

  - GET /estrutura-organizacional: retorna a árvore hierárquica da empresa.

- Banco de Dados: O PostgreSQL será usado como sistema de banco de dados relacional. A estrutura será otimizada com uso de índices em colunas frequentemente consultadas (como colaboradorId, equipeId, periodo), e read replicas poderão ser configuradas para leitura paralela em ambientes com maior demanda. Será implementado versionamento de schema com TypeORM. Além disso, dados sensíveis serão protegidos com criptografia em repouso.

- Integração com Frontend: A comunicação entre frontend e backend ocorrerá por meio de APIs RESTful. O frontend irá consumir endpoints para renderizar componentes dinâmicos com base em dados atualizados. Por exemplo, dashboards e gráficos de performance serão construídos com dados obtidos via requisições assíncronas a endpoints como:

  - GET /dashboard/kpis/equipe/:equipeId
  - GET /historico/kpis?colaborador=123&periodo=2024Q1

  Para melhorar o desempenho, será adotado caching com Redis em endpoints críticos (como dashboards e dados estruturais). O cache terá expiração automática e fallback para o banco em caso de invalidação.

Todo o backend será responsável por:

- Gerenciamento de usuários e permissões
- Controle de hierarquia de equipes
- Avaliação de desempenho e KPIs
- Exportação de estrutura organizacional

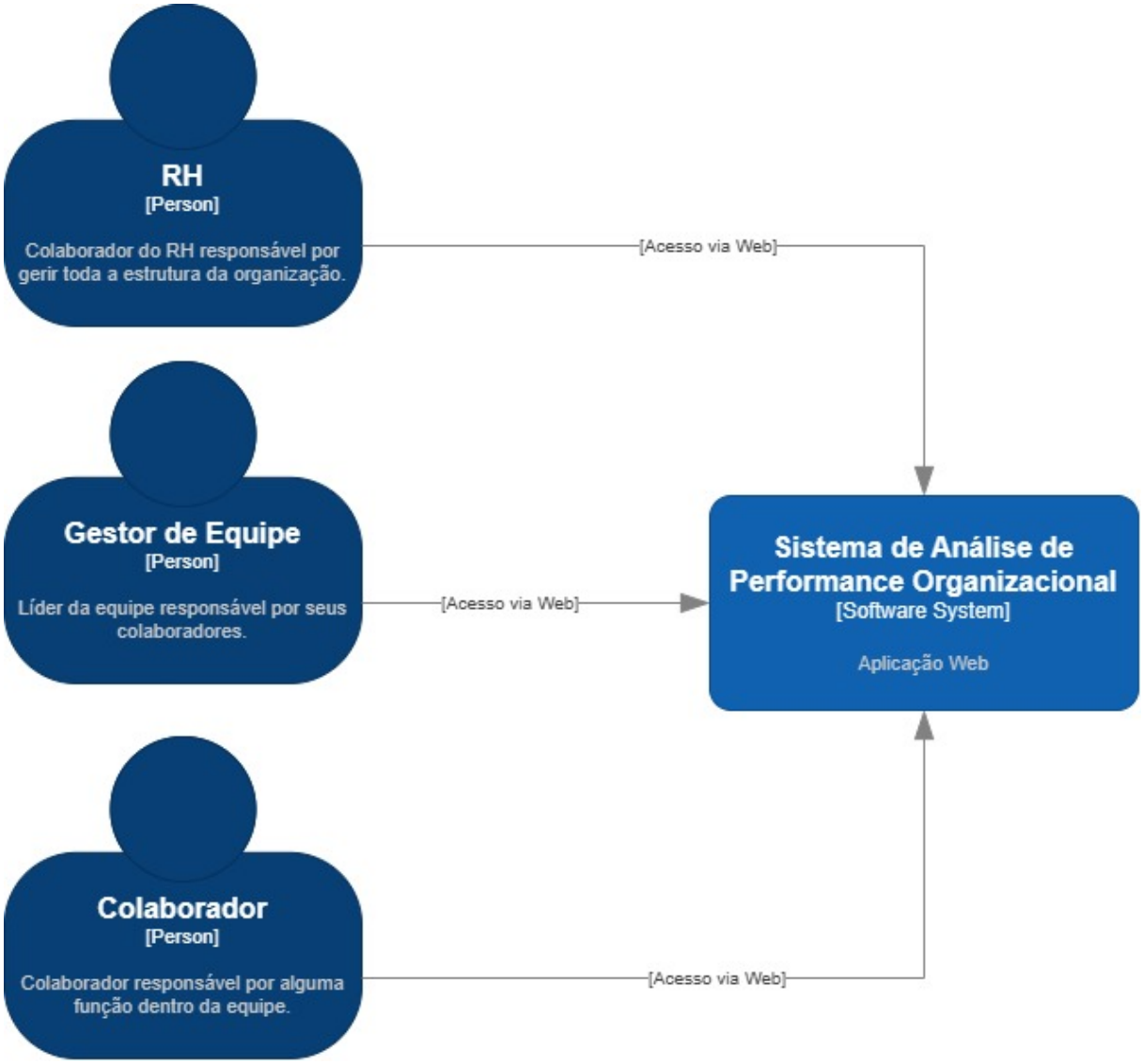
Padrões de Arquitetura

- Monolito Modularizado: O backend é um monólito organizado em módulos dentro de src (como auth, users, hr, kpi, team, org, person, reviews, geo, monitoring), cada um com seus próprios controllers, services, entidades, DTOs e testes.
- Arquitetura em camadas (Controller–Service–Repository):
- Controllers: expõem as APIs REST.
- Services: concentram as regras de negócio.
- Repositórios / database: acesso ao banco de dados isolado na camada de persistência (configurações e estratégia de acesso na pasta database), facilitando manutenção e reutilização.
- RESTful APIs: Para comunicação frontend-backend.

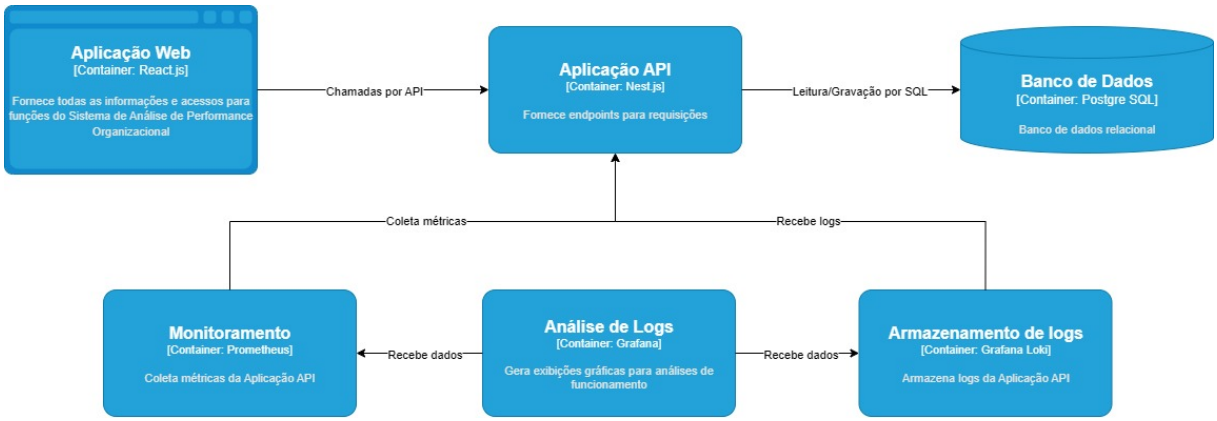
- Monitoramento e observabilidade: O módulo monitoring centraliza a exposição de métricas da aplicação (para Prometheus, Grafana etc.), permitindo acompanhar saúde e desempenho do sistema.

Modelos C4

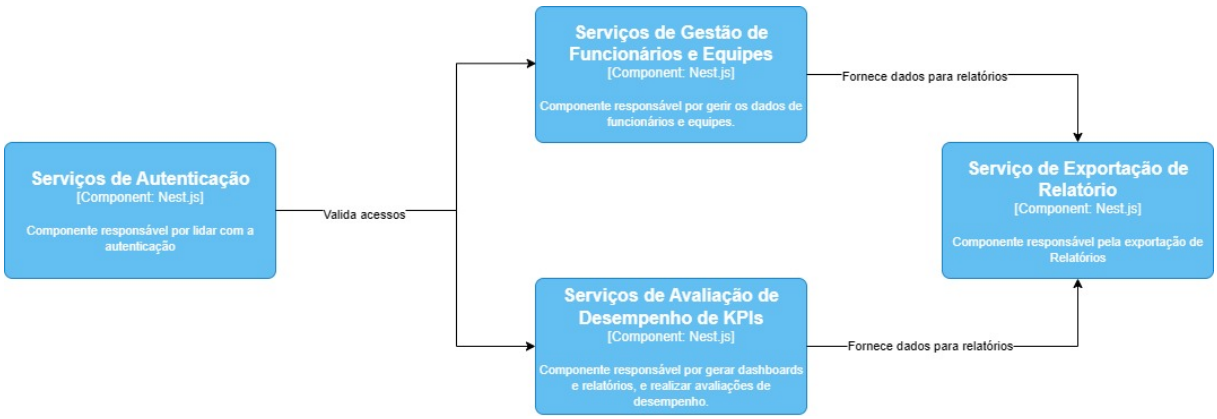
Contexto:



Containers:



Componentes:



3.4 Stack Tecnológica

Linguagens de Programação

JavaScript/TypeScript (Frontend e Backend):

Justificativa: O JavaScript é a linguagem de programação padrão para desenvolvimento web. É possível utilizar JavaScript tanto no frontend quanto no backend. A escolha por TypeScript no lugar de JavaScript para o backend oferece benefícios de tipagem estática, o que ajuda a detectar erros em tempo de desenvolvimento e torna o código mais fácil de manter e escalar.

Frameworks e Bibliotecas

NestJS (Backend):

Justificativa: O NestJS é um framework Node.js para construção de aplicações backend escaláveis, eficientes e de fácil manutenção Também promove uma estrutura organizada com suporte nativo a injeção de dependências, criação de módulos, controladores e serviços. Ele facilita a implementação de APIs REST e GraphQL, integra-se facilmente com bibliotecas como TypeORM.

React.js (Frontend):

Justificativa: O React é uma biblioteca de interface de usuário amplamente utilizada para construir interfaces de usuário interativas. Será utilizado para criar os componentes da interface web, possibilitando uma experiência de usuário dinâmica e responsiva.

TypeORM Migrations (ORM para PostgreSQL):

Troca: antes Prisma, agora TypeORM Justificativa: O TypeORM é um ORM maduro para Node.js e TypeScript, oferecendo integração nativa com NestJS. Ele facilita a modelagem das entidades, gerenciamento de relacionamentos e execução de operações de CRUD. Além disso, fornece suporte robusto a migrations, garantindo versionamento do esquema do banco de dados.

PostgreSQL em VPS Azure (Banco de Dados Relacional):

Troca: antes "PostgreSQL em Railway ou Neon", agora "PostgreSQL em VPS Azure" Justificativa: O PostgreSQL foi escolhido como sistema gerenciador de banco de dados relacional (SGBD) por sua robustez, conformidade com padrões SQL e escalabilidade adequada para aplicações web modernas. No contexto deste projeto, o banco de dados é hospedado em uma VPS na Azure, permitindo maior controle sobre configuração, segurança e recursos de infraestrutura.

- Ideal para modelar a hierarquia organizacional (ex.: relações entre colaboradores, gestores, filiais, KPIs e períodos) de forma consistente e normalizada.
- Permite ajuste fino de desempenho (tuning de parâmetros, índices, políticas de cache) e escalabilidade vertical de acordo com a demanda da aplicação.
- Integra-se de forma fluida com o ORM TypeORM, facilitando a definição de entidades, migrações e repositórios, além de possuir suporte a Docker e orquestração via *docker-compose* na própria VPS Azure.

TypeORM Migrations (Migrations para Banco de Dados):

Troca: antes "Prisma Migrate" Justificativa: O TypeORM fornece suporte a migrations, permitindo versionamento de esquemas e simplificando o processo de alteração e atualização do banco de dados ao longo do desenvolvimento.

Recharts (Visualização de Dados - Dashboards):

Justificativa: Recharts é uma biblioteca de gráficos construída especificamente para React, o que facilita sua integração com os componentes e o gerenciamento de estado da aplicação. Será utilizada para construir dashboards interativos que exibem métricas e KPIs de desempenho, permitindo análises detalhadas de funcionários e equipes. A biblioteca oferece alto grau de personalização e diversos tipos de gráficos (linhas, barras, pizza, radar, entre outros), o que favorece a adaptação das visualizações para diferentes tamanhos de tela e resoluções. No contexto deste projeto, serão aplicadas práticas de responsividade e redução da complexidade visual em dispositivos móveis, como o uso de gráficos simplificados, limitação da quantidade de dados exibidos por vez e carregamento assíncrono. Isso garante que os dashboards mantenham boa performance e legibilidade mesmo em telas menores, proporcionando uma experiência fluida tanto em desktops quanto em smartphones e tablets.

Ferramentas de Desenvolvimento e Gestão de Projeto

VSCode (Editor de Código):

Justificativa: O VSCode é uma das IDEs mais populares, oferecendo suporte completo para JavaScript, TypeScript, React, NestJS e TypeORM. Ele também tem integração com Git, o que facilita o versionamento e o gerenciamento do código.

Git/GitHub (Controle de Versão e Colaboração):

Justificativa: O Git é a ferramenta padrão para controle de versão, e o GitHub será utilizado para hospedar o repositório, facilitando a colaboração e o versionamento do código ao longo do desenvolvimento.

GitHub Actions (CI/CD):

Justificativa: Ferramenta de CI/CD integrada ao GitHub para automatizar os fluxos de integração e entrega contínuos. Com o GitHub Actions, é possível configurar pipelines para automação de testes, deploy e verificação de qualidade de código.

Docker (Containerização):

Justificativa: O Docker será utilizado para containerizar a aplicação, o que facilita a criação de ambientes consistentes e a implantação do sistema em diferentes servidores e plataformas. Isso também ajuda no isolamento do ambiente de desenvolvimento e produção.

Azure Database for PostgreSQL (Hospedagem do Banco de Dados):

Troca: antes Neon, agora Azure Database for PostgreSQL Justificativa: O Azure Database for PostgreSQL – Flexible Server será utilizado como banco de dados em nuvem, com alta disponibilidade, backups automáticos e escalabilidade configurável. Ele oferece:

- Total compatibilidade com TypeORM e frameworks NestJS.
- Pooling de conexões e suporte nativo a aplicações serverless hospedadas no Azure.
- Criptografia em repouso e em trânsito, garantindo conformidade com padrões de segurança corporativos.
- Integração nativa com Azure Monitor e Application Insights, facilitando o monitoramento de performance.

Jest (Testes Unitários e de Integração):

Justificativa: O Jest é um framework de testes JavaScript utilizado para escrever testes unitários e de integração. Ele garantirá que o sistema esteja funcionando conforme o esperado e ajudará a evitar regressões no código durante o desenvolvimento.

Postman (Testes de API):

Justificativa: O Postman será utilizado para testar as APIs desenvolvidas no NestJS. Ele permite simular requisições HTTP, verificar respostas e garantir que as integrações com o banco de dados e outros componentes estão funcionando corretamente.

Figma (Design e Prototipagem):

Justificativa: O Figma será utilizado para criar protótipos da interface de usuário (UI), garantindo que o design da aplicação seja bem planejado e alinhado com as necessidades do usuário. Ele permite colaboração em tempo real, o que é útil para equipes de desenvolvimento e design.

Azure App Service (Hospedagem da Aplicação):

Troca: antes Vercel, agora Azure App Service Justificativa: A aplicação frontend (React) e o backend (NestJS) serão hospedados no Azure App Service, que oferece suporte completo para aplicações Node.js e APIs. Essa mudança garante:

- Deploy automatizado integrado ao GitHub (CI/CD via GitHub Actions ou Azure DevOps).
- Escalabilidade horizontal e vertical sob demanda.
- Balanceamento de carga e distribuição global via Azure Front Door ou Azure CDN, otimizando latência.
- Suporte a containers (Docker) e ambientes isolados, ideal para aplicações full-stack modernas.

Prometheus + Grafana (Monitoramento)

Justificativa: O Prometheus foi selecionado por sua robustez na coleta de métricas em tempo real, possibilitando o acompanhamento detalhado de recursos como uso de CPU, memória, tempo de resposta de requisições e disponibilidade de serviços. Ele se integra facilmente a aplicações Node.js e NestJS, permitindo a instrumentação da aplicação com métricas customizadas que refletem o comportamento do sistema sob diferentes cargas de trabalho. Já o Grafana será utilizado como plataforma de visualização dessas métricas, possibilitando a construção de dashboards dinâmicos e interativos que facilitam o diagnóstico de problemas, a tomada de decisões operacionais e a identificação de tendências de uso.

Grafana Loki (Gerenciamento de logs)

Justificativa: Integração direta com o Grafana proporciona uma experiência unificada de visualização, onde métricas e logs podem ser analisados lado a lado em um mesmo painel. Essa abordagem aumenta significativamente a capacidade de resposta a falhas, uma vez que a correlação entre eventos e dados numéricos torna o processo de investigação mais ágil e eficaz. A escolha dessas ferramentas se justifica por sua maturidade, comunidade ativa, compatibilidade com Docker e facilidade de uso mesmo em ambientes de desenvolvimento com recursos limitados, como Railway ou Neon.

Jira (Gestão de projetos):

Justificativa: Ferramenta de gestão de projetos utilizada para acompanhar o progresso do desenvolvimento, planejar sprints e gerenciar tarefas. O Jira será usado para criar e acompanhar as histórias de usuários, tarefas e bugs ao longo do desenvolvimento do projeto.

3.5 Considerações de Segurança

Ao desenvolver um sistema de gestão de recursos humanos (RH), é crucial implementar medidas de segurança para proteger os dados sensíveis dos colaboradores e a integridade do sistema. Abaixo estão algumas das principais questões de segurança que devem ser consideradas, junto com as estratégias para mitigá-las.

Proteção contra Ataques de Autenticação

Problema:

A autenticação inadequada pode permitir que usuários não autorizados acessem o sistema.

Mitigação:

Autenticação com JWT (JSON Web Tokens): Utilizar JWT para autenticação e OAuth para delegação de autorização. A utilização de tokens de curto prazo (expiração) e renovação automática (refresh tokens) ajuda a prevenir o uso indevido de tokens expirados.

Autenticação Multifatorial (MFA): Implementar autenticação multifatorial para aumentar a segurança no login dos usuários.

Proteção de Senhas: Utilizar uma política de senhas fortes (mínimo de caracteres, mistura de letras, números e caracteres especiais) e armazená-las de forma segura utilizando hashing com bcrypt.

Exposição de Dados Sensíveis

Problema:

Armazenamento ou transmissão de dados sensíveis, como informações pessoais dos funcionários, sem criptografia pode expor o sistema a vazamentos de dados.

Mitigação:

Criptografia de Dados: Utilizar HTTPS (TLS/SSL) para criptografar a comunicação entre o cliente e o servidor, garantindo que dados como senhas e informações pessoais não sejam transmitidos em texto claro.

Criptografia de Dados em Repouso: Armazenar dados sensíveis (como senhas, documentos e informações financeiras) em formato criptografado no banco de dados, utilizando algoritmos robustos como AES.

Controle de Acesso e Privilégios

Problema:

A concessão inadequada de permissões pode resultar em escalonamento de privilégios e acesso indevido aos dados.

Mitigação:

Princípio do Menor Privilégio: Implementar um controle de acesso baseado em funções (RBAC - Role-Based Access Control). Apenas usuários com os privilégios adequados devem ter acesso a informações sensíveis ou realizar ações específicas no sistema.

Revisão de Permissões: Realizar auditorias regulares e revisões de permissões de usuários para garantir que apenas pessoas autorizadas tenham acesso a determinadas funcionalidades.

Armazenamento e Análise de Logs de Auditoria

Para garantir a rastreabilidade e a auditoria das ações realizadas no sistema, será implementado o armazenamento seguro dos logs de auditoria utilizando o Grafana Loki. Essa ferramenta é uma solução gratuita e eficiente para coleta, armazenamento e consulta de logs textuais em larga escala.

Além disso, o armazenamento dos logs será realizado de forma segura e criptografada, garantindo a confidencialidade das informações armazenadas. O acesso aos logs será restrito a usuários autorizados, reforçando o princípio do menor privilégio.

Além das medidas técnicas de segurança, o sistema também será desenvolvido em conformidade com a **Lei Geral de Proteção de Dados (LGPD)**, garantindo que os dados pessoais dos colaboradores sejam tratados de forma ética, segura e transparente. Isso inclui a coleta mínima necessária de informações, a obtenção de consentimento explícito para o tratamento de dados sensíveis, o direito de acesso e exclusão dos dados pelo titular, bem como a implementação de registros de auditoria para rastrear o uso e acesso a informações pessoais. O sistema também oferecerá níveis de permissão para limitar o acesso de usuários a dados confidenciais, de acordo com seus papéis organizacionais, respeitando o princípio do menor privilégio.

4 Próximos Passos

Revisões e Ajustes nos Documentos (26/06/2025 a 06/07/2025)

Revisão contínua do documento: Realizar revisões no documento, melhorando a clareza e a coerência das ideias, com base no feedback recebido.

Refinamento da documentação técnica: Refinar a descrição dos requisitos, arquitetura e decisões de design para garantir que o documento esteja pronto para a aprovação.

Aprovação Formal (07/07/2025 a 10/07/2025)

Obter a assinatura de professores responsáveis para validar o projeto, certificando que todas as etapas teóricas foram corretamente cumpridas, e entregar primeira etapa do Portfólio.

Início do Portfólio II (11/07/2025)

Sprint 1 — Planejamento Detalhado e Setup Inicial (08/07/2025 a 28/07/2025)

- Definir backlog inicial do projeto no Jira, com tarefas, subtarefas e prioridades.
- Atualizar repositório no GitHub com estrutura inicial dos diretórios.
- Configurar ambiente local com Docker, NestJS, React, PostgreSQL e TypeORM.
- Elaborar protótipo de baixa fidelidade no Figma para validação da interface inicial.
- Apresentar cronograma e arquitetura técnica ao orientador para validação.

Marco 1: Cronograma validado + ambiente de desenvolvimento configurado.

Sprint 2 — Estrutura de Autenticação e Hierarquia (29/07/2025 a 13/08/2025)

- Implementar sistema de autenticação com JWT e controle de permissões (RBAC).
- Desenvolver módulo de cadastro e login com validação de credenciais.
- Estruturar entidades de usuários, equipes e hierarquias organizacionais no banco.
- Iniciar testes com Postman e Jest para garantir a segurança da autenticação.

Marco 2: Usuário consegue autenticar, ver sua equipe e posição na hierarquia.

Sprint 3 — Gestão de KPIs e Dashboards (14/08/2025 a 31/08/2025)

- Implementar rotas para criação, definição e avaliação de KPIs.
- Adicionar suporte a tipos de KPI (maior/melhor, menor/melhor, binário).
- Criar componentes de dashboard em React com dados reais via API.
- Integrar Recharts para gráficos interativos.
- Iniciar coleta de métricas com Prometheus.

Marco 3: Sistema funcional de KPIs + visualização inicial em dashboard.

Sprint 4 — Logs, Auditoria e Exportações (01/09/2025 a 21/09/2025)

- Configurar Grafana Loki para registro e consulta de logs de auditoria.
- Implementar exportação da árvore organizacional.
- Adicionar logs para ações críticas (login, avaliação, alteração de dados).
- Validar políticas de LGPD com logs criptografados e segregação por acesso.

Marco 4: Logs ativos e exportações funcionando com segurança garantida.

Sprint 5 — Testes e Ajustes (22/09/2025 a 31/09/2025)

- Testes integrados com Jest e validações manuais via Postman.
- Revisar documentação técnica, código e diagramas.
- Verificar conformidade do sistema com requisitos

Marco 5: Requisitos validados e sistema com cobertura de testes.

Sprint 6 — Otimizações de Desempenho e Caching (01/10/2025 a 21/10/2025)

- Otimizar consultas SQL com índices e joins eficientes.
- Reduzir tempo de resposta para páginas de dashboards (¡1s).
- Incluir carregamento assíncrono de dados e lazy loading em componentes React.

Marco 6: Sistema estável e com desempenho otimizado.

Sprint 7 — Testes de Usabilidade e Ajustes de Interface (22/10/2025 a 11/11/2025)

- Realizar sessões de teste com usuários simulados.
- Coletar feedback sobre a usabilidade dos principais fluxos (cadastro, KPI, dashboards).
- Melhorar layout responsivo e comportamento mobile-first.
- Ajustar textos, ícones e feedback visual de ações no frontend.

Marco 7: Interface validada com boa usabilidade.

Sprint 8 — Refino da Documentação e Preparação da Apresentação (12/11/2025 a 20/11/2025)

- Atualizar documentação técnica com prints, diagramas e estruturas finais.
- Revisar código e aplicar padronização (lint, formatação, nomeclaturas).
- Criar slides da apresentação de defesa do projeto.
- Ensaiar apresentação e alinhar tópicos técnicos com objetivos do TCC.

Marco 8: Sistema finalizado e documentação pronta.

Sprint 9 — Submissão Acadêmica e Revisão Final (21/11/2025 a 30/11/2025)

- Fazer submissão formal dos artefatos obrigatórios (PDF, código-fonte, documentação).
- Corrigir eventuais pendências indicadas por orientador ou banca.
- Testar deploy final em Azure com todos os serviços ativos.
- Backup completo do sistema e logs para garantia de integridade.

Marco 9: Projeto entregue e homologado academicamente.

Sprint 10 — Apresentação Final e Encerramento (01/12/2025 a 10/12/2025)

- Apresentar defesa do TCC para banca avaliadora.
- Coletar pareceres, registrar observações e responder a perguntas técnicas.
- Encerrar repositório público com README documentando arquitetura, deploy e uso.
- Realizar encerramento formal da disciplina Portfólio II.

Marco Final: TCC apresentado, encerrado e documentado.

5 Referências

Frameworks e Bibliotecas

- NestJS: Framework para desenvolvimento full-stack com React.
- React.js: Biblioteca para construção de interfaces de usuário dinâmicas.
- TypeORM: ORM para interação segura com o banco de dados PostgreSQL.
- JWT (JSON Web Token): Para autenticação e autorização segura.
- Recharts: Biblioteca para visualização de dados e geração de gráficos interativos.
- Material UI: Biblioteca de componentes React para interfaces consistentes e responsivas.
- bcrypt: Biblioteca para hashing de senhas.
- Docker: Para containerização e simplificação de ambientes de desenvolvimento e produção.

Ferramentas de Desenvolvimento e Gestão

- GitHub Actions: Para CI/CD (Integração Contínua e Entrega Contínua).
- Jira: Para gestão de projetos e controle de tarefas.
- VS Code: Editor de código utilizado para o desenvolvimento.
- PostgreSQL: Banco de dados relacional para o armazenamento de dados.
- Azure: Para o deploy do sistema na nuvem em ambiente de produção e banco de dados relacional gratuito.
- Postman: Para testes de API e controle de requisições HTTP.

Documentação e Referências

- Documentação oficial do NestJS: Para o desenvolvimento de aplicações full-stack.
- TypeORM Documentation: Para entender como implementar o ORM e fazer consultas seguras.
- JWT.io: Para entender como implementar autenticação segura com JWT.
- Recharts Documentation: Para aprender como criar gráficos e dashboards dinâmicos.

Artigos Acadêmicos

- Boon, C., Hartog, D., & Lepak, D. (2019). A Systematic Review of Human Resource Management Systems and Their Measurement. *Journal of Management*, 45, 2498–2537. Disponível em: https://journals.sagepub.com/doi/10.1177/0149206318818718

- Chalisa, R., & Prawitasari, D. (2024). Analysis of Employee Performance Indicators Using the Human Resource Scorecard Approach and Analytical Hierarchy Process. *Jurnal Ilmiah Manajemen Kesatuan*. Disponível em: https://jurnal.ibik.ac.id/index.php/jimkes/article/view/2391

- (2022). Effectiveness of Human Resource Information System on HR Functions of an Organization. *Management Dynamics*. Disponível em: https://managementdynamics.researchcommons.org/journal/vol15/iss2/6/

Assinaturas dos Orientadores/Professores

Edicarsia Barbiero Pillon

Claudinei Dias

Paulo Rogério Pires Manseira