****

**UNIVERSIDADE ESTADUAL DO CEARÁ - UECE**

**CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT**

**DOCUMENTO DE ESPECIFICAÇÃO**

**(Sistema de Gestão de Funcionários)**

**Fortaleza – CE**

**Setembro de 2023**

**SUMÁRIO**

[**1.** **INTRODUÇÃO** 3](#_Toc146174282)

[**2.** **REQUISITOS** 3](#_Toc146174283)

[**3.** **CASOS DE USO** 4](#_Toc146174284)

[**4.** **ARQUIETURA** 5](#_Toc146174285)

[**5.** **ANEXO** 5](#_Toc146174286)

[Código: 5](#_Toc146174287)

[Estrutura da tabela 5](#_Toc146174288)

1. **INTRODUÇÃO**

Documento de especificação do sistema de gestão de funcionários com Requisitos Funcionais, Não Funcionais, Caso de Uso e Anexos para link do Github com código fonte e script de criação das tabelas.

1. **REQUISITOS**

**REQUISITOS FUNCIONAIS**

**RF01: Adicionar Funcionário**

* + Os usuários devem poder adicionar novas tarefas à lista
  + Cada tarefa deve ter um um ID nome, cargo, telefone e data de admissão

**RF02: Listar Funcionário**

* + Os usuários devem poder visualizar a lista de tarefas existentes.
  + A lista de tarefas deve mostrar o título e a descrição de cada tarefa.

**RF03: Atualizar Funcionário**

* + Os usuários devem poder atualizar o título e a descrição de uma tarefa existente.
  + A atualização de uma tarefa deve ser identificada pelo seu ID exclusivo.

**RF04: Remover Funcionário**

* + Os usuários devem poder remover uma tarefa da lista.
  + A remoção de uma tarefa deve ser identificada pelo seu ID exclusivo.

**REQUISITOS NÃO FUNCIONAIS**

**RNF01 - Usabilidade:**

* + A interface do usuário deve ser intuitiva, com fácil navegação e compreensão das funcionalidades oferecidas.
  + As mensagens de erro devem ser claras e informativas para auxiliar o usuário a entender e corrigir problemas.

**RNF02 - Manutenibilidade**

* + O código do sistema deve seguir boas práticas de programação, facilitando a manutenção futura e incorporação de novos recursos.
  + Deve haver uma documentação abrangente que explique a arquitetura, o código e os procedimentos de manutenção.

**RNF03 - Confiabilidade**

* + O sistema deve ser confiável, minimizando falhas e erros inesperados.
  + Deve ser implementado um mecanismo de backup e recuperação para garantir a integridade dos dados em caso de falha do sistema.

**RNF04 - Desempenho:**

* + O tempo de carregamento das páginas no frontend não deve exceder um limite aceitável, mesmo em condições de carga máxima.
  + O sistema deve oferecer boa responsividade ao usuário, garantindo tempos de resposta aceitáveis para todas as operações.

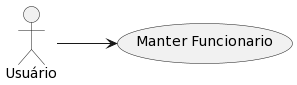
**RNF05 - Arquitetura de Três Camadas**

* + O sistema deve ser implementado seguindo uma arquitetura de três camadas, garantindo a separação clara das camadas de apresentação, lógica de negócios e acesso a dados. Isso proporcionará uma estrutura organizada e facilitará a manutenção e expansão do sistema.

**RNF06 – Tecnologias e Ferramentas**

* O sistema deve ser desenvolvido para Web utilizando React e tecnologias associadas no Frontend e Java como plataforma e linguagem de programação no BAckend
* O sistema deve ser desenvolvido usando as versões específicas das tecnologias recomendadas, incluindo o SQLite 3.43.0.0, Spring Boot 2.7.15 e React. A equipe de desenvolvimento deve manter essas versões durante todo o ciclo de vida do projeto.
* As tecnologias selecionadas, como o SQLite e o Spring Boot, devem ser capazes de oferecer o desempenho necessário para lidar com a quantidade prevista de dados e transações no sistema.
* As tecnologias selecionadas devem ser compatíveis com as IDEs e ferramentas de desenvolvimento recomendadas, como Eclipse STS, DBeaver, Postman e VSCode, para garantir uma integração suave do fluxo de trabalho de desenvolvimento.
* A integração entre o backend (Spring Boot) e o frontend (React) deve ser facilitada pelas tecnologias recomendadas, como REST e JSON, para permitir a transferência eficiente de dados entre as camadas.

1. **ESPECIFICAÇÃO - CASOS DE USO**



**Caso de Uso: Manter Funcionário**

**Ator Principal:** Usuário

**Resumo:** Este caso de uso descreve como o usuário pode criar, listar, atualizar, remover funcionários.

**Pré-condições:**

* O usuário deve estar autenticado no sistema (se a autenticação for um requisito não funcional).
* Existem tarefas cadastradas (para os casos de atualização, remoção e marcação de conclusão).

**Fluxo Básico:**

1. O sistema exibe a lista de tarefas disponíveis, incluindo opções para criar, listar, atualizar, remover e marcar tarefas como concluídas.
2. **Criar Funcionário:**

* O usuário escolhe a opção "Cadastras Funcionário".
* O sistema solicita o título e a descrição da nova tarefa.
* O usuário fornece o título e a descrição.
* O sistema cria a nova tarefa e a adiciona à lista de tarefas.

1. **Listar Funcionários:**

* O usuário escolhe a opção "Listar Tarefas".
* O sistema exibe a lista de tarefas, mostrando o título e a descrição de cada tarefa.

1. **Atualizar Funcionários:**

* O usuário escolhe a opção "Atualizar Tarefa".
* O sistema solicita o ID da tarefa que deseja atualizar.
* O usuário fornece o ID da tarefa.
* O sistema exibe a tarefa existente e solicita os novos valores para o título e a descrição.
* O usuário fornece os novos valores.
* O sistema atualiza a tarefa com os novos valores.

1. **Remover Tarefa:**

* O usuário escolhe a opção "Remover Tarefa".
* O sistema solicita o ID da tarefa que deseja remover.
* O usuário fornece o ID da tarefa.
* O sistema verifica se a tarefa existe.
* Se a tarefa existe, o sistema a remove da lista de tarefas.

**Pós-condições:**

* As operações de criação, atualização, remoção e marcação de conclusão de tarefas são refletidas na lista de tarefas.
* O sistema exibe mensagens de sucesso ou erro após cada operação.

**Exceções:**

* Se o usuário fornecer informações inválidas durante a criação ou atualização de tarefas, o sistema exibirá uma mensagem de erro.
* Se o usuário tentar atualizar ou remover uma tarefa que não existe, o sistema exibirá uma mensagem de erro.
* Se ocorrerem falhas no sistema, o usuário será notificado com uma mensagem de erro genérica.

**Requisitos Associados:**

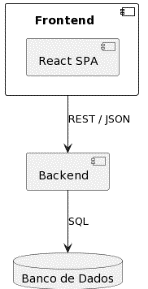
* RF1: Adicionar Tarefa
* RF2: Listar Tarefas
* RF3: Atualizar Tarefa
* RF4: Remover Tarefa
* RF5: Concluir Tarefa

1. **ARQUIETURA**

**Backend:** A Arquitetura de 3 Camadas é um padrão arquitetural que divide uma aplicação em três camadas distintas: Apresentação, Negócio e Persistência. Cada camada tem uma responsabilidade específica na aplicação.[Modelo Geral das Camadas:](https://github.com/UeceComp-2022/Exemplos_PCC/tree/main/Exemplo3ArquieturaTresCamadas#modelo-geral-das-camadas-na-arquitetura-de-3-camadas)

* **Camada de Apresentação**: Responsável pela interação com o usuário, exibição de informações e coleta de entrada do usuário.
* **Camada de Negócio**: Contém a lógica de negócios da aplicação, incluindo regras de negócios e manipulação de dados.
* **Camada de Persistência**: Responsável pelo acesso aos dados e interação com sistemas de armazenamento de dados, como bancos de dados.

**Frontend:** Esta aplicação é um exemplo de arquitetura de Single Page Application (SPA), representando uma página simples, um MVP (Produto Mínimo Viável), projetada para testar a interação com o backend. Utilizando React, ela integra gerenciamento de estado e comunicação assíncrona para operações CRUD, destacando a capacidade de manipulação dinâmica de dados de funcionários. A renderização condicional é empregada para interações de usuário na mesma página, e o layout, definido com CSS, é responsivo. Essa estrutura enfatiza a simplicidade e a eficácia em proporcionar uma interação fluida e reativa com o backend em um ambiente de teste**.**

**Integração**: o estilo arquitetural REST (Representational State Transfer) é utilizado na comunicação entre o frontend e o backend. A estrutura baseada em React permite realizar operações CRUD através de chamadas HTTP assíncronas, seguindo os princípios do REST, que incluem a simplicidade de interface e a comunicação stateless. Isso significa que cada requisição do frontend para o backend é autocontida, com todas as informações necessárias para processar a solicitação. Essa abordagem garante uma integração eficiente e organizada, crucial para testar e validar as funcionalidades da aplicação em um ambiente controlado.

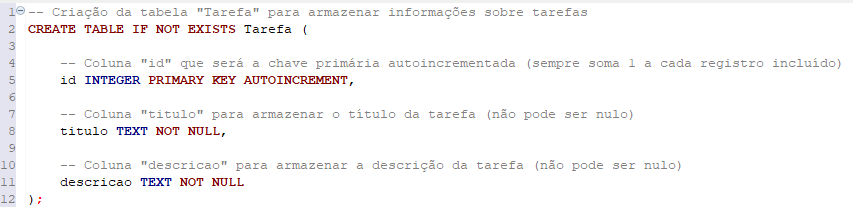
1. **ANEXO**

* **Código:**

<https://github.com/meu-projeto>

* **Estrutura da tabela**

**Conteúdo do** [script.sql](https://github.com/UeceComp-2022/Exemplos_PCC/blob/main/Exemplo3ArquieturaTresCamadas/script.sql):

****