## Histórico de Alterações:

| Data: | Autor: | Versão: | Descrição: Descrição: |
|---|---|---|---|
| 23/08/2024 | Jean lukas | 0.0 | Criação do documento de, identificação dos requisitos descrição dos requisitos funcionais. |

| Data: | Revisão: | Versão: | Descrição: |
|---|---|---|---|
| 23/08/2024 | Gabriel Bravo | 1.0 | |

---

## stakeholders:

- Engenheiro de Software (Autor do documento e responsável pelo funcionamento do sistema de acordo com os requisitos fornecidos)
- Cliente (Parte interessada no sistema, e fornecedor de informações para melhor clareza no documento)

## 1. REQUISITOS FUNCIONAIS:

**1.1 -** Sistema de gerenciamento de tarefas para equipes remotas.

O sistema permitirá que os usuários criem, editem, organizem e acompanhem tarefas, além de oferecer funcionalidades de colaboração e integração com ferramentas de comunicação.

1.2 - Criação de tarefa:

O sistema terá um menu com a opção de criação de tarefa, com essa opção terá a criação de tarefas voltadas ao usuário.

1.3 - Edição das tarefas:

A opção de edição ficara disponível no menu principal para que o usuário possa editar quando quiser. Assim que o usuário abrir a edição de tarefas ele poderá.

. Mover tarefas com base em sua importância

. Excluir tarefas

. Renomear tarefas

1.4 - Acompanhar tarefas:

No menu também terá a ferramenta de acompanhar tarefas. Quando acessado você poderá escolher qual tarefa você deseja acompanhar. Isso te permitirá ter acesso a tarefa dos integrantes da sua equipe, podendo também criar comentários para avaliação da tarefa.

1.5 - Envio de relatório:

O sistema deve permitir que os usuários exportem relatórios de tarefas em formatos como PDF ou Excel. O usuário seleciona um formato e as tarefas são exportadas de acordo com os filtros aplicados.

1.6 - Integração com ferramentas de comunicação:

O sistema deve se integrar com ferramentas de comunicação como Slack ou Microsoft Teams, para que os usuários possam discutir tarefas dentro dessas plataformas.

1.7 - Atribuir tarefas aos usuários:

A atribuição de tarefas aos usuários ficara disponível para os organizadores da equipe. Nessa aba será possível pesquisar o nome do usuário e adicionar uma tarefa no seu perfil, essa ação será notificada ao membro da equipe, possibilitando a realização e visualização da tarefa.

1.8 - Envio de notificações automáticas:

Todas as ações que forem atribuídas por um organizador a um usuário serão notificadas automaticamente ao mesmo, de modo que seja possível ao usuário clicar na notificação e ser direcionado diretamente a ela. Também notificação para tarefas que estão muito tempo sem ser realizadas estarão disponíveis no sistema.

1.9 - Cadastramento do Usuário:

Ao iniciar o sistema terá a possibilidade de cadastramento dos usuários.

1.10 - Calendário:

Fornecera um calendário que terá as datas de realização de cada tarefa.

1.11 - Histórico de Edições:

O sistema deve manter um histórico das edições feitas em cada tarefa. Os usuários podem visualizar o histórico de mudanças de uma tarefa, incluindo quem fez a alteração e quando.

1.12 - Visualização de Tarefas por Status:

O sistema deve permitir que os usuários filtrem as tarefas por status, como "Pendente", "Em Andamento", e "Concluída". O usuário escolhe um filtro, e a lista de tarefas é atualizada para mostrar apenas as tarefas com o status selecionado.

**2. REQUISITOS NÃO FUNCIONAIS:**

2.1 - Desempenho do sistema:

O sistema deverá ser capaz de suportar vários usuários utilizando o sistemas e organizar no mínimo 1000 usuários simultâneos, e agrupar dados de maneira eficiente com um tempo de resposta de 2 segundos.

2.2 - Segurança:

*O sistema deve implementar criptografia para proteger os dados e registro. Deve ter controle de acesso robustos para garantir que apenas autorizados, possam acessar e modificar informações. O sistema deve fazer uma cópia da segurança dos dados semanalmente.*

*2.3* - Usabilidade:

O sistema deve ter uma interface amigável e fácil de usar e com um design responsivo para dispositivos móveis e computadores.

*2.4* - Compatibilidade:

O sistema deve ser compatível com os dispositivos moveis ou desktops definidos pelo(a) Usuário como Chrome, Firefox, Edge e dispositivos desktop, tablet, mobile e fornecer adaptabilidade de interface para cada dispositivo.

*2.5 - Manutenção:*

O sistema deve ser modular, facilitando a manutenção, atualizando semanalmente e a adição de novas funcionalidades.

2.6 - Escalabilidade:

O sistema deve ser escalável para acompanhar o aumento no número de usuários e tarefas.

2.7 - Disponibilidade:

O sistema deve estar sempre ativo, com um tempo de funcionamento alto, (uptime) de 99,9% ou mais, e deve ser capaz de lidar com falhas de componentes sem interrupção no serviço.

**3. - Perguntas e Melhorias:**

3.1 - Perguntas ao Usuário:

Criação de tarefa:

Quais são os tipos de tarefas?

Todos poderão ter acesso a essa ferramenta?

Edição das tarefas:

Todos poderão ter acesso a essa ferramenta?

Integração com ferramentas de comunicação:

Quais ferramentas de comunicação serão usadas?

Por quem será usada?

Cadastramento do Usuário:

Será por Email, por telefone ou CPF o cadastramento do usuário?

Terá um Email diferente caso haja um Organizador para a equipe?

Caso tenha um organizador será disponibilizado um Email específico para este usuário?

3.2 - Melhorias:

Opção de recuperação ou redefinição de senhas:

Caso o usuário tenha perdido a sua senha ou queira mudá-la será fornecido uma função que lhe permita essas duas ações.

Perfil:

Adicionar foto, redes sociais, no perfil.

Desempenho:

Otimização do sistema para melhor usabilidade dos usuários.

## Linguagens de Programação e Frameworks 1.

**JavaScript/TypeScript (com Node.js)**

**Vantagens**

Versatilidade: Pode ser usado tanto no frontend quanto no backend. Grande Ecossistema: Ampla gama de bibliotecas e frameworks disponíveis. Alta Performance em I/O: Node.js é excelente para aplicativos que precisam de alta performance em operações de entrada/saída, como servidores de APIs em tempo real. Suporte a TypeScript: Adiciona tipagem estática, reduzindo erros comuns. Comunidade Ativa: Grande base de desenvolvedores e suporte contínuo.

**Desvantagens:**

Single-Threaded: Pode enfrentar dificuldades em tarefas computacionalmente intensivas. Callback Hell: Uso excessivo de callbacks pode tornar o código difícil de manter (embora isso seja mitigado com async/await). Menos Madura para Aplicações de Grande Escala: Comparado a linguagens como Java e C#. Melhor Uso: Aplicações web de alta performance, APIs em tempo real, microservices.

**2. Python**

**Vantagens**

Fácil de Aprender e Usar: Sintaxe simples e legível. Versatilidade: Amplo uso em ciência de dados, automação, desenvolvimento web, etc. Grande Ecossistema: Módulos e bibliotecas disponíveis para quase qualquer necessidade. Popularidade em IA e ML: amplamente utilizado em inteligência artificial e aprendizado de máquina.

**Desvantagens:**

Desempenho: É mais lento em comparação com linguagens compiladas como Java ou C#. Multithreading: GIL (Global Interpreter Lock) pode limitar a execução de threads em paralelo. Não ideal para Aplicações de Alta Performance: Em tarefas que exigem grande desempenho, outras linguagens podem ser preferíveis. Melhor Uso: Desenvolvimento rápido de protótipos, ciência de dados, scripts automatizados.

**3. Java**

**Vantagens:**

Portabilidade: "Write once, run anywhere" graças à JVM. Maturidade: Mais de duas décadas de evolução, com um grande ecossistema. Segurança: Forte suporte a segurança embutido na linguagem e no ecossistema. Performance: Bom desempenho e excelente gerenciamento de memória. Escalabilidade: Ideal para sistemas corporativos de grande escala.

**Desvantagens:**

Verbosidade: Código mais extenso em comparação com outras linguagens modernas. Tempo de Inicialização: Pode ser lento em inicialização comparado a Node.js ou Go. Curva de Aprendizado: Embora robusto, pode ser mais complexo para iniciantes. Melhor Uso: Aplicações corporativas, sistemas de grande escala, aplicativos móveis (Android).

**4. C# (com .NET)**

**Vantagens:**

Integração com o Ecossistema Microsoft: Excelente suporte e integração com produtos da Microsoft. Versatilidade: Pode ser usado para desenvolvimento de desktop, web, jogos (com Unity), entre outros. Performance: Boa performance, com suporte a multi-threading e paralelismo. Ferramentas de Desenvolvimento: Suporte de alto nível com IDEs como Visual Studio.

**Desvantagens:**

Dependência do Ecossistema Microsoft: Embora open-source, muitos desenvolvedores o associam com produtos da Microsoft. Portabilidade: Embora tenha melhorado, ainda pode ser limitado em plataformas não Windows. Melhor Uso: Aplicações empresariais, software desktop para Windows, desenvolvimento de jogos (Unity).

**5. Go (Golang)**

**Vantagens:**

Alta Performance: Muito rápido, com excelente suporte para concorrência. Simplicidade: Sintaxe simples e limpa, fácil de aprender. Compilado: Gera binários executáveis nativos, sem dependências externas.

Escalabilidade: Ótimo para microservices e sistemas distribuídos.

**Desvantagens:**

Menor Ecossistema: Comparado com Java ou Python, possui um ecossistema menor. Menos Suporte para OOP: Go é menos orientado a objetos, o que pode ser limitante para certos designs. Melhor Uso: Sistemas distribuídos, microservices, aplicações de alta performance.

## Bancos de Dados

**1. PostgreSQL**

**Vantagens:**

SQL Completo: Suporte completo a SQL e muitos recursos avançados (CTEs, JSONB, índices GIN/GiST). Extensibilidade: Suporta a criação de

tipos de dados personalizados, funções, operadores. Confiabilidade: Reconhecido por sua estabilidade e conformidade com ACID.

**Desvantagens:**

Complexidade: Pode ser mais complexo de gerenciar comparado a MySQL/MariaDB. Performance em Alta Escala: Em algumas situações, pode ser superado por bancos NoSQL. Melhor Uso: Aplicações que requerem consultas complexas, sistemas OLTP.

**2. MySQL/MariaDB**

**Vantagens:**

Popularidade: Amplamente utilizado, com vasto suporte em hospedagens.

Performance: Desempenho sólido em muitas situações, especialmente com otimizações. Facilidade de Uso: Mais simples de configurar e usar do que o PostgreSQL.

**Desvantagens:**

Funcionalidades Limitadas: Falta alguns dos recursos avançados do PostgreSQL. Menos Extensível: Menor suporte para tipos e funcionalidades customizadas. Melhor Uso: Aplicações web padrão, onde simplicidade e performance são chave.

**3. MongoDB**

**Vantagens:**

NoSQL: Flexível, ideal para dados não estruturados ou semi-estruturados. Escalabilidade Horizontal: Facilita a distribuição de dados em várias máquinas. Modelo de Documentos: Trabalha bem com JSON, muito usado em aplicações web modernas.

**Desvantagens:**

Consistência Eventual: Em cenários onde a consistência forte é necessária, pode não ser ideal. Transações Limitadas: Suporte a transações melhorou, mas ainda é inferior ao de bancos SQL.

Melhor Uso: Aplicações que requerem flexibilidade no esquema de dados, sistemas com necessidade de escalabilidade horizontal.

**4. SQLite**

**Vantagens:**

Leveza: Muito leve e fácil de embutir em aplicativos. Simplicidade: Sem necessidade de servidor, ideal para aplicativos menores ou desenvolvimento local. Desempenho Local: Ótimo para operações locais em dispositivos móveis ou aplicativos standalone.

**Desvantagens:**

Limitações de Escala: Não é adequado para aplicações com alta carga de dados ou usuários simultâneos. Funcionalidades Limitadas: Falta muitos dos recursos avançados de outros bancos de dados SQL.

Melhor Uso: Aplicativos móveis, desenvolvimento local, protótipos.

**5. Microsoft SQL Server**

**Vantagens:**

Integração com Ecossistema Microsoft: Ideal para ambientes Windows. Recursos Avançados: Amplo suporte para recursos de BI, OLAP, e integração com outras ferramentas Microsoft. Segurança: Suporte robusto a segurança e conformidade.

**Desvantagens:**

Custo: Pode ser caro, especialmente para grandes implementações. Dependência do Ecossistema Microsoft: Menor suporte para outros sistemas operacionais.

Melhor Uso: Aplicações corporativas, integração com sistemas Microsoft, ambientes que requerem suporte a BI e OLAP. 5) Escolha das Melhores Tecnologias Para o Desenvolvimento de Backend e API: Java/C#: Para sistemas corporativos de grande escala, com requisitos robustos de segurança e performance. Para Banco de Dados: PostgreSQL: Se a aplicação requer consultas complexas, suporte a JSON, e alta conformidade ACID.