DESCRIÇÃO DA ARQUITETURA DO SOFTWARE - ARCHITECTURE NOTEBOOK

Propósito e Visão Geral

O Sistema de Gerenciamento Kanban foi criado para tornar a gestão de projetos mais intuitiva e colaborativa, seguindo os princípios eficazes do método Kanban. Com um foco claro na organização visual, os usuários têm a capacidade de criar quadros Kanban, facilitando a visualização do fluxo de trabalho e aprimorando a eficiência na conclusão de tarefas.

O software oferece a funcionalidade de registro e autenticação de usuários, proporcionando serviços relacionados à conta, como visualização, edição e eliminação. No âmbito dos projetos, os usuários podem criar, visualizar, mover e eliminar quadros, cada um representando uma iniciativa específica.

A interação com os cartões dentro dos quadros é essencial para gerenciar tarefas individuais. Os usuários podem criar, visualizar, mover e eliminar cartões, representando atividades específicas em um quadro no estilo Kanban.

Para garantir um uso consistente, o sistema estabelece regras práticas, como permitir que apenas o criador de um quadro retire-o. A eliminação de um quadro resulta na exclusão dos cartões ligados a ele, e há um número máximo de cartões permitidos na coluna "Trabalho em Progresso".

Ao utilizar tecnologias modernas, como NodeJS, TypeScript, Prisma e SQLite, o Sistema de Gerenciamento Kanban oferece uma arquitetura eficiente e amigável. Com uma abordagem cuidadosa da metodologia Kanban, o software visa facilitar a gestão de projetos, promovendo uma colaboração eficaz entre as equipes.

Metas e Filosofia

A arquitetura do Sistema de Gerenciamento Kanban é pautada por princípios simples e eficientes, alinhados com o método Kanban. O foco dessa abordagem é tornar a aplicação fácil de usar, flexível para diferentes ambientes de negócios e ágil para otimizar o gerenciamento de projetos.

Princípios-Chave:

1. Simplicidade e Usabilidade: A arquitetura busca proporcionar uma experiência intuitiva, com um sistema fácil de usar.

2. Adaptação: A flexibilidade é um ponto-chave que permite ajustes a diferentes necessidades de negócios e integração com sistemas antigos, se necessário.

3. Eficiência de Desempenho: O objetivo principal é um sistema ágil, capaz de responder rapidamente mesmo em situações de grande demanda.

Metas Importantes:

1. Organização Intuitiva: A estrutura foi projetada para ser visualmente clara e fácil de entender para os usuários.

2. Personalização Fácil: Os usuários têm a capacidade de personalizar quadros e cartões para atender às suas necessidades específicas.

3. Manutenção Simplificada: A arquitetura foi pensada para facilitar a manutenção ao longo do tempo, com códigos limpos e documentação clara.

Desafios Apresentados:

1. Independência de Hardware: A arquitetura garante que o sistema seja independente de plataformas específicas, funcionando em diferentes ambientes.

2. Eficiência em Situações Difíceis: Foi implementada uma otimização para garantir que o sistema funcione bem, mesmo em condições desafiadoras, como picos de trabalho ou recursos limitados.

Suposições e Dependências do Projeto:

1. Experiência da Equipe:

- Suposição: A equipe conta com sua habilidade para implementar a arquitetura planejada e desejada.

- Dependência: É essencial que a equipe possua conhecimento em NodeJS, TypeScript, Prisma, Tailwind CSS, e outras tecnologias do projeto.

2. Disponibilidade de Recursos:

- Suposição: Espera-se que os recursos, como servidores, armazenamento e largura de banda, estejam sempre disponíveis.

- Dependência: A infraestrutura necessária precisa estar pronta e funcionando para um desenvolvimento sem problemas.

3. Integração com Interfaces Legadas:

- Suposição: O sistema pode precisar se conectar a sistemas mais antigos.

- Dependência: Deve-se entender e considerar as interfaces e protocolos desses sistemas ao projetar a arquitetura.

4. Qualidade dos Dados Iniciais:

- Suposição: Parte do pressuposto de que os dados iniciais, como informações de contas e quadros, estão corretos.

- Dependência: A precisão dos dados iniciais é fundamental para o funcionamento adequado do sistema.

5. Adoção do Método Kanban:

- Suposição: Acredita que os usuários seguirão o método Kanban para gerenciar seus projetos.

- Dependência: O sucesso do sistema depende da compreensão e aceitação dos usuários em relação ao Kanban.

6. Manutenção Contínua:

- Suposição: Prevê a necessidade de manutenção regular para correções e melhorias.

- Dependência: Deve garantir recursos suficientes para manter o sistema de forma eficaz.

7. Segurança dos Dados:

- Suposição: Coloca a segurança dos dados (informações de contas e quadros) como prioridade.

- Dependência: É crucial implementar e manter medidas robustas de segurança.

Referencias da Arquitetura e sua Implementação

As referências para a implementação da arquitetura do projeto incluem a documentação oficial do Node.js, que oferece orientações abrangentes sobre instalação, conceitos fundamentais e práticas recomendadas (https://nodejs.org/en/docs/). O TypeScript, superset do JavaScript, é abordado em (https://www.typescriptlang.org/docs), fornecendo recursos desde guias de início rápido até referências detalhadas de linguagem. O Prisma, utilizado como ORM, tem sua documentação completa disponível em (https://www.prisma.io/docs), abrangendo configuração inicial e tópicos avançados de modelagem de dados. O Tailwind CSS, framework de design de páginas web, e suas informações podem ser encontradas em (https://tailwindcss.com/docs), cobrindo instalação, utilização e personalização. O Visual Studio Code, editor de texto utilizado no projeto, pode ser explorado em

(https://code.visualstudio.com). O SQLite, banco de dados escolhido, possui uma documentação oficial que pode ser acessada em (https://www.sqlite.org/docs.html). O Express.js, framework para criação de projetos na web, acessado em (https://expressjs.com/). Além disso, a plataforma Notion, utilizada para gerar documentos, pode ser explorada em (https://www.notion.so).

Decisões, Restrições e Justificativas

1. Decisão: Utilização do NodeJS e TypeScript
   - Justificativa:
     - DO: Desenvolvimento eficiente e consistente.
     - DO: TypeScript adiciona tipagem estática, reduzindo erros durante o desenvolvimento.
     - DON'T: Evitar escolhas de tecnologia que poderiam prejudicar a produtividade.

2. Decisão: Implementação do Prisma ORM para Interação com o Banco de Dados SQLite
   - Justificativa:
     - DO: Facilita a manipulação de dados no banco de dados.
     - DO: Utilização de código TypeScript/JavaScript para operações no banco de dados.
     - DON'T: Evitar abordagens manuais que podem resultar em erros e ineficiência.

3. Decisão: Adoção do Framework Tailwind CSS para Design de Páginas Web
   - Justificativa:
     - DO: Facilita a criação de interfaces modernas e responsivas.
     - DO: Reduz o tempo de desenvolvimento com estilos pré-definidos.
     - DON'T: Evitar estilos manuais que podem levar a inconsistências de design.

4. Restrição: Número Máximo de Cartões na Coluna "Trabalho em Progresso"
   - Justificativa:
     - DO: Garantir um fluxo de trabalho equilibrado e evitar sobrecarga.
     - DO: Reflete a limitação física do trabalho que a equipe pode realizar simultaneamente.
     - DON'T: Evitar a sobrecarga de trabalho que pode resultar em atrasos e baixa eficiência.

5. Decisão: Utilização do Docker para Fazer o Conteiner

- Justificativa:

- DO: Simplifica a implantação e execução consistente do sistema em diferentes ambientes.

- DO: Melhora a escalabilidade e a consistência entre ambientes de desenvolvimento e produção.

- DON'T: Evitar inconsistências e dificuldades na implantação sem contêineres.

Mecanismos de Arquitetura

SQLite

O primeiro mecanismo arquitetural é o SQLite. Ele é usado para guardar as informações importantes do sistema Kanban de uma forma leve e eficiente. As tabelas, como Conta, Quadro e Cartao, são usadas para organizar os dados.

NodeJS

O segundo mecanismo é o ambiente de execução JavaScript dado pelo NodeJS. Ele ajuda a rodar o código JavaScript no servidor, sendo essencial para as operações no backend, como interagir com o banco de dados SQLite e lidar com solicitações HTTP.

TypeScript

O terceiro mecanismo é o TypeScript. Ele adiciona tipos ao JavaScript, ajudando a evitar erros durante o desenvolvimento. O TypeScript é usado para melhorar a solidez e clareza do código do sistema Kanban.

Tailwind CSS

O quarto mecanismo é o Tailwind CSS. Ele é usando para criar uma interface diferente e atual no frontend do sistema Kanban, fornecendo estilos predefinidos e utilitários que facilitam o desenvolvimento.

ExpressJS

O quinto mecanismo é o ExpressJS. Ele é usado para construir serviços web no backend do sistema Kanban, agilizando a comunicação entre o front e o back.

Principais Abstrações

1. Quadro: Onde os usuários jogam suas tarefas e projetos. Cada quadro tem colunas que mostram os estágios do trabalho.

2. Cartão: Representa tarefas ou coisas específicas em um quadro. Eles têm detalhes como descrição, status e quando foram criados.

3. Conta: É o perfil do usuário. Tem inserções como e-mail, nome e senha. Serve para entrar no sistema e usar tudo de um jeito personalizado.

4. Coluna: Colunas são como seções em um quadro. Cada uma representa um passo diferente no trabalho, tipo "A Fazer", "Em Andamento" e "Concluído".

5. Banco de Dados: Guarda informações importantes como dados de usuários, configurações, quadros e cartões.

Camadas ou estrutura arquitetônica

O Kanban usa uma arquitetura consistente baseada no padrão de camadas. Essa abordagem organiza o sistema em diferentes níveis, onde cada camada possui responsabilidades específicas e se comunica com outras camadas para funcionar corretamente.

Aqui as seguintes camadas:

1. Camada de Apresentação:

- Responsável por lidar com a interação do usuário e a apresentação dos dados.

- Inclui funcionalidades como registro, autenticação e interação com quadros e cartões.

2. Camada de Lógica de Negócios:

- Gerencia as regras de negócios e a lógica da aplicação.

- Implementa os serviços relacionados a contas, quadros e cartões.

3. Camada de Acesso a Dados:

- Lida com operações de banco de dados, incluindo a prototipagem do banco de dados utilizando SQLite.

- Responsável pela persistência e recuperação de dados relacionados a usuários, quadros e cartões.

4. Camada de Serviços:

- Inclui serviços como Prisma ORM, responsável pela interação com o banco de dados e a manipulação dos dados no nível de lógica de negócios.

5. Camada de Ferramentas e Frameworks:

- Engloba ferramentas e frameworks como NodeJS, TypeScript, ExpressJS, Tailwind CSS e Docker, fornecendo suporte técnico e facilidades para o desenvolvimento.

Visão de Arquitetura

As visões para descrever a arquitetura de software do sistema Kanban são as seguintes:

1. Visão Lógica: Descreve a estrutura e o comportamento das partes significativas da arquitetura, incluindo a estrutura de pacotes, interfaces críticas, classes, subsistemas e relações entre esses elementos.

2. Visão Operacional: Descreve os nodos físicos do sistema, destacando os processos, threads e componentes que operam nesses nodos físicos. Essencial para compreender a infraestrutura de implementação do sistema.

3. Visão de Casos de Uso: Lista ou representa graficamente os casos de uso que contêm requisitos arquitetônicos significativos, proporcionando insights sobre as interações entre usuários e sistema.

4. Visão de Prototipagem do Banco de Dados: Descreve a estrutura e o design do banco de dados, incluindo tabelas relevantes e suas relações, proporcionando uma compreensão detalhada da persistência de dados no sistema.

5. Visão de Ferramentas Utilizadas: Destaca as tecnologias e ferramentas específicas escolhidas para o desenvolvimento do sistema, oferecendo uma perspectiva sobre o ambiente de desenvolvimento e suas influências na arquitetura do sistema.