FEPI Faculdade de Ensino e Pesquisa de Itajubá

PROJETO EXTENSÃO CALENDÁRIO DE EVENTOS

AGENDA COLABORATIVA PARA EVENTOS COMUNITÁRIOS

ITAJUBÁ

2025

Justificativa e Objetivos do Sistema

Equipe: Carlos, Taila, Luana, Renan, Laryh

Justificativa

O projeto Agenda Colaborativa para Eventos Comunitários visa resolver a dificuldade que organizadores de eventos, como feiras, quermesses e pequenos comerciantes, enfrentam para gerenciar e divulgar suas atividades. Atualmente, a falta de uma plataforma centralizada e de fácil uso resulta em pouca visibilidade e coordenação ineficiente. A plataforma proposta oferece uma solução robusta e colaborativa, promovendo a interação e facilitando o acesso de interessados aos eventos.

Objetivos

- **Centralizar a Divulgação de Eventos Comunitarios:** Fornecer uma plataforma unificada onde organizadores podem cadastrar, editar e gerenciar seus eventos de forma intuitiva.
- **Aumentar a Visibilidade:** Melhorar a divulgação de eventos locais, permitindo que a comunidade encontre facilmente atividades de seu interesse.
- **Promover a Colaboração:** Criar um ambiente onde múltiplos usuários (organizadores e público) possam interagir, tornando o planejamento e a participação mais dinâmicos.
- **Facilitar o Acesso:** Oferecer uma interface amigável e acessível, que pode ser usada tanto por organizadores experientes quanto por iniciantes.

Requisitos do Sistema

Requisitos Funcionais

Estes são os recursos que o sistema deve ter:

- **Acesso e Autenticação:**
  - **Login de Usuário:** Permitir que usuários acessem o sistema com credenciais.
  - **Cadastro de Usuário:** Permitir que novos usuários se registrem na plataforma.

- **Gerenciamento de Eventos:**
    - **Cadastro de Evento:** Permitir que organizadores criem novos eventos, incluindo nome, descrição, data, horário, local e categoria.
    - **Edição de Evento:** Permitir que organizadores atualizem as informações de eventos existentes.
    - **Visualização de Eventos:** Exibir uma listagem e um calendário de eventos disponíveis.
    - **Busca e Filtros:** Permitir que usuários busquem eventos por nome, data, categoria ou localização.
- **Pagina inicial:**
    - **Página inicial:** Exibe eventos próximos do visitante.
    - **Cadastro de evento:** Permitir que organizadores criem posts (atualizações, fotos) sobre seus eventos.
    - **Reação a um Post:** Permitir que usuários reajam a posts.
    - **Interação com eventos:** Permitir que usuários favoritem um evento

Requisitos Não Funcionais

Estes são os critérios de qualidade do sistema:

- **Usabilidade:** A interface deve ser intuitiva e fácil de usar para todos os perfis de usuário.
- **Desempenho:** A plataforma deve ser rápida e responsiva, especialmente na visualização dos eventos e em operações de busca.
- **Segurança:** As informações dos usuários e eventos devem ser protegidas, com um sistema de autenticação seguro.
- **Escalabilidade:** A arquitetura do sistema deve ser capaz de lidar com um aumento no número de usuários e eventos sem comprometer o desempenho.
- **Compatibilidade:** A aplicação web deve ser compatível com os principais navegadores.

Arquitetura Proposta e Tecnologias

A arquitetura do projeto será baseada em microsserviços, separando a lógica de negócio em componentes menores e independentes.

- **Arquitetura:** Microsserviços
- **Backend: Node.js** com Express.js para a API.
- **Banco de Dados: MySQL** para persistência de dados estruturados, como informações de usuários, eventos e posts.

- **Mensageria: Kafka** será usado para lidar com eventos assíncronos, como o envio de notificações em tempo real. Isso garante que o front-end não fique aguardando a finalização de tarefas pesadas.
- **Cache: Redis** será utilizado para caching de dados frequentemente acessados (como listagens de eventos populares) e para gerenciamento de sessões, melhorando o desempenho da aplicação.
- **Orquestração de Containers: Docker** e **Docker Compose** serão fundamentais para garantir que o ambiente de desenvolvimento seja consistente e fácil de configurar para toda a equipe, como você já mencionou.
- **Controle de Versão: Git** para gerenciar o código-fonte, com o uso de um fluxo de trabalho de *branching* (e.g., Git Flow ou Feature Branching).
- **Metodologia: Scrum** para o gerenciamento ágil do projeto, com sprints, reuniões diárias e planejamento de backlog.

Descrição da API (Backend)

A API REST do backend, desenvolvida em Node.js, será o cérebro da aplicação, atuando como um intermediário entre o front-end e o banco de dados/serviços externos. Ela será responsável por toda a lógica de negócio.

Estrutura da API:

- **Camada de Roteamento:** Define os endpoints da API. Por exemplo, `/api/v1/events` para eventos, `/api/v1/users` para usuários, etc.
- **Camada de Controladores:** Recebe as requisições do roteador, processa a lógica de negócio e interage com a camada de serviço.
- **Camada de Serviços:** Contém a lógica principal de negócio e interage com a camada de modelos para acessar o banco de dados.
- **Camada de Modelos:** Define a estrutura dos dados e interage diretamente com o MySQL.

Fluxo de Funcionamento (Exemplo - Cadastro de Evento):

1. **Requisição do Front-end:** O front-end envia uma requisição **POST** para o endpoint `/api/v1/events` com os dados do novo evento.
2. **Validação:** A API Node.js valida os dados recebidos para garantir que estão corretos e completos.
3. **Processamento da Lógica:**
    - A API interage com o **MySQL** para criar um novo registro na tabela de eventos.

    - Após a criação, a API envia uma mensagem para um tópico do **Kafka**, por exemplo, `events-created`.
    - Outros serviços (como um microserviço de notificações) que estão "ouvindo" esse tópico no Kafka podem processar essa mensagem de forma assíncrona, por exemplo, para enviar uma notificação aos usuários interessados.
4. **Resposta:** A API Node.js retorna uma resposta de sucesso (HTTP 201 Created) para o front-end.

Comunicação com o Redis:

A API utilizará o **Redis** para:

- **Cache:** Antes de fazer uma busca no MySQL por uma lista de eventos populares, a API verificará se os dados já estão no Redis. Se estiverem, a resposta será muito mais rápida.
- **Sessões de Usuário:** Armazenar dados de sessão do usuário para manter o estado da autenticação, melhorando a performance em vez de consultar o banco de dados a cada requisição.