Desafio Técnico: Desenvolvedor Web FullStack

Contexto do Negócio

No modelo de cooperativismo, decisões importantes são tomadas em assembleias por meio de votação. Estamos desenvolvendo uma plataforma digital para modernizar esse processo, permitindo que associados participem de votações de forma remota, segura e transparente. Sua missão é construir o coração dessa plataforma.

Objetivo Geral

Desenvolver uma aplicação Web FullStack para gerenciamento de sessões de votação. A solução deve conter um **backend com uma API RESTful em Python**, um **frontend reativo em React** e um processo de **autenticação via token**.

Toda a aplicação (backend, frontend e quaisquer outros serviços) deve ser executada de forma integrada utilizando Docker.

Requisitos Funcionais

1. Autenticação

- **Backend:**
  - Deve haver um endpoint `POST /register` para um cadastro simples de usuário (ex: recebendo `name`, `cpf`, `password`).
  - Deve haver um endpoint `POST /login` que recebe as credenciais do usuário (`cpf`, `password`), valida-as e retorna um token de acesso (JWT é recomendado).
  - Todos os endpoints de gerenciamento de pautas e votos (exceto listagem geral) devem ser protegidos, exigindo um token de acesso válido no cabeçalho `Authorization: Bearer <token>`.
- **Frontend:**
  - Deve haver uma tela de login.
  - Após o login, o token recebido deve ser armazenado e enviado em todas as requisições para endpoints protegidos.
  - A interface deve se comportar de maneira diferente para usuários logados e não logados (ex: esconder botões de voto se o usuário não estiver autenticado).

2. Backend (API em Python)

- **Gerenciamento de Pautas:**
  - `POST /topics` - (Protegido) Cadastra uma nova pauta para votação.
  - `GET /topics` - (Público) Lista todas as pautas cadastradas.
- **Gerenciamento da Sessão de Votação:**
  - `POST /topics/{topic_id}/session` - (Protegido) Abre uma nova sessão de votação para uma pauta. A requisição pode receber `duration_minutes`; se não, a duração padrão é de 1 minuto.
- **Registro de Votos:**
  - `POST /topics/{topic_id}/vote` - (Protegido) Recebe o voto de um associado. O corpo deve conter o voto (`'Sim'` ou `'Não'`). O ID do usuário deve ser extraído do token de autenticação. Cada associado pode votar apenas uma vez por pauta.
- **Resultados da Votação:**
  - `GET /topics/{topic_id}/result` - (Público) Exibe o resultado da votação após o fechamento da sessão.

3. Frontend (React.js)

- **É obrigatório o uso de Redux** para o gerenciamento de estado da aplicação. Isso deve incluir o estado de autenticação (usuário logado, token) e o estado dos dados da aplicação (pautas, resultados, etc.).
- **Telas:**
    - **Dashboard:** Lista as pautas e seus status ("Aguardando Abertura", "Sessão Aberta", "Votação Encerrada").
    - **Tela de Votação:** Permite que um usuário autenticado vote em uma pauta com sessão aberta.
    - **Tela de Resultados:** Exibe a contagem final de votos.
    - **Tela de Login/Registro.**

Requisitos Técnicos (Obrigatórios)

- **Backend:** Python 3.x (uso de frameworks como Flask, FastAPI ou Django é permitido).
- **Frontend:** React.js e Redux.
- **Persistência de Dados:** Os dados devem ser persistidos em um banco de dados à sua escolha (ex: PostgreSQL, SQLite).
- **Containerização:** Todo o ambiente deve ser orquestrado com **Docker e Docker Compose**. A aplicação deve ser executável com um único comando.
- **Testes:** O backend deve conter testes unitários que validem as regras de negócio críticas.

Tarefas Bônus (Diferenciais)

- **Notificação em Tempo Real (MQTT):** Ao fechar uma sessão, o backend publica o resultado em um tópico MQTT. O frontend pode se inscrever para exibir o resultado instantaneamente.
- **Verificação de Associado (API Externa):** Antes de computar um voto, valide se o CPF do usuário (armazenado durante o registro) está apto a votar consultando a API externa `https://user-info.herokuapp.com/users/{cpf}`.
- **Backend em GoLang:** Como diferencial, desenvolva o backend em Go ao invés de Python.

Critérios de Avaliação e Entrega

- **O que será avaliado:** Arquitetura, separação de conceitos, lógica, código limpo, uso de Design Patterns, testes, tratamento de erros, documentação, organização dos commits e funcionalidade da containerização com Docker.
- **Dívidas Técnicas:** Se não conseguir implementar algo da forma ideal, documente em uma seção `DÍVIDAS TÉCNICAS` no seu `README.md`, explicando o porquê e qual seria a solução ideal.
- **Formato de Entrega:** Link para um repositório **público** no GitHub.
- **Prazo:** O desafio será enviado na sexta-feira ao final do dia, e a submissão do link deve ser feita até a segunda-feira seguinte, às 09h00 da manhã.