Teste para Desenvolvedor Full Stack (Redis + MySQL + Laravel + API + React + Next + MUI)

Introdução

Este teste avalia suas habilidades em trabalhar com grandes volumes de dados (big data), criar soluções performáticas para sincronização e manipulação de dados, e desenvolver interfaces responsivas e intuitivas com tecnologias modernas.

Cenário

Você foi contratado para implementar uma solução que:

1. Migre dados de uma API para um banco de dados MySQL.
2. Sincronize diariamente novos dados da API com o banco de dados.
3. Desenvolva um componente de interface para buscar e selecionar dados com Autocomplete.
4. Permita adicionar novas opções diretamente no banco de dados caso nenhuma corresponda à busca do usuário.

Detalhes da API

- Para este teste, utilize a API JSONPlaceholder. Ela fornece dados fictícios e permite testar operações com um volume considerável de registros.
- Documentação da API: https://jsonplaceholder.typicode.com/guide/.
- A API retorna registros no formato JSON. Considere utilizar o endpoint `/posts`, que possui os seguintes campos relevantes:
    - `id`: Identificador único (number).
    - `title`: Título do item (string).
    - `body`: Conteúdo do item (string).

Requisitos

Backend

1. **Migração Inicial**
    - Implemente uma rota CLI (`php artisan`) para:
        - Fazer requisições paginadas à API e salvar os dados no MySQL.
        - Evitar registros duplicados (baseie-se no campo `id`).
        - Cachear as requisições no Redis por 24 horas.

2. **Sincronização Diária**
    - Implemente um Job para:
        - Buscar novos registros da API diariamente e atualizar o banco de dados.
        - Registrar logs no banco com o número de registros processados.
3. **Endpoints RESTful**
    - Crie uma rota GET `/items` que permita:
        - Busca de itens por `title`.
        - Paginação.
    - Crie uma rota POST `/items` para adicionar novos registros ao banco de dados.
4. **Manifestações Kubernetes**
    - Forneça manifestos Kubernetes que contemplem:
        - Deployment do backend e do frontend.
        - Configuração de serviços (Services) e ingressos (Ingress) para expor os aplicativos.
        - Configuração de volumes persistentes, caso necessário.

Frontend

1. **Componente Autocomplete**
    - Implemente um componente React com MUI que:
        - Consuma o endpoint `/items` para exibir opções enquanto o usuário digita.
        - Renderize dados de forma performática, lidando com listas grandes (100.000+ registros).
        - Permita adicionar uma nova opção caso nenhuma corresponda à busca.
        - Envie a nova opção ao backend usando o endpoint POST `/items` e atualize a lista automaticamente.

Diferenciais (Não obrigatórios)

Docker e CI/CD

1. **Docker**
    - Forneça um `Dockerfile` funcional para o backend e o frontend.
2. **CI/CD**
    - Configure um pipeline de CI/CD que:
        - Realize build e testes automatizados.
        - Gere e publique imagens Docker em um registro de contêiner.
        - Faça o deploy automático nos manifestos Kubernetes fornecidos.

Documentação

1. **README.md**
    - Adicione documentação que inclua:
        - Descrição do projeto e sua arquitetura.

        - Instruções de instalação, configuração e execução.
        - Passos para configurar os pipelines de CI/CD e deploy no Kubernetes.
2. **Sphinx**
    - Utilize o Sphinx para produzir uma documentação profissional e detalhada do projeto, incluindo:
        - Documentação do backend (rotas, modelos, e Jobs).
        - Guia de configuração e uso do frontend.
        - Configurações e comandos do Kubernetes e CI/CD.

Testes Automatizados

1. **Backend**
    - Implemente testes unitários e de integração para garantir a funcionalidade correta dos endpoints.
2. **Frontend**
    - Implemente testes para o componente Autocomplete.
3. **Pipeline**
    - Adicione os testes ao pipeline de CI para execução automática.

Instruções

1. **Configuração do Ambiente**
    - Use Laravel 11 e MySQL para o backend.
    - Use React 18, Next.js, e MUI para o frontend.
    - Configure o Redis como cache.
2. **Entrega**
    - Submeta o projeto em um repositório no GitHub com instruções claras de instalação e execução.

Critérios de Avaliação

1. **Backend**
    - Qualidade do código (estrutura, boas práticas, e legibilidade).
    - Uso eficiente de Eloquent, Redis e Jobs.
    - Manutenibilidade e extensibilidade do código.
2. **Frontend**
    - Responsividade e usabilidade do componente.
    - Performance em grandes listas de dados.
    - Integração com o backend.
3. **Infraestrutura**
    - Qualidade dos manifestos Kubernetes.
4. **Diferenciais**
    - Implementação de Docker e CI/CD.
    - Documentação completa no README.md e/ou gerada com Sphinx.
    - Testes automatizados para backend e frontend.
5. **Performance**

- Eficiência na manipulação de grandes volumes de dados.
- Uso correto de caching e otimizações de banco de dados.

Output Esperado

1. **Banco de Dados**
   - Tabela `items` com os campos `id`, `title`, `body`, `created_at`, `updated_at`.
   - Log de sincronizações na tabela `sync_logs`.
2. **Frontend**
   - Um componente Autocomplete funcional, responsivo e performático.
3. **Infraestrutura**
   - Manifestos Kubernetes completos e funcionais.
4. **Repositório**
   - Projeto completo, com instruções de setup e uso no arquivo README.md.

Boa sorte!