Hiring Challenge (Frontend)

Desafio técnico para posições de software engineering focados em frontend.

Take home challenge

1. Depois da screening interview você receberá um e-mail com um link pra esta página descrevendo o escopo do desafio;
2. **Esse desafio será revisado por você juntamente com um membro do time Kanastra durante a próxima etapa**;
3. Sinta-se livre para expandir o desenho da aplicação descrita;
4. Para entregar você deverá responder este e-mail com um link do GitHub contendo o repositório com a implementação. O Repositório deve conter um **README.md**, descrevendo quais tecnologias foram utilizadas, como rodar (com versão específica do node, caso haja algum problema de compatibilidade) e o link para uma demo.

Requisitos

Gostaríamos de conhecer um pouco mais das suas habilidades por meio de uma página responsiva, a qual tem como objetivo:

- Consumir a API do Spotify utilizando as melhores práticas de segurança. O candidato deve ser capaz de ler a documentação da API e utilizar ela da forma recomendada.
- Listar os artistas. Sinta-se livre para utilizar imagens, animações, entre outros artifícios para uma UI chamativa.
- Ao clicar em um artista devemos conseguir visualizar o nome e a popularidade dele, bem como as suas principais musicas e os seus albuns.
- A listagem de albuns deve conter 20 por página com controle manual para navegar entre as páginas.
- Fique a vontade para explorar mais sobre a API caso queira adicionar mais coisas.
- Devemos conseguir filtrar as listas de artistas e albuns com os seguintes requisitos:
  - Filtrar pelo nome
  - Filtrar pelo nome do album

  A criatividade aqui pode ser explorada, criando filtros na UI da maneira que você preferir.
- `Tente construir toda essa experiência com uma UI chamativa`.

Prazo máximo

Nós esperamos que você implemente esse problema em no máximo 5 dias corridos.

Pontos de atenção

1. É ideal utilizar React, CSS ou Tailwind (pode ser com `shadcn`), Context API para gerenciar estados da aplicação e para as requests pode usar `axios` e caso utilize `react query` com cache das mesmas será um diferencial.
2. É ideal utilizar Typescript. Um diferencial é o domínio das tipagens por meio de generics.

3. **Testes unitários** não são obrigatórios, mas com certeza diferencias; O mesmo vale para testes **end-to-end**. Gostamos bastante do Cypress para esse último.
4. Estilização nas variações de componentes. *Exemplo*: inputs, botões, textos etc.
5. Promise handling side effects
6. Error handling
7. Loading handling
8. Filters and pagination side effects
9. Tradução utilizando *tokenização* com duas ou mais linguagens
10. Animações não são obrigatórias, mas são diferenciais.
11. Gerenciamento de estados globais sem prop drilling, prezando pela performance e pelo armazenamento inteligente e dinâmico.