Sobre o sistema:

- O intuito é construir um sistema que encurte as URLs.

Informações de desenvolvimento:

- Deverá ser implementado um projeto com NodeJS na última versão estável, sendo construído como API REST. Leve em consideração que o sistema será implementado em uma infraestrutura que escala verticalmente.
- O sistema deve possibilitar o cadastro de usuários e autenticação dos mesmos.
- O sistema deve possibilitar que a partir de um url enviado, ele seja encurtado para no máximo 6 caracteres. Exemplo:
    - Entrada: https://teddy360.com.br/material/marco-legal-das-garantias-sancionado-entenda-o-que-muda/
      Saída: http://localhost/aZbKq7
- Qualquer um pode solicitar que o URL seja encurtado e para encurtar deve existir apenas um endpoint, mas caso seja um usuário autenticado, o sistema deve registrar que o URL pertence ao usuário.
- Um usuário autenticado pode listar, editar o endereço de destino e excluir URLs encurtadas por ele.
- Todo acesso a qualquer URL encurtado deve ser contabilizado no sistema.
- Quando um usuário listar os urls deve aparecer na listagem a quantidade de cliques.
- Todos os registros devem ter uma forma de saber quando foram atualizados.
- Os registros só poderão ser deletados logicamente do banco, ou seja, deverá ter um campo que guarda a data de exclusão do registro, caso ela esteja nula é porque ele é válido, caso esteja preenchida é porque ele foi excluído e nenhuma operação de leitura ou escrita pode ser realizada por ele.

Sobre a entrega:

- Construir uma estrutura de tabelas que faça sentido para o projeto usando um banco relacional.
- Construir endpoints para autenticação de e-mail e senha que retorna um Bearer Token.
- Construir apenas um endpoint para encurtar o URL, ele deve receber um URL de origem e deve aceitar requisições com e sem autenticação, deve retornar o url encurtado - incluindo o domínio..
- Definir o que deve e não deve ser variável de ambiente..
- Construir endpoints que aceitam apenas requisições autenticadas:
    - Listagem de URL Encurtados pelo usuário com contabilização de clicks
    - Deletar URL Encurtado
    - Atualizar a origem de um URL encurtado.
- README ou CONTRIBUTING explicando como rodar o projeto.
- Construir um endpoint que ao receber um URL encurtado, redirecione o usuário para o URL de origem e contabilize.
- Maturidade 2 da API REST

Entrega com diferencial:

- Utilizar docker-compose para subir o ambiente completo localmente.
- Ter testes unitários
- API está documentada com OPEN API ou Swagger
- Ter validação de entrada em todos os lugares necessários.
- Ter instrumentação de observabilidade (implementação real ou abstração) de um ou vários tipos:
    - Logs
    - Métricas
    - Rastreamento

    Observação: Se a implementação for real, não é obrigatório que tenha no docker compose, ao rodarmos a aplicação podemos inserir as credenciais de serviços como Elastic, Sentry, Datadog, New Relic, Open Telemetry, Jagger, Prometheus e etc desde que esteja em variáveis de ambiente e uma explicação de como configurar. Também é interessante ter uma variável de ambiente que ativa ou desativa o uso da ferramenta.
- Dar deploy no ambiente em um cloud provider e expor no readme o link.
- Deixar no README pontos de melhoria para caso o sistema necessite escalar horizontalmente e quais serão os maiores desafios.

- Monorepo com separação de serviços como gerenciamento de identidade e acesso e regra de negócio de encurtar URL com comunicação entre os serviços. Obrigatório docker-compose neste cenário.
- Configurar um api gateway como KrankeD na frente dos serviços.
- Utilizar changelog com a realidade do seu desenvolvimento
- Git tags definindo versões de release, por exemplo release 0.1.0 como encurtador criado, 0.2.0 como autenticação, 0.3.0 como operações de usuário no encurtador, 0.4.0 como contabilização de acessos.
- Construir deployments do Kubernetes para deploy.
- Construir artefatos do Terraform para deploy.
- Construir github actions para lint e testes automatizados.
- Transformar o sistema em multi tenant.
- Construir funcionalidades a mais que acredite ser interessante para o “domínio do negócio” da aplicação.
- Definir e assegurar quais versões do NodeJS são aceitas no projeto.
- Configurar pré commit ou pre push hooks.
- Código tolerante a falhas.

O mesmo teste é utilizado para avaliação de **todas as senioridades**, então a maior partes dos diferenciais são voltados ao público mais sênior.

A avaliação é feita pela escrita do código, configurações da base de código, padrões de projetos, design patterns, boas práticas, commits, tags, arquitetura e tecnologias utilizadas. Levamos em consideração também quando o candidato recebe o teste e a data de entrega do teste.

Caso o teste não rode no ambiente local do avaliador por falta de instruções, conflito de dependências que necessite “*force*” na instalação ou código quebrado, impactará negativamente a avaliação.

O código deve estar armazenado em um repositório público preferencialmente **GitHub**.

Desejamos boa sorte em seu processo!