Desafio Encurtador de URL

Um encurtador de URL é um serviço que recebe uma URL qualquer e retorna uma outra, geralmente menor que a original. Ex: bit.ly, TinyURL.

O seu objetivo é criar uma API RESTful para um encurtador de URLs.

Leia com atenção: Algumas premissas existem para esse projeto e deverão fazer parte da forma como você soluciona o problema. Esses fatores serão importantes na sua avaliação:

- O sistema rodará em um ambiente com ubuntu server 14.04 LTS em uma máquina virtualizada na Amazon.
- O serviço deverá ser stateless. Cada novo hit de um usuário pode bater em diferentes instâncias da sua aplicação.
- As instâncias poderão compartilhar um mesmo banco de dados, não estamos interessado em sua capacidade de escalar o banco.
- É esperado que sua solução preze pelo desempenho do serviço principal. `GET /url/:id`
- Você deverá se preocupar em como esse serviço rodará **em produção**. Esperamos poder rodar o seu software em um servidor **limpo** seguindo somente as instruções fornecidas no `README.md` do projeto.
- Você deve usar GIT como controle de versão em seu projeto. Faça commits regulares.
- Para manter a simplicidade, não haverá autenticação nas APIs.

Forma de Entrega

Você deverá entregar um `.tar.gz` ou `.zip` do seu projeto **contendo o diretório** `.git/` e um README.md e enviar como resposta ao email que usamos para enviar o desafio.

NÃO COMPARTILHE ESSE DESAFIO OU A SUA SOLUÇÃO.

Avaliação

A avaliação será feita por desenvolvedores acostumados em colocar sistemas escaláveis no ar. Consideraremos a simplicidade da sua solução em conjunto com a efetividade da mesma. Não existe um único jeito certo de se fazer um sistema contanto que esse seja efetivo, eficiente e fácil de manter. Atente aos seguintes pontos quando estiver desenvolvendo seu projeto:

- **Simplicidade**: se dois métodos resolvem o mesmo problema, preferimos o mais legível e simples. Atente para a **facilidade de deploy e operação** da sua applicação.
- **Banco de Dados**: Qualquer banco. Se sua aplicação utilizar um banco de dados, este será instalado em uma instância **separada** da aplicação global. Sua documentação deve incluir o procedimento para isso.
- Testes: testaremos os endpoints da sua API usando automação (**+ para código desenvolvido com foco em testes**).
- Desempenho: todos os testes rodarão em uma instância padrão na Amazon, o desempenho relativo da sua solução frente a outras será considerado. Lembrando que não estamos em busca do código mais rápido, mas definitivamente não queremos operar o mais lento.
- Code style: não preferimos jeito A ou jeito B, desde que haja consistência ao longo do código.
- Linguagem: Vale tudo. Escolha a linguagem que preferir, atente para que sua escolha de linguagem faça sentido dentro do contexto de uma API Web. Atente para as premissas, desempenho e legibilidade antes de escolher algo como Whitespace ou Brainfuck.

Endpoints

Os seguintes endpoints são necessários no seu projeto. O escopo total do projeto corresponde somente a esses endpoints e como colocá-los em produção. Todos os endpoints com exceção do primeiro deverão ser RESTful com `Content-Type: application/json`.

GET /url/:id

Deve retornar um 301 redirect para o endereço original da URL.

```
301 Redirect
Location: <url>
```

Caso o `id` não existe no sistema, o retorno deverá ser um `404 Not Found`.

POST /users/:userid/urls

Cadastra uma nova url no sistema

```
{"url": "http://sdkjflsdjf.com/?sdfsdf"}
```

A resposta deverá ser um objeto JSON igual ao da chamada `GET /stats/:id`.

```
{
    "id": "23094",
    "hits": 0,
    "url": "",
    "shortUrl": ""
}
```

GET /stats

Retorna estatísticas globais do sistema.

```
{
    "hits": "", // Quantidade de hits em todas as urls do
sistema
    "urlCount": "", // Quantidade de urls cadastradas
    "topUrls": [ // 10 Urls mais acessadas
        // Objeto stat por id, igual ao /stats/:id,
ordernado por hits decrescente
    ]
}
```

GET /user/:userId/stats

Retorna estatísticas das urls de um usuário. O resultado é o mesmo que `GET /stats` mas com o escopo dentro de um usuário.

Caso o usuário não exista o retorno deverá ser com código `404 Not Found`.

GET /stats/:id

Retorna estatísticas de uma URL específica

```
{
    "id": "23094", // ID da url
    "hits": 123, // Quantidade de hits nela via /url/23094
    "url": "", // A url original
    "shortUrl": "" // A url curta formada
}
```

DELETE /url/:id

Apaga uma URL do sistema (duh!). Deverá retornar vazio em caso de sucesso.

POST /user

Cria um usuário. O conteúdo do request deverá ser um objeto JSON com o conteúdo no seguinte formato.

```
{
    "id": "jibao"
}
```

DELETE /user/:userId

Apaga um usuário. :)