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 /urls/: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. Inclua o **diretório** `.git/` e um README.md e envie como resposta ao email que usamos para enviar o desafio. Para facilitar forneça um script de `install.sh` e de `start.sh` .

- README.md - Dependências do projeto, descrição geral de como funciona, arquitetura, como executar o build e os testes;
- install.sh - Script para instalar o projeto;
- start.sh - Script para iniciar o projeto;

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 /urls/:id

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

```
301 Redirect
Location:
```

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://www.chaordic.com.br/folks"}
```

A resposta deverá ser um objeto JSON igual ao da chamada `GET /stats/:id` com código `201 Created`.

```
{
    "id": "23094",
    "hits": 0,
    "url": "http://www.chaordic.com.br/folks",
    "shortUrl": "http://<host>[:<port>]/asdfeiba"
}
```

GET /stats

Retorna estatísticas globais do sistema.

```
{
    "hits": 193841, // Quantidade de hits em todas as urls do sistema
    "urlCount": 2512, // Quantidade de urls cadastradas
    "topUrls": [ // 10 Urls mais acessadas
        // Objeto stat por id, igual ao /stats/:id, ordernado por hits decrescente
        {
            "id": "23094",
            "hits": 153,
            "url": "http://www.chaordic.com.br/folks",
            "shortUrl": "http://<host>[:<port>]/asdfeiba"
        },
        {
            "id": "23090",
            "hits": 89,
            "url": "http://www.chaordic.com.br/chaordic",
            "shortUrl": "http://<host>[:<port>]/asdfeiba"
        },
        // ...
    ]
}
```

GET /users/: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": 0, // Quantidade de hits nela
    "url": "http://www.chaordic.com.br/folks", // A url original
    "shortUrl": "http://short.url.com/asdfeiba" // A url curta formada
}
```

DELETE /urls/:id

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

POST /users

Cria um usuário. O conteúdo do request deverá ser com código `201 Created` e retornar um objeto JSON com o conteúdo no seguinte formato. Caso já exista um usuário com o mesmo `id` retornar código `409 Conflict`

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

DELETE /user/:userId

Apaga um usuário. :)