Encurtador de Links

Por: Rafael Raupp

Instalação:

1. **Baixar** o projeto completo via github no link "https://github.com/chaordic/developer-intern-challenge".
2. Adicionar um novo host no servidor local preferencialmente com a url "**http://chr.dc**".
3. Criar um **banco de dados**.

API:

1. Entrar no diretório ".../project_path/**api**".
2. Executar o comando "**composer install**" via linha de comando para instalar todas as dependências do projeto.
3. Abrir o arquivo "**.env.example**" para edição.
4. Editar as variaveis "**`DB_DATABASE`**", "**`DB_USERNAME`**" e "**`DB_PASSWORD`**" para as configurações do seu banco de dados.
5. Renomear o arquivo de "**.env.example**" para "**.env**".
6. Executar o comando "**php artisan migrate**" via linha de comando para criar as tabelas no banco.
7. Executar o comando "**php artisan key:generate**" via linha de comando criar uma chave de criptografia.
8. Assim que o servidor estiver rodando, a **API** estará pronta para receber as requisições.

VIEW:

1. Entrar no diretório ".../project_path/**view**".
2. Executar o comando "**npm install**" via linha de comando para instalar todas as dependências do projeto.
3. Caso a **API** não esteja rodando na URL "**http://chr.dc**" deve ser editada a variável "**API_ROOT**" do arquivo ".../project_path/view/src/app/services/**api-paths.service.ts**".
4. Executar o comando "**ng serve**" via linha de comando para iniciar o servidor (o servidor deve preferencialmente rodar na porta **4200**, caso contrário deve ser editado o arquivo "**Cors.php**" da **API** e mudar a porta na linha **12**).

Estrutura:

API:

Controllers:

HitController:

Esse controller é responsável por responder qual o **total** de **hits** em **urls**.

RedirectController:

Esse controller é responsável por **redirecionar** o usuário quando ele acessa uma das **urls encurtadas**.

UrlController:

Esse controller é responsável pela **inserção** e **recuperação** de urls.
Ele possui os seguintes **métodos**:

index:

Esse método é acessado através de uma **GET REQUEST** para a url “/api/urls”, por padrão ele retorna as **5 urls mais clicadas**, porém caso seja passado uma **query parameter** com o nome “**url**” o método retorna o **objeto json** contendo essa url ou **nenhum conteúdo** caso ela não exista no banco de dados.

show:

Esse método é acessado através de uma **GET REQUEST e** retorna um objeto **json** de **url** a partir do seu **id**, que é passado diretamente na url “**/api/urls/{{ID}}**”.

store:

Esse método é acessado através de uma **POST REQUEST**, ele **formata** a url recebida, **gera um id** para a url encurtada, salva no banco de dados e **retorna o objeto json** da url que foi criada.

format_url:

Esse método é privado e formata uma url recebida, removendo a "**/**" do final (caso exista), e depois verificando se a url já possui "**http://**" ou "**https://**", caso ela não possua, ele se encarrega de incluir, verificando antes se essa url **já existe no banco de dados** com o protocolo **HTTPS** incluído, para evitar que ela seja duplicada, uma com **HTTP** e outra com **HTTPS**.

gen_id():

Esse método é privado e gera uma id única para ser usada na url encurtada.

Models:

Url:

Essa classe **estende** a classe "**Model**" do **Laravel**, isso faz com que ela facilite a **inserção**, **recuperação** e **atualização** da tabela pela qual é responsável no banco de dados, que nesse caso é a tabela "**urls**".

Migrations:

create_urls_table:

Essa migration **cria** a tabela de urls, e depois disso **mapeia** e **insere** os dados contidos no arquivo "**urls.json**" postado no github.

Middlewares:

Cors:

Esse middleware serve para possibilitar que a aplicação **angular**, possa acessar a API mesma estando em outro domínio.

VIEW:

Interfaces:

PaginatedResponseInterface:

Essa interface define a estrutura de uma resposta paginada do servidor, contendo um array de objetos do tipo **UrlInterface** e um objeto contendo os **metadados** da paginação.

UrlInterface:

Essa interface define a estrutura de um objeto **Url**.

UrlResponseInterface:

Essa interface define a estrutura de uma resposta do servidor contendo um objeto do tipo **UrlInterface.**

Services:

ApiPathsService:

Esse serviço é responsável por definir os caminhos para a API.

ApiRequestService:

Esse serviço é responsável por fazer as requests para a API. Possui os metodos:

getTop5:

Faz uma GET request à API para buscar as 5 **urls com mais hits**.

getTotalHits:

Faz uma GET request à API descobrir o **total de hits** em urls.

postUrl:

Faz uma POST request à API para **encurtar uma url**.

HitsChangeService:

Esse serviço é responsável por manter a página sempre atualizada.
Ele possui um **Subject** que é inscrito pelo componente "**TopComponent**" para ser notificado sempre que o número total de hits for modificado no componente "**TotalComponent**".

Components:

ShortenerComponent:

Esse componente é responsável por **encurtar as urls**.

TopComponent:

Esse componente é responsável por mostrar as **urls mais clicadas**.
Ele possui o método **getTotalHits()** que causa um grande **overhead** na API por fazer uma requisição por segundo para cada página abert, sendo assim, com um número considerável de usuários o **desempenho cairia muito**.

TotalComponent:

Esse componente é responsável por mostrar o **total de clicks** em urls.

HeaderComponent:

Esse componente contém o **cabeçalho** da aplicação.

FooterComponent:

Esse componente contém o **rodapé** da aplicação.