Cristiane

API não deve ser responsável por redirecionar, se não encontrar deve retornar 404, e o front deve redirecionar

```php
public function index($slug)
{
    $link = Shortener::where('slug', $slug)
        ->where('expiration_date', '>', now())
        ->where('disable', '=', false)
        ->first();
    if (!$link) {
        return redirect('http://localhost:8080/404');
    }

    return redirect()->away($link->url);
}
```

Vamos ganhar uns pontinhos? Ao invés de realizar validação dentro do Controller podemos fazer validação antes de chegar nele utilizando Requests Validations:

[Documentoçãa](https://laravel.com/docs/10.x/validation)
[Medium](https://medium.com/@kamerk22/the-smart-way-to-handle-request-validation-in-laravel-5e8886279271)

```php
public function store(Request $request)
{
    $validate = Validator::make($request->all(), [
        'url' => 'required|url|max:255'
    ]);

    if ($validate->fails()) {
        return response()->json($validate->messages());
    }
```

Conseguimos criar um hash único sem loop?
Exemplo de lei de formação: [slug][data][uniquid]

```php
try {
    do {
        $id = uniqid();
        $slug = $this->hash($id);
    } while (Shortener::where('slug', $slug)->exists());
```

Vamos ganhar mais uns pontos? Criando uma camada de Serviços.
Basicamente criar uma Classe dentro de App > Http > Services e migrar os processamentos para dentro dela.

O que jogar dentro dela? Todo e qualquer processamento do tipo:

- Consultas
- Criação de hashtags
- Salvar os dados

```php
do {
    $id = uniqid();
    $slug = $this->hash($id);
} while (Shortener::where('slug', $slug)->exists());
```

```php
$link = new Shortener;
$link->url = $request->url;
$link->slug = $slug;
$link->expiration_date = now()->addDays(7);
$link->save();
```


```php
do {
    $id = uniqid();
    $slug = $this->hash($id);
} while (Shortener::where('slug', $slug)->exists());
```

Quando tivermos vários registros aqui vai dar ruim, vamos utilizar paginação?

```php
    public function show()
    {
        $links = Shortener::all();
        $now = now();
```

exemplo (acho que é isso):
$links = Shotener::paginate(20);

Podemos melhorar

```php
foreach ($links as $link) {
    if ($link->expiration_date < $now) {
        $link->expired = true;
    } else {
        $link->expired = false;
    }
}
```

Sugestão -> através do serviço, buscar os registros e na consulta adicionar um novo campo

(Acho que é isso)
Model::where()->select('*', DB::RAW('minha nova coluna'))->paginate();

Preferir utilizar o Carbon no lugar do now()

```php
{
    $links = Shortener::all();
    $now = now();
```

(Acho que é isso)
$now = new Carbon();

Padronizar os retornos, mesmo se for um array de resultados

```php
    return response()->json($links);
}
```

```php
return response()->json(['success' => true, 'data' => $link]);
```

Aqui o campo expiration_date não deve ser setado como a data atual?

```php
public function disable($id)
{
    $link = Shortener::where('id', $id)
        ->where('disable', '=', false)
        ->first();

    if (!$link) {
        return response()->json(['success' => false, 'message' => 'Link not found or cannot be disabled']);
    }

    $link->disable = true;
    $link->save();

    return response()->json(['success' => true, 'message' => "The link {$link->slug} was disabled successfully"]);
}
```


Vamos ganhar mais alguns pontos? Mostre a eles que você conhece a estrutura de uma Model

Adicionar:

Protected $table = ‘nome da tabela’
Protected $fillable = [
‘nome’,
‘dos campos’,
‘que serão utilizados no insert’
];

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Shortener extends Model
{
    use HasFactory;
}
```


Vamos mostrar que você manja de Banco?

$table->increments('id')->unsigned(); <<<<<<<<<<>>>>>>>>>> assim você mostra que sabe que a fala unsigned não permite inserir valores negativos e aumenta o range para os números positivos

```php
public function up(): void
{
    Schema::create('shorteners', function (Blueprint $table) {
        $table->id();
        $table->string('url')->unique();
        $table->string('slug')->unique();
        $table->timestamps();
        $table->boolean('disable')->default(false);
        $table->date('expiration_date');
    });
}
```


Expiration_date é date ou datetime?
Veja, caso queiramos o prazo de 24 horas, precisamos saber o horário d criação (timestesmp()) e 24 horas após

```
        $table->date('expiration_date');
```


Remover traços web, pois a API não deve ter interesse nessas informações