Controllers

Já tivemos nosso primeiro contato com controllers até este momento. Os controllers são parte importantíssimas nesta arquitetura utilizando o Laravel. Relembrando, de forma simples, os controllers são o ponto de delegação entre Model e View, recebendo a requisição e entregando para quem é de direito.

Utilizando o artisan podemos automatizar, como já fizemos, a geração dos nossos controllers. Como por exemplo fizemos no módulo passado ao executarmos o comando abaixo:

```
1    php  artisan  make:controller HelloWorldController
```

Agora vamos explorar mais opções dentro deste comando e buscar mais produtividade na execução de nossos projetos.

Controllers como Recurso

Podemos gerar controllers com métodos para cada operação de CRUD e por meio de uma configuração de rota termos também as rotas automáticas para cada um destes métodos.

Este tipo de controller chamamos de controller como recurso. Vamos gerar e entender como são. Em seu terminal na raiz do seu projeto execute o comando abaixo:

```
1    php  artisan  make:controller UserController --resource
```

Controllers como Recurso

Perceba agora que usei o mesmo comando para gerar um novo controller, no caso acima o controller UserController mas adicionei um parâmetro, o --resource que criará um controller com os seguintes métodos abaixo:

index;
create;
store;
show;
edit;
update;
destroy;

Controllers como Recurso

Agora com o controller como recurso gerado, podemos expor rotas para cada um dos métodos acima. Neste caso, para facilitar mais ainda nossas vidas, temos um método dentro do Route chamado de resource que já expõe para nós as rotas para cada um destes métodos simplificando ainda mais as coisas.

Em seu arquivo de rotas web adicione a rota abaixo:

```
1    Route::resource('/users',  'UserController');
```

Se nós debugarmos as rotas existentes em nosso projeto até o momento, podemos encontrar, focando no controller como recurso e em suas rotas.

Controllers como Recurso

Para debugarmos as rotas existentes no Laravel, definidas dentro de sua aplicação, basta executar na raiz do seu projeto o comando abaixo:

```
1    php  artisan route:list
```

Obtendo a lista de rotas de sua aplicação. Veja o destaque nas rotas criadas pelo método resource do Route:

Controllers como Recurso

```
blog: php artisan route:list
+--------+-----------+--------------------------+--------------+-----------------------------------------------------------------+-----------------------------------------------+
| Domain | Method    | URI                      | Name         | Action                                                          | Middleware                                    |
+--------+-----------+--------------------------+--------------+-----------------------------------------------------------------+-----------------------------------------------+
|        | GET|HEAD  | /                        |              | Closure                                                         | web                                           |
|        | POST      | _ignition/execute-solution |            | Facade\Ignition\Http\Controllers\ExecuteSolutionController      | Facade\Ignition\Http\Middleware\IgnitionEnabled |
|        | GET|HEAD  | _ignition/health-check   |              | Facade\Ignition\Http\Controllers\HealthCheckController          | Facade\Ignition\Http\Middleware\IgnitionEnabled |
|        | GET|HEAD  | _ignition/scripts/{script} |            | Facade\Ignition\Http\Controllers\ScriptController               | Facade\Ignition\Http\Middleware\IgnitionEnabled |
|        | POST      | _ignition/share-report   |              | Facade\Ignition\Http\Controllers\ShareReportController          | Facade\Ignition\Http\Middleware\IgnitionEnabled |
|        | GET|HEAD  | _ignition/styles/{style} |              | Facade\Ignition\Http\Controllers\StyleController                | Facade\Ignition\Http\Middleware\IgnitionEnabled |
|        | GET|HEAD  | api/user                 |              | Closure                                                         | api,auth:api                                  |
|        | GET|HEAD  | hello-world              |              | App\Http\Controllers\HelloWorldController@index                 | web                                           |
|        | GET|HEAD  | post/{slug}              |              | Closure                                                         | web                                           |
|        | GET|HEAD  | users                    | users.index  | App\Http\Controllers\UserController@index                       | web                                           |
|        | POST      | users                    | users.store  | App\Http\Controllers\UserController@store                       | web                                           |
|        | GET|HEAD  | users/create             | users.create | App\Http\Controllers\UserController@create                      | web                                           |
|        | GET|HEAD  | users/{user}             | users.show   | App\Http\Controllers\UserController@show                        | web                                           |
|        | PUT|PATCH | users/{user}             | users.update | App\Http\Controllers\UserController@update                      | web                                           |
|        | DELETE    | users/{user}             | users.destroy | App\Http\Controllers\UserController@destroy                    | web                                           |
|        | GET|HEAD  | users/{user}/edit        | users.edit   | App\Http\Controllers\UserController@edit                        | web                                           |
+--------+-----------+--------------------------+--------------+-----------------------------------------------------------------+-----------------------------------------------+
blog:
```

Controllers como Recurso

Além de termos gerado o controller com os métodos para serem utilizados dentro de um CRUD completo, temos também a criação das rotas para cada um dos métodos do controller, por meio da chamada de apenas um método, o resource.

O método resource recebe o nome da rota como primeiro parâmetro e o nome do controller, gerado como recurso, no segundo parâmetro, simples assim.

Perceba que na imagem anterior temos as rotas, temos também apelidos para estas rotas e ainda temos rotas para cada um dos verbos HTTP.

Rota	Verbo	Controller@método
/users/	GET	UserController@index;
/users/	POST	UserController@store;
/users/create	GET	UserController@create;
/users/{user}	GET	UserController@show;
/users/{user}	PUT ou PATCH	UserController@update;
/users/{user}	DELETE	UserController@destroy;
/users/{user}/edit	GET	UserController@edit;

Lembrando que você precisa adicionar sua lógica para cada um dos métodos, entretanto, todas as rotas já estão definidas como vimos acima e quando geramos o controller temos as definições dos métodos também.

Single Action Controllers

Single Action Controllers são úteis quando você necessita somente de um método no Controller:

```php
class IndexController extends Controller
{
    //
    public function __invoke()
    {
        return view('index', ['name' => 'Joãozinho']);
    }
}
```

Laravel 8

Rota:

```php
Route::get('/', IndexController::class);
```

Formulários, Request & Response

Agora vamos conhecer um ponto crucial para a criação das nossas aplicações, como manipular informações recebidas em nossas requests e as particularidades do response ou respostas HTTP dentro do Laravel.

Aproveito também e já abordo isso com dados vindos de um formulário onde teremos pequenos pontos importantes a serem abordados neste aspecto.

Então vamos lá!

Iniciando pelo formulário

Para começarmos, vamos criar um formulário para envio dos dados para nosso controller.

Para isso crie um arquivo chamado de create.blade.php dentro da pasta de views, colocando ele dentro da pasta posts, então crie esta pasta também.

O caminho completo até o arquivo ficará assim:

resources/views/posts/create.blade.php

Iniciando pelo formulário

Veja o formulário abaixo a ser adicionado no create.blade.php:

```
1    <form  action="{{route('posts.store')}}" method="post">
2
3            <div   class="form-group">
4                    <label>Titulo</label>
5                    <input type="text"  name="title"  class="form-control">
6            </div>
7
8            <div   class="form-group">
9                    <label>Descrição</label>
10                    <input type="text"  name="description"  class="form-control">
11            </div>
12
```

Iniciando pelo formulário

```
13          <div   class="form-group">
14                  <label>Conteúdo</label>
15                  <textarea  name="content" id=""  cols="30"  rows="10" class="form-control"></te\
16  xtarea>
17          </div>
18
19          <div   class="form-group">
20                  <label>Slug</label>
21                  <input type="text"  name="slug" class="form-control">
22          </div>
23
24          <button class="btn btn-lg btn-success">Criar  Postagem</button>
25  </form>
```

Iniciando pelo formulário

No formulário temos 4 campos diretamente, são eles:

- Titulo;
- Descrição;
- Conteúdo;
- Slug (futuramente vamos automatizar esta geração);

Nosso formulário trabalhará com envio dos dados via método POST e já adicionamos, a action do formulário, a chamada de uma rota pelo seu apelido, a posts.store. Vamos gerar nosso controller e nossas rotas para que possamos fazer esse form funcionar.

Iniciando pelo formulário

Gere o controller a partir do seu terminal, com o comando abaixo:

```
1    php  artisan  make:controller Admin/PostController
```

Perceba mais um apredizado acima, como coloquei Admin/ nosso controller será criado dentro da pasta Admin dentro da pasta de controllers. Isso simplifica também nossas gerações.

Pro nosso exemplo não vou gerar como recurso porque nosso foco ainda não é gerarmos um crud e sim conhecermos esse trabalho entre as requisições e o envio de dados.

Iniciando pelo formulário

Uma vez criado o controller, adicione os métodos abaixo:

```
1    public function create()
2    {
3                return view('posts.create');
4    }
```

e também:

```
1    public function store(Request $request)
2    {
3
4    }
```

Iniciando pelo formulário

O primeiro método será para exibição de nossa view criada anteriormente e o segundo para manipularmos o que for enviado do formulário de criação. Este segundo método veremos mais adiante seu conteúdo.

Até o momento nosso controller fica desta maneira:

```
1   <?php
2
3   namespace App\Http\Controllers\Admin;
4
5   use   Illuminate\Http\Request;
6   use   App\Http\Controllers\Controller;
7
```

Iniciando pelo formulário

```
8   class  PostController  extends Controller
9   {
10          public function create()
11          {
12                  return view('posts.create');
13          }
14
15          public function store(Request $request)
16          {
17
18          }
19  }
```

Iniciando pelo formulário

View criada, controller e métodos definidos vamos adicionar no arquivo de rotas web.php nossas definições de rota para este trabalho. Vamos lá!

Veja as rotas adicionadas abaixo:

```
1    Route::prefix('admin')->namespace('Admin')->group(function(){
2
3                Route::prefix('posts')->name('posts.')->group(function(){
4
5                        Route::get('/create',  [PostController::class,'create'])->name('create');
6
7                        Route::post('/store', [PostController::class,'store'])->name('store');
8
9                });
10
11   });
```

Perceba acima que isolei o prefixo admin e o namespace Admin para o grupo de rotas que contêm o nosso post. E também isolei o prefixo posts, bem como o seu apelido posts. para o conjunto de rotas pensadas para este módulo.

Iniciando pelo formulário

No fim ao acessarmos /admin/posts/create veremos nosso formulário, ainda sem estilo, carregado em nossa tela.

E também ao enviarmos nosso form ele mandará os dados para nossa rota

/admin/posts/store.

Com nosso servidor ligado, através do comando php artisan serve podemos acessar nosso link

http://aulta.test/admin/posts/create

Feito isso vamos conhecer o request!

Request, manipulando requisições

Vamos começar este trecho tentando enviar qualquer dado para o nosso backend a partir do formulário. Podemos enviar até o form vazio mesmo, então vamos clicar em Criar Postagem em nosso formulário.

Se você recebeu a tela de expirado, não se preocupe.

Por que isso aconteceu? Perceba, pela sua barra de endereços, que ele enviou a requisição para a rota correta: /admin/posts/store então porque recebemos uma tela de requisição expirada?

Quando enviamos dados via POST, PUT ou PATCH o Laravel faz uma validação na requisição para evitar que fontes externas enviem dados ou falsifiquem nossa requisição. Esse controle é chamado de CSRF.

Obs.: O CSRF não é algo exclusivo do Laravel, é um tópico de segurança e recomendo fortemente a leitura sobre o assunto.

Request, manipulando requisições

Agora como podemos adicionar esta possibilidade em nosso formulário? Vamos lá então, adicione após a abertura da tag form o seguinte input abaixo:

```
1    <input  type="hidden"  name="_token" value="{{csrf_token()}}">
```

Acima estamos enviando o token csrf, para validar a procedência e envio dos dados do nosso formulário através de nossa requisição. Feito isso, volte para a página do formulário atualize e tente enviar novamente.

Agora você verá tudo branco e a página de expirado já não existe mais, como não temos nada definido no método store lá no controller o resultado será mesmo uma página em branco mas nossa requisição post, vinda do formulário, já bate na execução do método.

O input acima, com o token csrf, pode ser substituido completamente pelo: `{{csrf_field()}}`

Que adicionará o input completo como fizemos na mão anteriormente. Ou ainda podemos simplificar mais, utilizando uma diretiva disponível do blade,: `@csrf`

Que também adiciona o input como fizemos anteriormente.

Request, manipulando requisições

O formulário agora fica desta forma:

```
1     <form  action="{{route('posts.store')}}" method="post">
2
3         @csrf
4
5         <div   class="form-group">
6             <label>Titulo</label>
7             <input type="text"  name="title"  class="form-control">
8         </div>
9
10          <div   class="form-group">
11              <label>Descrição</label>
12              <input type="text"  name="description"  class="form-control">
```

Request, manipulando requisições

```
13          </div>
14
15          <div   class="form-group">
16                  <label>Conteúdo</label>
17                  <textarea  name="content" id=""  cols="30"  rows="10" class="form-control"></te\
18  xtarea>
19          </div>
20
21          <div   class="form-group">
22                  <label>Slug</label>


23                  <input type="text"  name="slug" class="form-control">
24          </div>
25
26          <button class="btn btn-lg btn-success">Criar  Postagem</button>
27  </form>
```

Manipulando os dados da requisição

Agora que nosso form já está 'funcionando' e nossa requisição está chegando no método como podemos manipular os dados dentro do método store?

Se voltarmos ao método store do PostController, vamos perceber que em sua assinatura temos a definição de um parâmetro.

```
1    public function store(Request $request)
2    {
3
4    }
```

O parâmetro $request, e seu tipo esperado é um objeto Request do próprio Laravel. O objeto Request vêm do namespace Illuminate\Http\Request e ele é automaticamente resolvido pelo container de injeção de dependências do Laravel, mais a frente falaremos sobre esse container.

Com isso, por meio do parâmetro $request temos a possibilidade de acesso, das mais variadas formas, aos dados enviados na requisição para nosso método, e em nosso caso, vindos do formulário.

Manipulando os dados da requisição

Vamos ver um panorama geral dos dados enviados. Adicione o seguinte trecho dentro do método

store:

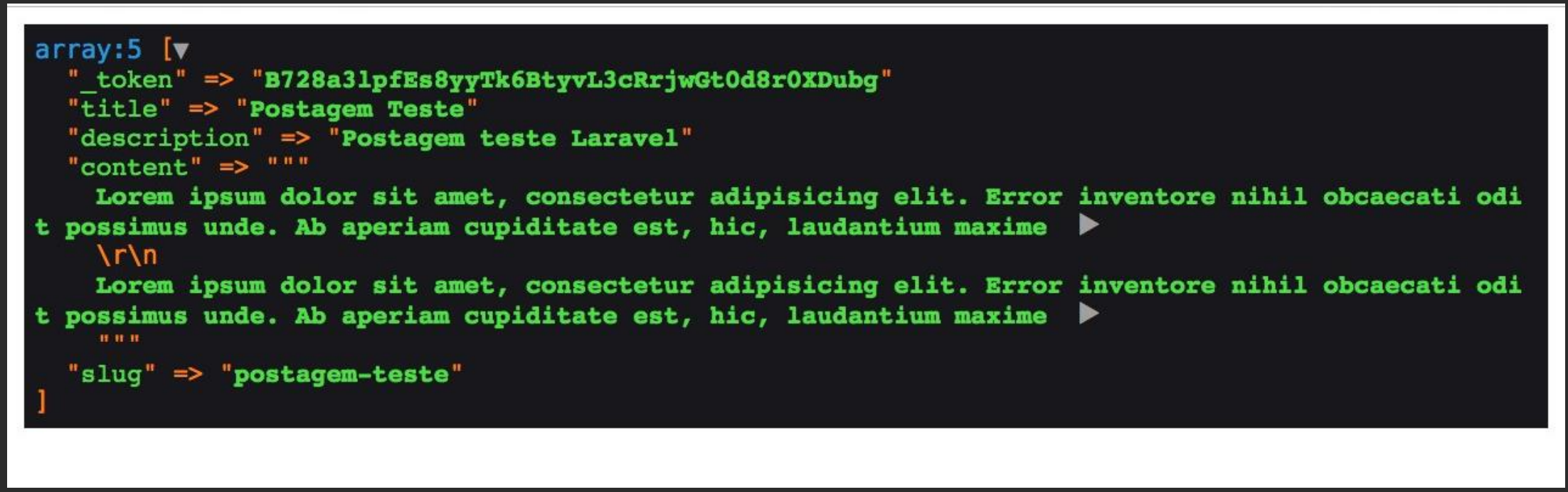
```
1    dd($request->all());
```

Acima temos contato com mais uma função helper do Laravel, o dd que simplesmente faz uma dump mais customizado dos dados informados e em seguida joga um die travando a continuação da execução do código. Por isso o dd ou dump and die.

Continuando...

Manipulando os dados da requisição

Preencha dados em seu formulário e envie novamente a requisição. Veja o resultado.



O resultado acima são todos os campos vindos do nosso formulário, o resultado do dump é o que temos acima mas os dados estão sendo resgatados por meio do método all do objeto Request que traz todos os dados enviados por meio desta requisição.

Recuperando valores específicos

Você pode acessar os campos diretamente, por exemplo se eu quiser acessar o titulo da postagem enviado do formulário eu posso acessar das seguintes maneiras:

```
1    dd($request->get('title'));
```

ou

```
1    dd($request->title);
```

ou ainda

```
1    dd($request->input('title'));
```

Isso vale para cada campo que você envia e que deseja recuperar o valor.

Verificando a existência de um dado na requisição

Podemos verificar ainda se determinado campo, parâmetro ou input foi informado em nossa requisição. Para isso podemos utilizar o método has do objeto request.

Veja abaixo:

```
1    if($request->has('title')) {
2            dd($request->title);
3    }
```

Com isso podemos realizar determinadas operações para a existência ou não existência dos campos. Se você deseja verificar a existência de vários inputs, você pode utilizar o método hasAny, veja abaixo:

```
1    if($request->hasAny(['title', 'content', 'slug'])) {
2            dd($request->title);
3    }
```

Recuperando campos específicos

Podemos recuperar inputs a nosso gosto caso precisarmos. Por exemplo, se eu quiser recuperar apenas o campo title e o campo slug da nosso request:

```
1    $request->only(['title', 'slug']);
```

Desta maneira vamos receber um array com os dois campos informados e seus respectivos valores. Caso você queira ignorar campos específicos também é possível, por exemplo:

```
1    $request->except(['title']);
```

Neste caso receberemos um array com todos os campos e somente o campo title não estará presente neste array. São método bem úteis quando precisamos destes comportamentos.

Trabalhando com Query Strings ou Parâmetros de URL

Parâmetros de url, as famosas query strings, são parâmetros informados em nossa url sempre após a ? e respeitando chave=valor e quando temos mais de um parâmetro são concatenados pelo &.

Por exemplo, como eu poderia recuperar o parâmetro search da seguinte url

http://aula.test?search=teste

Para acessarmos seu valor, nós podemos utilizar o método query para recuperar este input exclusivo vindo da url.

Por exemplo:

```
1    $request->query('search');
```

Trabalhando com Query Strings ou Parâmetros de URL

Este método, o query, assim como o get e o input, aceita um segundo parâmetro pro caso da não existência do parâmetro ou input solicitado. Podemos definir um valor default que será carregado caso não tenhamos o input em questão.

Veja abaixo:

```
1    $request->query('search', 'este valor será retornado caso não  tenhamos o  parâmetro s\
2    earch na  query string');
3
4    //Pro input e  pro   get
5
6    $request->get('title', 'este valor será retornado caso não  tenhamos o  parâmetro titl\
7    e  na  requisição');
8
9    $request->input('title', 'este valor será retornado caso não  tenhamos o  parâmetro ti\
10     tle na  requisição');
```

Manipulando as respostas HTTP

Podemos manipular as respostas http em nossos controllers ou funções anônimas em nossas rotas utilizando a função helper response. Veja abaixo uma utilização simples:

```
1    return  response('Retornando uma resposta',  200);
```

Acima retornei uma resposta com o conteúdo Retornando uma resposta e o status code http 200, que se refere a status de sucesso.

Podemos definir cabeçalhos HTTP em nossa resposta também. Por exemplo, posso dizer que o tipo da minha resposta é um json da seguinte maneira:

```
1    return  response('Retorno do  tipo json',  200)
2                        ->header('Content-Type', 'application/json');
```

Agora o tipo da resposta, que por default seria do tipo 'text/html', será do tipo 'application/json' por meio da manipulação do cabeçalho http Content-Type e informando o mime-type por meio deste cabeçalho, em nosso caso, colocando o tipo para json: application/json.

Manipulando as respostas HTTP

Podemos ainda manipular cookies e enviar quantos cabeçalhos forem necessários por meio do objeto response. Por exemplo, se você quiser retornar um cookie em uma determinada resposta, basta utilizar como abaixo:

```
1    return  response('Retorno do  tipo json',  200)
2                        ->cookie('nome_cookie', 'valor_cookie',  'tempo_em_minutos_de_validade_do\
3    _cookie');
```

Redirecionamentos

Dentro desta manipulação do response que têm a ver com o retorno de nossas rotas (quer seja em função anônima quer seja no método de um controller) precisamos abordar, também, sobre redirecionamentos.

Em alguns momentos vamos precisar apenas retornar um redirecionamento pós realização de uma determinada execução. Para isso temos a função helper redirect:

```
1    return redirect('/');
```

Após um determinado processo posso redirecionar o usuário para uma determinada url, acima redireciono ele para a página inicial do nosso site mas podemos melhorar mais ainda esses redirecionamentos, acima redirecionando ele para uma rota.

Redirecionamentos

Podemos redirecionar o usuário para a rota desejada por meio do apelido desta rota. Veja abaixo:

```
1    return redirect()->route('home');
```

Chamo a função redirect sem parâmetros, com isso terei o retorno do objeto \Illuminate\Routing\Redirector

e com isso posso ter acesso ao método route, que inclusive já utilizamos aqui nas aulas. Agora basta que eu informe o apelido da rota desejada, e se essa rota possuir parâmetros dinâmicos basta informar em um array no segundo parâmetro do route como já vimos anteriormente.

Redirecionamentos

Podemos utilizar também um redirect para o estado anterior de uma requisição, isso é perfeito para momentos de erro no processamento de determinado acesso ou envio de dados. Por exemplo, quando enviamos os dados de um formulário para o backend e temos algum erro de validação nos dados.

Para isso temos a função helper back. Por exemplo:

```
1    return back();
```

Que retornará o usuário para o estado anterior da requisição assim como o botão back do browser ou melhor simulando esse comportamento.

Geralmente usamos essa função back para momentos de erro de processo ou validações, e nestes casos você precisa voltar o usuário pro momento anterior, caso o usuário tenha mandado dados de um formulário podemos retornar os campos já digitados por ele também como podemos ver abaixo:

```
1    return back()->withInput();
```

Acima além de voltarmos pro estado da requisição anterior, estamos mandando de volta as inputs digitados também. Para manipularmos lá na view, no form e exibir os valores vindos do withInput podemos usar a função helper old nos inputs do nosso formulário.

Redirecionamentos

Lembra do nosso formulário? Olha como ele fica após adicionarmos essa possibilidade de pegar o valor novamente dos campos digitados anteriormente pelo usuário:

```
1   <form  action="{{route('posts.store')}}" method="post">
2
3           @csrf
4
5           <div   class="form-group">
6                   <label>Titulo</label>
7                   <input type="text"  name="title"  class="form-control" value="{{old('title')}}">
8
9           </div>
10
11           <div   class="form-group">
12                  <label>Descrição</label>
13                  <input type="text"  name="description"  class="form-control" value="{{old('description')}}">
14
15           </div>
16
17           <div   class="form-group">
18                  <label>Conteúdo</label>
```

Redirecionamentos

```
19                  <textarea  name="content" id=""  cols="30"  rows="10" class="form-control">{{old('content')}}</textarea>
20
21              </div>
22
23              <div   class="form-group">
24                  <label>Slug</label>
25                  <input type="text"  name="slug" class="form-control" value="{{old('slug')}}">
26              </div>
27
28              <button class="btn btn-lg btn-success">Criar  Postagem</button>
29      </form>
```

Perceba no textarea e nos atributos value de cada input que temos agora o print do retorno da função helper old onde informados o nome do campo que queremos recuperar o valor digitado anteriormente. Caso aquele campo em questão não esteja na requisição, adicionado pelo withInput, o campo simplesmente fica em branco e o usuário preenche tranquilamente.

Isso é perfeito para o usuário, que não precisará digitar tudo novamente caso tenhamos algum problema no backend e precisamos retornar para o estado anterior.

Exercício

Criar um formulário e um Controller para manipular a inserção do mural de recados.

Os dados a serem solicitados são:
Nome:
E-mail:
Cidade:
Recado:

Validar no PHP se algum dado não foi informado, caso isso ocorra retornar ao formulário com todos os dados digitados anteriorment preenchidos.