

TÓPICOS ESPECIAIS EM COMPUTAÇÃO II - A

João Vitor Veronese Vieira

FILTRAGEM DE DADOS EM UMA TELA DE CONSULTA

Erechim, 02 de Dezembro de 2017.

(https://github.com/joao-vieira/Tutorial_FiltragemDeDados)

→ Introdução

- Basicamente, este tutorial vai demonstrar como criar uma tela de listagem (contendo *todos os estados do Brasil*) e fazer com que um filtro possa ser aplicado sobre esses resultados.
- Este filtro consistirá de um *campo texto* acima da tabela, que filtrará os resultados conforme uma nova letra for digitada.

→ Introdução

Tutorial - Filtragem de ... x

localhost:8000/listagem

Search

Tutorial

Digite o valor que deseja aqui

#	Nome	UF
1	Acre	AC
2	Alagoas	AL
3	Amapá	AP
4	Amazonas	AM
5	Bahia	BA
6	Ceará	CE
7	Distrito Federal	DF
8	Espírito Santo	ES
9	Goiás	GO
10	Maranhão	MA
11	Mato Grosso do Sul	MS
12	Mato Grosso	MT
13	Minas Gerais	MG

→ Introdução

Tutorial - Filtragem de ... x

localhost:8000/listagem

Search

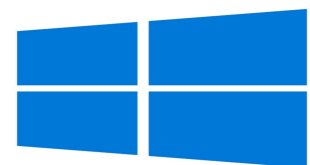
Tutorial

Rio

#	Nome	UF
19	Rio de Janeiro	RJ
20	Rio Grande do Norte	RN
21	Rio Grande do Sul	RS

→ Introdução

- Pontos Importantes:
 - Este tutorial considera que o ambiente propício para o desenvolvimento de uma aplicação *Laravel* já esteja *configurado* (inclusive conexão com Banco de Dados).
 - Este tutorial foi feito no sistema operacional *Linux*, portanto, qualquer diferença de detalhes deve ser ignorada.
 - Quando o tutorial referenciar a aplicação *Terminal*, pode-se entender *cmd* (para Windows).



→ Introdução

Criando o Projeto !

→ Criando o Projeto

- *Execute o comando*

```
# composer create-project --prefer-dist laravel/laravel /var/www/Tutorial_FiltragemDeDados
```

```
root@joao:/home/joao_vieira# composer create-project --prefer-dist laravel/laravel /var/www/Tutorial_FiltragemDeDados
Do not run Composer as root/super user! See https://getcomposer.org/root for details
Installing laravel/laravel (v5.4.30)
- Installing laravel/laravel (v5.4.30): Loading from cache
Created project in /var/www/Tutorial_FiltragemDeDados
> php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 59 installs, 0 updates, 0 removals
- Installing symfony/css-selector (v3.3.10): Loading from cache
- Installing tijsverkoyen/css-to-inline-styles (2.2.0): Loading from cache
- Installing doctrine/inflector (v1.1.0): Loading from cache
```

→ Criando o Projeto

- Pronto! A aplicação básica, utilizando o framework *Laravel*, está criada.
- Abra este projeto em um editor de texto de sua preferência. Este tutorial utilizará o *Sublime Text 3*.

→ Criando o Projeto

Configurando o Banco de Dados !

→ Configurando o B.D.

- Agora, criaremos um novo *modelo*, o qual chamaremos de Estados.

```
# php artisan make:model Estados -m
```

```
root@joao:/var/www/Tutorial_FiltragemDeDados# php artisan make:model Estados -m  
Model created successfully.  
Created Migration: 2017_11_09_231416_create_estados_table
```

- O parâmetro *-m* indica que queremos criar uma *migração*, vinculada ao modelo *Estados*.

→ Configurando o B.D.

- A migração que acabou de ser criada será alterada. Ao fim da alteração, a classe deve ficar assim:

Caminho:

/database/migrations/<migracao_estados>

```
1  <?php
2
3  use Illuminate\Support\Facades\Schema;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Database\Migrations\Migration;
6
7  class CreateEstadosTable extends Migration
8  {
9      /**
10       * Run the migrations.
11       *
12       * @return void
13       */
14     public function up()
15     {
16         Schema::create('estados', function (Blueprint $table) {
17             $table->increments('id');
18             $table->string('nome', 60);
19             $table->string('uf', 2);
20             $table->timestamps();
21         });
22     }
23
24     /**
25      * Reverse the migrations.
26      *
27      * @return void
28      */
29     public function down()
30     {
31         Schema::dropIfExists('estados');
32     }
33 }
34
```

→ Configurando o B.D.

Estados.php (dentro da pasta 'app')

- Na classe do modelo que também acabou de ser criado, adicione a seguinte linha:

```
protected $fillable = ['id', 'nome', 'uf'];
```

→ Configurando o B.D.

- Para que seja evitado o trabalho de realizar um cadastro dos estados, tendo em vista que o objetivo deste tutorial é realizar a filtragem dos dados, vamos criar uma *Seeder* (ou semeador).
- Toda vez que o nosso banco for criado, utilizaremos um parâmetro a mais que invoca esta “Seeder”, fazendo com que ela “semeie” os registros em nossa tabela.

→ Configurando o B.D.

- *Execute o comando*

php artisan make:seeder EstadosTableSeeder

```
root@joao:/var/www/Tutorial_FiltragemDeDados# php artisan make:seeder EstadosTableSeeder  
Seeder created successfully.
```

→ Configurando o B.D.

- Acesse o link abaixo e faça download do arquivo `EstadosTableSeeder.php`:

LINK

- O arquivo que você baixou é a Seeder pronta. Copie o conteúdo presente no arquivo e cole na sua própria Seeder, que se encontra em:

`/database/seeds/EstadosTableSeeder.php`

→ Configurando o B.D.

- Agora, só precisamos “mostrar ao Laravel” que esta Seeder deve ser executada sempre que o comando de migração for acionado.
- Abra o arquivo **DatabaseSeeder.php** (/database/seeds/) e o deixe desta forma:

```
1  <?php
2
3  use Illuminate\Database\Seeder;
4
5  class DatabaseSeeder extends Seeder
6  {
7      /**
8       * Run the database seeds.
9       *
10      * @return void
11      */
12      public function run()
13      {
14          $this->call(EstadosTableSeeder::class);
15      }
16  }
```


→ Configurando o B.D.

- Depois que tiver feito isso, rode o seguinte comando:

```
# php artisan migrate --seed
```

- Se todos os passos listados até aqui foram seguidos, o banco escolhido deverá conter uma tabela *estados*, desta forma:

	id [PK] serial	nome character varying(60)	uf character varying(2)	created_at timestamp(0) without time zone	updated_at timestamp(0) without time zone
1	1	Acre	AC	2017-11-10 01:25:29	2017-11-10 01:25:29
2	2	Alagoas	AL	2017-11-10 01:25:29	2017-11-10 01:25:29
3	3	Amapá	AP	2017-11-10 01:25:29	2017-11-10 01:25:29
4	4	Amazonas	AM	2017-11-10 01:25:29	2017-11-10 01:25:29
5	5	Bahia	BA	2017-11-10 01:25:29	2017-11-10 01:25:29
6	6	Ceará	CE	2017-11-10 01:25:29	2017-11-10 01:25:29

→ Configurando o B.D.

Criando a Listagem !

→ Criando a Listagem

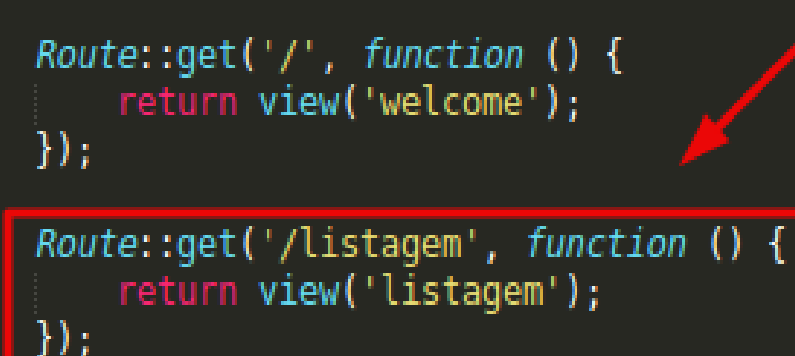
- Agora que temos dados suficientes para preencher nossa tabela, vamos criar a listagem dos estados.
- Antes de mais nada, vamos facilitar nosso acesso à listagem. Abra seu arquivo `welcome.blade.php` (`/resources/views/welcome.blade.php`).
- Quase no fim do arquivo (linha 91), adicione:

```
<a href="https://github.com/laravel/laravel">GitHub</a>  
<a href="{{ url('/listagem') }}">Listagem</a>  
</div>
```

→ Criando a Listagem

- Com isso, fica mais prático acessarmos nossa página de listagem. Porém, precisamos definir a rota desta página.
- Acesse seu arquivo **web.php**, dentro da pasta *routes* e adicione:

```
13
14 Route::get('/', function () {
15     return view('welcome');
16 });
17
18 Route::get('/listagem', function () {
19     return view('listagem');
20 });
```



→ Criando a Listagem

- Podemos reparar que nossa rota, ao ser acionada, nos direciona para uma visão ' *listagem* '.
- Porém, ainda não criamos esta visão. Vamos fazer isso agora.
 - Crie um novo arquivo dentro de: `resources/views/`
 - Nomeie este arquivo como: `listagem.blade.php`

→ Criando a Listagem

- Abra o arquivo de listagem que você acabou de criar.
- Deixe o conteúdo dele como na imagem abaixo:

```
listagem.blade.php x
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Tutorial - Filtragem de Dados</title>
5     <link href="{{ asset('/css/app.css') }}" rel="stylesheet">
6   </head>
7
8   <body>
9     <nav class="navbar navbar-default">
10       <div class="container-fluid">
11         <div class="navbar-header">
12           <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1">
13             <span class="sr-only">Toggle Navigation</span>
14             <span class="icon-bar"></span>
15             <span class="icon-bar"></span>
16             <span class="icon-bar"></span>
17           </button>
18           <a class="navbar-brand" href="#">Tutorial</a>
19         </div>
20       </div>
21     </nav>
22
23     <div class="container">
24       <div class="row">
25         <table class="table table-striped table-bordered table-hover">
26           <thead>
27             <tr>
28               <th>#</th>
29               <th>Nome</th>
30               <th>UF</th>
31             </tr>
32           </thead>
33           <tbody>
34
35           </tbody>
36         </table>
37       </div>
38     </div>
39   </body>
40 </html>
```

→ Criando a Listagem

- Feito isso, inicie o seu projeto. Acesse o diretório onde ele está localizado e execute:

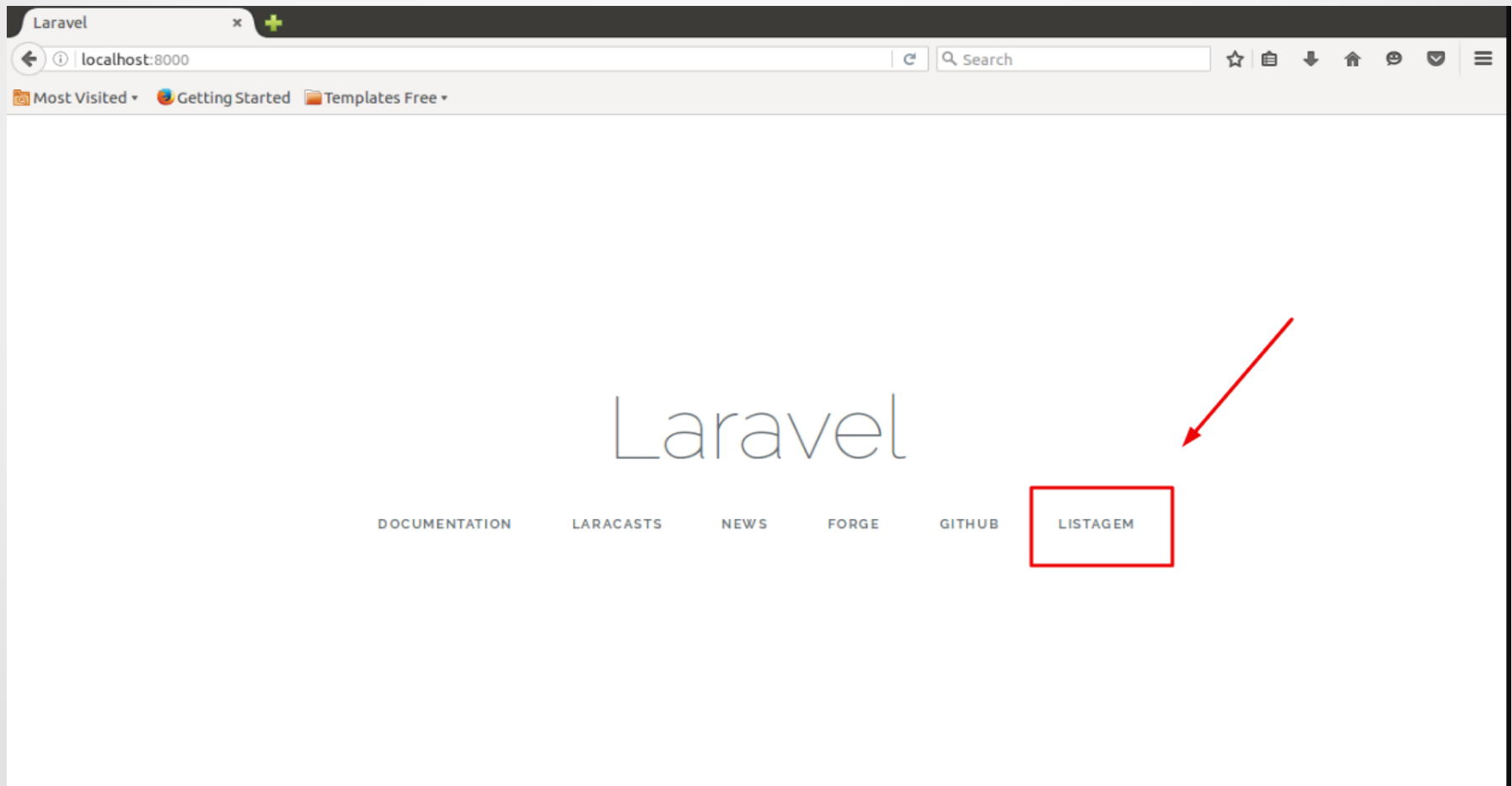
```
# php artisan serve
```

- Agora, acesse seu projeto:

```
http://localhost:8000/
```

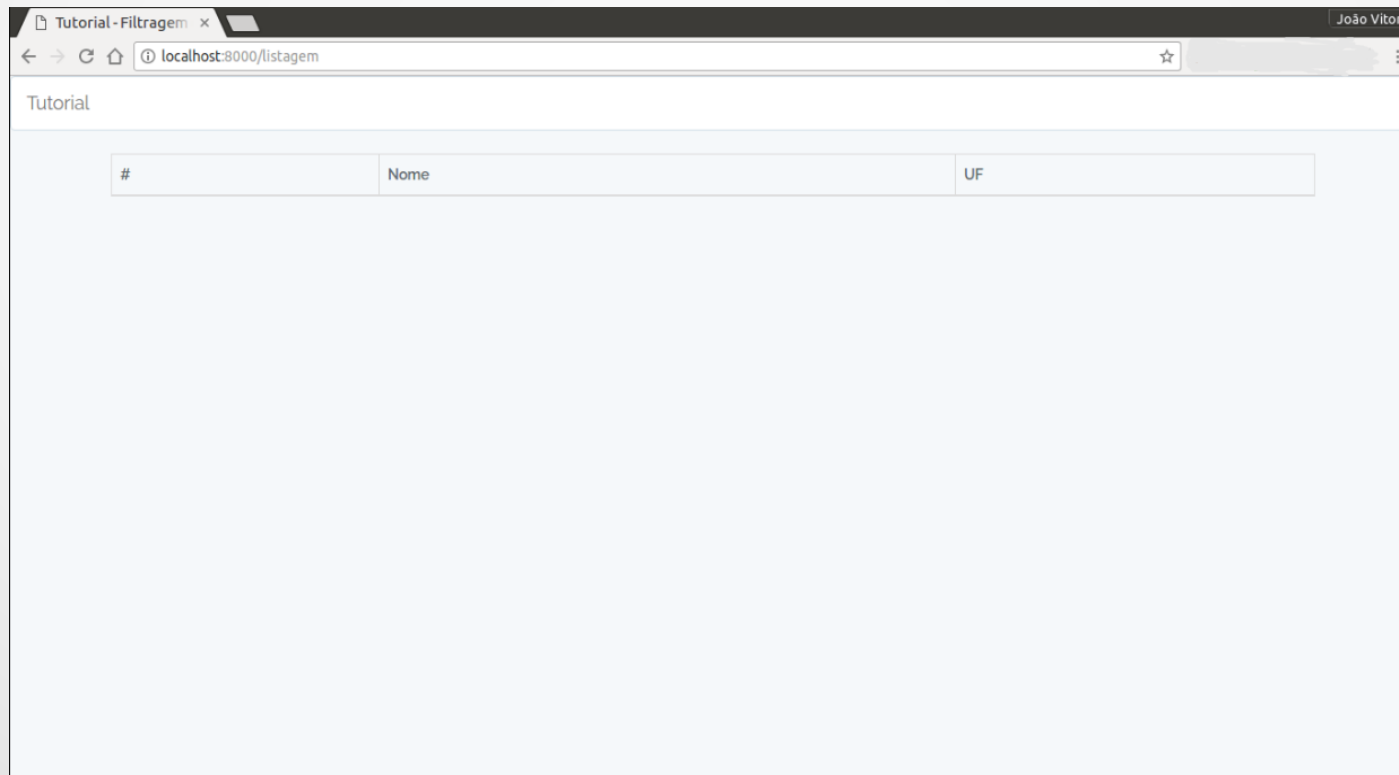
→ Criando a Listagem

- Note que as ações que executamos geraram o seguinte resultado:



→ Criando a Listagem

- Clique em “Listagem”
- Esta é a tela onde iremos filtrar os estados:



→ Criando a Listagem

Preenchendo a Tabela !

→ Preenchendo a Tabela

- Agora, para que os estados que já estão cadastrados no banco de dados sejam listados, teremos de fazer algumas modificações.
- Antes de mais nada, rode o seguinte comando no Terminal:

```
# php artisan make:controller EstadosController
```

```
root@joao:/var/www/Tutorial_FiltragemDeDados# php artisan make:controller EstadosController  
Controller created successfully.
```

→ Preenchendo a Tabela

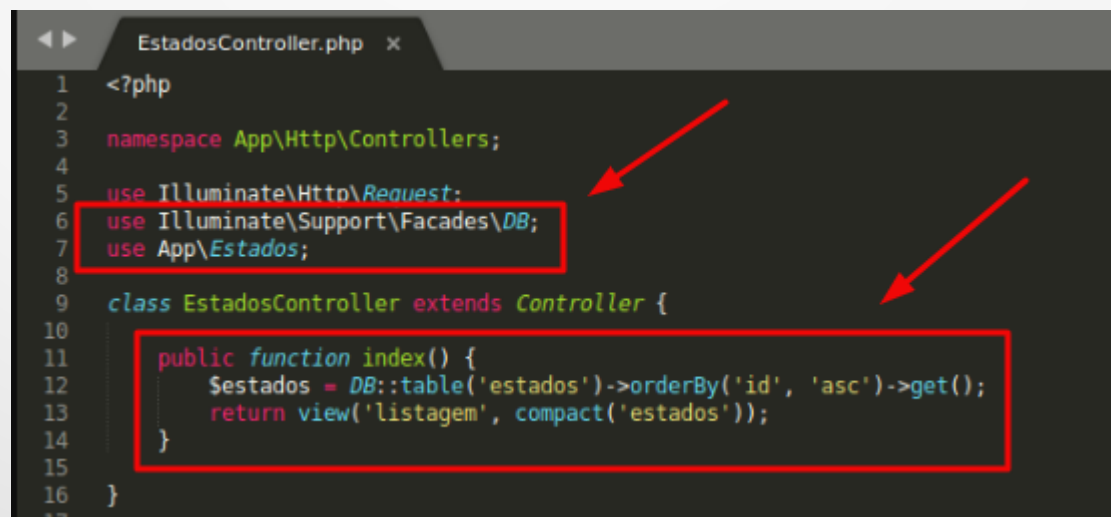
- Em seguida, abra seu arquivo `web.php` (rotas)
- No local onde estávamos retornando uma *visão*, mude o código, tornando-o igual ao que segue abaixo:

```
13
14 Route::get('/', function () {
15     return view('welcome');
16 });
17
18 Route::get('/listagem', 'EstadosController@index');
```



→ Preenchendo a Tabela

- Esta mudança faz com que agora, ao clicar em “Listagem” na tela de boas-vindas, a função *index* do controlador *EstadosController* seja invocada.
- Deixe seu arquivo **EstadosController.php** desta forma:



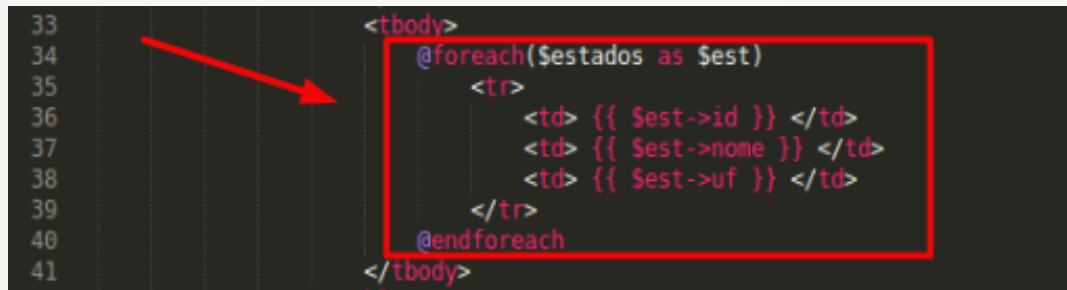
```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\DB;
7 use App\Estados;
8
9 class EstadosController extends Controller {
10
11     public function index() {
12         $estados = DB::table('estados')->orderBy('id', 'asc')->get();
13         return view('listagem', compact('estados'));
14     }
15
16 }
```

→ Preenchendo a Tabela

- Agora, nossa visão já possui os dados dos estados cadastrados no banco de dados, pois o controlador realizou a consulta e está enviando esses dados até ela
- Faltam apenas, portanto, mostrar essas informações na tela.
- Para isso, abra seu arquivo [listagem.blade.php](#)

→ Preenchendo a Tabela

- Incremente o seguinte trecho de código neste arquivo :



```
33
34
35
36
37
38
39
40
41

    @foreach($sestados as $sestado)
        <tr>
            <td> {{ $sestado->id }} </td>
            <td> {{ $sestado->nome }} </td>
            <td> {{ $sestado->uf }} </td>
        </tr>
    @endforeach
</tbody>
```

- Agora, vá até o navegador em que a tela de listagem está aberta e *atualize* a página

→ Preenchendo a Tabela

- Esta é a tela que criamos até agora:

Tutorial		
#	Nome	UF
1	Acre	AC
2	Alagoas	AL
3	Amapá	AP
4	Amazonas	AM
5	Bahia	BA
6	Ceará	CE
7	Distrito Federal	DF
8	Espírito Santo	ES
9	Goiás	GO
10	Maranhão	MA
11	Mato Grosso do Sul	MS
12	Mato Grosso	MT
13	Minas Gerais	MG
14	Paraná	PR

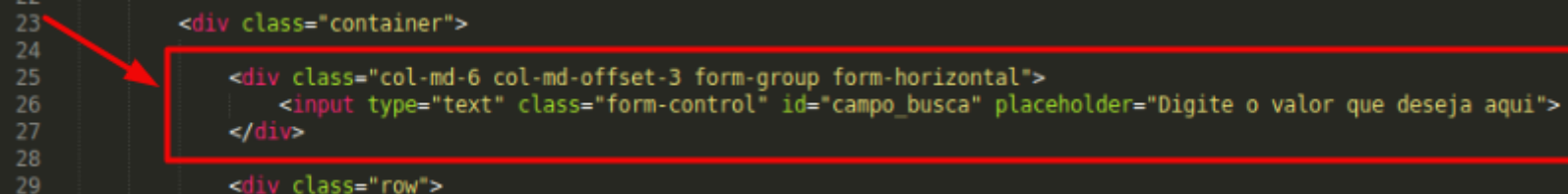
→ Preenchendo a Tabela

Criando o Filtro !

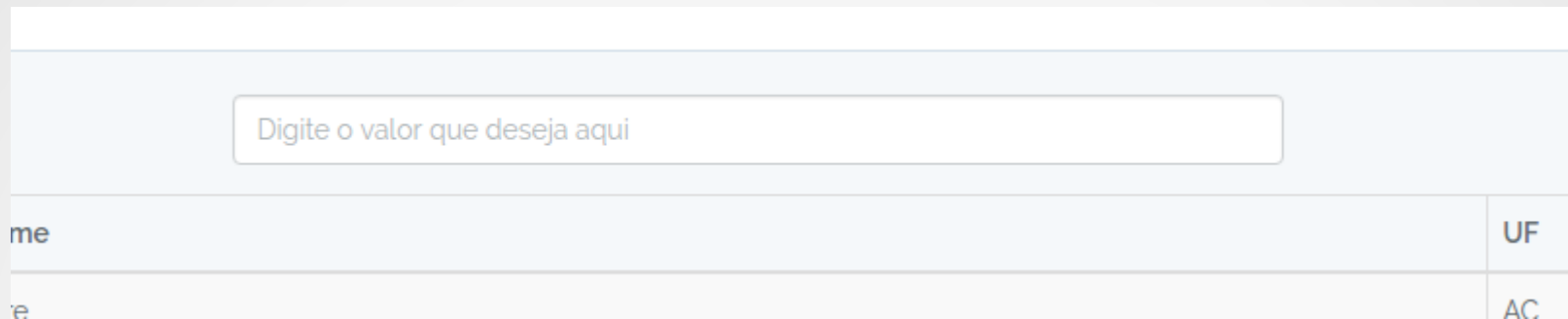
→ Criando o Filtro

- Agora, vamos iniciar a criação da estrutura que irá filtrar os valores presentes em nossa tabela dinamicamente.
- O primeiro passo é criar o campo de texto na tela de listagem (`listagem.blade.php`)

```
23 <div class="container">
24
25     <div class="col-md-6 col-md-offset-3 form-group form-horizontal">
26         <input type="text" class="form-control" id="campo_busca" placeholder="Digite o valor que deseja aqui">
27     </div>
28
29     <div class="row">
```



→ Criando o Filtro



The screenshot shows a web interface with a search bar and a table. The search bar is a light blue rectangle with the placeholder text "Digite o valor que deseja aqui". Below it is a table with two columns. The first column contains the names of Brazilian states, and the second column contains their abbreviations (UF). The first row shows "me" and "UF". The second row shows "e" and "AC".

	UF
me	
e	AC

- Agora que o campo de texto está criado, vamos construir o código que irá filtrar os estados listados na tabela.
- Para realizarmos esta ação, usaremos *jQuery* e *Javascript*.

→ Criando o Filtro

- Adicione a propriedade *id* à tag HTML `<tbody>` de sua tabela.

```
</tr>
</thead>
<tbody id="corpo_tabela">
  @foreach($estados as $est)
    <tr>
```

- No mesmo arquivo, antes da tag `</body>`, adicione:

```
</div>
<script type="text/javascript">
</script>
</body>
```

- É dentro desta tag `<script>` que escreveremos o código de filtragem.

→ Criando o Filtro

- **keyup()**

- O método Javascript *keyup* é disparado quando uma tecla do teclado é pressionada em um local escolhido
- Desta forma, vamos especificar que, quando uma tecla for pressionada dentro do campo de texto que criamos (referenciado pelo seu id), algo será executado:

```
<script type="text/javascript">  
  $("#campo_busca").keyup(function () {  
    });  
</script>
```

Obs: Note que duas linhas foram adicionadas dentro da *tag* `<script>` que criamos anteriormente e que “campo_busca” é o *id* do *input* que criamos acima de nossa tabela.

→ Criando o Filtro

- Agora, toda vez que uma tecla for pressionada enquanto o cursor estiver dentro do campo de texto que criamos, uma função será executada.
- Apenas à título de conhecimento, adicione esta linha dentro da função criada no slide anterior:

```
console.log(this.value);
```

Obs: Atualize a página, abra a ferramenta de desenvolvedor do navegador (clique em F12 no seu teclado), vá até a aba “Console” e digite algo dentro do campo de texto. Note que *this*, no comando acima, faz referência ao campo de texto, por isso o valor que você digitar irá aparecer nos “logs” do navegador.

→ Criando o Filtro

- Depois de entendermos a função que estamos criando, caso tenha feito o teste sugerido no slide anterior, apague a última linha adicionada (*console.log.....*).
- *No lugar desta linha, adicione:*

```
var linhasTabela = $("#corpo_tabela").find("tr").hide();
```

Obs: Este trecho de código “procura” todas as linhas da tabela de estados quando a função é invocada e as “esconde”. Com isso, após digitado algum parâmetro de filtro no campo de texto, a função mostra apenas as linhas que satisfazem a condição de filtro.

→ Criando o Filtro

- Agora, adicionaremos uma condicional:
 - Se existir algum *caracter* presente no campo de texto, iremos mostrar apenas as linhas que contêm este(s) *caractere(s)*.
 - Se não, mostramos todas as linhas novamente (ex: quando o conteúdo do campo de texto for removido).
- Nosso método está assim, por enquanto:

```
<script type="text/javascript">
  $("#campo_busca").keyup(function () {
    var linhasTabela = $("#corpo_tabela").find("tr").hide();

    if (this.value.length) {
    } else {
      linhasTabela.show();
    }
  });
</script>
```


→ Criando o Filtro

- **split()**

- O método *split* divide um objeto *String* em um *array* de *strings*, separando o conteúdo selecionado através de um parâmetro.
- Em nosso caso, nosso parâmetro será um *espaço* (*character* em branco).
- Isso será feito para que o filtro se aplique em mais de um registro. Ex: Duas palavras são digitadas e uma palavra aparece em uma linha e a outra em outra linha. Com isso, as duas linhas continuarão sendo mostradas após a filtragem.

→ Criando o Filtro

- Portanto, adicione a seguinte linha dentro do condicional if de nosso método:

```
var palavrasChaveBusca = this.value.split(" ");
```

- Esta linha abrange a explicação contida no slide anterior.
- Por fim, vamos *percorrer* todas as palavras digitadas no campo de texto com o método *\$.each*, do jQuery.

→ Criando o Filtro

- Para meios de comparação, nosso método está assim até agora:

```
<script type="text/javascript">
    $("#campo_busca").keyup(function () {
        var linhasTabela = $("#corpo_tabela").find("tr").hide();

        if (this.value.length) {
            var palavrasChaveBusca = this.value.split(" ");
            $.each(palavrasChaveBusca, function (indice, palavra) {

            });
        } else {
            linhasTabela.show();
        }
    });
</script>
```

→ Criando o Filtro

- **filter()**

- O método *filter* cria um novo *array* com todos os elementos que passaram no teste implementado pela função fornecida.
- Ou seja, no nosso caso, todas as linhas que contiverem o valor digitado no campo de busca, serão adicionadas nesse *array* momentâneo do método *filter* e, após, serão mostradas novamente.

→ Criando o Filtro

- O último passo é adicionar a seguinte linha dentro do método each:

```
linhasTabela.filter(":contains('" + palavra + "')").show();
```

→ Criando o Filtro

- Versão final do método:

```
52     <script type="text/javascript">
53         $("#campo_busca").keyup(function () {
54             var linhasTabela = $("#corpo_tabela").find("tr").hide();
55
56             if (this.value.length) {
57                 var palavrasChaveBusca = this.value.split(" ");
58                 $.each(palavrasChaveBusca, function (indice, palavra) {
59                     linhasTabela.filter(":contains('" + palavra + "')").show();
60                 });
61             } else {
62                 linhasTabela.show();
63             }
64         });
65     </script>
```

- Com isso, a filtragem sobre os estados listados na tabela está pronta. Bom trabalho!

→ Criando o Filtro

Atualização: versão 2!

→ Atualização: versão 2

- Com listagens simples, que envolvam poucos registros, esse filtro é extremamente agradável
- Agora, já imaginou remover, mostrar e esconder as linhas da tabela com mais de 80.000 registros ?
- Poisé, a gente fará isso. Acesse novamente a pasta no Google Drive e, agora, faça o download do arquivo:
[EstadosTableSeeder_v2.php](#)

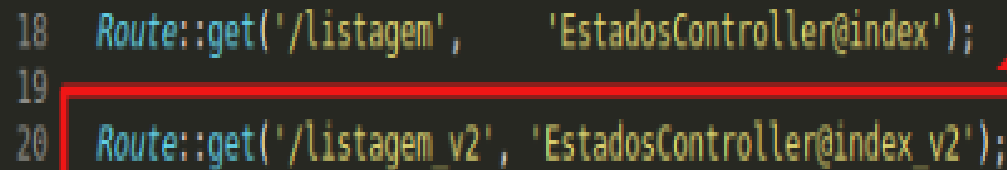
→ Atualização: versão 2

- Antes de colarmos o conteúdo deste arquivo em uma Seeder que iremos criar, vamos configurar as rotas e alterar alguns pontos para que criemos *duas telas de filtros*
 - Uma será para poucos registros, com filtro em *Javascript* e *jQuery*
 - A outra será para vários registros, usando o próprio *Laravel* e seu poderoso *Eloquent ORM*

→ Atualização: versão 2

- ***web.php***

```
18 Route::get('/listagem', 'EstadosController@index');  
19  
20 Route::get('/listagem_v2', 'EstadosController@index_v2');
```

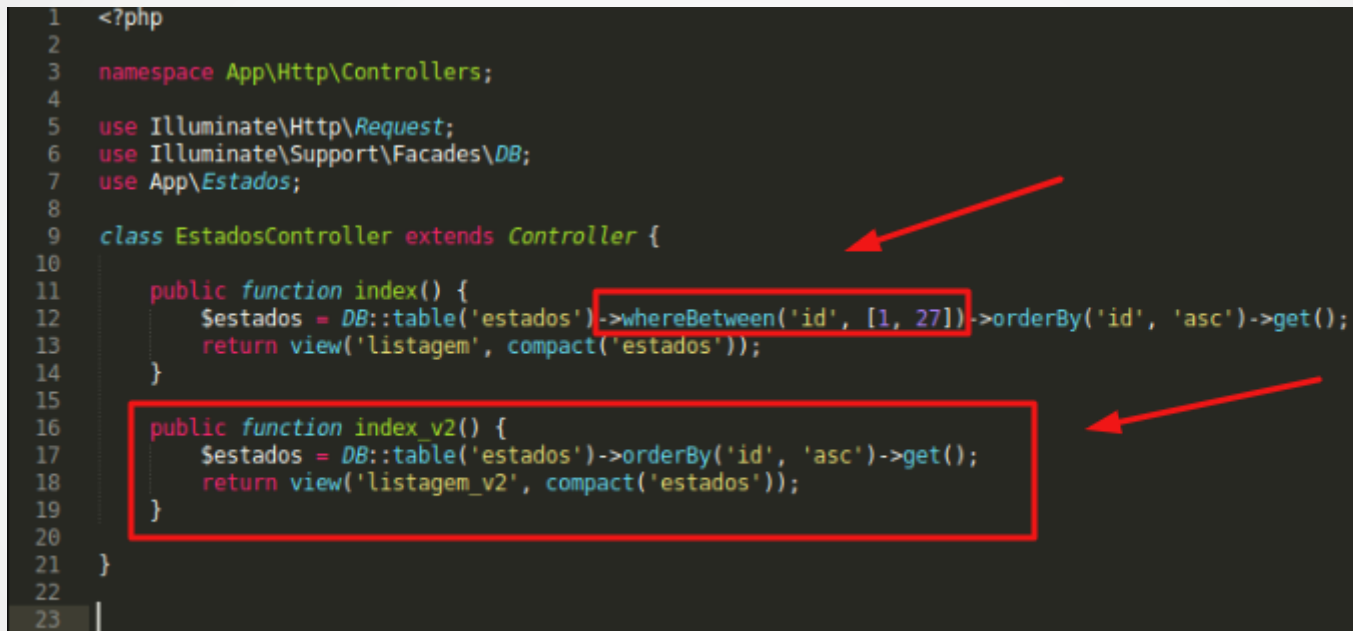


Explicação: Criamos mais uma rota, chamada “*listagem_v2*”, para capturar esse novo teste que iremos fazer e invocar a função “*index_v2*”, do controlador *Estados*.

→ Atualização: versão 2

- **EstadosController.php**

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\DB;
7 use App\Estados;
8
9 class EstadosController extends Controller {
10
11     public function index() {
12         $estados = DB::table('estados')->whereBetween('id', [1, 27])->orderBy('id', 'asc')->get();
13         return view('listagem', compact('estados'));
14     }
15
16     public function index_v2() {
17         $estados = DB::table('estados')->orderBy('id', 'asc')->get();
18         return view('listagem_v2', compact('estados'));
19     }
20 }
21
22
23 |
```



Explicação: Alteramos nossa consulta ao banco no método *index*, limitando para os estados originais (1 até 27) e criamos um novo método: *index_v2*.

→ Atualização: versão 2

- ***listagem_v2.blade.php***

```
24 <div class="container">
25
26 <div class="col-md-6 col-md-offset-3 form-group">
27 <form class="form-inline" method="POST" action="{{ url('submit') }}">
28 {{ csrf_field() }}
29 <label for="input_busca" class="sr-only">Pesquise ao lado:</label>
30 <input type="text" class="form-control" id="input_busca" name="input_busca" placeholder="Pesquisa aqui" style="width: 80%">
31 <button type="submit" class="btn btn-success">Buscar</button>
32 </form>
33 </div>
34
35 <div class="row">
```

Explicação: Copie o arquivo *listagem.blade.php* (dentro da pasta *resources/views/*), cole na mesma pasta e altere o nome do arquivo para *listagem_v2.blade.php*.

Feito isso, faça duas coisas:

1. Apague a tag `<script>` e a função Javascript que criamos.
2. Troque o campo de texto pelo código mostrado na imagem acima.

→ Atualização: versão 2

- Ok, feito isso, vamos criar mais de **80.000** registros!
- *Execute o comando*

php artisan make:seeder EstadosTableSeeder_v2

```
root@joao:/var/www/Tutorial_FiltragemDeDados# php artisan make:seeder EstadosTableSeeder_v2
```

→ Atualização: versão 2

- Abra este arquivo que acabamos de criar e, dentro dele, coloque todo o conteúdo contido no arquivo que você baixou do Google Drive ([EstadosTableSeeder_v2.php](#))
- Feito isso, abra o arquivo [DatabaseSeeder.php](#)

```
12 public function run()  
13 {  
14     $this->call(EstadosTableSeeder::class);  
15     $this->call(EstadosTableSeeder_v2::class);  
16 }
```

Deixe seu método *run* como na imagem.

→ Atualização: versão 2

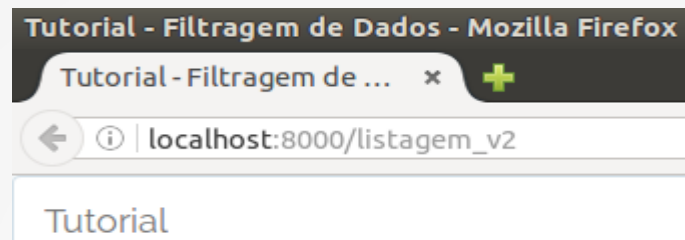
- Depois que tiver feito isso, rode o comando:

```
# php artisan migrate --seed
```

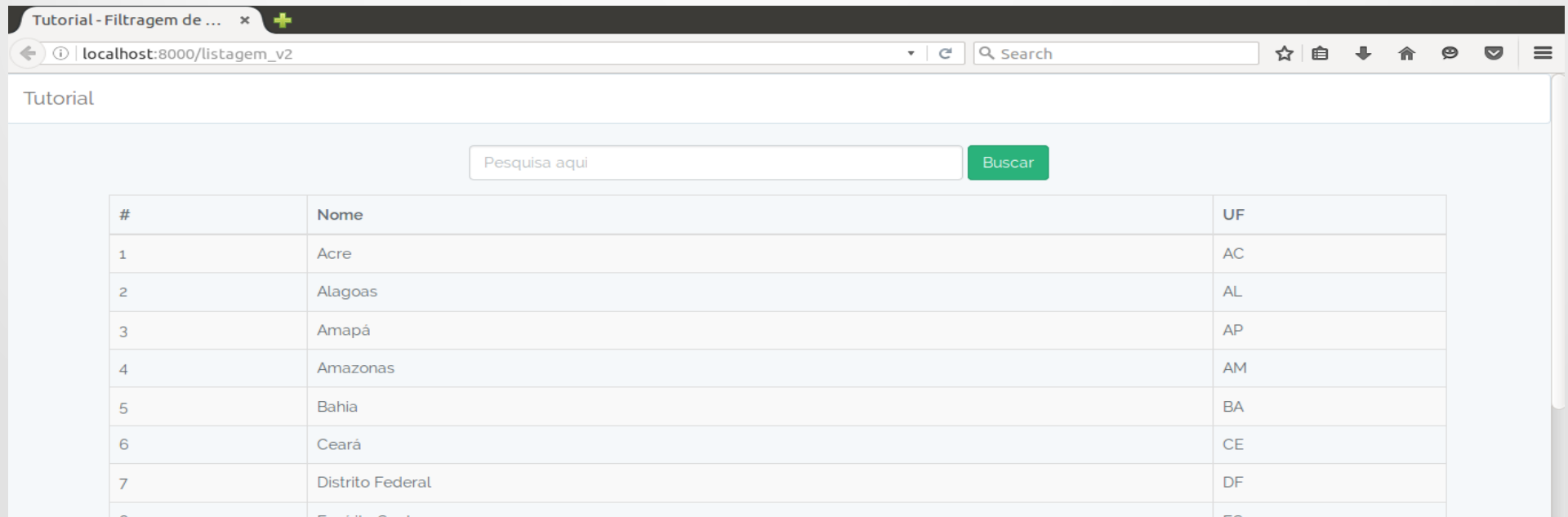
- **Vá buscar um café!** (isso vai demorar...)
- Se você ainda está aí quando o comando acabou de ser executado, verifique sua tabela. Ela deve ter mais de **84.000** registros!

→ Atualização: versão 2

- Agora, acesse em seu browser:



- A tela está assim ?



→ Atualização: versão 2

- Perfeito. Agora resta apenas criarmos uma rota para quando o *formulário* de busca que criamos for enviado e um método no controlador para executar nosso filtro.

- **web.php**

```
21  
22 Route::post('submit', 'EstadosController@busca');
```

Adicione a rota acima no arquivo **web.php**

→ Atualização: versão 2

- **EstadosController.php**

```
22     public function busca(Request $request) {  
23         $pesquisa = $request->input_busca;  
24         $estados = DB::table('estados')->where('nome', 'like', "%{$pesquisa}%")->orderBy('id', 'asc')->get();  
25         return view('listagem_v2', compact('estados'));  
26     }  
27
```

E, adicione o método acima no arquivo **EstadosController.php**

- **Pronto!** Nosso filtro está pronto. (Lembrando que estamos filtrando apenas pelo **nome do estado**, por enquanto)

→ Atualização: versão 2

Tutorial - Filtragem de ... x +

localhost:8000/listagem_v2 Search

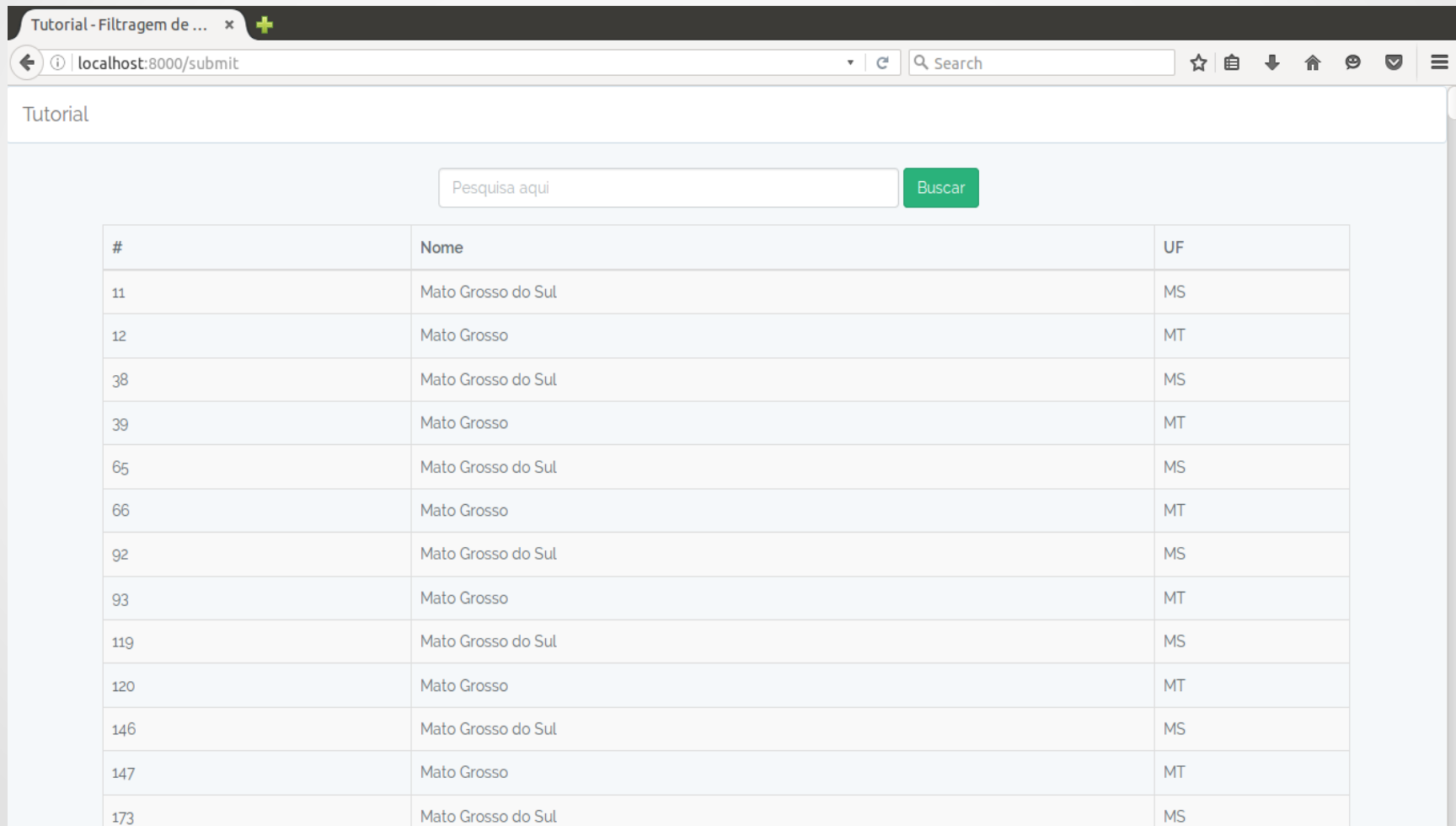
Tutorial

Pesquisa aqui Buscar

#	Nome	UF
1	Acre	AC
2	Alagoas	AL
3	Amapá	AP
4	Amazonas	AM
5	Bahia	BA
6	Ceará	CE
7	Distrito Federal	DF
8	Espírito Santo	ES
9	Goiás	GO
10	Maranhão	MA
11	Mato Grosso do Sul	MS
12	Mato Grosso	MT
13	Minas Gerais	MG

→ Atualização: versão 2

- Após ter realizado a busca por “Mato”:



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/submit'. The page title is 'Tutorial'. Below the title, there is a search bar with the placeholder text 'Pesquisa aqui' and a green 'Buscar' button. The search results are displayed in a table with three columns: '#', 'Nome', and 'UF'. The table contains 13 rows of data, all of which are 'Mato Grosso do Sul' or 'Mato Grosso' with corresponding state abbreviations 'MS' or 'MT'.

#	Nome	UF
11	Mato Grosso do Sul	MS
12	Mato Grosso	MT
38	Mato Grosso do Sul	MS
39	Mato Grosso	MT
65	Mato Grosso do Sul	MS
66	Mato Grosso	MT
92	Mato Grosso do Sul	MS
93	Mato Grosso	MT
119	Mato Grosso do Sul	MS
120	Mato Grosso	MT
146	Mato Grosso do Sul	MS
147	Mato Grosso	MT
173	Mato Grosso do Sul	MS

→ Atualização: versão 2

O B R I G A D O !