



Marcos Rodrigo Momo, M.Sc.
e-mail: marcos.momo@ifsc.edu.br

Programação para Internet II



**INSTITUTO
FEDERAL**
Santa Catarina

Câmpus
Gaspar

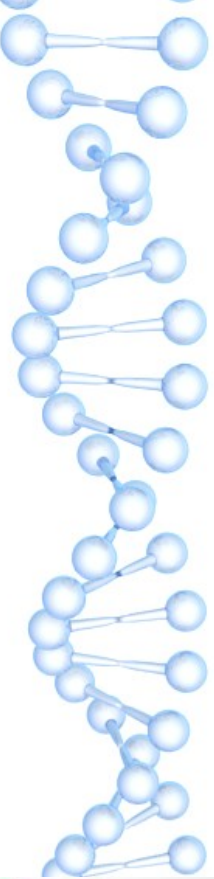
Gaspar, maio 2021.

Roteiro

- Métodos do sistema de rotas
- Métodos de controller resource
 - `php artisan make:controller meucontrolador - -resource`
- Práticas
- Atividade extraclasse

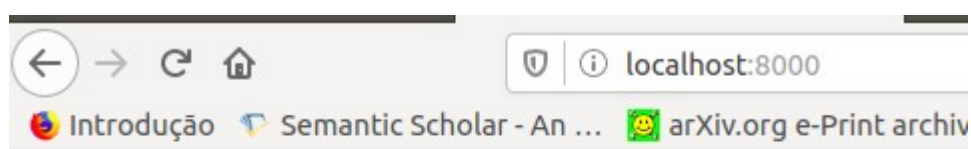
Criando um controlador resource

`php artisan make:controller ClienteControlador - -resource`



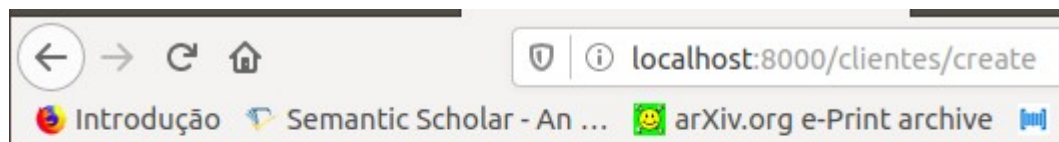
Métodos criados no Laravel meucontrolador

- public function index()
- public function create()
- public function store(Request \$request)
- public function show(\$id)
- public function edit (\$id)
- public function update(Request \$request, \$id)
- public function destroy(\$id)



Sistema de Cadastro

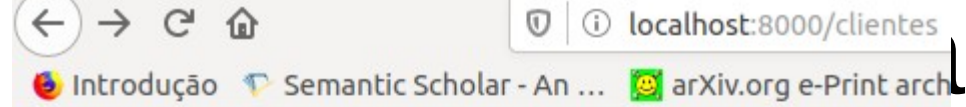
[Cadastrar cliente](#)



Novo Cliente

[Voltar](#)

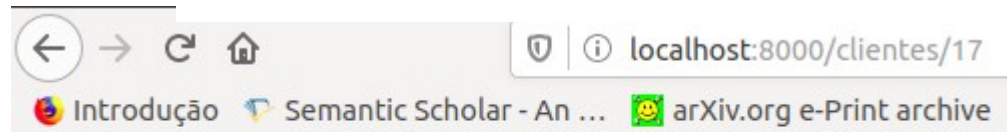
Final da atividade!!!!



Clientes:

[NOVO CLIENTE](#)

1. Maria| [Editar](#) [Mostrar](#) [Voltar](#)
2. João| [Editar](#) [Mostrar](#) [Voltar](#)
3. Paulo| [Editar](#) [Mostrar](#) [Voltar](#)
4. Pedro| [Editar](#) [Mostrar](#) [Voltar](#)



Dados do Cliente

ID: 17

Nome: Maria

[Voltar](#)

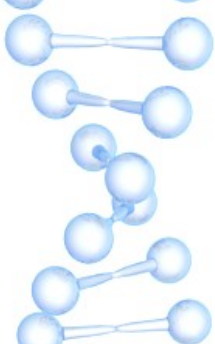
Arquivo de rota

- Associar cada rota aos métodos criados pelos meucontrolador

web.php ×

routes > web.php > ...

```
39 Route::get('multiplica/{n1}/{n2}', [MeuControlador::class, 'mul  
40  
41 Route::resource('cliente', ControladorCliente::class);  
42
```



Associando as rotas aos métodos criados

Fazer o teste!!!!

```
$ php artisan route:list
```

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	/		Closure	web
	GET HEAD	api/user		Closure	api auth:api
	POST	cliente	cliente.store	App\Http\Controllers\ControladorCliente@store	web
	GET HEAD	cliente	cliente.index	App\Http\Controllers\ControladorCliente@index	web
	GET HEAD	cliente/create	cliente.create	App\Http\Controllers\ControladorCliente@create	web
	PUT PATCH	cliente/{cliente}	cliente.update	App\Http\Controllers\ControladorCliente@update	web
	GET HEAD	cliente/{cliente}	cliente.show	App\Http\Controllers\ControladorCliente@show	web
	DELETE	cliente/{cliente}	cliente.destroy	App\Http\Controllers\ControladorCliente@destroy	web
	GET HEAD	cliente/{cliente}/edit	cliente.edit	App\Http\Controllers\ControladorCliente@edit	web
	GET HEAD	dados		App\Http\Controllers\ControladorDados@dados	web
	POST	dados		App\Http\Controllers\ControladorDados@dados	web
	GET HEAD	divide/{n1}/{n2}		App\Http\Controllers\MeuControlador@divisão	web
	GET HEAD	idade		App\Http\Controllers\MeuControlador@getidade	web
	GET HEAD	multiplica/{n1}/{n2}		App\Http\Controllers\MeuControlador@multiplicar	web
	GET HEAD	nome		App\Http\Controllers\MeuControlador@getnome	web
	GET HEAD	produtos		App\Http\Controllers\MeuControlador@produtos	web
	GET HEAD	soma/{n1}/{n2}		App\Http\Controllers\MeuControlador@somar	web
	GET HEAD	subtrai/{n1}/{n2}		App\Http\Controllers\MeuControlador@subtrair	web

```
$ □
```


Criar uma variável cliente no controlador com os seguintes dados

```
app > Http > Controllers > ControleCliente.php > ControleCliente > index
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class ControleCliente extends Controller{
8      private $cliente = [
9          ['id'=>1, 'nome'=>'João'],
10         ['id'=>2, 'nome'=>'Maria'],
11         ['id'=>3, 'nome'=>'José'],
12         ['id'=>4, 'nome'=>'Pedro'],
13     ];
14 }
```


Função index() e teste

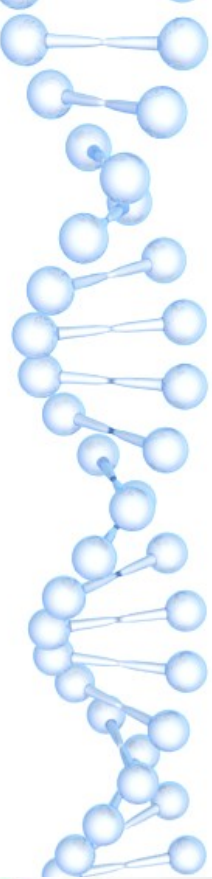
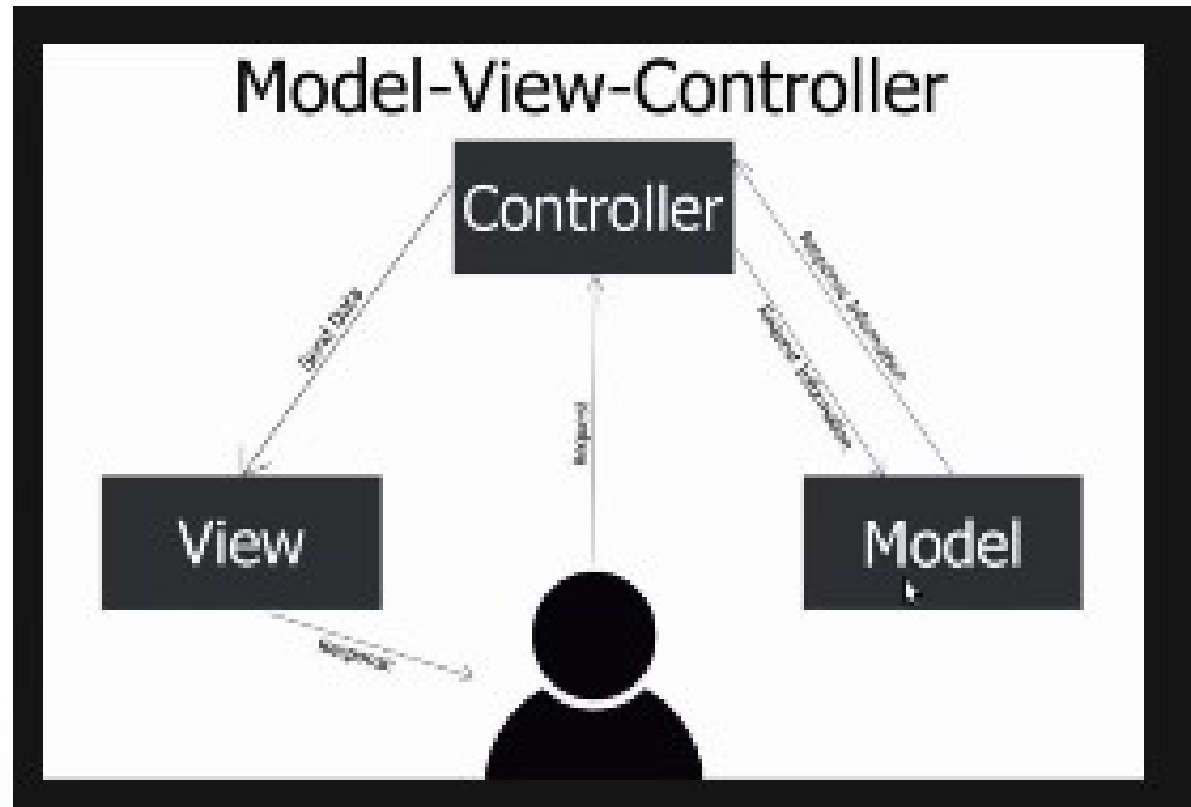
```
public function index(){  
    echo "<ol>";  
    foreach ($this->cliente as $c) {  
        echo "<li>" . $c['nome'] . "</li>";  
    }  
    echo "</ol>";  
}
```

← → ↻ 🏠 ⓘ 127.0.0.1:8000/clientes

🎨 Apps 🔄 Nova guia 🔴 Inversão da Or... ED AS M

1. João
2. Maria
3. José
4. Pedro

Conceito MVC



Passando os dados para uma view com a função compact

```
public function index(){  
    $clientes = $this->clientes;  
    return view('clientes.index', compact(['clientes']));  
}
```

Criando a view index, dentro da pasta cliente

```
resources > views > clientes > index.blade.php > ol  
1 <h3>Clientes</h3>  
2 <ol>  
3     @foreach ($clientes as $c)  
4         <li>{{ $c['nome'] }}</li>  
5     @endforeach  
6 </ol>
```

Teste

← → ↻ 🏠 ⓘ 127.0.0.1:8000/clientes
Apps Nova guia Inversão da Or... ED AS

Clientes

1. João
2. Maria
3. José
4. Pedro

Criar link para o edit apartir do ID do cliente

```
<h3>Clientes</h3>
<ol>
  @foreach ($clientes as $c)
    <li>
      {{ $c['nome'] }} |
      <a href="{{ route ('clientes.edit', $c['id'] ) }}">Editar</a>
    </li>
  @endforeach
</ol>
```

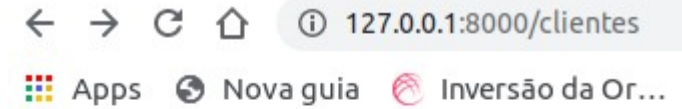
← → ↻ 🏠 ⓘ 127.0.0.1:8000/clientes
🎲 Apps 🔄 Nova guia 🌐 Inversão da Or...

Clientes

1. João | [Editar](#)
2. Maria | [Editar](#)
3. José | [Editar](#)
4. Pedro | [Editar](#)

Criar link para novo cliente

```
b.php  ControleCliente.php  index.blade.php X
res > views > clientes > index.blade.php > ol
<h3>Clientes</h3>
<a href="{{ route ('clientes.create') }}">NOVO CLIENTE</a>
<ol>
  @foreach ($clientes as $c)
    <li>
      {{ $c['nome'] }} |
      <a href="{{ route ('clientes.edit', $c['id'] ) }}">Editar</a>
    </li>
  @endforeach
</ol>
```



Clientes

NOVO CLIENTE

1. João | [Editar](#)
2. Maria | [Editar](#)
3. José | [Editar](#)
4. Pedro | [Editar](#)



Retorno do método index

Cientes:

Resultado

1. Ademir
2. Pedro
3. Paulo
4. Maria
5. João
6. José

View create

- Criaremos um formulário para adicionar um cliente
- Em views na pasta criada cliente criar uma view chamada create.blade.php

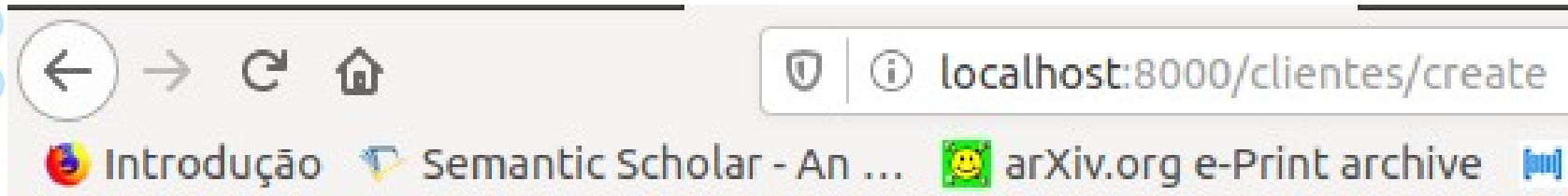
```
<h3>Novo Cliente</h3>
<form Action="{{ route('clientes.store') }}" method="POST">
    @csrf
    <input type="text" name="nome">
    <input type="submit" value="Salvar">
    <a href="{{ route ('clientes.index') }}">Voltar</a>
</form>
```


Método create

```
public function create()  
{  
    return view('cliente.create');  
}
```

View create

Resultado!!!!



Novo Cliente

Salvar

[Voltar](#)

Tratamento do POST

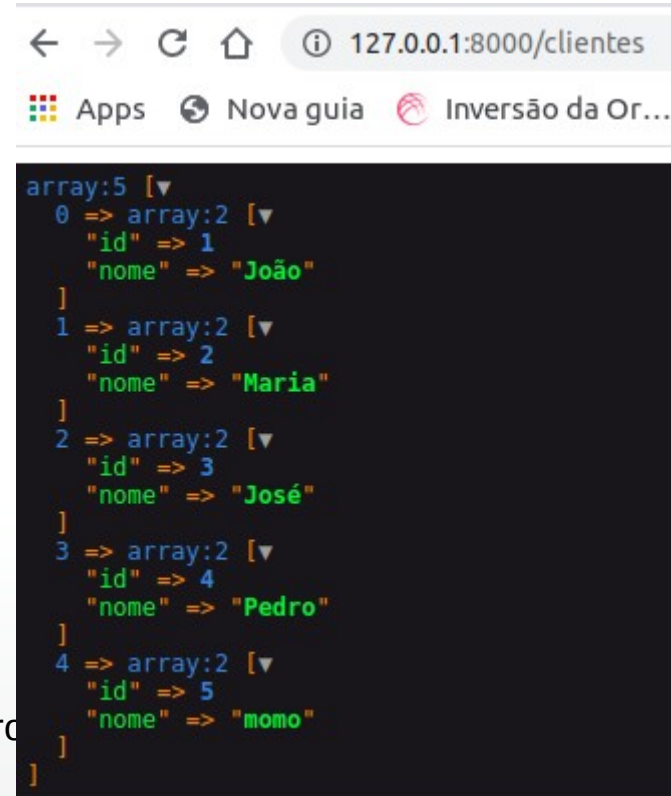
- O método Store vai fazer o tratamento das requisições de POST
-

Método Store

```
public function store(Request $request){  
  
    $id = count($this->clientes)+1;  
    $nome = $request->nome;  
    $dados = ["id"=>$id, "nome"=>$nome];  
    $this->clientes[] = $dados;  
    return redirect()->route('clientes.index');  
}
```

Método Store

- Fazer o teste utilizando o debbuger
 - `dd(this->clientes);`



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8000/clientes`. The browser's developer tools are open, showing a JSON array of five client objects. Each object has an `id` and a `nome` property.

```
array:5 [▼  
  0 => array:2 [▼  
    "id" => 1  
    "nome" => "João"  
  ]  
  1 => array:2 [▼  
    "id" => 2  
    "nome" => "Maria"  
  ]  
  2 => array:2 [▼  
    "id" => 3  
    "nome" => "José"  
  ]  
  3 => array:2 [▼  
    "id" => 4  
    "nome" => "Pedro"  
  ]  
  4 => array:2 [▼  
    "id" => 5  
    "nome" => "momo"  
  ]  
]
```

Criando sessão e gravando o array clientes

- Vamos gravar os dados em sessão
- Atualiza a exibição dos dados cada vez que realiza o cadastro
- Vamos criar um construtor no controller
- O construtor vai iniciar com os dados existentes

```
public function __construct(){  
    //Gravando os dados na primeira vez  
    $clientes = session('clientes');  
    if (!isset($clientes))  
        session(['clientes'=>$this->clientes]);  
}
```

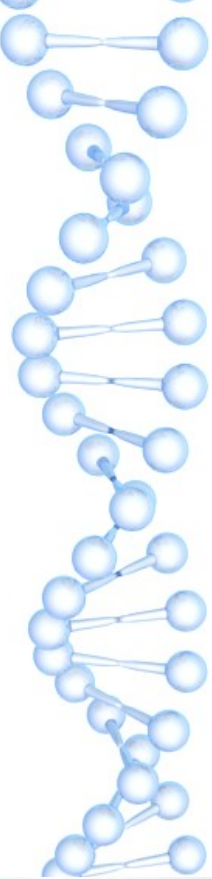
Usando a session no método index

```
public function index(){  
    //pegar os dados do cliente da sessão  
    $clientes = session('clientes');  
    return view('clientes.index', compact(['clientes']));  
}
```


Método store

- Não precisamos salvar o token
- Vamos salvar apenas o id e o nome na sessão
- Retornar para index

```
public function store(Request $request){  
    //pegando os dados do cliente de session  
    $clientes = session('clientes');  
    //o valor do id agora da variável $cliente da session  
    $id = count($clientes)+1;  
    //pegando o campo nome do form  
    $nome = $request->nome;  
    //criando o array  
    $dados = ["id"=>$id, "nome"=>$nome];  
    //atribuindo o array dados aos dados clientes (atualizando)  
    $clientes[] = $dados;  
    //atualizando session com os novos dados  
    session(['clientes'=>$clientes]);  
    //redirecionando para a view clientes.index  
    return redirect()->route('clientes.index');  
}
```



Método show

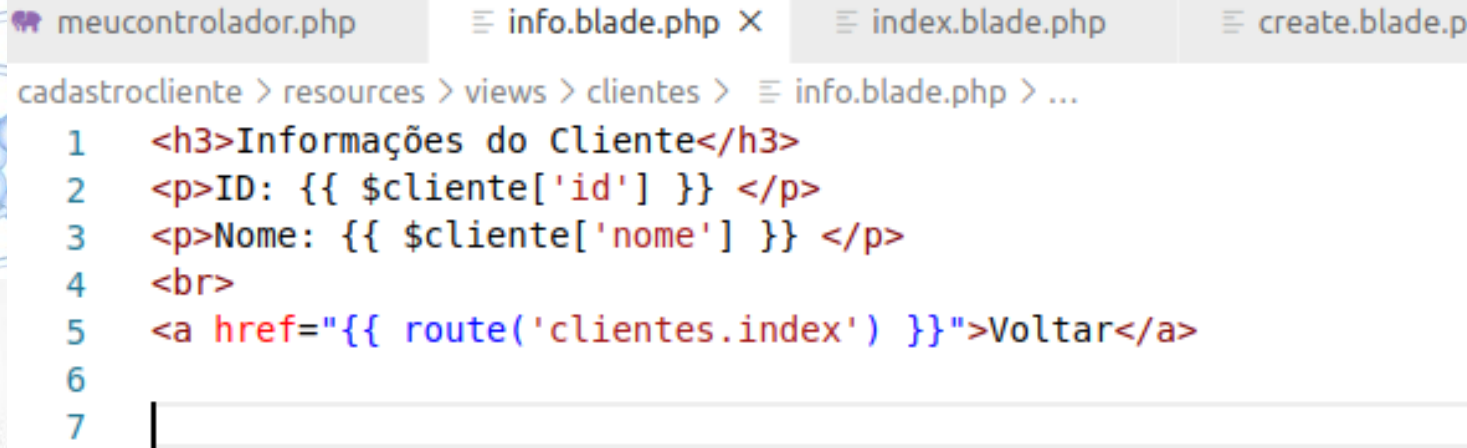
- O objetivo do método show é recuperar todos os dados de um cliente através do id e

```
public function show($id)
{
    //Recuperando todos os dados de clientes
    $clientes=session('clientes');
    $cliente = $clientes[$id-1];
    return view('clientes.info',compact(['cliente']));
}
```

Método show

- O método show vai retornar uma view
 - Podemos criá-la chamada de info.blade.php

Fazer o teste!!!!



The screenshot shows a web application interface with a code editor. At the top, there are tabs for 'meucontrolador.php', 'info.blade.php' (selected), 'index.blade.php', and 'create.blade.p'. Below the tabs, the breadcrumb path is 'cadastrocliente > resources > views > clientes > info.blade.php > ...'. The code editor displays the following HTML code:

```
1 <h3>Informações do Cliente</h3>
2 <p>ID: {{ $cliente['id'] }} </p>
3 <p>Nome: {{ $cliente['nome'] }} </p>
4 <br>
5 <a href="{{ route('clientes.index') }}">Voltar</a>
6
7 |
```

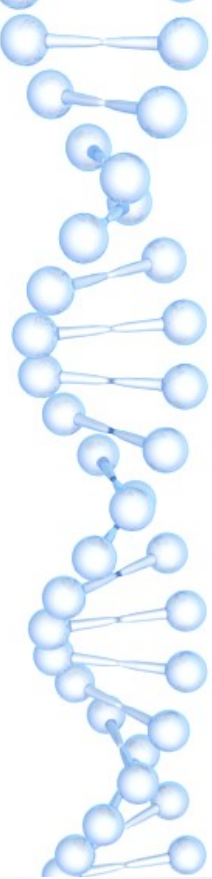
Fazendo o link para carregar a view de dados do cliente

cadastrocliente > resources > views > clientes > index.blade.php > ol > li > a

```
1 <h3>Clientes</h3>
2 <a href="{{ route ('clientes.create') }}">NOVO CLIENTE</a>
3 <ol>
4     @foreach ($clientes as $c)
5         <li>
6             {{ $c['nome'] }} |
7             <a href="{{ route ('clientes.edit', $c['id'] ) }}">Editar</a>
8             <a href="{{ route ('clientes.show', $c['id'] ) }}">Info</a>
9         </li>
10    @endforeach
11 </ol>
```

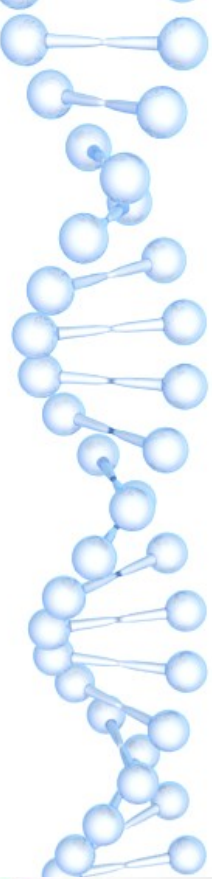
Métodos edit e update

- Método edit mostra um formulário para a edição dos dados
- Método update atualiza os dados de uma sessão ou no BD



Método edit

```
public function edit($id)
{
    //Recuperando todos os dados de clientes
    $clientes=session('clientes');
    $cliente = $clientes[$id-1];
    return view('clientes.edit',compact(['cliente']));
}
```



Criar a view para o método edit

```
cadastrocliente > resources > views > clientes > edit.blade.php > form
1  <h3>Atualizar Cliente</h3>
2
3  <form action="{{ route('clientes.update', $cliente['id']) }}" method="POST">
4      @csrf
5      @method('PUT')
6      <input type="text" name="nome" value="{{ $cliente['nome'] }}">
7      <input type="submit" value="Atualizar">
8  <a href="{{ route('clientes.index') }}">Voltar</a>
9  </form>
```

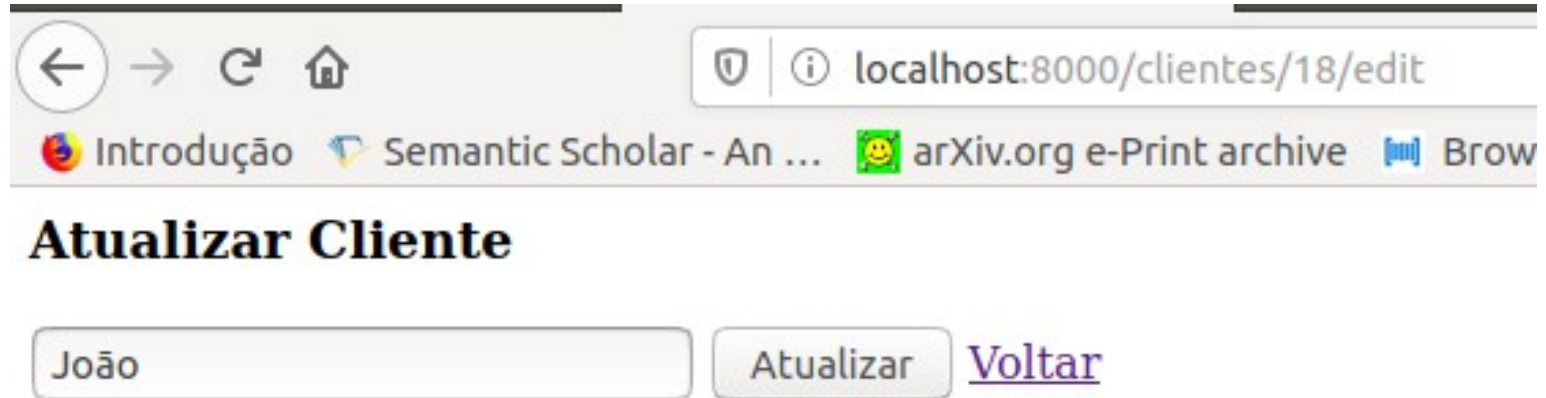
ANALISAR A ROTA DO UPDATE

Método Update

```
public function update(Request $request, $id)
{
    $clientes=session('clientes');
    $clientes[$id-1]['nome']=$request->nome;
    session(['clientes'=>$clientes]);
    return redirect()->route('clientes.index');
}
```

Atividade criar a view para o método edit

Resultado!!!!!!



Fazer testes analisar código

- Métodos HTTP PUT/DELETE no Laravel
- Value nome
- Recuperando o ID
- Chamando o método UPDATE
- Retornando para Index

Método destroy

Ação do botão apagar

- O objetivo do método destroy é enviar uma requisição ao controlador para apagar um registro
- Vamos criar um botão no index responsável para apagar um cadastro
- Vamos precisar enviar uma requisição de delete, portanto precisamos criar um formulário



Método destroy

Ação do botão apagar

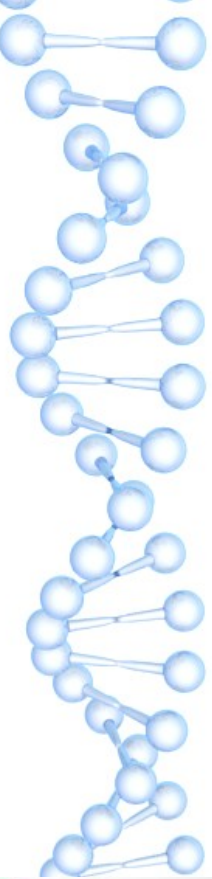
astrocliente > resources > views > clientes > index.blade.php > ol

```
1 <h3>Clientes</h3>
2 <a href="{{ route ('clientes.create') }}">NOVO CLIENTE</a>
3 <ol>
4     @foreach ($clientes as $c)
5         <li>
6             {{ $c['nome'] }} |
7             <a href="{{ route ('clientes.edit', $c['id'] ) }}">Editar</a>
8             <a href="{{ route ('clientes.show', $c['id'] ) }}">Info</a>
9             <form action="{{ route('clientes.destroy', $c['id']) }}" method="POST">
10                 @csrf
11                 @method('DELETE')
12                 <input type="submit" value="Apagar">
13                 <a href="{{ route('clientes.index') }}">Voltar</a>
14             </form>
15         </li>
16     @endforeach
17 </ol>
```



Método destroy

```
public function destroy($id) {  
    //recuperando o array de clientes  
    $clientes=session('clientes');  
    //obtendo a coluna do id  
    $ids = array_column($clientes, 'id');  
    //busca o id recebido  
    $index = array_search($id, $ids);  
    //apaga o id encontrado  
    array_splice($clientes, $index, 1);  
    //atualiza o array de clientes em session  
    session(['clientes' => $clientes]);  
    //redireciona para view clientes index  
    return redirect()->route('clientes.index');  
}
```



Dica

- Podemos alterar o marcado no HTML do Index para mostrar o ID
- Alterar o marcador para ul e adicionar o id do array cliente

cadastrocliente > resources > views > clientes > index.blade.php > ul

```
1 <h3>Clientes</h3>
2 <a href="{{ route ('clientes.create') }}">NOVO CLIENTE</a>
3 <ul>
4     @foreach ($clientes as $c)
5         <li>
6             {{ $c['id'] }} | {{ $c['nome'] }} |
7             <a href="{{ route ('clientes.edit', $c['id'] ) }}">Editar</a>
8             <a href="{{ route ('clientes.show', $c['id'] ) }}">Info</a>
9             <form action="{{ route('clientes.destroy', $c['id']) }}" method="POST">
10                 @csrf
11                 @method('DELETE')
12                 <input type="submit" value="Apagar">
13                 <a href="{{ route('clientes.index') }}">Voltar</a>
14             </form>
15         </li>
16     @endforeach
17 </ul>
```



Apps Nova gui

Clientes

NOVO CLIENTE

- 2 | Maria | [Editar](#) [Info](#)
 [Voltar](#)
- 3 | José | [Editar](#) [Info](#)
 [Voltar](#)

Correção problema do índices (id)

- O problema ocorre quando apagamos um cliente
- O índice perde a referência
 - `$clientes[$id-1]['nome']=$request - > nome;`
- Solução:
 - Criar uma função que retorna um cliente com o seu respectivo ID
 - Depois fazer as alterações em todos os lugares que usava a solução anterior

Criando a função GetIndex()

```
private function getIndex($id, $clientes){  
    //obtendo a coluna do id  
    $ids = array_column($clientes, 'id');  
    //busca o id recebido  
    $index = array_search($id, $ids);  
    //retorna o ID do cliente  
    return $index;  
}
```

Corrigindo a captura dos Ids dos Clientes

- Agora com a função `getIndex()`, podemos alterar os métodos:
 - Destroy
 - Update
 - Edit
 - Show

```
public function show($id){
    //Recuperando todos os dados de clientes
    $clientes=session('clientes');
    $index = $this->getIndex($id, $clientes);
    $cliente = $clientes[$index];
    return view('clientes.info',compact(['cliente'])):
}
```

```
public function update(Request $request, $id) {
    $clientes=session('clientes');
    //Utilizando a função getIndex() para capturar o id
    $index = $this->getIndex($id, $clientes);
    $clientes[$index]['nome']=$request->nome;
    session(['clientes'=>$clientes]);
    return redirect()->route('clientes.index');
}
```

```
public function edit($id){
    //Recuperando todos os dados de clientes
    $clientes=session('clientes');
    $index = $this->getIndex($id, $clientes);
    $cliente = $clientes[$index];
    return view('clientes.edit',compact(['cliente']));
}
```

```
public function destroy($id) {
    //recuperando o array de clientes
    $clientes=session('clientes');
    //utilizando a função getIndex passando os parâmetros
    $index = $this->getIndex($id, $clientes);
    //busca o id recebido
    array_splice($clientes, $index, 1);
    //atualiza o array de clientes em session
    session(['clientes' => $clientes]);
    //redireciona para view clientes index
    return redirect()->route('clientes.index');
}
```

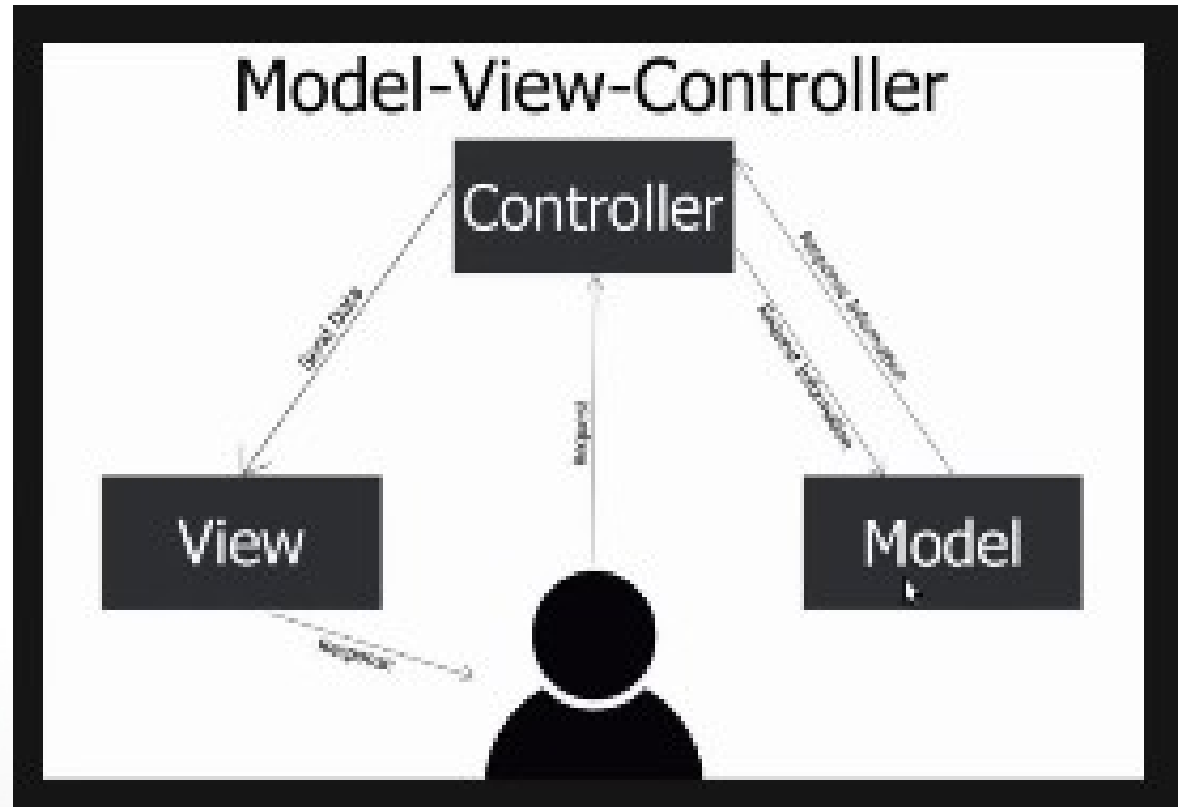
Em Store corrigindo o incremento de Id

- Na função Store devemos alterar o método do cálculo do id
- Utilizaremos a função end + 1

- \$id = end(\$clientes)['id']+1;

```
public function store(Request $request){  
    //pegando os dados do cliente de session  
    $clientes = session('clientes');  
    //calcular o id com a função end + 1  
    $id = end($clientes)['id']+1;  
    //pegando o campo nome do form  
    $nome = $request->nome;  
    //criando o array  
    $dados = ["id"=>$id, "nome"=>$nome];  
    //atribuindo o array dados aos dados clientes (atualizando)  
    $clientes[] = $dados;  
    //atualizando session com os novos dados  
    session(['clientes'=>$clientes]);  
    //redirecionando para a view clientes.index  
    return redirect()->route('clientes.index');  
}
```

Concluindo





Atividade extraclasse:

Enunciado: utilizando os métodos do Controller Resource (index, create, store, show, edit, update e destroy), implementar um cadastro de calunos com os seguintes dados: nome, idade, curso, e-mail.

Funcionalidades: Cadastrar, Editar, Apagar e Consultar alunos

Forma de entrega: o aluno deverá gravar um vídeo (máximo 5 minutos) apresentando as funcionalidades conforme solicitadas e as respectivas classes (controller, rota e views) e postar o link do vídeo no SIGAA.

Data de entrega: 22/05/2021 (sábado)

Referências

- Laravel. <https://laravel.com>
- Controllers. <https://laravel.com/docs/8.x/controllers>