

## LARAVEL Funções CRUD

Neste exemplo irão ser desenvolvidas as funções CRUD (**Create Read Update Delete**) para uma tabela simples. Esta tabela tem unicamente quatro campos (ID, description, created\_at e updated\_at). Com a exceção do campo 'description', todos os outros são criados automaticamente aquando da criação do modelo de dados (Model) e da migração (migration).

O projeto exemplo irá utilizar o projeto já criado anteriormente (exampleApp).

### Acrescentar ao projeto alguns pacotes necessários

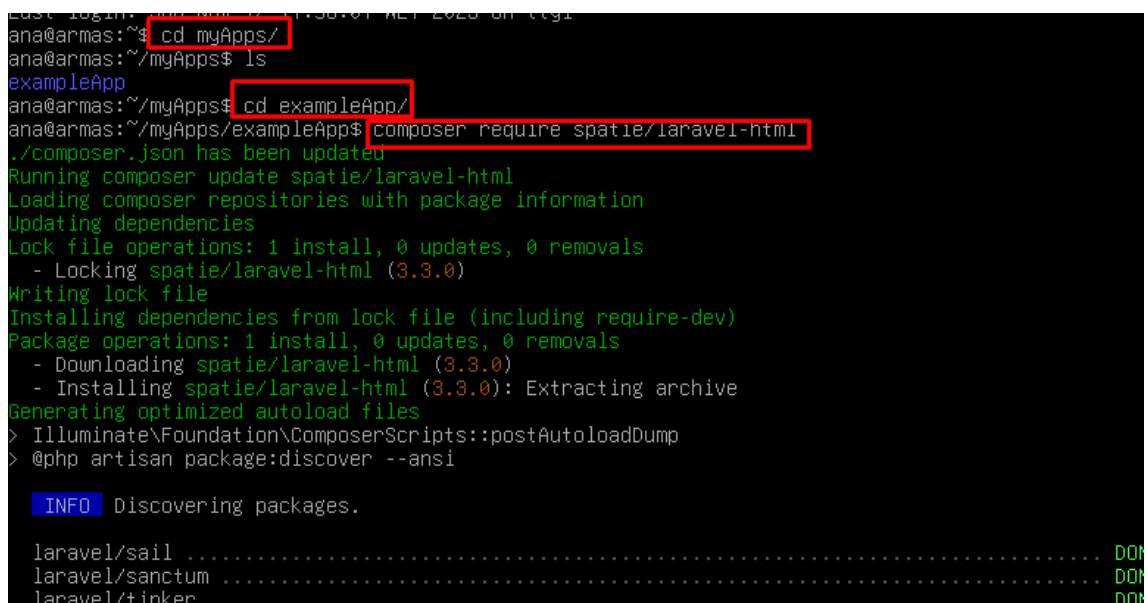
Vamos acrescentar alguns pacotes que irão ser necessários. Esses pacotes são os seguintes:

spatie/laravel-html – Encapsula HTML para blade

Para isso, fazer os seguintes passos:

Na consola, dentro da pasta "exampleApp" fazer:

```
composer require spatie/laravel-html
```



```
ana@armas:~$ cd myApps/
ana@armas:~/myApps$ ls
exampleApp
ana@armas:~/myApps$ cd exampleApp/
ana@armas:~/myApps/exampleApp$ composer require spatie/laravel-html
./composer.json has been updated
Running composer update spatie/laravel-html
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
- Locking spatie/laravel-html (3.3.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
- Downloading spatie/laravel-html (3.3.0)
- Installing spatie/laravel-html (3.3.0): Extracting archive
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi

 INFO  Discovering packages.

laravel/sail ..... DONE
laravel/sanctum ..... DONE
laravel/tinker ..... DONE
```

Acrescentar os *ServiceProvider* ao array de *providers* no ficheiro config/app.php

```
'providers' => [
    ...
    Spatie\Html\HtmlServiceProvider::class,
],
```

Neste momento, e se tudo correu bem, o ficheiro 'composer.json' tem este aspeto:

```
{
  "name": "laravel/laravel",
  "type": "project",
  "description": "The Laravel Framework.",
  "keywords": ["framework", "laravel"],
  "license": "MIT",
  "require": {
    "php": "^8.0.2",
    "guzzlehttp/guzzle": "^7.2",
    "laravel/framework": "^9.19",
    "laravel/sanctum": "^3.0",
    "laravel/tinker": "^2.7",
    "spatie/laravel-html": "^3.3"
  },
  "require-dev": {
    "fakerphp/faker": "^1.9.1",
    "laravel/pint": "^1.0",
    "laravel/sail": "^1.0.1",
    "mockery/mockery": "^1.4.4",
    "nunomaduro/collision": "^6.1",
    "phpunit/phpunit": "^9.5.10",
    "spatie/laravel-ignition": "^1.0"
  },
  "autoload": {
    "psr-4": {
      "App\\": "app/",
      "Database\\Factories\\": "database/factories/",
      "Database\\Seeders\\": "database/seeders/"
    }
  },
  "autoload-dev": {
    "psr-4": {
      "Tests\\": "tests/"
    }
  }
}
```

## Criação do modelo de dados e migração

Vamos criar o modelo de dados já com a respetiva *migration*. Para isso fazemos o seguinte (a o modelo de dados vai se chamar 'Article'):

```
php artisan make:model Article -m
```

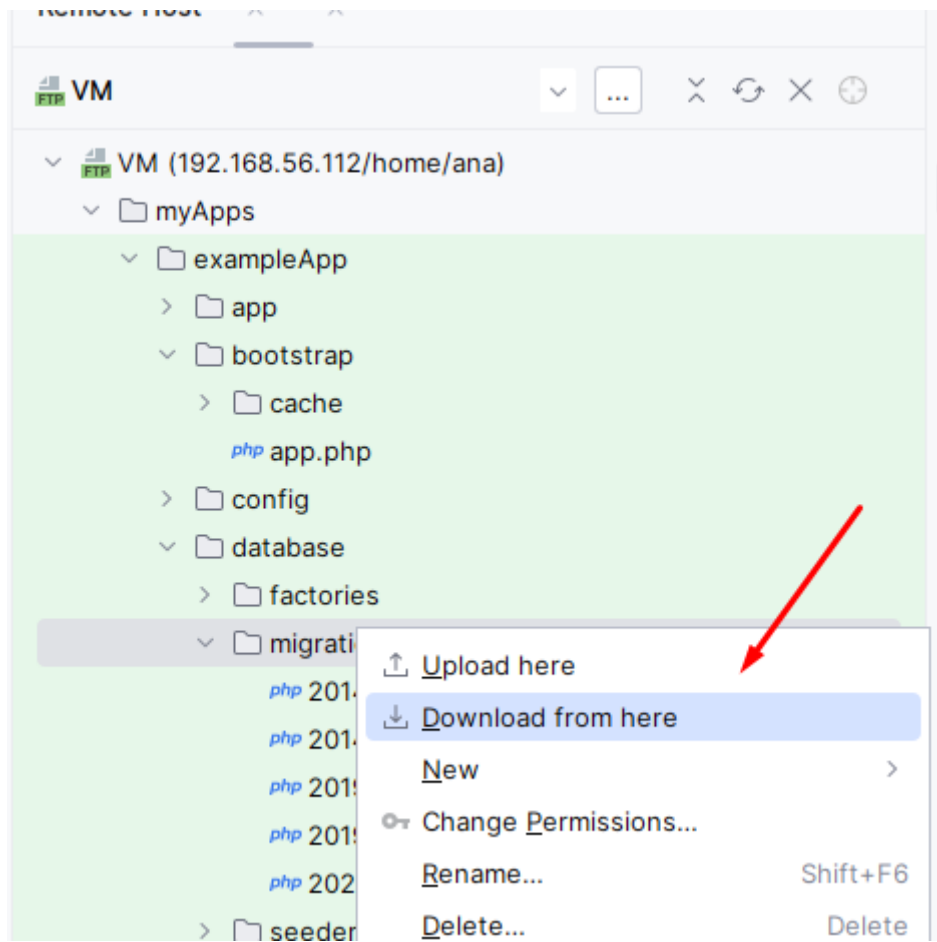
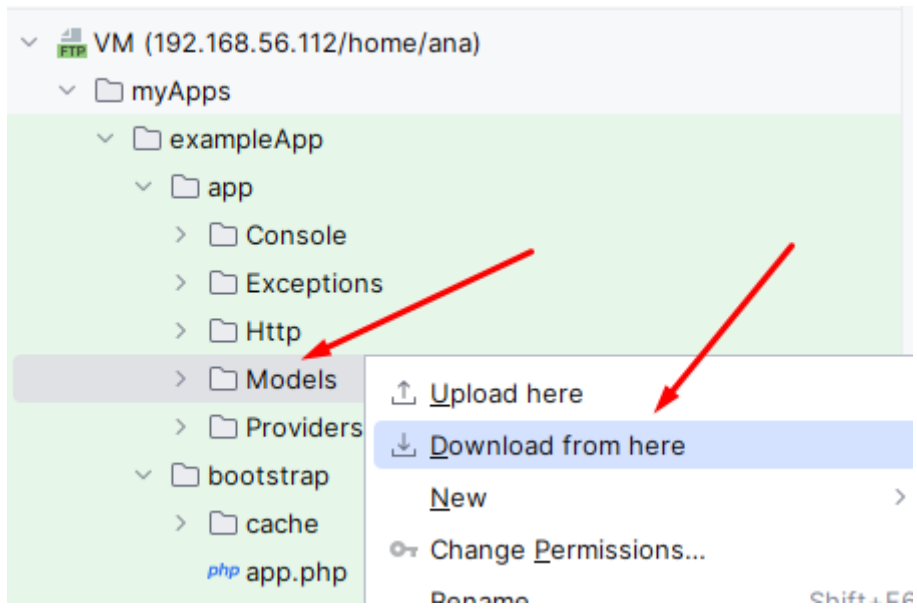
O parâmetro "-m" implica a criação da 'migration'.

A execução deste comando determinou a criação de dois novos ficheiros:

'app/Models/Article.php' – o model

database/migrations/yyyy\_mm\_dd\_hhmmss\_create\_articles\_table.php' – a migration

Podemos, nesta altura, fazer o download, para a pasta local, das migrações e dos models



### Alteração da 'migration'

Editar o ficheiro 'yyyy\_mm\_dd\_hhmmss\_create\_articles\_table.php' e acrescentar os campos desejados. Neste caso, o campo 'description'. Deve ficar assim:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('articles', function (Blueprint $table) {
            $table->id();
            $table->string('description');
            $table->timestamps();
        });
    }
}
```

Neste momento, o Model Article.php não tem a indicação dos campos possíveis de serem preenchidos.

Editar agora o ficheiro 'app/Models/Article.php'. Deve ficar assim:

```
<?php

namespace App\Models;

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

class Article extends Model
{
    use HasFactory;

    //protected $table = 'articles';
    protected $fillable=[
        'description',
        'created_at',
        'updated_at',
    ];
}
```

## Criação da base de dados

Para que possamos manipular dados é necessário criar previamente uma base de dados MySQL. Para isso podemos utilizar a ferramenta phpmyadmin (instalada por 'default' aquando da instalação do wampserver) ou utilizando comandos de consola. A base de dados vai-se chamar 'crud'.

### Utilizando a consola

```
ana@armas:~/public_html/CRUD$ mysql --user=root --password=potter
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 9
Server version: 10.11.4-MariaDB-1~deb12u1 Debian 12

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE crud;
Query OK, 1 row affected (0.003 sec)

MariaDB [(none)]> exit
Bye
ana@armas:~/public_html/CRUD$
```

### Alteração do ficheiro '.env'

O ficheiro '.env' é onde se colocam todas as variáveis de ambiente, incluindo a definição da base de dados (nome, user, password,...)

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=crud
DB_USERNAME=root
DB_PASSWORD=potter
```

deveríamos criar um user para esta APP  
NUNCA usar "root"

## Configuração do ficheiro 'config/database.php'

```
'mysql' => [
    'driver' => 'mysql',
    'url' => env( key: 'DATABASE_URL'),
    'host' => env( key: 'DB_HOST', default: '127.0.0.1'),
    'port' => env( key: 'DB_PORT', default: '3306'),
    'database' => env( key: 'DB_DATABASE', default: 'forge'),
    'username' => env( key: 'DB_USERNAME', default: 'forge'),
    'password' => env( key: 'DB_PASSWORD', default: ''),
    'unix_socket' => env( key: 'DB_SOCKET', default: ''),
    'charset' => 'utf8mb4',
    'collation' => 'utf8mb4_unicode_ci',
    'prefix' => '',
    'prefix_indexes' => true,
    'strict' => true,
    'engine' => 'InnoDB',
    'options' => extension_loaded( extension: 'pdo_mysql') ? array_filter([
        PDO::MYSQL_ATTR_SSL_CA => env( key: 'MYSQL_ATTR_SSL_CA'),
    ]) : [],
],
```

## Correr as *migrations* para a criação de todas as tabelas

```
php artisan migrate
```

```
INFO  Preparing database.

Creating migration table ..... 51ms DONE

INFO  Running migrations.

2014_10_12_000000_create_users_table ..... 82ms DONE
2014_10_12_100000_create_password_resets_table ..... 308ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 74ms DONE
2019_12_14_000001_create_personal_access_tokens_table ... 140ms DONE
2023_11_19_165119_create_articles_table ..... 43ms DONE
```

Se for necessário alterar a estrutura de alguma tabela, antes é necessário fazer o *rollback* da *migration*. Para isso utiliza-se o comando 'php artisan migrate:rollback', obtendo-se o resultado mostrado na figura seguinte.

```
php artisan migrate:rollback
```

```
INFO  Rolling back migrations.

2023_11_19_165119_create_articles_table ..... 38ms DONE
2019_12_14_000001_create_personal_access_tokens_table ... 31ms DONE
```

```
2019_08_19_000000_create_failed_jobs_table ..... 39ms DONE
2014_10_12_100000_create_password_resets_table ..... 29ms DONE
2014_10_12_000000_create_users_table ..... 27ms DONE
```

## Criação do Controller

Vamos agora criar o *Controller* que permitirá a manipulação dos dados. Neste caso, vamos-lhe chamar 'articleController'. Como queremos que *controller* tenha os métodos necessários para a manipulação dos dados, teremos que utilizar o parâmetro '--resource'. Assim, na consola, fazer:

```
php artisan make:controller articleController --resource
```

Após o comando executar é criado um ficheiro, já com os métodos necessários, chamado 'app/Http/Controllers/articleController.php'

## Atualização das rotas

As rotas disponíveis estão no ficheiro 'routes/web.php'. Alterar para:

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\articleController;

/*
|-----
| Web Routes
|-----
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get( uri: '/', function () {
    return view( view: 'welcome');
});

Route::resource( name: 'article', controller: articleController::class);
```

Assim, foram criadas todas as rotas para a manipulação dos dados da tabela 'articles'.

A primeira rota, é a rota que vem por *default* após a criação de um novo projeto. Poderá, obviamente, ser alterada posteriormente.

Através do comando seguinte, poder-se-á ver as rotas disponíveis na aplicação:

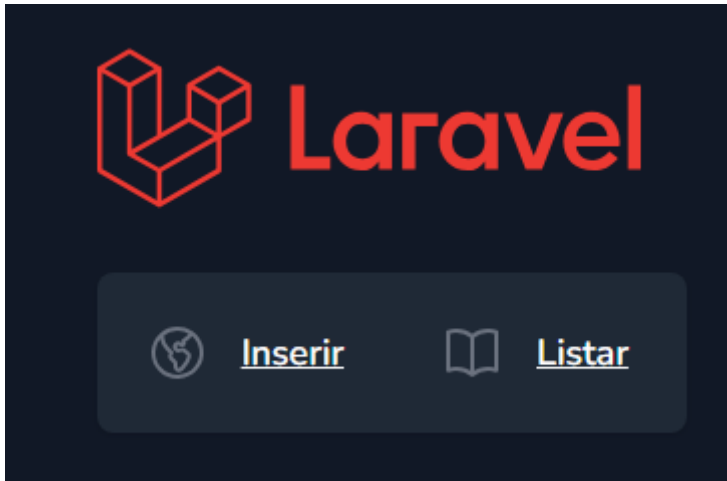
```
php artisan route:list
```

## Criação da View 'welcome'

Todas as views estão localizadas na pasta 'resources/views'.

Os nomes das views cumprem com a seguinte forma: **nome\_da\_view.blade.php**. Assim, o ficheiro da view 'welcome' terá o nome **welcome.blade.php**

Neste momento, e a título de exemplo, vamos alterar a view 'welcome' de modo a ficar com o aspeto seguinte:



```
<div class="p-6">
  <div class="flex items-center">
    <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24" stroke-width="1.5" str
    <div class="ml-4 text-lg leading-7 font-semibold"><a href="article/create" class="underline te
  </div>
</div>
<div class="p-6">
  <div class="flex items-center">
    <svg xmlns="http://www.w3.org/2000/svg" fill="none" viewBox="0 0 24 24" stroke-width="1.5" str
    <div class="ml-4 text-lg leading-7 font-semibold"><a href="article/list" class="underline text
  </div>
```

## Desenvolvimento do método "create"

No *controller* 'articleController', alterar o método create

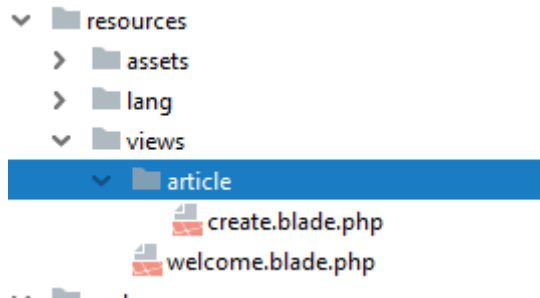
```
public function create()
{
    return view( view: 'article.create' );
}
```

O método 'create' vai retornar a view 'create' que está dentro da pasta 'article'. Podemos usar '.' ou '/' para separador de caminhos.



## Desenvolvimento do formulário para o método 'create'

Depois de criar o método 'create', é necessário proceder com o desenvolvimento da view associada a esse método. Para isso, navegue até o diretório 'resources/view', crie uma nova pasta chamada 'article' e, em seguida, crie o arquivo da view denominado 'create.blade.php'.



O seu conteúdo é o seguinte:

```
<!-- Bootstrap -->
<!--
Ver mais em:https://getbootstrap.com/docs/4.3/getting-started/download/ -->
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy60rQ6VrjIEaFf/njGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
<!-- Fonts fontawesome-->
<!-- Ver mais em:https://fontawesome.bootstrapcdn.com/ -->
<link href="http://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
rel="stylesheet">
<!-- Formulário usando componentes bootstrap e o pacote laravelcollective-->
<div class="jumbotron">
    {{ html()->form('POST', url('/article'))->open() }}
    <div class="panel panel-primary">
        <div class="panel-body">
            <div class="form-group" style="text-align: left;">
                {{ html()->label(descrição: ', 'description') }}
                {{ html()->text('description') }}
            </div>
            <div class="form-group">
                {{ html()->submit('OK')->class('btn btn-success frm-control') }}
                <a href="{{url('/') }}" class="btn btn-primary"
role="button">Cancelar</a>
            </div>
        </div>
    </div>
    {{ html()->form()->close() }}
</div>

<!-- Fim do Formulário -->
<!-- Bootstrap js -->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
integrity="sha384-UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1cllHTMGA3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
```

description:

OK

Cancelar

Ao clicar em "OK", acionaremos o método 'store' no controlador 'articleController', enquanto ao clicar em "Cancelar", seremos redirecionados para a página anterior.

## Desenvolvimento do método 'store'

```
public function store(Request $request)
{
    /* Atenção:
     * Para utilizar, não esquecer de colocar, em cima,
     * use Carbon\Carbon
     */
    $now=Carbon::now()->toDateTimeString();
    /*
     * verificar se está, em cima, use App\Models\Article;
     */
    $input= new Article(array(
        'description'    => $request->description,
        'created_at'     => $now,
        'updated_at'     => $now,
    ));
    $input->save();
    return redirect(url('/'));
}
```

## Desenvolvimento da função 'Listar'

A operação de listagem não está incluída nas funções fundamentais de CRUD. Portanto, é necessário adicionar manualmente o método ao controlador, assim como à rota. Vale ressaltar que a ordem das rotas escritas impacta o seu funcionamento.

Assim, o ficheiro 'routes\web.php' deve ficar assim:

```
Route::get('/', function () {
    return view('welcome');
});

Route::get('article/list', [articleController::class, 'list']);
Route::resource('article', articleController::class);
```

Vamos agora desenvolver o método 'list' no *Controller* 'articleController'.

O método deve ficar (+/-) assim:

```
public function list() {
    $rs=Article::select('articles.*')
        ->orderBy('articles.id','asc')
        ->get();
    return view('article._list')->with(compact('rs'));
}
```

Agora, só falta criar a *View*. Esta *View* vai ser criada na diretoria 'resources\views\article' e vai-se chamar `_list.blade.php`.

```
<!-- Bootstrap -->
<!--
Ver mais em:
https://getbootstrap.com/docs/4.3/getting-started/download/
-->
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap
.min.css" integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.m
in.js" integrity="sha384-
JjSmVggyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDs4x0xIM+B07jRM"
crossorigin="anonymous"></script>
<!-- Fonts fontawesome-->
<!--
Ver mais em:
https://fontawesome.bootstrapcdn.com/
-->
<link href="http://maxcdn.bootstrapcdn.com/font-
awesome/4.7.0/css/font-awesome.min.css" rel="stylesheet">
<!-------
----->
<!-- LISTAGEM -->
<div class="jumbotron">
    <table class="table table-striped">
        <thead>
            <tr>
                <th>#</th>
                <th>Descrição</th>
            </tr>
        </thead>
        <tbody>
            @foreach($rs as $r)
                <tr>
                    <th scope="row">{{ $r->id }}</th>
                    <td>{{ $r->description }}</td>
                    <td style="vertical-align:middle;">
                        <a href="{{url('/') }}/article/{{ $r->id }}/edit"
class="btn btn-xs btn-primary" role="button">
                            <i class="fa fa-pencil-square-o"></i>
                        </a>
                    </td>
                    <td>
                        {{ html()->form('DELETE', url('/article').'/'.$r-
>id)->id('deleteform')->open() }}
                        <button type="submit" class="btn btn-xs btn-
```

```

danger" id="deletebutton">
    <i class="fa fa-trash-o"></i>
    </button>
    {{ html()->form()->close() }}
</td>
</tr>
</tbody>
</table>
<!-- Botao de voltar-->
<div class="form-group">
    <a href="{!!URL::to('/') !!}" class="btn btn-primary"
role="button">Voltar</a>
</div>
</div>
<!-- FIM DA LISTAGEM -->
<!-- Bootstrap js -->

```

## Desenvolvimento do método “edit”

```

public function edit($id)
{
    $rs=Article::findOrFail($id);
    return view('article.edit')->with(compact('rs'));
}

```

## Desenvolvimento do formulário para o método ‘edit’

```

...
<div class="jumbotron">
    {{ html()->modelForm($rs,'PATCH', url('/article')..'/'.$rs->id)->open() }}
    <div class="panel panel-primary">
        <div class="panel-body">
            <div class="form-group" style="text-align: left;">
                {{ html()->label('id: ','id') }}
                {{ html()->text('id')->isReadonly() }}
            </div>
            <div class="form-group" style="text-align: left;">
                {{ html()->label('descrição: ','description') }}
                {{ html()->text('description') }}
            </div>
            <div class="form-group">
                {{ html()->submit('OK')->class('btn btn-success frm-control') }}
                <a href="{{url('/') }}" class="btn btn-primary"
role="button">Cancelar</a>
            </div>
        </div>
    </div>
    {{ html()->closeFormModel() }}
</div>
...

```

## Desenvolvimento do método 'update'

```
public function update(Request $request, $id)
{
    $now=Carbon::now()->toDateTimeString();
    $fiels_to_update=array(
        'description' => $request->description,
        'updated_at'   => $now,
    );
    Article::where('id',$id)
        ->update($fiels_to_update);
    return redirect (url('/article/list'));
}
```

## Desenvolvimento do método 'destroy'

```
public function destroy($id)
{
    Article::where('id',$id)->delete();
    return redirect (url('/article/list'));
}
```

## Otimização das Views | layouts

Observando o código desenvolvido anteriormente, percebemos a presença de repetições em algumas partes. Dessa forma, é possível realizar otimizações. Para isso, vamos criar um modelo de visualização (View) para todas as páginas. Essa View será denominada 'app.blade.php' e será armazenada no diretório 'resources/views/layouts', que precisamos criar antecipadamente.

Esta View será escrita tendo como base a View 'welcome' com a adição de Bootstrap.

A View app.blade.php ficaria assim (extrato):

```
...
<div class="relative flex items-top justify-center min-h-screen bg-
gray-100 dark:bg-gray-900 sm:items-center py-4 sm:pt-0">
    @if (Route::has('login'))
        <div class="hidden fixed top-0 right-0 px-6 py-4 sm:block">
            @auth
                <a href="{{ url('/home') }}" class="text-sm text-gray-
700 dark:text-gray-500 underline">Home</a>
            @else
                <a href="{{ route('login') }}" class="text-sm text-
gray-700 dark:text-gray-500 underline">Log in</a>

                @if (Route::has('register'))
                    <a href="{{ route('register') }}" class="ml-4
text-sm text-gray-700 dark:text-gray-500 underline">Register</a>
                @endif
            @endauth
        </div>
    @endif

    @yield('content')

</div>
...
```

A View create.bade.php, por exemplo, ficaria assim:

```
@extends('layouts.app')
@section('content')
<div class="jumbotron">
    {{ html()->form('POST', url('/article'))->open() }}
    <div class="panel panel-primary">
        <div class="panel-body">
            <div class="form-group" style="text-align: left;">
                {{ html()->label('description: ', 'description') }}
                {{ html()->text('description') }}
            </div>
            <div class="form-group">
                {{ html()->submit('OK')->class('btn btn-success frm-control') }}
                <a href="{{ url('/') }}" class="btn btn-primary"
role="button">Cancelar</a>
            </div>
        </div>
    </div>
    {{ html()->form()->close() }}
</div>
@endsection
```

Vamos agora criar uma *View* com o nome `list.blade.php`

```
@extends('layouts.app')
@section('content')
    <div class="jumbotron">
        @include('article._list')
        <!-- Botao de voltar-->
        <div class="form-group">
            <a href="{{URL::to('/') }}" class="btn btn-primary"
role="button">Voltar</a>
        </div>
    </div>
@endsection
```

No `articleController`, no método `list()` devemos alterar

```
return view('article._list')->with(compact('rs'));
```

para

```
return view('article.list')->with(compact('rs'));
```

## Otimização das *Views* | formulários

Podemos observar no código implementado que os formulários de criação (`create`) e de edição (`edit`) são praticamente idênticos. Para otimizar nosso trabalho, podemos fazer o seguinte:

1. Criar uma `blade` (`_frm.blade.php`) com o código igual
2. Utilizá-lo, com recurso à diretiva `@include` nas *blades* “`create`” e “`edit`”.

Assim, a `blade` `_frm.blade.php`

```
<div class="panel panel-primary">
    <div class="panel-body">
        <div class="form-group" style="text-align: left;">
            {{ html()->label('description: ', 'description') }}
            {{ html()->text('description') }}
        </div>
        <div class="form-group">
            {{ html()->submit('OK')->class('btn btn-success frm-
control') }}
            <a href="{{url('/') }}" class="btn btn-primary"
role="button">Cancelar</a>
        </div>
    </div>
</div>
```

A blade “create” ficaria assim:

```
@extends('layouts.app')
@section('content')
<div class="jumbotron">
    {{ html()->form('POST', url('/article'))->open() }}
    @include('article._frm')
    {{ html()->form()->close() }}
</div>
@endsection
```

E a blade “edit”, assim:

```
@extends('layouts.app')
@section('content')
<div class="jumbotron">
    {{ html()->modelForm($rs, 'PATCH', url('/article')..'/'.$rs->id)->open() }}
    <div class="form-group" style="text-align: left;">
        {{ html()->label('id: ', 'id') }}
        {{ html()->text('id')->isReadonly() }}
    </div>
    @include('article._frm')
    {{ html()->closeModelForm() }}
</div>
@endsection
```

## Validação de formulários

Nos métodos *store* e *update*, deveremos acrescentar as linhas de validação dos campos.

```
$validateData=$request->validate([
    'description' => 'required | min:5'
], [
    'description.required' => 'A descrição é obrigatória',
    'description.min' => 'A descrição tem de ter, no mínimo, 5 caracteres',
]);
```

Para que as mensagens de erro de validação apareçam, teremos de criar uma blade (\_error.blade.php) com o seguinte conteúdo:

```
@if($errors->any())
    <ul class="alert alert-danger">
        @foreach($errors->all() as $e)
            <li>{{ $e }}</li>
        @endforeach
    </ul>
@endif
```



Iremos incluir esta blade nas blade \_frm.blade.php, ficando assim:

```
<div class="panel panel-primary">
  <div class="panel-body">
    <div class="form-group" style="text-align: left;">
      {{ html()->label('descrição: ', 'description') }}
      {{ html()->text('description') }}
    </div>
    <div class="form-group">
      {{ html()->submit('OK')->class('btn btn-success frm-
control') }}
      <a href="{{url('/') }}" class="btn btn-primary"
role="button">Cancelar</a>
    </div>
  </div>
</div>
@include('article._error')
```

Podem ver todas as regras de validação disponíveis aqui:

<https://laravel.com/docs/9.x/validation#available-validation-rules>