



# Criando projeto com laravel 10

## Criando projeto

```
composer create-project laravel/laravel crud-com-laravel-10
```

## Alterando o banco de dados

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

## Criando a primeira rota nomeada

```
<?php

use Illuminate\Support\Facades\Route;

Route::get('/', [HomeController::class, 'index'])->name('home');
```

## Startando o Servidor PHP Laravel pelo Cmd

```
php artisan serve
```

## Criando a HomeController via Cmd

```
php artisan make:controller HomeController
```

## Ajustando a HomeController para direcionar a view()

```

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class HomeController extends Controller
{
    public function index(){
        return view(view:'home');
    }
}

```

## Criando a view em resources\views

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Crud Com Laravel 10</title>
</head>
<body>
    <div class="container">
        @yield('content')
    </div>
</body>
</html>

```

```

@extends('master')

@section('content')

<h2>Home</h2>

@endsection

```

## Dicas sobre a master

A view `master.blade.php` no Laravel é um arquivo de template mestre que define a estrutura básica e o layout comum para todas as outras views do seu aplicativo. Ele é usado para criar uma estrutura consistente em todas as páginas do seu aplicativo web.

Ao criar uma view `master.blade.php`, você define o esqueleto básico do seu aplicativo, incluindo elementos como cabeçalho, rodapé, barra de navegação e

outros elementos de layout comuns. Essa view pode conter seções vazias ou placeholders onde o conteúdo específico de cada página será inserido.

Por exemplo, você pode ter um arquivo `master.blade.php` que contenha a estrutura HTML geral do seu aplicativo:

```
<!DOCTYPE html>
<html>
<head>
    <title>Meu Aplicativo</title>
    <!-- Incluir os estilos CSS, scripts JavaScript e outras dependências comuns -->
</head>
<body>
    <!-- Incluir uma barra de navegação comum a todas as páginas -->
    <nav>
        <!-- Links de navegação -->
    </nav>

    <!-- Incluir o conteúdo específico de cada página -->
    <div class="content">
        @yield('content')
    </div>

    <!-- Incluir um rodapé comum a todas as páginas -->
    <footer>
        <!-- Conteúdo do rodapé -->
    </footer>

    <!-- Incluir scripts JavaScript comuns -->
</body>
</html>
```

Observe o uso de `@yield('content')` no arquivo `master.blade.php`. Esse é um espaço reservado onde o conteúdo específico de cada página será inserido. Ao criar outras views, você pode estender a view `master.blade.php` e preencher essa seção com o conteúdo desejado.

Por exemplo, suponha que você tenha uma view `home.blade.php` para a página inicial do seu aplicativo. Para estender a `master.blade.php`, você pode fazer o seguinte:

```
@extends('master')

@section('content')
    <h1>Bem-vindo à minha página inicial</h1>
    <p>Conteúdo da página inicial...</p>
@endsection
```


Dessa forma, a seção `@yield('content')` na view `master.blade.php` será substituída pelo conteúdo definido na `home.blade.php`. Isso permite que você mantenha a estrutura geral do aplicativo em uma view mestre e defina o conteúdo específico de cada página em views separadas.

Em resumo, a view `master.blade.php` no Laravel é um arquivo de template mestre que define a estrutura e o layout comum para todas as outras views do seu aplicativo, proporcionando consistência e facilitando a manutenção do código.

## Rotas Nomeadas ligadas ao CRUD

### Laravel - The PHP Framework For Web Artisans

Laravel is a PHP web application framework with expressive, elegant syntax. We've already laid the foundation — freeing you to create without sweating the small things.

 <https://laravel.com/docs/10.x/controllers#actions-handled-by-resource-controller>

```
<?php
Route::get('/users/{user}', function (User $user) {
    return $user;
});

Route::post('/users', function (CreateUserRequest $request) {
    $user = User::create($request->validated());
    Mail::to($user->email)->send(new WelcomeMessage);
    return $user;
});
```

```
//Rotas criadas com base na documentação
//https://laravel.com/docs/10.x/controllers#actions-handled-by-resource-controller

Route::get('/', [HomeController::class, 'index'])->name('home');
Route::get('/users', [UserController::class, 'index'])->name('users.index');
Route::get('/users/create', [UserController::class, 'create'])->name('users.create');
Route::post('/users', [UserController::class, 'store'])->name('users.store');
Route::get('/users/{user}', [UserController::class, 'show'])->name('users.show');
Route::get('/users/{user}/edit', [UserController::class, 'edit'])->name('users.edit');
Route::patch('/users/{user}', [UserController::class, 'update'])->name('users.update');
Route::delete('/users/{user}', [UserController::class, 'destroy'])->name('users.destroy');
```

## Criando a Controller dos usuários

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

Esse comando gera a controller de usuário com todos os metodos para o crud que já foram roteados no arquivo web.php

## Criando a instância da model do usuário na controller

```
<?php

namespace App\Http\Controllers;
```

```

use App\Models\User;
use Illuminate\Http\Request;

class UserController extends Controller
{
    public readonly User $user;

    public function __constructor(){
        $this->user = new User();
    }

    public function index()
    {
        $users = $this->user->all();
        return view('users', ['users' => $users]);
    }
}
?>

```

## Criando a Controller dos usuários

Para criar a controller dos usuários com todos os métodos para CRUD, execute o seguinte comando:

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

Esse comando irá gerar a controller `UserController` com todos os métodos necessários para realizar as operações de criação, leitura, atualização e exclusão de usuários.

## Criando a instância da model do usuário na controller

Na controller `UserController`, é necessário criar uma instância da model `User` para realizar operações no banco de dados. Adicione o seguinte código ao início da classe:

```

<?php

namespace App\Http\Controllers;

use App\Models\User;
use Illuminate\Http\Request;

class UserController extends Controller
{
    public readonly User $user;

    public function __construct()

```

```

{
    $this->user = new User();
}

// Restante dos métodos da controller...
}

```

Isso criará uma propriedade protegida `$user` na controller, que será uma instância da model `User`. Essa instância será usada para executar consultas e modificações no banco de dados relacionadas aos usuários.

## Método `index`

O método `index` é responsável por exibir a listagem de usuários. Ele retorna uma view chamada `'users'` e passa a variável `$users` para a view, contendo todos os usuários do banco de dados. O código do método `index` é o seguinte:

```

public function index()
{
    $users = $this->user->all();
    return view('users', ['users' => $users]);
}

```

## Método `create`

O método `create` é responsável por exibir o formulário de criação de um novo usuário. Ele retorna uma view chamada `'user_create'`. O código do método `create` é o seguinte:

```

public function create()
{
    return view('user_create');
}

```

## Método `store`

O método `store` é responsável por armazenar um novo usuário no banco de dados. Ele recebe um objeto `Request` contendo os dados do novo usuário e realiza a criação no banco de dados. Em seguida, redireciona de volta à página anterior com uma mensagem de sucesso ou erro. O código do método `store` é o seguinte:

```

public function store(Request $request)
{

```

```

    $created = $this->user->create([
        'image' => '',
        'firstname' => $request->input('firstname'),
        'lastname' => $request->input('lastname'),
        'email' => $request->input('email'),
        'password' => password_hash($request->input('password'), PASSWORD_DEFAULT),
    ]);

    if ($created) {
        return redirect()->back()->with('message', 'Successfully Created');
    }
    return redirect()->back()->with('message', 'Error Created');
}

```

## Método `show`

O método `show` é responsável por exibir os detalhes de um usuário específico. Ele recebe o ID do usuário como parâmetro e pode ser implementado conforme necessário. O código do método `show` é o seguinte:

```

public function show(string $id)
{
    // Implementação dos detalhes do usuário
}

```

## Método `edit`

O método `edit` é responsável por exibir o formulário de edição de um usuário específico. Ele recebe um objeto `User`

contendo os dados do usuário a ser editado e retorna uma view chamada `'user_edit'`, passando o usuário como parâmetro. O código do método `edit` é o seguinte:

```

public function edit(User $user)
{
    return view('user_edit', ['user' => $user]);
}

```

## Método `update`

O método `update` é responsável por atualizar um usuário no banco de dados. Ele recebe um objeto `Request` contendo os dados atualizados e o ID do usuário a ser atualizado. Em seguida, realiza a atualização no banco de dados e redireciona de

volta à página anterior com uma mensagem de sucesso ou erro. O código do método `update` é o seguinte:

```
public function update(Request $request, string $id)
{
    $updated = $this->user->where('id', $id)->update($request->except(['_token', '_method', 'updated_at']));

    if ($updated) {
        return redirect()->back()->with('message', 'Successfully Updated');
    }
    return redirect()->back()->with('message', 'Error Updated');
}
```

## Método `destroy`

O método `destroy` é responsável por excluir um usuário do banco de dados. Ele recebe o ID do usuário a ser excluído e realiza a exclusão no banco de dados. Em seguida, redireciona para a rota `'users.index'`. O código do método `destroy` é o seguinte:

```
public function destroy(string $id)
{
    $this->user->where('id', $id)->delete();

    return redirect()->route('users.index');
}
```