

**CURSOS  
TÉCNICOS**

**DESENVOLVIMENTO DE  
SISTEMAS WEB II**

**Eixo Informática para Internet**

**Unidade 6**

## SUMÁRIO

### UNIDADE 6

1 PHP com framework moderno: Introdução ao Laravel.....	2
1.1 O que é um Framework?.....	3
1.2 O que é o Laravel?.....	4
2 Composer - Gerenciamento de Dependências.....	4
3 Instalando o NODE e NPM.....	5
4 Iniciando o primeiro projeto Laravel.....	6
4.1 Conhecendo a estrutura de um projeto Laravel.....	9
4.2 Arquitetura MVC.....	10
4.3 Configurando banco de dados no Laravel.....	11
5 Criando Controllers e definindo rotas.....	12
5.1 Definição da rota.....	13
6 Definindo uma View.....	14
6.1 Definindo uma View com Blade.....	16
6.2 Diretivas do Blade .....	17
6.3 Componentes do Blade.....	19
7. Estilizando com Bootstrap.....	20
8. Fixando o conhecimento: Desafios.....	22
9. Referências.....	23

## UNIDADE 6

### 1 PHP com framework moderno: Introdução ao Laravel

Nesta unidade, o foco será em PHP com framework moderno e suas particularidades.

#### 1.1 O que é um Framework?

Um framework, em termos gerais, é uma estrutura ou conjunto de ferramentas que **fornece uma base para o desenvolvimento de software**. Ele oferece uma abstração reutilizável de código para ajudar os desenvolvedores a criar aplicativos, sistemas ou módulos de software de maneira mais eficiente e consistente.

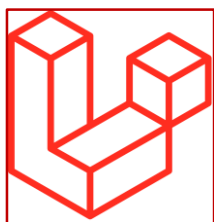
Os frameworks geralmente **incluem bibliotecas, convenções de codificação, padrões de projeto** e outros componentes que facilitam o desenvolvimento, permitindo que os desenvolvedores se concentrem mais na lógica específica do aplicativo em vez de se preocuparem com detalhes de implementação comuns. Além disso, os **frameworks podem fornecer uma estrutura organizacional para o código**, promovendo boas práticas de desenvolvimento e facilitando a manutenção e expansão do software ao longo do tempo.

#### Exemplos de frameworks:

→ **Frontend (Next, Angular e VUE)**



→ **Backend ou Fullstack (Laravel, Symfony e Spring Boot)**



Aqui destacamos o **Laravel** e o **Symfony** que são frameworks voltados para o ecossistema de desenvolvimento PHP. Existem outros, porém estes são um dos mais populares.

## 1.2 O que é o Laravel?

O Laravel é um framework de desenvolvimento web para a linguagem de programação PHP. Ele foi criado por Taylor Otwell e é conhecido por fornecer uma estrutura elegante e expressiva para o desenvolvimento de aplicativos web modernos. É um dos frameworks de desenvolvimento PHP mais populares e possui uma vasta comunidade de devs, e também, uma ótima documentação para quem está iniciando ou busca referências.

### **Alguns recursos que o Laravel inclui:**

- ✓ Sistema de roteamento;
- ✓ Eloquent ORM;
- ✓ Artisan Console;
- ✓ Laravel Mix;
- ✓ Middlewares;
- ✓ Ferramentas de segurança (SQL Injection e CSRF - Cross-Site Request Forgery);
- ✓ Sistemas de Autenticação;
- ✓ Templates HTML;
- ✓ Estrutura pronta e adequada ao padrão MVC;
- ✓ Conexão e modelagem de banco de dados facilitada.

Em suma, são muitos recursos que o Laravel fornece logo ao iniciar o projeto. Isto pode tornar o desenvolvimento mais ágil e prático.

## **2 Composer - Gerenciamento de Dependências**

O Composer é uma ferramenta de gerenciamento de dependências para PHP. Ele é amplamente utilizado na comunidade de desenvolvimento PHP para facilitar a inclusão e o gerenciamento de bibliotecas e pacotes de terceiros em projetos PHP. O Composer ajuda a lidar com as dependências do projeto de forma eficiente.

### **Dos principais pontos, destaca-se:**

- **Autoloading** → Composer facilita o carregamento automático de classes

- Gerenciamento de dependências** → Composer permite a possibilidade de definir as dependências do projeto no arquivo **composer.json**. Analisando o arquivo mencionado, o Composer instala as bibliotecas necessárias, além das suas próprias dependências.
- Instalação de Pacotes** → Composer instala pacotes (bibliotecas, frameworks, etc) a partir de um repositório online chamado packagist, que é um repositório central para bibliotecas do PHP.

Segundo a documentação oficial, o composer **é fortemente inspirado pelo Node Package Manager (npm)** e pelo **bundle do Ruby**. Saiba mais em: [Introdução - Composer](#).

### **Para instalar:**

1. Acesse o site da documentação oficial. [Link Documentação oficial](#).

2. Navegue até a seção “Installation - Windows”.

\* Caso você possua outro Sistema operacional, confira as outras seções de instalação do Composer em MacOS ou Linux.

3. Clique no link “**Composer-Setup.exe**” para **baixar o arquivo instalador** da última versão do Composer.

#### **Installation - Windows #**

##### **Using the Installer #**

This is the easiest way **Clique aqui!** it up on your machine.

Download and run **Composer-Setup.exe**. It will install the latest Composer version and set up your PATH so that you can call `composer` from any directory in your command line.

**Note:** Close your current terminal. Test usage with a new terminal: This is important since the PATH only gets loaded when the terminal starts.

4. Clique no instalador do composer

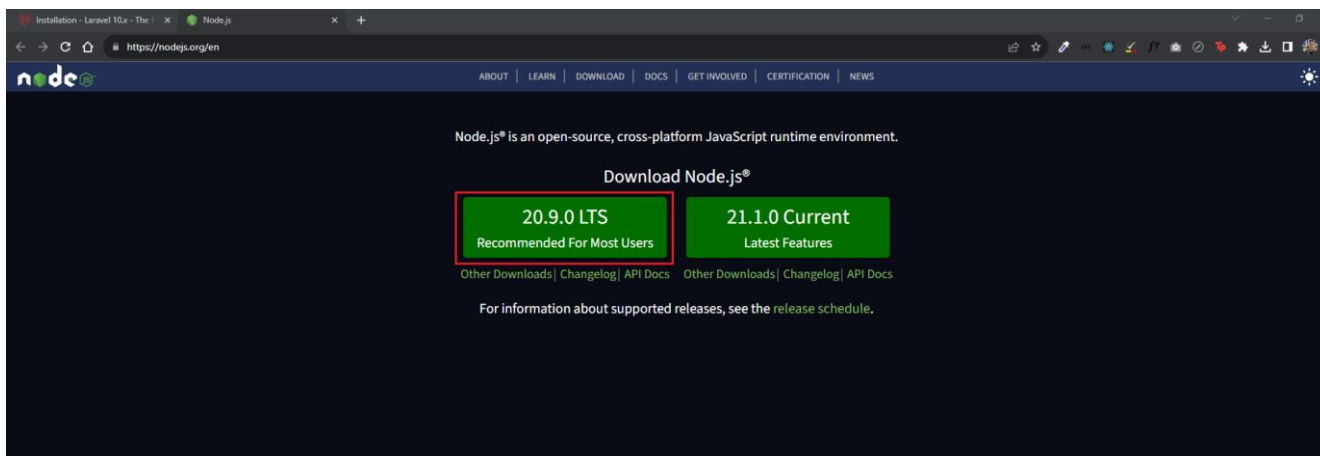
5. Faça o processo de instalação dando o aceite nas etapas. (Este processo não deve demorar muito)

### **3 Instalando o NODE e NPM**

O npm (*Node Package Manager*) é o sistema de gerenciamento de pacotes para o Node.js. Ele permite que os desenvolvedores instalem, compartilhem e gerenciem dependências de projetos Node.js de maneira eficiente. **Para iniciar um projeto Laravel, é recomendado pela documentação fazer a instalação do NODE e NPM.**

**Para instalar:**

1. Acesse o link da página do Node → <https://nodejs.org/en>
2. Clique no botão para instalar a versão LTS do Node.



3. Execute o arquivo .msi e dê o aceite em todas as etapas. Este processo não deve demorar.cmd
4. Caso queira verificar se a instalação ocorreu bem, abra seu terminal de comando
5. Digite o comando: **node -v** e **npm -v**
6. Ambos comandos devem retornar a versão das instalações.

#### ***4 Iniciando o primeiro projeto Laravel***

Com o **Composer** e **Node** instalados você pode criar um novo projeto **Laravel**.

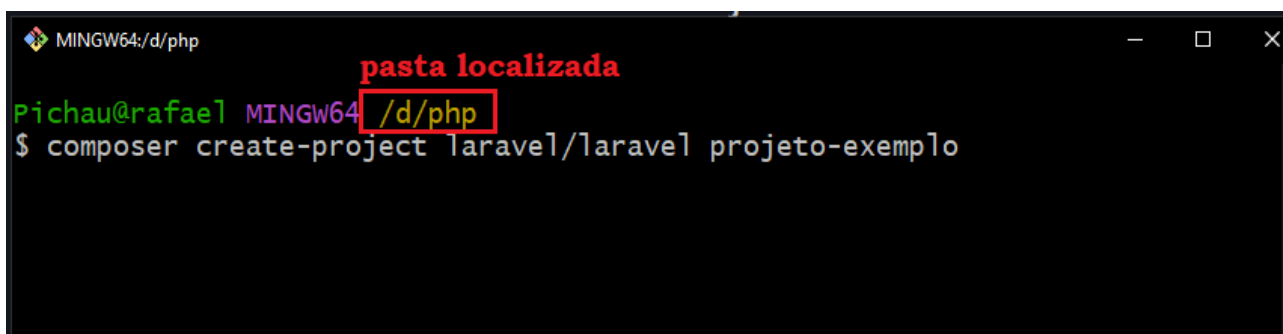
**Caso queira acompanhar pela documentação oficial:**

→ <https://laravel.com/docs/10.x/installation#your-first-laravel-project>

1. Abra um terminal de comando em uma pasta que você deseja iniciar um projeto Laravel.
2. Rode o comando

**composer create-project laravel/laravel projeto-exemplo**

**Exemplo** - Terminal de comando e comando de criação de projeto Laravel



Neste exemplo, foi aberto o terminal do **GIT Bash** para executar o comando. Uma vez digitado o comando **é só apertar enter** e aguardar a instalação.

**3.** Vão ser instalados inúmeros arquivos. Se tudo der certo, nenhum erro será gerado.

```
- Installing spatie/laravel-ignition (2.3.1): Extracting archive
63 package suggestions were added by new dependencies, use 'composer suggest'
to see details.
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
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE
spatie/laravel-ignition ..... DONE

82 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force

 INFO  No publishable resources for tag [laravel-assets].

No security vulnerability advisories found.
> @php artisan key:generate --ansi

 INFO  Application key set successfully.
```

**4.** Entre na pasta do projeto criado com: **cd <nome da pasta>** e execute.

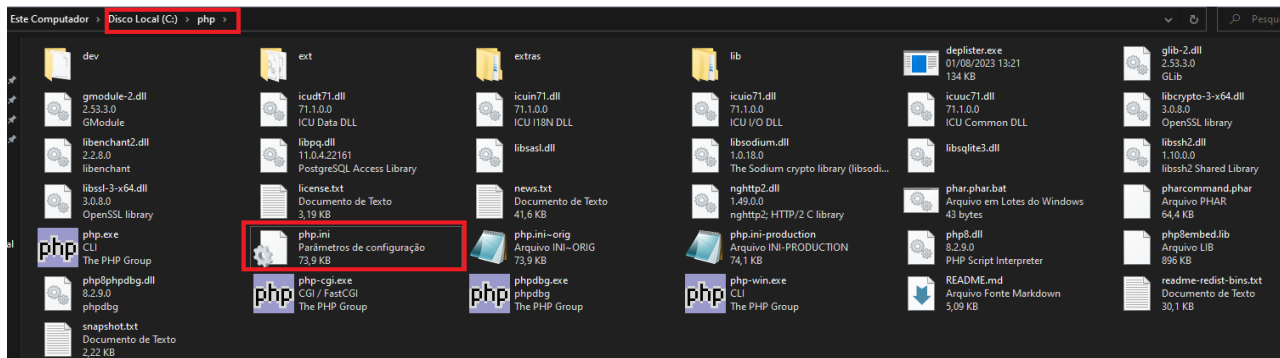
**5.** Uma vez dentro da pasta execute o comando **code** . para abrir o VSCode na pasta do projeto gerado

```
Pichau@rafael MINGW64 /d/php anterior
$ cd projeto-exemplo/

Pichau@rafael MINGW64 /d/php/projeto-exemplo atual
$ code .| abrir VSCode
```

**6.** Antes de tudo, encontre dentro do seu computador um arquivo chamado **php.ini**.

Geralmente este arquivo **fica no diretório C:** do seu computador.



**7.** Você deve abrir o arquivo (como se fosse um bloco de notas) pesquisar pela extensão **fileinfo** e REMOVER O COMENTÁRIO (“;”), que é um ponto e vírgula. Segue a imagem de como deve ficar:

```

php.ini - Bloco de Notas
Arquivo  Editar  Formatar  Exibir  Ajuda

; Dynamic Extensions ;
;
; If you wish to have an extension loaded automatically, use the following
; syntax:
;
;   extension=modulename
;
; For example:
;
;   extension=mysqli
;
; When the extension library to load is not located in the default extension
; directory, You may specify an absolute path to the library file:
;
;   extension=/path/to/extension/mysqli.so
;
; Note : The syntax used in previous PHP versions ('extension=<ext>.so' and
; 'extension='php_<ext>.dll') is supported for legacy reasons and may be
; deprecated in a future PHP major version. So, when it is possible, please
; move to the new ('extension=<ext>') syntax.
;
; Notes for Windows environments :
;
; - Many DLL files are located in the ext/
;   extension folders as well as the separate PECL DLL download.
;   Be sure to appropriately set the extension_dir directive.
;
extension=bz2

; The ldap extension must be before curl if OpenSSL 1.0.2 and OpenLDAP is used
; otherwise it results in segfault when unloading after using SASL.
; See https://github.com/php/php-src/issues/8620 for more info.
extension=ldap
extension=curl
extension=ffi
extension=ftp
extension=fileinfo
extension=gd
  
```

**Não esqueça de SALVAR as alterações (ctrl + s).**

**8.** Com o VSCode aberto, no terminal. Digite o comando: **composer install** e aguarde a instalação.

```

Arquivo  Editar  Seleção  Ver  Acessar  Executar  ...
projeto-exemplo

PROJETO-EXEMPLO
├── app
├── bootstrap
├── config
├── database
├── public
├── resources
├── routes
├── storage
├── tests
├── vendor
├── .editorconfig
├── .env
├── .env.example
├── .gitattributes
├── .gitignore
├── artisan
├── composer.json
├── composer.lock
├── package.json
├── phpunit.xml
├── README.md
└── vite.config.js

Iniciar
├── Novo Arquivo...
├── Abrir o Arquivo...
├── Abrir a Pasta...
├── Clonar um Repositório Git...
├── Abrir um passo a passo...
└── Conectar a...

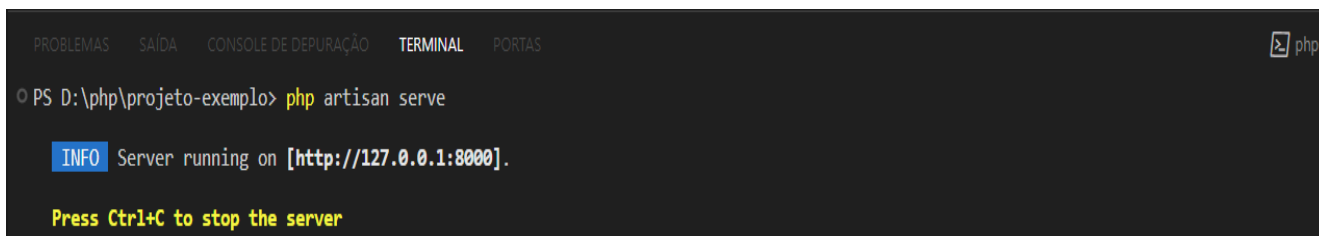
Recente
├── C:\
├── controle-series  D:\php\laravel
├── api-example      D:\php\laravel
├── 04-ignite-shop   D:\projetos\next
├── acessibilidade-exemplo  D:\vemser13\projetos
├── html             D:\vemser13\exemplos
├── ignite-github-blog  D:\ignite\03-consumo-de-api-e-performance
├── 03-dt-money       D:\ignite\03-consumo-de-api-e-performance
├── node             D:\projetos
├── questoes-gabarito  D:\vemser12\prova-tecnica
└── Mais...

Mostrar a página inicial na inicialização

PROBLEMAS  SAÍDA  CONSOLE DE DEBUGAÇÃO  TERMINAL  PORTAS
PS D:\php\projeto-exemplo> composer install
  
```



9. Após instalado, basta rodar o comando: **php artisan serve** para iniciar um servidor local com a aplicação Laravel.



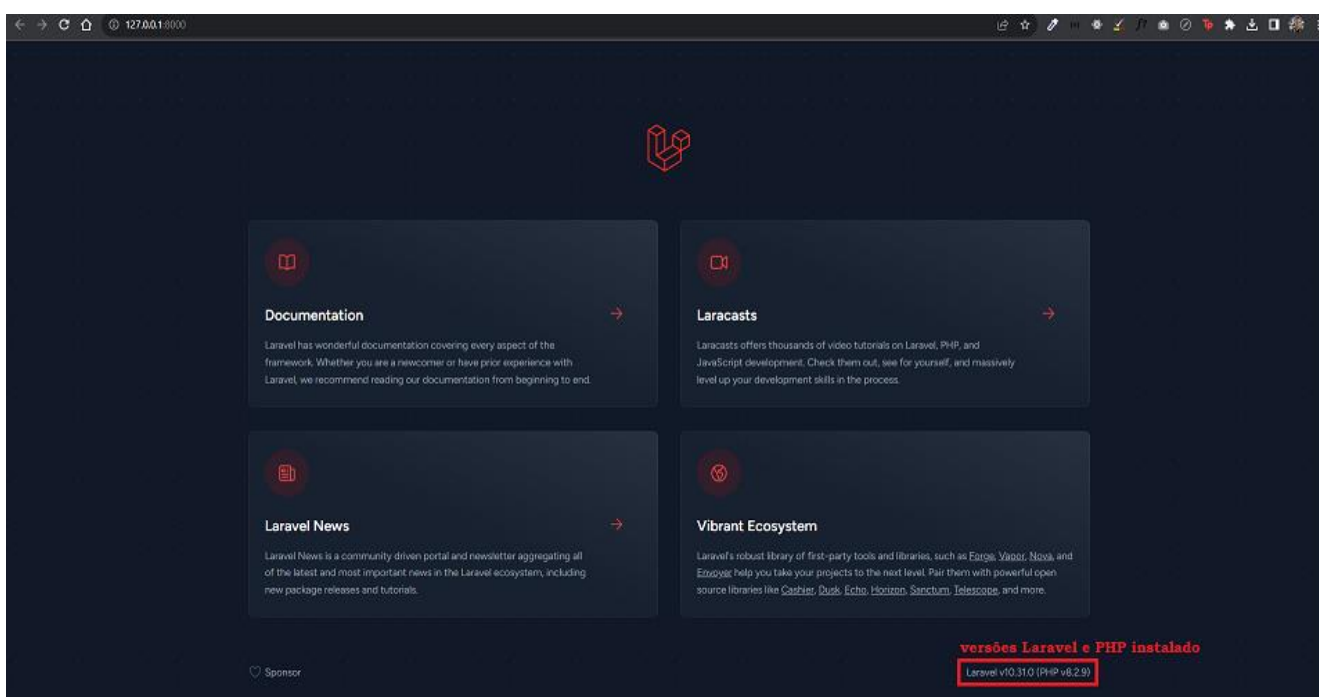
```

PROBLEMAS  SAÍDA  CONSOLE DE DEPUÇÃO  TERMINAL  PORTAS
PS D:\php\projeto-exemplo> php artisan serve

[INFO] Server running on [http://127.0.0.1:8000].

Press Ctrl+C to stop the server
  
```

10. Abra no seu **navegador** o servidor de acordo com o **ip** e **porta** gerada. Neste caso, seria: **127.0.0.1:8000** OU **localhost:8000**.

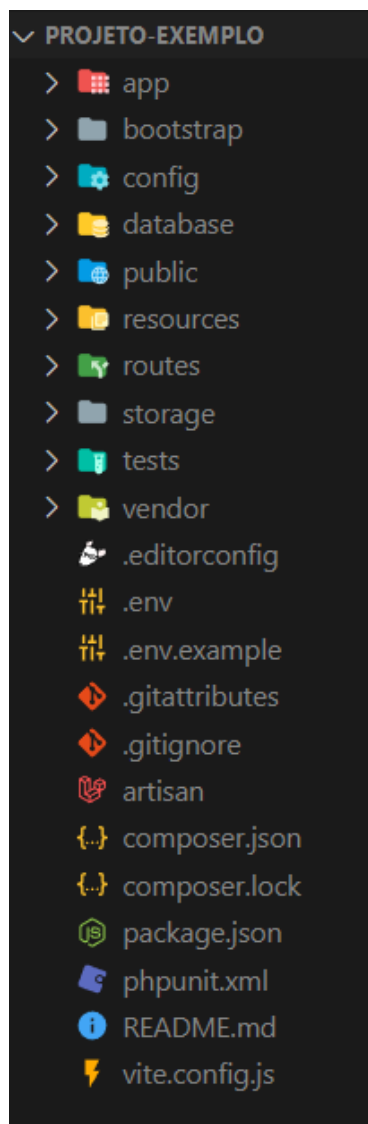


## 4.1 Conhecendo a estrutura de um projeto Laravel

Conhecer a estrutura e a organização de diretório de um novo framework é essencial para encontrar os recursos, e também manter o padrão de projeto estabelecido.

**No Laravel, temos a seguinte estrutura de pastas:**

Basicamente, iremos utilizar os diretórios sob demanda. Com base em algo que precisamos, navegamos até o diretório e fazemos as alterações. Alguns repositórios serão mais acessados ao longo do desenvolvimento. Segue:



**app** → Deve conter o principal código da aplicação. É neste que teremos os **controllers** e **models**.

**config** → Como o nome implica o diretório **config** possui arquivos de configurações do projeto. Inclusive informações sobre o **banco de dados utilizado**, **idioma** e **horário de referência** da aplicação.

**database** → O diretório **database** contém as **migrations** do seu banco de dados, além de **model factories (modelos fabricados)**.

**public** → **Public** contém o arquivo **index.php**, que é o ponto de entrada para todas as solicitações que entram em sua aplicação e configura o carregamento automático.

**resources** → No diretório **resources** contém as **views** bem como seus ativos brutos, não compilados, como CSS e JavaScript.

**routes** → **routes** contém todas as definições de rotas para sua aplicação. Por padrão, vários arquivos de rotas estão incluídos no Laravel: **web.php**, **api.php**, **console.php** e **channels.php**. **Podemos definir tanto rotas para nossa aplicação frontend, quanto rotas de uma API criada em Laravel.**

→ Para descrição completa dos diretórios: [Diretórios - Laravel](#)

## 4.2 Arquitetura MVC

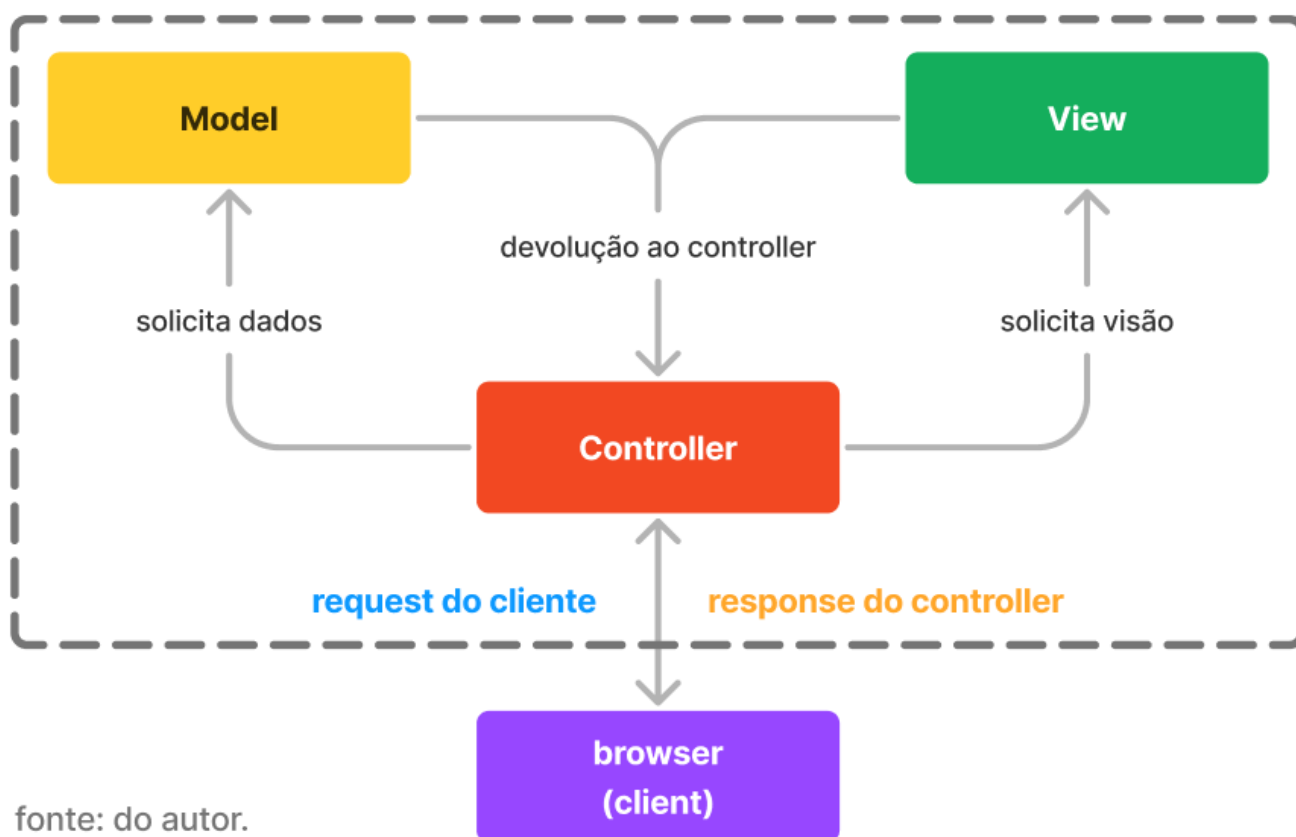
A arquitetura **MVC (Model-View-Controller)** é um padrão de projeto amplamente utilizado no desenvolvimento de software para separar os componentes principais de uma aplicação dividindo a aplicação em três componentes principais:

**1. Model (Modelo):** Representa a **camada de dados e lógica de negócios da aplicação**. Ele lida com o **acesso aos dados**, a validação, as regras de negócios e outras operações relacionadas à **manipulação dos dados**.

**2. View (Visão):** É responsável pela **apresentação da interface do usuário** e exibição dos dados fornecidos pelo modelo. A visão recebe as informações do modelo e as **renderiza para**

que o usuário possa interagir. Ela não contém lógica de negócios; seu foco é na representação visual.

**3. Controller (Controlador):** Atua como **intermediário entre o modelo e a visão**. Ele **recebe as entradas do usuário**, processa essas entradas (por meio do modelo) e atualiza a visão de acordo. **O controlador é responsável por gerenciar o fluxo de controle na aplicação**. Segue esquema para exemplificar:



fonte: do autor.

A principal vantagem deste padrão de projeto é a **separação de responsabilidades**, que facilita a manutenção e o desenvolvimento de software, pois as alterações em uma parte do sistema têm impacto mínimo nas outras.

#### 4.3 Configurando banco de dados no Laravel

Por padrão, o Laravel utiliza o banco de dados do MySQL. Para configurar o projeto para utilizar outro banco de dados é necessário que isto seja informado nas variáveis de ambiente (arquivo **.env**). Os sistemas de banco de dados que o Laravel já fornece suporte são: **MySQL**, **SQLite**, **PostgreSQL**. Isso pode ser visualizado dentro em: **config > database.php**

**Para conectar ao seu banco de dados basta:**

1. Acessar arquivo **.env** e preencher de acordo com as suas credenciais.

```

.env database.php
.env
12 DB_CONNECTION=mysql
13 DB_HOST=127.0.0.1
14 DB_PORT=3306
15 DB_DATABASE=laravel_exemplo
16 DB_USERNAME="root"
17 DB_PASSWORD="sua_senha"
18

```

2. Uma vez configurado **conexão, host, porta, nome da base de dados, usuário** (o mesmo que aparece no seu MySQL Workbench) e **senha**, salve o arquivo.

3. Abra o terminal do seu VSCode e rode o comando, **php artisan migrate**

```

PS D:\php\projeto-exemplo> php artisan migrate

[WARN] The database 'laravel_exemplo' does not exist on the 'mysql' connection.

Would you like to create it? (yes/no) [no]
> yes

[INFO] Preparing database.

Creating migration table ..... 51ms DONE

[INFO] Running migrations.

2014_10_12_000000_create_users_table ..... 37ms DONE
2014_10_12_100000_create_password_reset_tokens_table ..... 13ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 23ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 15ms DONE

```

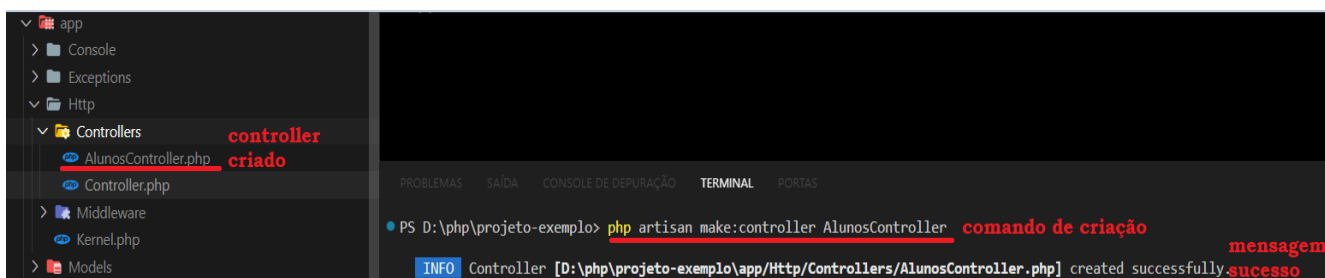
Com isso, foi criada uma estrutura inicial de tabelas. **Confira no MySQL Workbench.**

## 5 Criando Controllers e definindo rotas

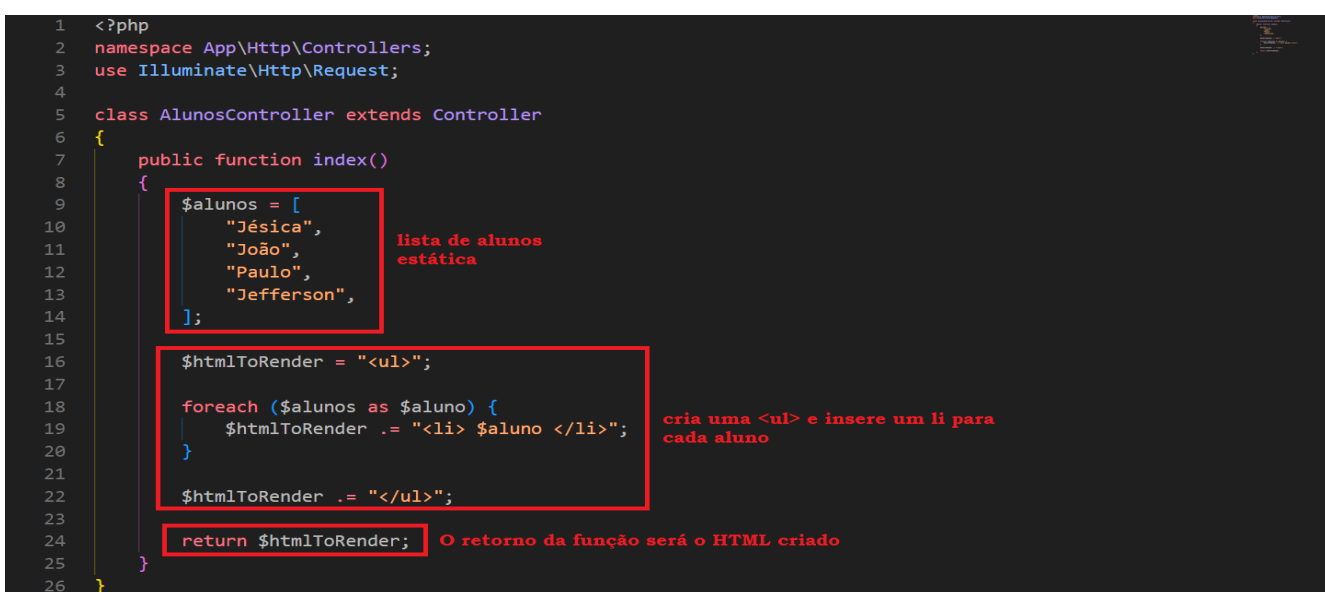
Podemos criar novos controllers para a nossa aplicação utilizando os comandos do **artisan make**. Rodando o comando: **php artisan make** você confere todos os comandos de criação possíveis no **artisan**. É possível criar controllers, models, views, migrations e etc., **tudo via linha de comando**.

Vamos criar um controller chamado **AlunosController** para posteriormente utilizarmos a tabela do desafio da semana passada. Para isso segue a estrutura do comando de criação de um controller usando artisan: **php artisan make:controller <NomeController>**

Com isso, é gerado um arquivo com a estrutura inicial de um controller estendendo as características (atributos e métodos) de um Controller padrão.



Utilizamos da estrutura padrão e montamos um código que servirá de exemplo didático para o entendimento dos **controllers e rotas**. Note que, criamos uma função de nome **index** para retornar uma lista HTML.



## 5.1 Definição da rota

Para definir uma nova rota para a aplicação frontend, acesse o diretório: **routes >**

**web.php**



**Esta é a estrutura padrão do arquivo de rotas.** Perceba que já há uma rota definida com endereço (“/” ou root) que retorna uma view chamada **“welcome”**, que nada mais é do que a página que vemos ao utilizar o comando **php artisan serve**.

**Exemplo** - Criação da nova rota vinculada ao Controller (AlunosController).

```

1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  use App\Http\Controllers\AlunosController;
6
7  Route::get('/', function () {
8      return view('welcome');
9  });
10
11 Route::get('/alunos', [AlunosController::class, 'index']);
12

```

**importação do Controller**

**função apontada**

**definição da nova rota**

É necessário realizar a **importação do Controller** no arquivo, e **definir uma nova rota**. A estrutura seria: **Route::get('ulr-da-rota', [NomeController::class, 'nome\_funcao']);**

Ao acessar: **localhost:8000/alunos** temos o retorno do controller em tela.



• Jéssica  
• João  
• Paulo  
• Jefferson

**NOTA:** O exemplo utilizado acima ilustra a conexão entre **uma nova rota** e **a lógica de retorno de um dado Controller**. No entanto, esta prática de retornar um HTML puro direto no retorno da função do Controller não é adotada nas práticas reais de mercado, pois **fere o conceito de separação de responsabilidades**.

Atualmente, no Controller retornamos uma view que está em outro arquivo. Assim conseguimos separar a parte lógica da parte visual.

**Vamos refatorar o exemplo acima neste próximo índice.**

## 6 Definindo uma View

Views são a parte visual, ou seja, o que o usuário enxerga, então normalmente dedicamos estes arquivos de view somente para definir a estrutura, estilização e renderizar esses elementos.

**Exemplo - Criando view “listar-aluno” (sem blade) e Refatorando o Controller.**

1. Podemos criar um novo arquivo <nome-arquivo.php> em: **resources > views**

```

1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Listar alunos</title>
7 </head>
8 <body>
9
10 <h1>Listar alunos</h1>
11 <hr>
12
13 <ul>
14 <?php foreach($alunos as $aluno) { ?>
15
16 <li><?= $aluno ?></li>
17
18 <?php } ?>
19 </ul>
20
21 </body>
22 </html>
    
```

2. Definimos a estrutura HTML do arquivo para executar a mesma funcionalidade do que estava sendo feito antes no Controller.

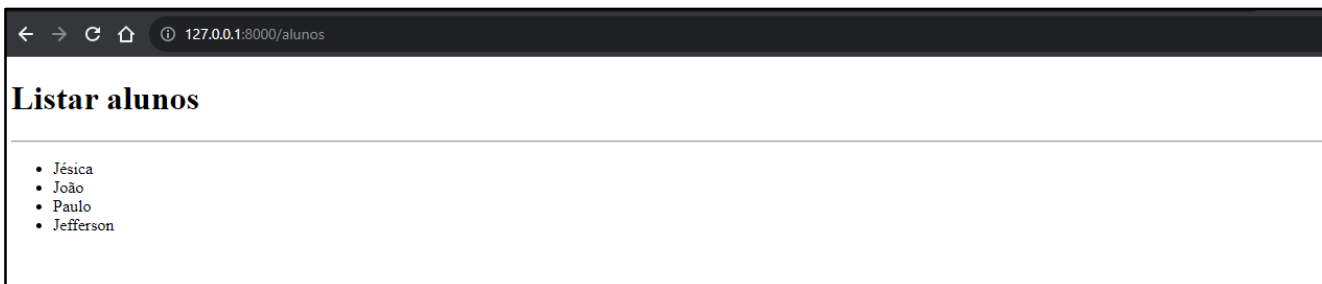
Perceba que apenas definimos a **estrutura visual** e **realizamos um foreach na lista de alunos que será enviada via Controller**. Veja que colocamos o código **entre tags PHP**. Algo muito similar ao que fizemos na semana de PHP para a WEB.

3. No Controller **AlunosController**, agora **precisamos definir o retorno da função index** como **uma view** e **enviar os dados da lista de alunos para a view**.

```

1 <?php
2 namespace App\Http\Controllers;
3 use Illuminate\Http\Request;
4
5 class AlunosController extends Controller
6 {
7     public function index()
8     {
9         $alunos = [
10             "Jésica",
11             "João",
12             "Paulo",
13             "Jefferson",
14         ];
15         nome da view
16         return view('listar-alunos', [
17             'alunos' => $alunos lista de dados que serão enviados para
18             a view
19         ]);
20     }
21 }
    
```

4. Acessando o navegador novamente temos o mesmo resultado, porém agora as lógicas da nossa aplicação estão muito melhor separadas no projeto.



### 6.1 Definindo uma View com Blade

No passado, a maioria das aplicações em PHP renderizava HTML para o navegador usando modelos HTML simples intercalados com declarações **PHP echo**, que exibem dados recuperados de um banco de dados durante a solicitação. Conforme exemplo anterior:

```

<ul>
  <?php foreach($series as $serie) { ?>
    <li><?= $serie ?></li>
  <?php } ?>
</ul>
  
```

No Laravel, ainda é possível alcançar essa abordagem de renderização HTML usando visões (views) e o Blade. O Blade é uma linguagem de **modelagem extremamente leve** que **oferece uma sintaxe conveniente e simplificada para exibir dados**, iterar sobre dados e muito mais.

**Exemplo** - Renderização com Blade

```

<div>
  @foreach ($users as $user)
    Hello, {{ $user->name }} <br />
  @endforeach
</div>
  
```

Note que a sintaxe é mais simplificada, neste caso. Para conferir mais informações sobre PHP para frontend:

→ [PHP Frontend - PHP & Blade](#)



**NOTA:** Ao construir aplicações dessa maneira, envios de formulários e outras interações na página geralmente recebem um documento HTML inteiramente novo do servidor, e **a página inteira é redesenhada pelo navegador.**

Apesar do usuário perder um pouco em UX pelo constante recarregamento de página, muitas aplicações mais simples podem ser construídas utilizando os templates HTML do Blade.

**É de extrema importância ressaltar que o Laravel é flexível podendo inclusive gerar API's que podem ser consumidas com frontends mais modernos como React, Next e Vue.**

### Exemplo - Refatorando o código HTML

1. Para refatorar o exemplo que criamos primeiro precisamos definir **"listar-alunos.php"** para **"listar-alunos.blade.php"** para definir um arquivo view de template HTML Blade.
2. Logo após renomear o arquivo, podemos alterar o código do foreach para o mostrado no exemplo utilizando a sintaxe do blade

**Exemplo** - Foreach com blade.

```
<ul>
    @foreach($alunos as $aluno)
        <li> {{$aluno}} </li>
    @endforeach
</ul>
```

Desta forma, utilizamos a sintaxe do template html blade, porém o resultado em tela seguirá o mesmo.

### 6.2 Diretivas do Blade

Além da herança de templates e da exibição de dados, o Blade também oferece atalhos convenientes para estruturas de controle PHP comuns, como declarações condicionais e loops. Esses atalhos proporcionam uma maneira limpa e concisa de trabalhar com estruturas de controle em PHP, ao mesmo tempo em que permanecem familiares aos seus equivalentes em PHP. Estes atalhos são chamados de **diretivas**.

→ **Diretivas If:** Podemos utilizar diretivas if para renderizar ou não um template HTML pré-definido

```

@if (count($records) === 1)
    I have one record!
@elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif
  
```

→ Diretivas **@isset** e **@empty**:

```

@isset($records)
    // $records is defined and is not null...
@endisset

@empty($records)
    // $records is "empty"...
@endempty
  
```

→ Diretivas **@auth** e **@guest**: Podem ser usadas para retornar diferentes templates HTML caso o usuário esteja autenticado ou não

```

@auth('admin')
    // The user is authenticated...
@endauth

@guest('admin')
    // The user is not authenticated...
@endguest
  
```

Para um overview de todas diretivas, basta acessar a documentação oficial:

→ [Diretivas - Blade](#)

### 6.3 Componentes do Blade

Para que não seja necessário repetir toda uma estrutura padrão do HTML como por exemplo a declaração da tag HTML, HEAD, Body, elementos padrões como cabeçalhos e rodapés, por exemplo, podemos criar componentes do blade que podem **receber parâmetros** e **estruturas HTML**.

#### 6.3.1 Criando componente

1. Para criar um componente precisamos criar uma pasta **components** dentro de **views**.
2. Criamos então o nosso componente, dentro desta **components**. Vamos criar um componente "layout.blade.php" para conter uma estrutura padrão.

```

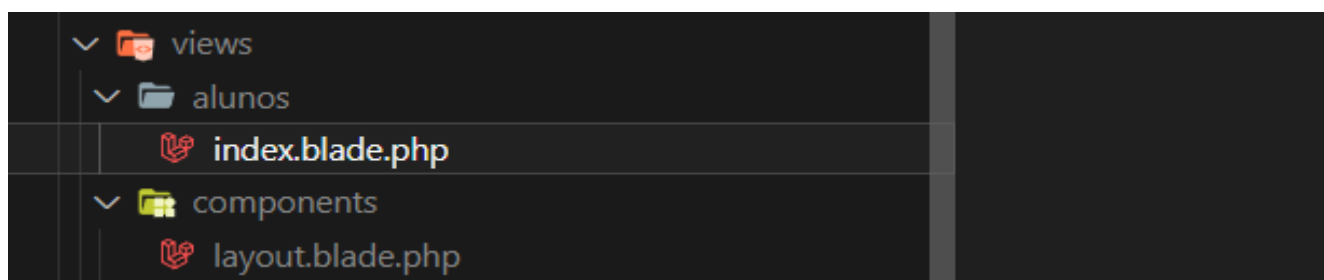
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>{{ $title }}</title>
7 </head>
8 <body>
9     <h1>{{ $title }}</h1>
10    <hr>
11
12    <!-- conteúdo dinâmico -->
13    {{ $slot }}
14 </body>
15 </html>
16

```

parâmetros dinâmico de título. Recebido na chamada do componente

dentro do slot será inserido todo conteúdo na chamada do componente

3. Vamos criar uma **pasta alunos** dentro de **view** para armazenar o **index.blade.php** da página de listagem de alunos para então, chamar o layout recém criado.



4. No arquivo **index.blade.php** vamos chamar o componente **layout.blade.php** e atribuir parâmetros e o slot.

```

1 <x-layout title="Alunos"> parâmetros
2     <ul>
3         @foreach($alunos as $aluno)
4             <li> {{ $aluno }} </li>
5         @endforeach
6     </ul>
7 </x-layout>
8

```

Conteúdo dinâmico: slot

x + nome componente = tag para utilizar componente

\* Perceba que utilizamos uma sintaxe de declaração de tag HTML para utilizar o componente.

5. Como alteramos o local do arquivo de renderização da lista de alunos, temos que refletir isso no AlunoController.

```
app > Http > Controllers > AlunosController.php
1  <?php
2  namespace App\Http\Controllers;
3  use Illuminate\Http\Request;
4
5  class AlunosController extends Controller
6  {
7      public function index()
8      {
9          $alunos = [
10             "Jésica",
11             "João",
12             "Paulo",
13             "Jefferson",
14         ];
15         alunos.index = procure pelo arquivo index DENTRO da pasta ALUNOS
16         return view('alunos.index', [
17             'alunos' => $alunos
18         ]);
19     }
20 }
```

6. **Faça o teste no seu servidor local** para verificar se há alguma mudança. Em termos visuais nada foi alterado, porém agora, temos um componente **para que não seja preciso repetir códigos em vão**. Caso o formato das páginas do nosso projeto seja em partes similares ou que se repitam é indicado utilizar componentes.

**NOTA:** É importante destacar que além de fornecer facilidades na escrita e organização de código o **Blade pode trazer mais segurança** para suas aplicações WEB utilizando de lógicas que proíbem a inserção de código malicioso via inputs ou envio de formulários.

## 7. Estilizando com Bootstrap

Podemos utilizar o **CDN** (content Delivery Network do **Bootstrap** para estilizar nossa aplicação de forma mais rápida com estilos prontos atribuídos através de classes.

Caso você queria saber mais sobre o Bootstrap, confira a documentação oficial:

→ [Bootstrap - Iniciando/Instalação](#)

\* **Vamos utilizar estes dois links do Bootstrap na tag Head do nosso layout.**

<b>CSS</b>	<a href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css">https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css</a>
<b>Javascript</b>	<a href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js">https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js</a>

**Exemplo** - Inserção dos links para utilizar Bootstrap no layout.blade.php

```

layout.blade.php
resources > views > components > layout.blade.php
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>{{ $title }}</title>
7   <!-- Estilos Bootstrap -->
8   <link
9     href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
10    rel="stylesheet" integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
11    crossorigin="anonymous"
12  >
13
14 </head>
15 <body>
16   <h1>{{ $title }}</h1>
17   <hr>
18
19   <!-- conteúdo dinâmico -->
20   {{ $slot }}
21
22   <!-- scripts - parte lógica bootstrap -->
23   <script
24     src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"
25     integrity="sha384-C6RzsynM9kWDrmNet87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL"
26     crossorigin="anonymous">
27   </script>
28 </body>
29 </html>
  
```

Com os links vinculados agora podemos estilizar a aplicação com todos os recursos disponíveis do Bootstrap.

### Exemplo de estilização básica

- Vamos estilizar o container da lista;
- A lista em si;
- Adicionar um botão para “Adicionar aluno” (para depois darmos continuidade ao projeto de exemplo) e vamos estilizar também.



Obtivemos este resultado no fim. Basicamente, foi adicionada uma tag `<main>` envolta do conteúdo da página no layout e algumas classes em `<ul>`, `<li>` e no `<a>` na página do **index.blade.php** de alunos.

### **8. Fixando o conhecimento: Desafios**

Realize aprimoramentos no projeto desenvolvido na apostila OU faça você mesmo um do zero.

1. **Crie um Controller** para uma entidade específica (Aluno, Animal, Bancos...)
2. Crie uma função no Controller para retornar uma View. **Repasse uma lista ou alguns dados para renderizar na view.**
3. **Crie uma view** específica utilizando templates, diretivas e/ou layouts com Blade.
4. **Vincule uma rota** vinculando o controller e a função que retorna a View.
5. Faça a estilização da página com Bootstrap.

**NOTA:** Faça o desafio para que você possa obter e fixar o conhecimento de uma parte da arquitetura MVC do Laravel.

**Na próxima semana trabalharemos mais com os Models e a conexão com o banco de dados para listar, criar, editar e deletar dinamicamente e refletir as alterações no banco.**

Confira o projeto desenvolvido até o momento: [Repositório Github - Controller e View](#)

→ **caso você queria clonar o repositório com os códigos da aula você deverá seguir o passo-a-passo:**

1. Rode **git clone** <url-de-clone-do-projeto>
2. Rode **composer install** (na pasta do projeto clonado)
3. Rode **cp .env.example .env**
4. Rode **php artisan key:generate**
5. Rode **php artisan migrate** (não esqueça de inserir as credenciais do seu banco de dados)
6. Rode **php artisan serve** para iniciar o server da aplicação
7. Vá para o link: [localhost:8000](http://localhost:8000)

**Não hesite em recorrer a documentação oficial do Laravel. Lá você vai encontrar com mais detalhes todo o conteúdo que iremos desenvolver ao longo das semanas.**

→ <https://laravel.com/docs/10.x>

## 9. Referências

Múltiplos autores: **Node.js**, não informado. Disponível em: <<https://nodejs.org/en>>. Acesso em 10 de novembro de 2023.

Múltiplos autores: **Composer: A Dependency Manager for PHP**, não informado. Disponível em: <<https://getcomposer.org/>>. Acesso em 10 de novembro de 2023.

Múltiplos autores: **Bootstrap: Get started with Bootstrap**, não informado. Disponível em: <<https://getbootstrap.com/docs/5.3/getting-started/introduction/>>. Acesso em: 12 de novembro de 2023.

Múltiplos autores: Laravel: **Installation**, não informado. Disponível em: <<https://laravel.com/docs/10.x>>. Acesso em: 12 de novembro de 2023.