

CREACIÓN DE UNA APLICACION WEB CON LARAVEL 8.0

En esta serie vamos a manejar lo siguiente:

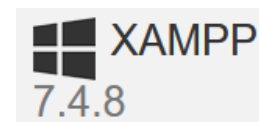
1. Practica. Creación de un proyecto en Laravel
2. Practica. Sistema de carpetas y archivos - Manejo de rutas, controladores y vistas
3. Practica. Blade Templates (Vista View – Pagina Web con Bootstrap)
4. Practica. Estructuras de control (condiciones y ciclos dentro de Blade)

Instructor Franco A Reina Argoty – ADSI CTPI

1. Practica. Creación de un proyecto en Laravel

A. Descargar los archivos de Instalación de PHP – Paquete de XAMPP

<https://www.apachefriends.org/es/download.html>



B. Descargue el instalador de Composer

(Descarga) <https://getcomposer.org/download/>

Ayuda Instalación Composer: <https://styde.net/instalacion-de-composer-y-laravel-en-windows/>

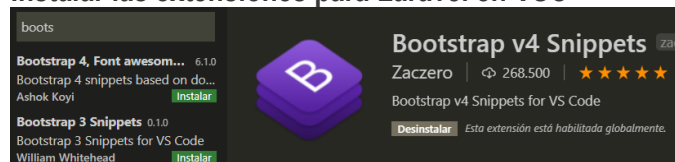
Composer conjunto de librerías que se manejan para PHP manejador de dependencias que permite trabajar elementos como la instalación de Laravel.

C. Descargue y configure las extensiones en el editor de código Visual Studio Code VSC

<https://code.visualstudio.com/download>



Instalar las extensiones para Laravel en VSC



Extensiones VSC

- Bootstrap v4 Snippets
- HTML Snippets
- IntelliSense for CSS class names in HTML
- Laravel Blade Snippets
- Laravel Snippets

- seti-icons
- Spanish Language Pack for Visual Studio Code
- Sublime Text Keymap and Settings Importer
- vscode-icons
- YAML
- Laravel Blade formatter

D. Instalar Composer

Ejecutamos la siguiente línea en la ventana de powerShell o CMD

```
composer global require laravel/installer
```

Comandos de CMD para desplazarnos entre carpetas (Terminal)

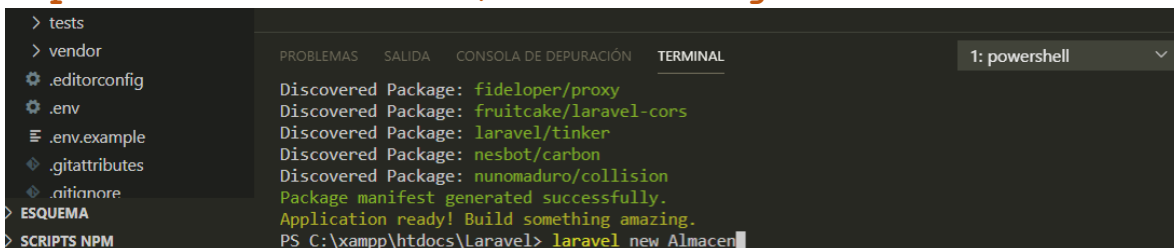
cd..	Permite salir de una carpeta o retroceder en el path (rutas)
Cd (nombre carpeta)	Permite ingresar a esa carpeta
Dir	Listar o mostrar las carpetas y archivos

E. Crear un Nuevo Proyecto

Para crear un nuevo proyecto desde Visual Studio Code>Menú principal >New Terminal
 Recuerda debes estar en la ruta C:\xampp\htdocs(forma simple)> **Laravel new Inventario**(nombre proyecto)←

Verificamos instalación del proyecto vamos al navegador>Localhost> Inventario/public←

También puedes ejecutar el comando (forma completa) **composer create-project --prefer-dist laravel/laravel blog**



Nota: Para trabajar sobre un proyecto previamente creado simplemente arrastramos la carpeta de proyecto hasta el área de contenido en MVC

Artisan es el nombre de la interfaz de línea de comandos incluida en Laravel (permite crear controladores – rutas – vistas)

Listar todos los comandos de artisan

```
php artisan list
```

Lanzar el servidor de Laravel a la web

Localhost> `php artisan serve`

Servidor: <http://localhost:8000>

<http://127.0.0.1:8000>

Cuando el navegador se encuentra lleno en los temporales usar

`php artisan view:cache`

Para cambiar el puerto donde se ejecuta el servidor de Laravel

`php artisan serve --port=####`

Nota: Puede crear un proyecto con una **versión** diferente con

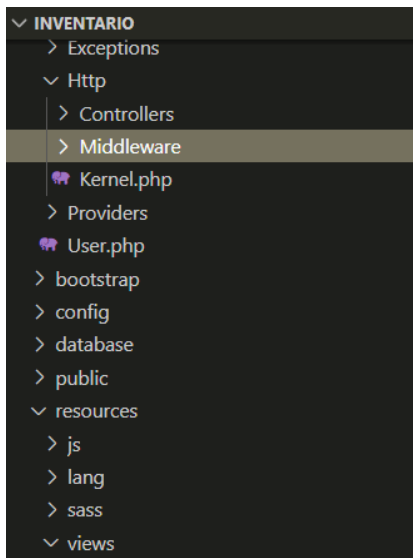
`composer create-project laravel/laravel="8.*" laravel8`

Nota: Puedes instalar **Laragon** (permite al estilo de XAMPP tener un servidor – un dominio virtual)

1. Practica. Manejo de controlador, métodos y rutas

2. Practica. Sistema de carpetas y archivos - Manejo de rutas, controladores y vistas

A. Sistema de archivos y carpetas manejado por Laravel

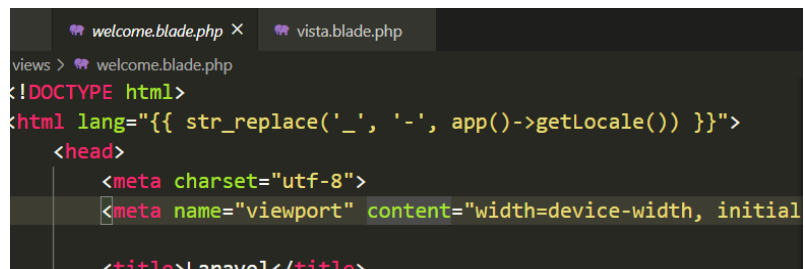


Views Carpetas con los archivos Html del sitio que se pueden visualizar por su **Route**(controlador). Utilizan la plantilla **Blade** que le permiten incorporar agilidad en la reutilización de código utiliza el **Framework Bootstrap** o diseño web con Html y CSS, conocidas para construir sitios web

Public: Carpeta de acceso y publicación de contenido a los usuarios. Te permite configurar las carpetas para imágenes, CSS, JS

Routes: Controlan todas las rutas de tu sitio web siendo **web.php** esta una de las más usadas (ruta que permite devolver a través de distintos métodos (GET, POST, DELETE, DESTROY) una vista o template como Index o Welcome).

Resources: Permite almacenar las distintas vistas de la aplicación, se encuentran dentro de la carpeta **Views/welcome.blade.php**



Migraciones y Modelos: Genera la construcción de los distintos elementos de BD almacenados en el proyecto.

Rutas

```
> views
└─ routes
  └─ api.php
  └─ channels.php
  └─ console.php
  └─ web.php
└─ storage
  └─ app
  └─ framework
```

```
24 Route::get('/categorias', 'App\Http\Controllers\PagesController@categorias');
25
26 //Route::get('/empleados', [EmpleadoController::class, 'create']);
27 //Route::get('/empleados', 'App\Http\Controllers\EmpleadoController@create');
28 //Route::get('/personas', 'App\Http\Controllers\PersonaController@create');
29
30 Route::resource('/empleados', 'App\Http\Controllers\EmpleadoController');
31 Route::resource('/personas', 'App\Http\Controllers\PersonaController');
32 Route::resource('/productos', 'App\Http\Controllers\ProductoController');
```

Controladores

Migraciones

```

15  ✓ PRACTICA1906984
16  > app
17  > bootstrap
18  > config
19  > database
20  > factories
21  > migrations
22  2014_10_12_000000_create_users...
23  2014_10_12_100000_create_pass...
24  2019_08_19_000000_create_failed...
25  2020_09_21_153425_create_categ...
26  2020_09_23_144106_create_empl...
27  2020_09_25_145627_create_perso...

Schema::create('personas', function (Blueprint $table) {
    //id, nombre, apellido, correo, celular, email, fecha_
    $table->id();
    $table->string('nombre', 30);
    $table->string('apellido', 40);
    $table->string('correo', 40);
    $table->string('celular');
    $table->string('email');
    $table->dateTime('fecha_reg');
    $table->timestamps();
});
```

Modelos

```

PRACTICA1906984
├── ProductoController.php M
│   ├── Middleware
│   ├── Kernel.php
│   └── Models
│       ├── Categoria.php
│       ├── Empleado.php
│       ├── Persona.php
│       ├── Producto.php
│       ├── User.php
│       └── Providers
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Empleado extends Model
9 {
10     use HasFactory;
11 }
12

```

Carpeta publica

```

> database
├── public
│   ├── .htaccess
│   ├── favicon.ico
│   └── index.php
├── robots.txt
├── resources
│   ├── css
│   ├── js
│   └── lang
47 $app = require_once __DIR__.'../bootstrap/app.php';
48
49 $kernel = $app->make(Kernel::class);
50
51 $response = tap($kernel->handle(
52     $request = Request::capture()
53 ))->send();
54
55 $kernel->terminate($request, $response);

```

Vistas o Template

```

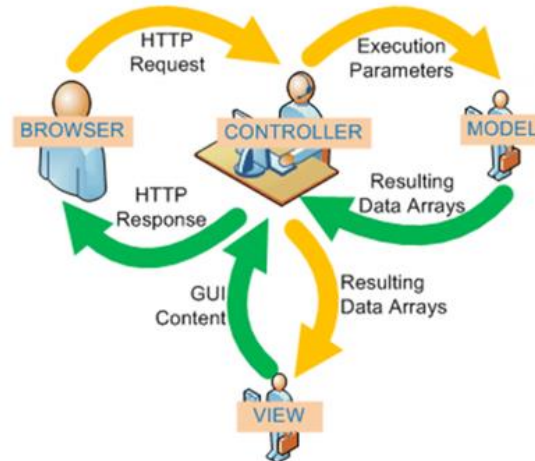
PRACTICA1906984
├── resources
│   ├── css
│   ├── js
│   ├── lang
│   └── views
│       ├── empleado
│       │   ├── create.blade.php M
│       │   ├── edit.blade.php
│       │   ├── form.blade.php
│       │   ├── index.blade.php M
│       ├── persona
│       └── producto
1 @extends('plantillaweb')
2
3 @section('secciondinamica')
4
5 <form action="{{url('/empleados')}}" method="POST" enctype="
6     @csrf
7     @error('nombre')
8     <div class="alert alert-danger alert-dismissible fade show
9         El nombre es requerido
10         <button type="button" class="close" data-dismiss="alert"
11             <span aria-hidden="true">&times;</span>
12         </button>

```

B. Manejo de controlador, métodos y rutas

Generalmente se encuentran en el archivo **web.php** donde podemos crear funciones, vistas, una cadena de texto para ser pasada a una vista o template. Los sitios web se basan en las rutas; para permitir dar un **orden** a los documentos del sitio y hacer que se cumpla el modelo MVC.

Aunque también las podemos encontrar en el archivo **api.php** donde podemos configurarla de acuerdo con nuestra necesidad las rutas de una API



Tipo de Ruta

Nombre ruta

Ruta del controlador y nombre

```
Route::get('articles', 'App\Http\Controllers\ArticleController@index');
```

Nombre del método llamado

Rutas

A partir de la ruta para la vista welcome podemos crear una nueva ruta

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

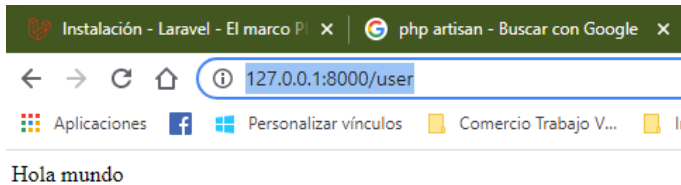
Crear una nueva ruta que retorna solo caracteres

```
Route::get('user/', function(){  
    return 'Hola mundo';  
});
```

Observe que return no tiene ningún método llamado

Mostramos la ruta en el navegador

`http://127.0.0.1:8000/user`



Si deseamos pasar un numero por la dirección Ej. User/50, ver un índice de user (1..100)

```
Route::get('user/{numero}', function ($numero) {
    return '<h2>Presentando a los usuarios:</h2> '.$numero;
});
```



Si no enviamos un parámetro para que muestre en la galería saldrá el mensaje página no encontrada. Podemos ubicar la siguiente línea, para que el parámetro sea optativo agregamos la ? en variable y el texto = 'No existe'

```
Route::get('user/{numero?}', function ($numero = 'no existe') {
    return '<h2>Presentando a los usuarios:</h2> '.$numero;
});
```

Parámetros opcionales

Si colocamos un nombre también lo muestra para que no pase esto, validamos con where (utilizando la documentación de Laravel). Copiamos el parámetro al finalizar el código, reemplazamos name, toma valores de 0-9 + infinitas veces. Así podemos validar lo que el usuario está pasando como información

```
->where('numero', '[0-9]+')
```



```
Route::get('user/{numero?}', function ($numero = 'no existe') {
    return '<h2>Presentando a los usuarios:</h2> '.$numero;
});
```

```
})->where('numero', '[0-9]+');
```

Métodos de enrutador disponibles

El enrutador le permite registrar rutas que responden a los métodos de envío de HTTP

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

Vistas

Podemos crear nuestra primera vista views/use-galery.php (crear el archivo en la carpeta) con código Html y un título que diga presentación de usuarios

```
<style>
h1{
    background-color: #fff;
    color: #636b6f;
    font-family: 'Nunito', sans-serif;
    font-weight: 200;
    height: 100vh;
    margin: 0;
}
</style>

<body>

<h3>Presentación de usuarios 2020</h3>

</body>
```

Debemos crear la respectiva ruta para visualizar vista. Dentro del paréntesis user-galery es el nombre de la vista, use-galery, es el archivo de la carpeta vista; no le colocamos .PHP

```
Route::view('user', 'user-galery');
```

Podemos retornar variables a través de la vista con un arreglo asociativo

```
Route::view('user', 'user-galery', ['numero'=>5000]);
```


NO lo enviamos por la URL el numero.

Definimos la variable en la vista (Una forma de trabajar directamente con PHP pero luego utilizaremos Blade)

```
<h3>Presentación de usuarios <?= $numero ?> </h3>
```

Si verificamos en la documentación de Laravel(<https://laravel.com/docs/8.x/controllers>). Un controlador es una clase de PHP heredada, que nos permite utilizar sus propiedades y métodos o funciones. Laravel utiliza la línea de comandos para generar esta clase.

Comando para crear un controlador básico

```
php artisan make:controller ClienteController
```

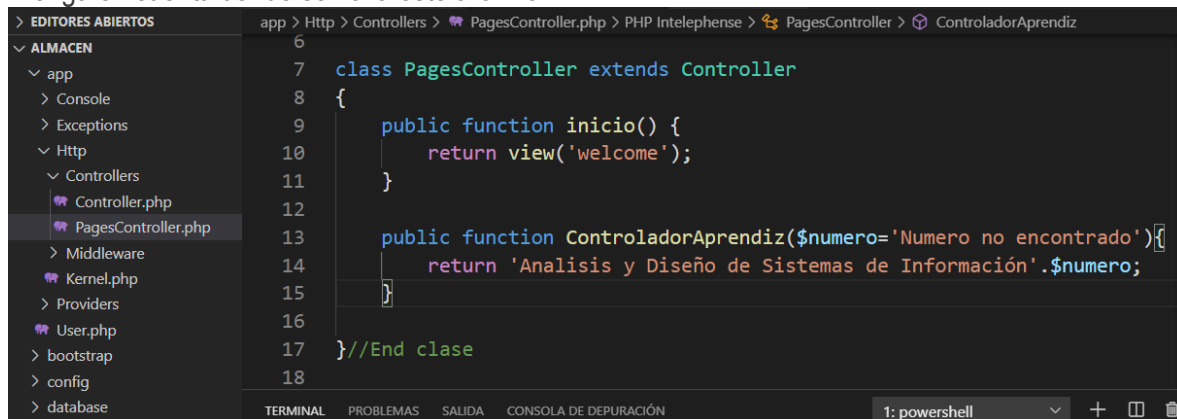
En el archivo **ClienteController.php** se definen los métodos quedando de esta manera

```
public function inicio()
{
    return view('producto');
}

public function saludo() {
    return 'Hello World';
}

} // End clase
```

Tenga en cuenta donde se halla este archivo



Controlador con manejo de parámetros

```
public function ContCategoria($dato=null){
```

```
$listaCat = ['Juan','Diego','Ana', 'Juliana', 'Jime
na', 'Luis'];
$j = 0;
while ($j<10){
    echo "<p>I'm looping forever.</p>";
    $j++;
}

return view('categoria', compact('listaCat', 'dato'
,'j'));
}
```

En el archivo de rutas web.php se debe hacer uso del archivo del controlador creado

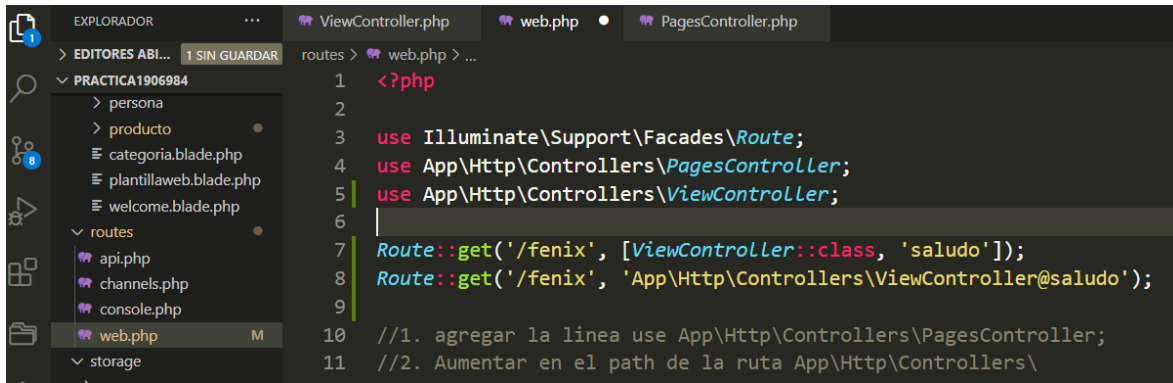
```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\PagesController;
use App\Http\Controllers\ViewController;
```

para crear la ruta de un controlador lo puedo hacer de **2 formas** donde **/producto** es el nombre de la ruta y **producto** es el nombre del método

```
Route::get('/producto', [ViewController::class, 'producto']
);

Route::get('/fenix', 'App\Http\Controllers\ViewController@saludo');
```



```

1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\PagesController;
5 use App\Http\Controllers\ViewController;
6
7 Route::get('/fenix', [ViewController::class, 'saludo']);
8 Route::get('/fenix', 'App\Http\Controllers\ViewController@saludo');
9
10 //1. agregar la linea use App\Http\Controllers\PagesController;
11 //2. Aumentar en el path de la ruta App\Http\Controllers\

```

3. Practica. Blade Templates (Vista View – Pagina Web con Bootstrap)

Utilizando el sistema de plantillas Blade. Sistema incorporado por Laravel para trabajar sus vistas (View). Para trabajar con Blade debemos ubicarlo al crear el archivo en el nombre ej.

`empleado.blade.php`

Podemos verificar que continúa trabajando con el script PHP, pero ahora utilizaremos la sintaxis de Blade

```
<h3>Presentación de usuarios {{$numero}} </h3>
```

Ahora trabajamos con la plantilla de **Bootstrap** – Traemos el código de su plantilla y lo pegamos en nuestro archivo `plantilla.blade.php`

Starter template

Be sure to have your pages set up with the latest design and development standards. That means using an HTML5 doctype and including a viewport meta tag for proper responsive behaviors. Put it all together and your pages should look like this:

```
<!doctype html>
<html lang="en">
<head>
```

Copy to clipboard

Copy

Para no repetir el código presente en la plantilla de Bootstrap trabajamos de la siguiente manera: Creamos un archivo nuevo `plantillaweb.blade.php` que hereda a las páginas `cliente.blade.php` y `producto.blade.php`

En la plantilla definimos una sección dinámica llamada `secciondinamica` dentro de `body`

```
<body>
  <div class="container">
    @yield('secciondinamica')
  </div>
```

La cual llamamos en los archivos

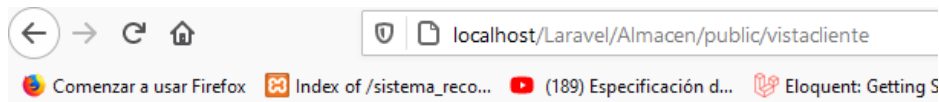
```
@extends('Plantillaweb')

@section('secciondinamica')
<h1>Entrada para Entrada de Clientes</h1>
@endsection
```

Creamos las rutas

```
Route::get('vistacliente', function(){
    return view('Cliente');
});
```

```
Route::get('vistaproducto', function(){
    return view('Producto');
});
```



Almacen Electro Lujo

Bienvenido Franco A Reina

Ahora vamos a agregar en **plantillaweb** dos botones para navegar entre archivos

```
<body>
  <a href="" class="btn btn-primary">Cliente</a>
  <a href="" class="btn btn-primary">Producto</a>
  <div class="container">
    @yield('secciondinamica')
  </div>
```

4. Practica. Estructuras de control (condiciones y ciclos dentro de Blade)

Blade es el motor de plantillas simple pero potente que se proporciona con Laravel. A diferencia de otros motores de plantillas PHP populares, Blade no le impide usar código PHP simple en sus vistas.

Dos de los principales beneficios de utilizar Blade son la herencia de **plantillas** y las **secciones**

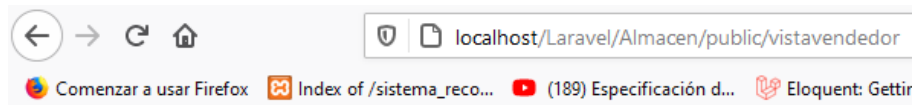
Creamos un nuevo archivo vendedor.blade.php, le heredamos de plantilla (sección), asignamos un nombre en route(vistaVen) y creamos un enlace en plantilla para su visualización. Crearemos un equipo de trabajo vendedores a través de un arreglo

Plantilla web.blade.php

```
<a href="{{route('vistaVen')}}" class="btn btn-
primary">Vendedor</a>
```

Web.php

```
Route::get('vistavendedor', function(){
    return view('Vendedor');
})->name('vistaVen');
```



Almacen Electro Lujo

Cliente

Producto

Vendedor

Vendedores Almerkar

Ahora creamos una variable para el arreglo, si estamos retornando una vista como hacemos para poder mostrar. Dentro de **View**, colocamos un array asociativo, el primer parámetro un nombre 'equipo' y la variable que queremos asociar.

```
Route::get('vistavendedor', function(){
    $equipo = ['Andrea', 'Juan Carlos', 'Alexa'];
    //para pasar el valor lo agregamos a view
    return view('Vendedor', ['equipo'=>$equipo]);
})->name('vistaVen');
```

Utilizamos una **estructura de control foreach** en el archivo vendedor.blade.php y colocamos la variable de arreglo \$equipo hasta \$v_nombre cada nombre del vendedor del arreglo (pueden cambiar el nombre de la variable)

```
@foreach($equipo as $v_nom)
```

```
<a href="#" class="h4 text-info">{{$v_nom}}</a><br>
@endforeach
```

Tipos de fuente de Bootstrap

<https://imacreste.com/bootstrap-colores-fuentes-y-tipografias/#Small>

Podemos utilizar compact para el paso de la variable tipo arreglo

```
return view('vendedor', compact('equipo'));
```

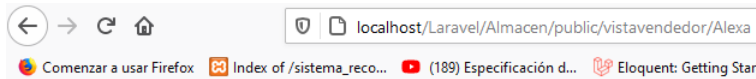
Si deseamos pasar un parámetro en el URL(variable) ej. Clic en Andrea muestre los datos del vendedor aumentamos en la función

```
Route::get('vistavendedor/{dato?}', function($dato=NULL){
//Route::get('vistavendedor', function(){
    $equipo = ['Andrea', 'Juan Carlos', 'Alexa'];
    return view('vendedor', compact('equipo', 'dato'));
})->name('vistaVen');
```

En la vista archivo Vendedor.blade.php creamos la estructura condicional

```
<? //5.3 creamos un if para validar datos de variable
    //si pasamos datos por ella indique cuando existen o no
    //en este caso el vendedor debe existir
?>

@if(!empty($dato))
    <p> La variable se encuentra</p>
@endif
```



Almacen Electro Lujo

[Cliente](#)
[Producto](#)
[Vendedor](#)

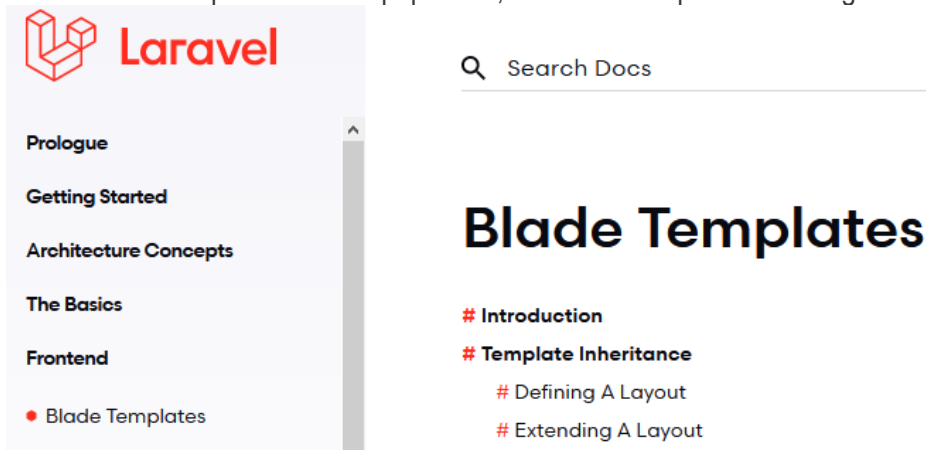
Vendedores Almerkar

[Andrea](#)
[Juan Carlos](#)
[Alexa](#)

El vendedor existe

Para trabajar más estructuras de control en Laravel en línea (<https://laravel.com/docs/8.x/blade>)

Blade es el motor de plantillas simple pero potente que se proporciona con Laravel. A diferencia de otros motores de plantillas PHP populares, Blade no le impide usar código PHP simple en sus vistas.



Ejercicio. Agregar una estructura de control que permita listar el numero de usuarios digitados por la ruta dirección con un ciclo for o while

```
@for ($i = 1; $i <= $id; $i++)
    usuario No. {{ $i }} <br>
@endfor
<br>
$j = 0;
@while ($j<10)
    <p>I'm looping forever.</p>
    $j++;
@endwhile
```



Bibliografía

<https://laravel.com/docs/8.x>

<https://www.youtube.com/watch?v=KKpXpWCTIbo&list=PLPI81lqbj-4KHPEGngoy5PSjjxcwnpCdb>

<https://aprendible.com/series/laravel-desde-cero>

<https://imacreste.com/bootstrap-colores-fuentes-y-tipografias/#Small>

<https://bluuweb.github.io/tutorial-laravel/bases-datos/>

http://www.oscarabadfolgueira.com/crear-una-base-datos-mysql-desde-consola/#Acceder_a_mysql_desde_la_consola

BEATI Hernán (2015). PHP - Creación de páginas Web dinámicas 2a edición. Editorial Alfaomega. 2015

W3C The World Wide Web Consortium (2016). HTML 5.1 W3C Recommendation.

Recuperado de: <https://www.w3.org/TR/html>

W3C The World Wide Web Consortium (2017). CSS Snapshot 2017. Recuperado de:

<https://www.w3.org/TR/CSS>

<https://www.cursosdesarrolloweb.es/cursos-gratuitos/>

CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor(es)	Franco Reina Argoty	Instructor	CTPI	Junio 2021