



Análisis y desarrollo de software.

ID: 2644590



www.sena.edu.co

@SENAComunica

Instructor

Diego Fernando Calderón Silva
Correo: dfcalderon@sena.edu.co

¿Qué necesito para crear mi primer hola mundo?

- Al ser un framework, se necesitan ciertas dependencias tales como:
- Composer
- XAMPP / Laragon
- Docker

¿Qué necesito para crear mi primer hola mundo?

- Vamos a instalar inicialmente composer:
 1. Nos dirigimos a la página oficial de composer
 2. <https://getcomposer.org>
 3. Pinchamos en descargar
 4. Se ejecuta el archivo descargado



A Dependency Manager for PHP

Latest: 2.6.3 (changelog)



¿Qué necesito para crear mi primer hola mundo?

5. Una vez instalado nos dirigimos a la terminal o CMD y escribimos “composer –v” para analizar la versión instalada

¿Qué necesito para crear mi primer hola mundo?

6. Vamos a acceder a la carpeta “www” si estamos usando Laragon o “htdocs” si estamos usando XAMPP
7. Para acceder debemos usar el comando “cd /ruta/”
8. Dentro de la misma CMD o terminal vamos a crear el primer proyecto con el comando:

“ composer create-project laravel/laravel
Nombre_del_proyecto ”

```
ingdiegoc@MacBook-Pro-de-Diego htdocs % composer create-project laravel/laravel primerHolamundo
Creating a "laravel/laravel" project at "./primerHolamundo"
Info from https://repo.packagist.org: #StandWithUkraine
Installing laravel/laravel (v10.2.6)
  - Installing laravel/laravel (v10.2.6): Extracting archive
Created project in /Applications/XAMPP/xamppfiles/htdocs/primerHolamundo
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies
Lock file operations: 110 installs, 0 updates, 0 removals
  - Locking brick/math (0.11.0)
  - Locking dflydev/dot-access-data (v3.0.2)
  - Locking doctrine/inflector (2.0.8)
  - Locking doctrine/lexer (3.0.0)
  - Locking dragonmantank/cron-expression (v3.3.3)
  - Locking egulias/email-validator (4.0.1)
  - Locking fakerphp/faker (v1.23.0)
  - Locking filp/whoops (2.15.3)
  - Locking fruitcake/php-cors (v1.2.0)
  - Locking graham-campbell/result-type (v1.1.1)
  - Locking guzzlehttp/guzzle (7.8.0)
  - Locking guzzlehttp/promises (2.0.1)
  - Locking guzzlehttp/psr7 (2.6.1)
  - Locking guzzlehttp/uri-template (v1.0.2)
  - Locking hamcrest/hamcrest-php (v2.0.1)
  - Locking laravel/framework (v10.24.0)
  - Locking laravel/pint (v1.13.2)
  - Locking laravel/prompts (v0.1.8)
  - Locking laravel/sail (v1.25.0)
  - Locking laravel/sanctum (v3.3.1)
```

¿Qué necesito para crear mi primer hola mundo?

9. Una vez instalado el proyecto en nuestra carpeta local del servidor, iremos al archivo .env que es donde gestionamos nuestra conexión a la base de datos
10. Configuramos la BD y procedemos a realizar nuestra primera migración para emparejar la base de datos de nuestro gestor de datos con nuestro primer hola mundo

```
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=laravel
15 DB_USERNAME=root
16 DB_PASSWORD=
```

¿Qué necesito para crear mi primer hola mundo?

11. Para realizar la primera migración usamos:
“php artisan migrate”
y nuestras bases de datos cargaran las tablas que traen por defecto. Si queremos revertir esos cambios,
usamos:
“php artisan migrate:rollback”

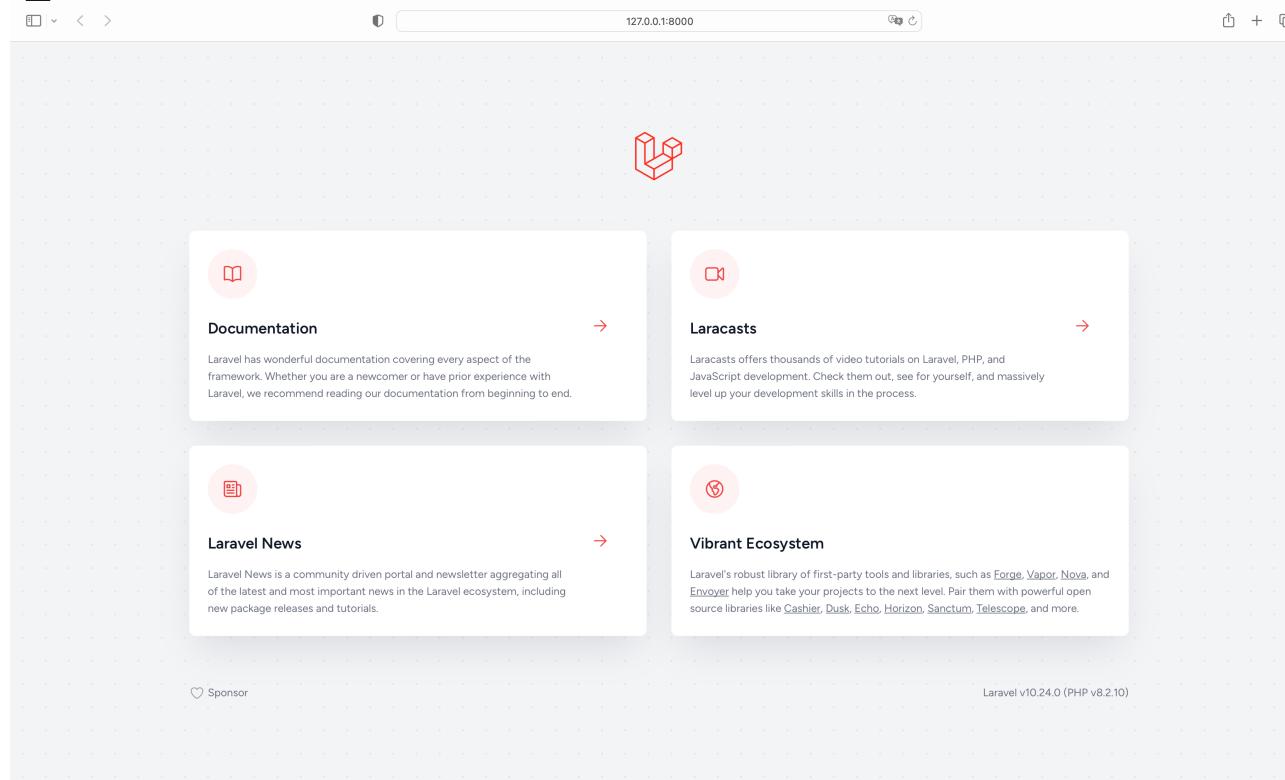
¿Qué necesito para crear mi primer hola mundo?

12. Vamos a iniciar la primera vista de laravel con el comando:
“php artisan serve”

Pero antes de escribir el comando debemos ingresar mediante “cd” al proyecto creado.

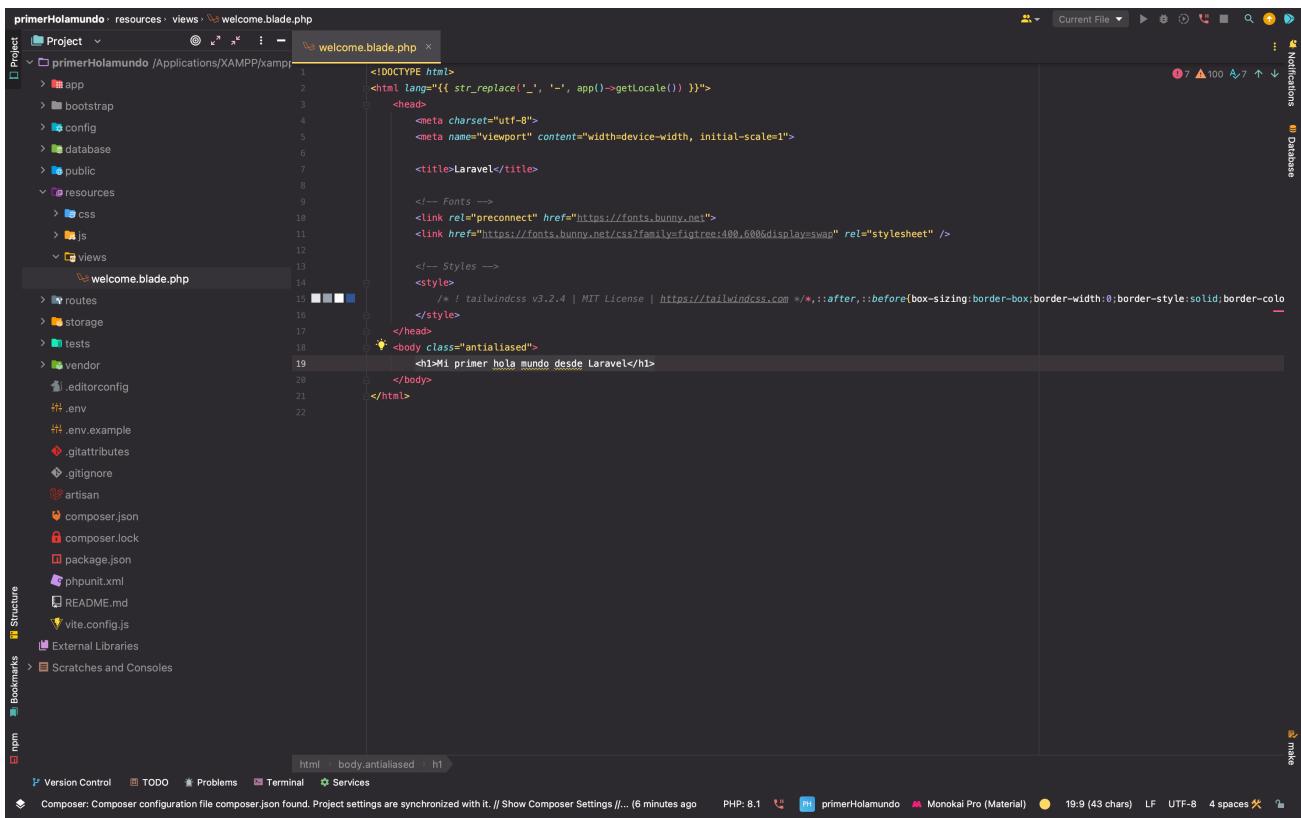
```
[ingdiegoc@MacBook-Pro-de-Diego htdocs % cd primerHolamundo
[ingdiegoc@MacBook-Pro-de-Diego primerHolamundo % php artisan serve
INFO Server running on [http://127.0.0.1:8000].
Press Ctrl+C to stop the server
```

¿Qué necesito para crear mi primer hola mundo?



¿Qué necesito para crear mi primer hola mundo?

13. Para empezar a escribir código y visualizar el primer hola mundo vamos al archivo resources/views/principal.blade.php y borramos todo lo que está dentro del body para reemplazar por:
<h1>Mi primer hola mundo desde Laravel</h1>



```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Laravel</title>
    <!-- Fonts -->
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link href="https://fonts.googleapis.com/css?family=figtree:400,600&display=swap" rel="stylesheet" />
    <!-- Styles -->
    <style>
        /* tailwindcss v3.2.4 | MIT License | https://tailwindcss.com */::after,::before{box-sizing:border-box; border-width:0; border-style:solid; border-color:#000;}</style>
</head>
<body class="antialiased">
    <h1>Mi primer hola mundo desde Laravel</h1>
</body>
</html>
```

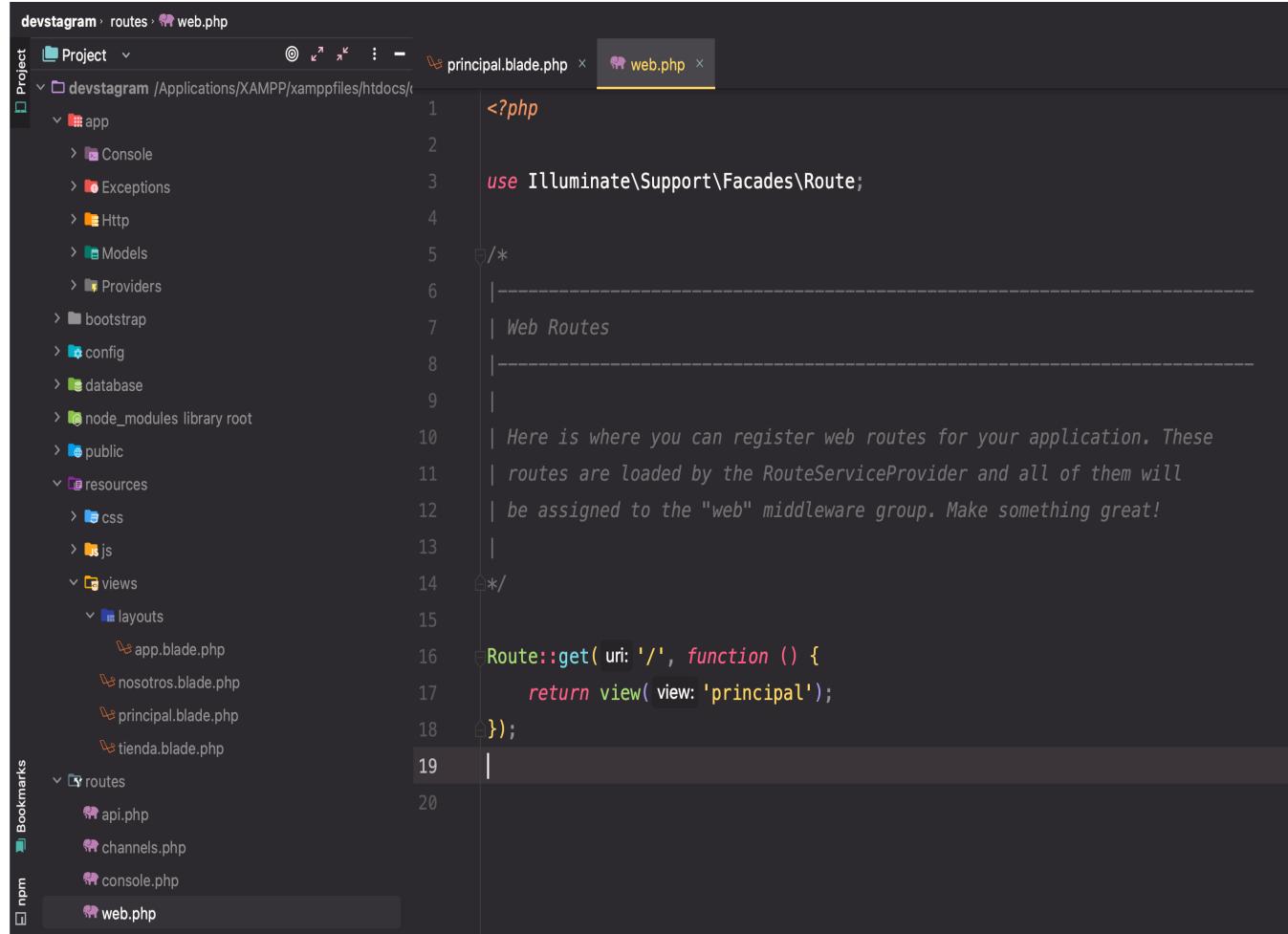
¡¡Recargamos la página del navegador y LISTO!!!
Hemos creado nuestro primer hola mundo desde laravel

Routing

La forma en que laravel maneja las rutas es mediante los archivos con extensión .Blade ubicados en resources/views.

Blade es un template engine por defecto que viene en laravel y como tal tiene su propia sintaxis, sin decir con esto que no se puedan usar sentencias básicas como el foreach, if, while...

Luego de crear mi primer hola mundo, se ve en pantalla precisamente “Hola mundo” y si vamos a la carpeta routes al archivo “ web.php ” encontraremos que la ruta nos lleva a la vista de bienvenida, en donde miraremos la sintaxis básica para visitar la url.



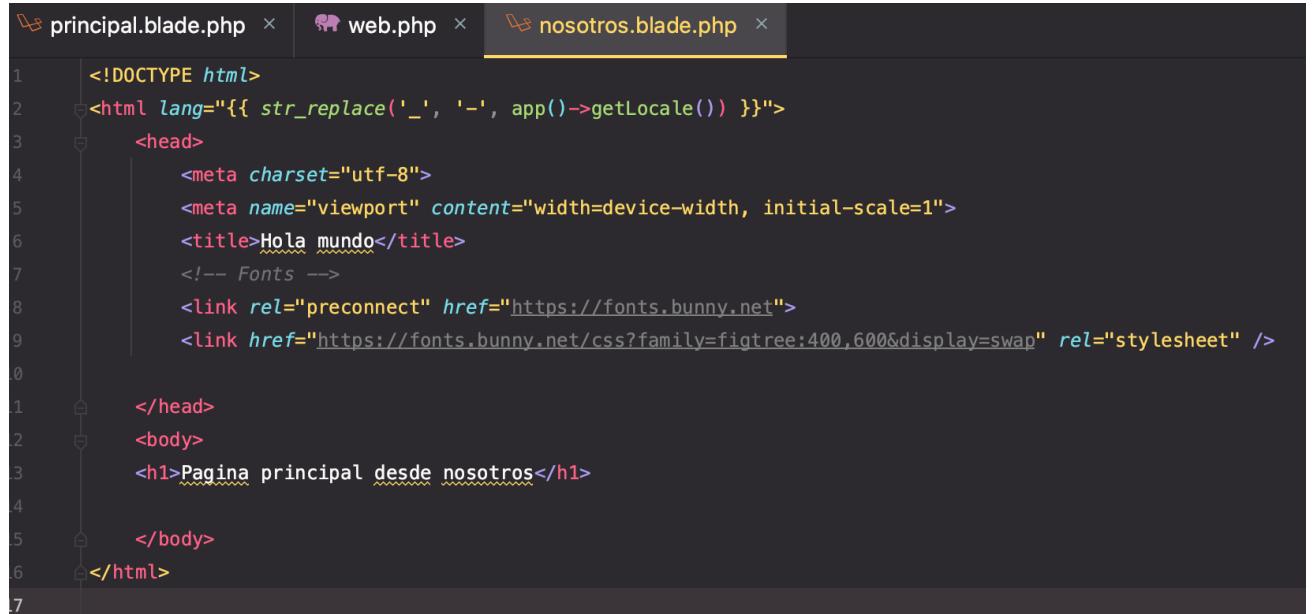
```
<?php  
use Illuminate\Support\Facades\Route;  
  
/*  
| Web Routes  
|  
| Here is where you can register web routes for your application. These  
| routes are loaded by the RouteServiceProvider and all of them will  
| be assigned to the "web" middleware group. Make something great!  
*/  
  
Route::get('/', function () {  
    return view('principal');  
});  
Route::get('/tienda', function () {  
    return view('tienda');  
});  
Route::get('/nosotros', function () {  
    return view('nosotros');  
});
```

Routing

Ahora vamos a crear una nueva vista llamada “nosotros.blade.php”; pero antes de esto, debemos crear la ruta. Para crear la ruta, en el archivo web.php copiamos de principal y pegamos con el nombre de nuestra nueva vista, es decir, nosotros.

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

Eso en la carpeta llamada routes, ahora, en la carpeta resources/views vamos a crear un nuevo archivo llamado “nosotros.blade.php” y copiamos el contenido del html de la página principal



```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale()) }}">
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Hola mundo</title>
        <!-- Fonts -->
        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link href="https://fonts.googleapis.com/css?family=Figtree:400,600&display=swap" rel="stylesheet" />
    </head>
    <body>
        <h1>Pagina principal desde nosotros</h1>
    </body>
</html>
```

Routing

Una vez creada la ruta y la vista, laravel ya entenderá que hay algo nuevo, por lo que no es necesario hacer más cambios, y para acceder a nuestra nueva ruta, debemos indicarle a la url de nuestro navegador que nos lleve a:

<http://127.0.0.1:8000/nosotros>



Hacer lo mismo para:

- Tienda

Página principal desde nosotros

Routing

Una vez creadas las vistas y para fines demostrativos vamos a crear un menú de navegación básico, sin estilos, ni nada relacionado para evidencias que podemos viajar entre las diferentes rutas. Para esto creamos un `<nav>` dentro de cada una de las vistas y le enseñamos la ruta así:

Dentro del href, es donde realmente le indicamos que ruta debe seguir.

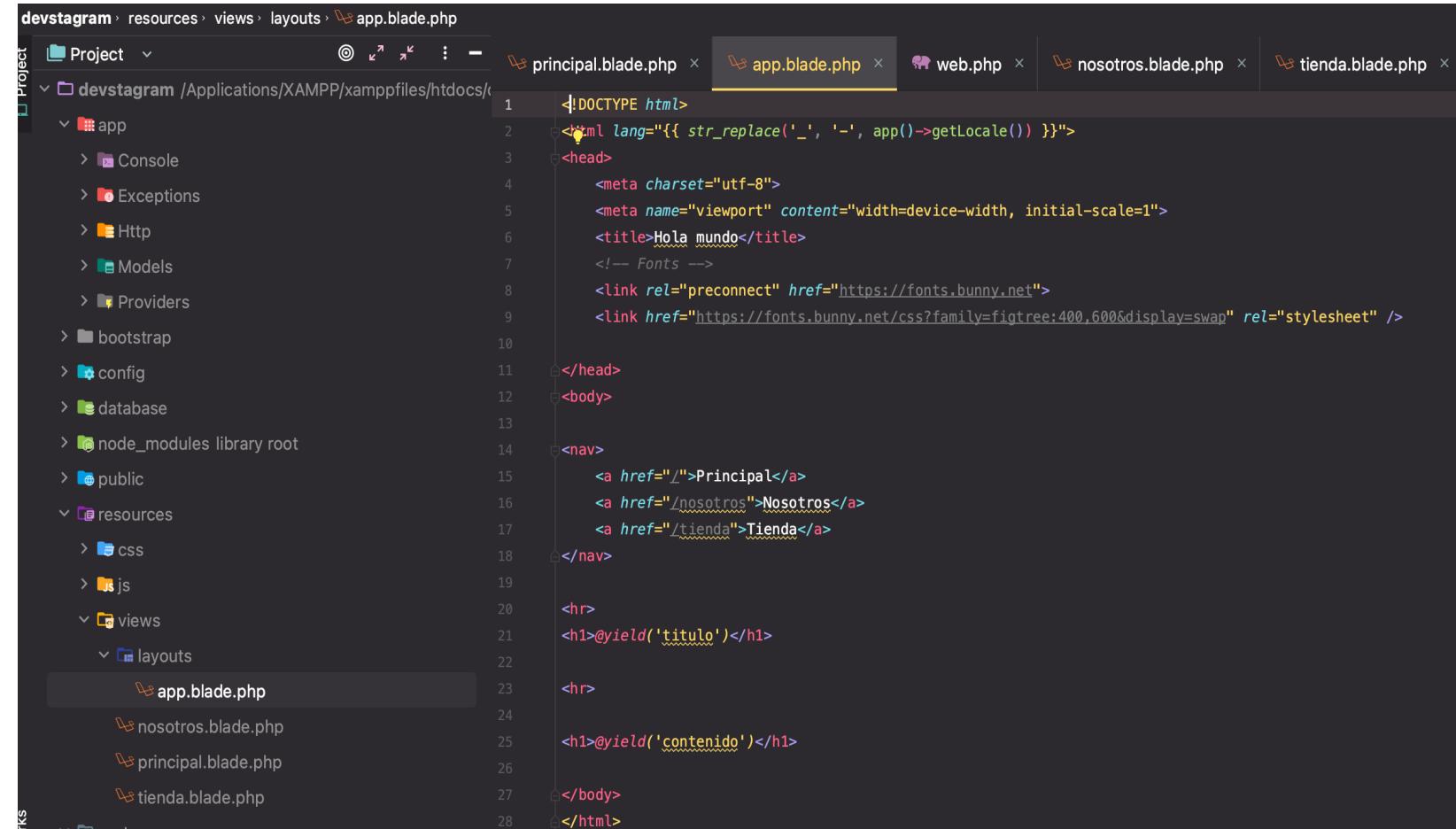
Se repite este proceso en cada uno de los scripts.

```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()>getLocale()) }}>
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>Hola mundo</title>
        <!-- Fonts -->
        <link rel="preconnect" href="https://fonts.bunny.net">
        <link href="https://fonts.bunny.net/css?family=figtree:400,600&display=swap" rel="stylesheet" />

    </head>
    <body>
        <h1>Pagina principal</h1>
        <nav>
            <a href="/">Principal</a>
            <a href="/nosotros">Nosotros</a>
            <a href="/tienda">Tienda</a>
        </nav>
    </body>
</html>
```

Layout

Si nos fijamos hasta ahora, se repite bastante código; hemos tenido que escribir el nav exactamente igual en cada una de las vistas y esto incumple desde todas las perspectivas los principios de la programación, es por esto que debemos crear un archivo global que contenga la información que se hereda en todas las vistas. Para lograr esto vamos a crear un template principal. Para esto debemos crear una carpeta layouts dentro de viwes y dentro de layout vamos a crear un script llamado “app.blade.php” y voy a copiar y pegar todo el contenido de principal.blade.php y a pegar dentro de app.



```
devstagram · resources · views · layouts > app.blade.php
Project devstagram /Applications/XAMPP/xamppfiles/htdocs/
  > app
    > Console
    > Exceptions
    > Http
    > Models
    > Providers
  > bootstrap
  > config
  > database
  > node_modules library root
  > public
  > resources
    > css
    > js
    > views
      > layouts
        > app.blade.php
        > nosotros.blade.php
        > principal.blade.php
        > tienda.blade.php
  > routes

principal.blade.php x app.blade.php x web.php x nosotros.blade.php x tienda.blade.php x

1 <!DOCTYPE html>
2 <html lang="{{ str_replace('_', '-', app()>getLocale()) }}>
3 <head>
4   <meta charset="utf-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1">
6   <title>Hola mundo</title>
7   <!-- Fonts -->
8   <link rel="preconnect" href="https://fonts.bunny.net">
9   <link href="https://fonts.bunny.net/css?family=figtree:400,600&display=swap" rel="stylesheet" />
10 </head>
11 <body>
12
13 <nav>
14   <a href="/">Principal</a>
15   <a href="/nosotros">Nosotros</a>
16   <a href="/tienda">Tienda</a>
17 </nav>
18
19 <hr>
20 <h1>@yield('titulo')</h1>
21
22 <hr>
23
24 <h1>@yield('contenido')</h1>
25
26
27 </body>
28 </html>
```

Layout

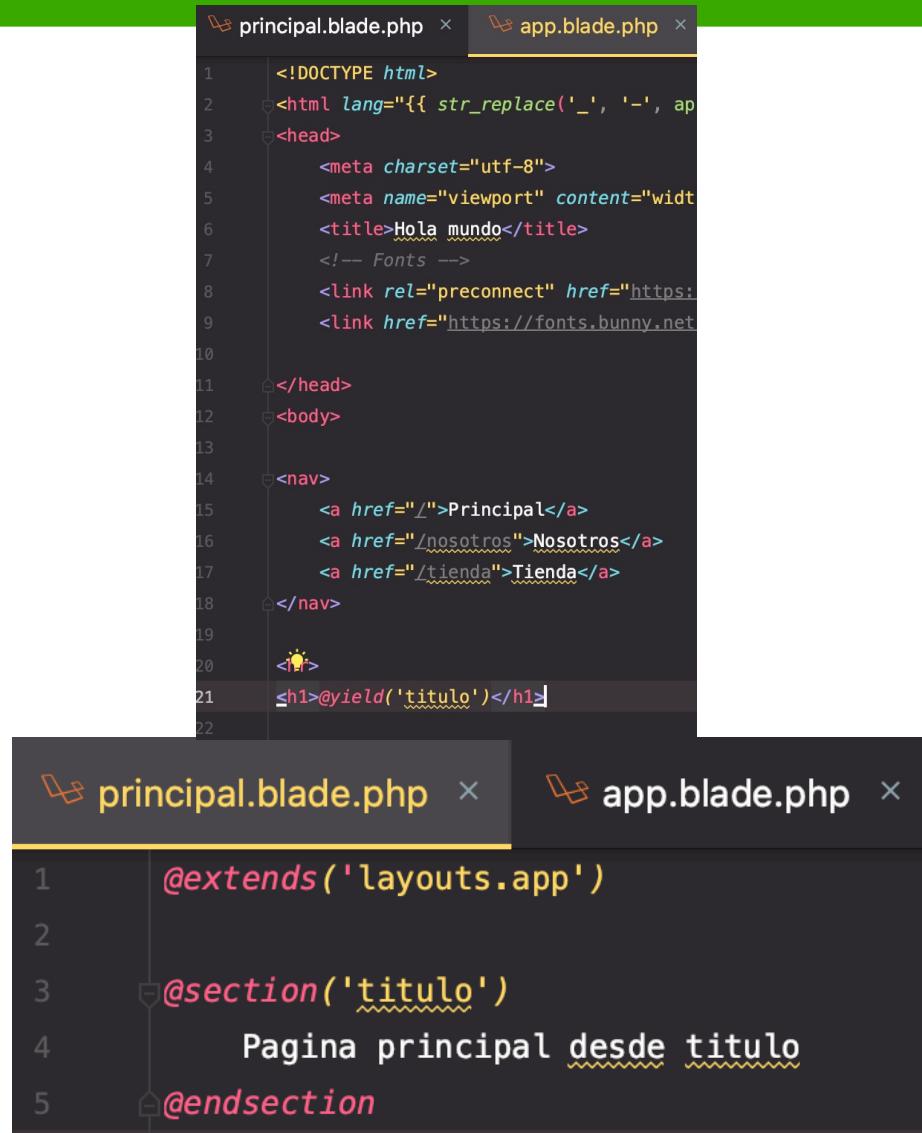
Con esto se busca eliminar la duplicidad de código, pero para que el sistema entienda a que vista se está refiriendo, dentro de app se debe escribir el contenedor `@yield` cada vez que necesitemos mostrar una sección específica, por ejemplo, dentro de app escribimos `<h1>@yield('titulo')</h1>` y para lograr mostrar algo desde la sección título, iremos a nuestra página principal y escribiremos las directivas en el archivo que debe estar completamente vacío:

```
@extends('layouts.app')
```

```
@section('titulo')
```

Página principal desde título

```
@endsection
```



```
principal.blade.php x app.blade.php x
1  <!DOCTYPE html>
2  <html lang="{{ str_replace('_', '-', app()->getLocale()) }}>
3  <head>
4      <meta charset="utf-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1">
6      <title>Hola mundo</title>
7      <!-- Fonts -->
8      <link rel="preconnect" href="https://fonts.googleapis.com">
9      <link href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;600;700&display=swap" rel="stylesheet">
10
11 </head>
12 <body>
13
14 <nav>
15     <a href="/">Principal</a>
16     <a href="/nosotros">Nosotros</a>
17     <a href="/tienda">Tienda</a>
18 </nav>
19
20 <h1>@yield('titulo')</h1>
21
22
```



```
principal.blade.php x app.blade.php x
1 @extends('layouts.app')
2
3 @section('titulo')
4     Página principal desde título
5 @endsection
```

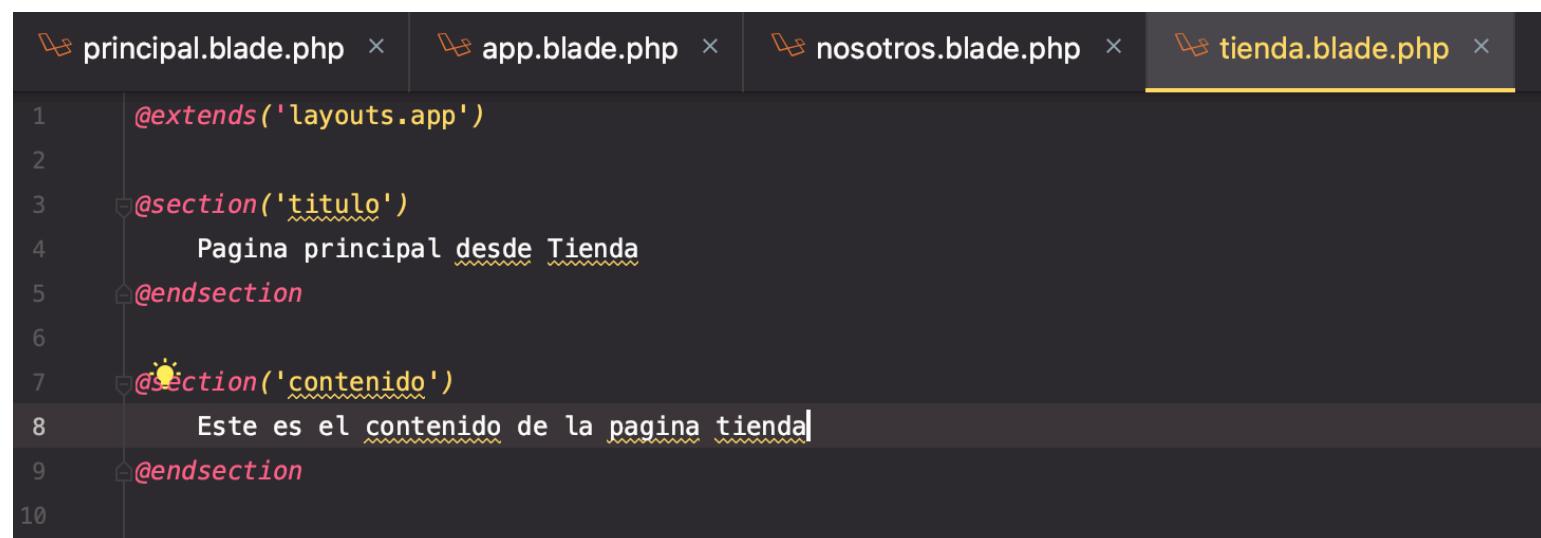
Layout

Dentro de cada vista podemos crear cuantas secciones necesitemos, lo único que debemos hacer es usarlas usando el contenedor en app. Vamos a repetir este proceso en las 3 vistas creadas, principal, nosotros y tienda. Mientras que, en app, crearemos un contenedor nuevo que se direccione a una sección llamada contenido:

```
<hr>
<h1>@yield('titulo')</h1>

<hr>

<h1>@yield('contenido')</h1>
```



```
principal.blade.php x app.blade.php x nosotros.blade.php x tienda.blade.php x
1 @extends('layouts.app')
2
3 @section('titulo')
4     Pagina principal desde Tienda
5 @endsection
6
7 @section('contenido')
8     Este es el contenido de la pagina tienda
9 @endsection
10
```

Layout

Cree una vista nueva llamada “Contacto”



Instalando Tailwind CSS con Vite



Para esto debemos verificar la versión de vite; una vez verificada, debemos confirmar que el archivo vite.config.js exista. Este archivo no existía en versiones anteriores a la 9 así que, de no existir, elimine el proyecto y vuélvalo a crear con una versión ≥ 9 .

Una vez realizada esta verificación en nuestra terminal y estando dentro de la carpeta del proyecto; ejecutaremos el comando:

```
npm install -D tailwindcss postcss autoprefixer
```

Si no se crea el archivo tailwind.config.js en la raíz del proyecto usa: npx tailwindcss init . Luego de instaladas las dependencias debemos agregar `@vite('resources/css/app.css')` En la cabecera del archivo app.blade.php para después lanzar el siguiente comando en una ventana nueva de nuestra terminal:

```
npm run dev
```

Y vamos a actualizar nuestra página, si no vemos ningún error es porque estamos listos para usar TailwindCss con vite

Nota: si no toma las clases, seguir este link: <https://tailwindcss.com/docs/guides/laravel>

Creando navegación y estilos para el layout principal



Vamos a crear la navegación inicialmente de la siguiente forma:

1. Creamos el logo principal de devstagram en app.Blade.php



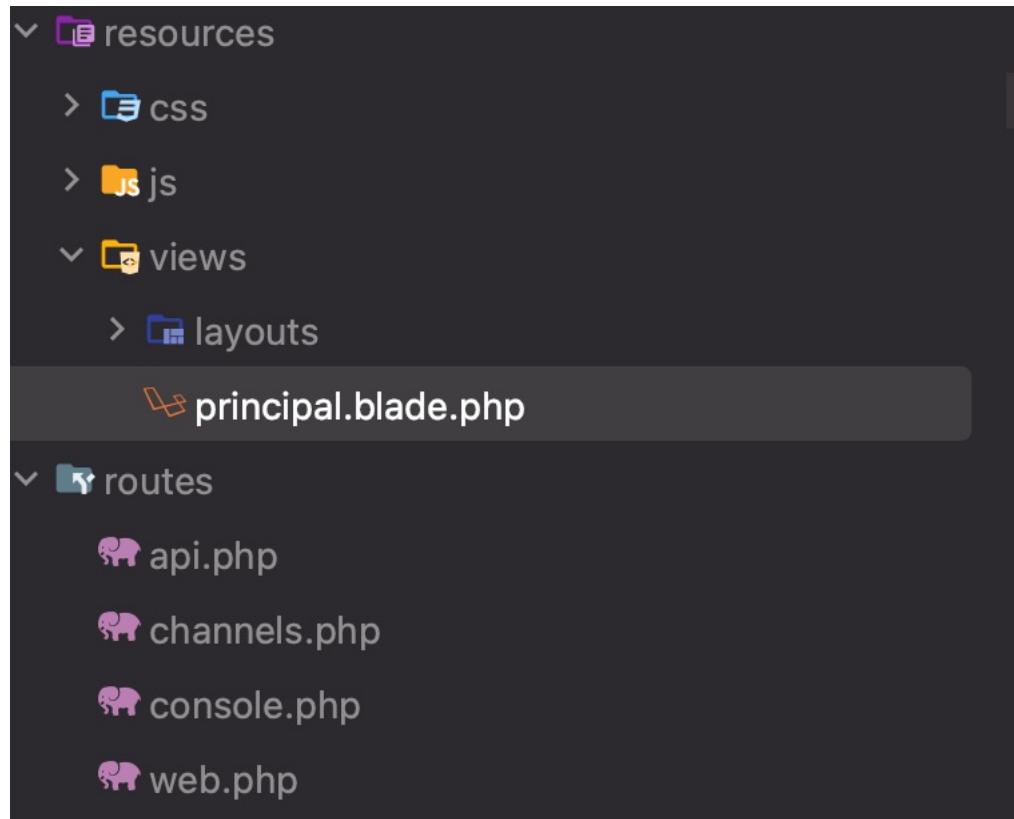
Devstagram

```
app.blade.php x vite.config.js x tailwind.config.js x app.css x
1 <!doctype html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     @vite('resources/css/app.css')
7   </head>
8   <body class="bg-gray-100">
9     {{--<h1 class="text-3xl font-bold underline">
10       Hello world!
11     </h1>--}}
12     <header class="p-5 border-b bg-white shadow">
13       <h1 class="text-3xl font-black">
14         Devstagram
15       </h1>
16     </header>
17
18
19   </body>
20 </html>
```

Creando navegación y estilos para el layout principal



Seguidamente vamos a eliminar todas las vistas y vamos a dejar únicamente principal.



Creando navegación y estilos para el layout principal



A continuación, debajo del header en la app; se creará un main el cual irá añadiendo dinámicamente la información de las vistas y finalmente un footer.

```
<header class="p-5 border-b bg-white shadow">
  <h1 class="text-3xl font-black">
    Devstagram
  </h1>
</header>
<main>
  
</main>
<footer>
</footer>
```

Creando navegación y estilos para el layout principal



Dando forma al contenido se crea lo siguiente

```
<header class="p-5 border-b bg-white shadow">
  <h1 class="text-3xl font-black">
    Devstagram
  </h1>
</header>
<main class="container mx-auto mt-10">
  <h2 class="font-black text-center text-3xl mb-10">
    @yield('titulo')
  </h2>
  @yield('contenido')
</main>
<footer class="text-center p-5 text-gray-500 font-bold uppercase">
  Devstagram – Todos los derechos reservados
</footer>
```

Creando navegación y estilos para el layout principal



Dando forma al contenido se crea lo siguiente

```
<header class="p-5 border-b bg-white shadow">
  <h1 class="text-3xl font-black">
    Devstagram
  </h1>
</header>
<main class="container mx-auto mt-10">
  <h2 class="font-black text-center text-3xl mb-10">
    @yield('titulo')
  </h2>
  @yield('contenido')
</main>
<footer class="text-center p-5 text-gray-500 font-bold uppercase">
  Devstagram – Todos los derechos reservados
</footer>
```

Creando navegación y estilos para el layout principal



Como es conocido por todos, hasta el momento para incluir código php dentro de un HTML debemos usar las etiquetas <?php echo date(Y);?> por ejemplo, pero de ahora en adelante al estar usando Blade, se reemplazaran esas etiquetas por {{ }} incluyendo el código php dentro de ellas así:

```
<header class="p-5 border-b bg-white shadow">
    <h1 class="text-3xl font-black">
        Devstagram
    </h1>
</header>
<main class="container mx-auto mt-10">
    <h2 class="font-black text-center text-3xl mb-10">
        @yield('titulo')
    </h2>
    @yield('contenido')
</main>
<footer class="text-center p-5 text-gray-500 font-bold uppercase">
    Devstagram - Todos los derechos reservados {{date("Y")}}
</footer>
</body>
```

Creando navegación y estilos para el layout principal



Esa es una forma de usar el código, otra forma es mediante las etiquetas `@php código php @endphp` y se ve de la siguiente manera:

```
<footer class="text-center p-5 text-gray-500 font-bold uppercase">
    Devstagram – Todos los derechos reservados @php echo date("Y") @endphp
</footer>
```

¿Qué son helpers en laravel?

Creando navegación y estilos para el layout principal



Cree un directorio dentro de views llamado auth y dentro de este directorio un archivo llamado: register.blade.php Una vez creado, debe crear un menú de navegación en la app.blade.php que se vea de la siguiente manera

A screenshot of a web browser window showing the Devstagram homepage. The address bar says "127.0.0.1:8000". The page title is "Devstagram". On the right, there are "LOGIN" and "CREAR CUENTA" buttons. The main content area has a heading "Pagina principal" and the text "Este es el contenido de la pagina principal". At the bottom, it says "DEVSTAGRAM - TODOS LOS DERECHOS RESERVADOS 2023-09-29 01:47:42".

Finalmente, debe darle funcionalidad a "CREAR CUENTA" y se debe dirigir a la nueva vista register

A screenshot of a web browser window showing the "CREAR CUENTA" registration page. The address bar says "127.0.0.1:8000/crear-cuenta". The page title is "Devstagram". On the right, there are "LOGIN" and "CREAR CUENTA" buttons. The main content area has a heading "Registrate en devstagram". At the bottom, it says "DEVSTAGRAM - TODOS LOS DERECHOS RESERVADOS 2023-09-29 01:51:27".

¿Cómo crear un controlador?



Para crear un controlador en Laravel se usa el comando:

```
php artisan make:controller RegisterController
```

Debe estar seguro antes de enviar este comando que está dentro de la carpeta de su proyecto

Una vez se ejecuta el comando se revisa la ruta :

/devstagram/app/Http/Controllers

Ahí debe estar creado nuestro nuevo controlador

A screenshot of a code editor window titled "devstagram" showing the file "RegisterController.php" in the "app/Http/Controllers" directory. The code editor has a dark theme. The file content is as follows:

```
<?php  
namespace App\Http\Controllers;  
use Illuminate\Http\Request;  
  
class RegisterController extends Controller  
{  
    //  
}
```

The "RegistersController.php" file is highlighted in the file tree on the left, and the "app.blade.php" file is also visible in the tabs bar.

¿Cómo crear un controlador?



Y modificamos el archivo web.php cortando la función de registrar y pegándola dentro del controlador así:

```
Route::get('/', function () {
    return view('principal');
});
Route::get('/crear-cuenta');
```

web.php

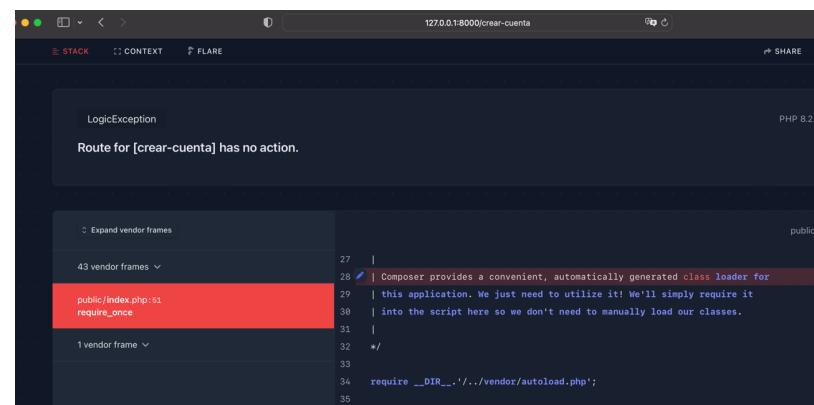
```
class RegisterController extends Controller
{
    no usages
    public function index() {
        return view('auth.register');
    }
}
```

RegisterController.php

Hasta el momento debe ir así nuestro código, pero aún no debe funcionar y esto lo verificamos en el navegador.

No funciona debido a que la ruta está esperando su extensión.

Así que debemos enseñarle así que para empezar entre corchetes pondremos el nombre del controlador así:



¿Cómo crear un controlador?



Debe tener en cuenta algo. Para que funcione el controlador debe importar el controlador dentro de web.php, para eso en la parte alta del código puede ingresar:

```
use App\Http\Controllers\RegisterController;
```

Seguidamente, nos dirigimos a la ruta de crear-cuenta y dentro de los corchetes escribimos

```
[RegisterController::class, 'index']
```

Donde se evidencia que le ingresamos el nombre del controlador seguido de ::class y finalmente el método que vamos a buscar, para este caso vamos a cambiar index por crear tanto en web como en RegisterControllers

```
Route::get('uri: '/crear-cuenta', [RegisterController::class, 'crear']);
```

web.php

```
class RegisterController extends Controller
{
    public function crear()
    {
        return view( view: 'auth.register' );
    }
}
```

RegisterController.php



G R A C I A S

Línea de atención al ciudadano: 01 8000 910270
Línea de atención al empresario: 01 8000 910682



www.sena.edu.co