

## instalar Laravel 8 en Windows 10 XAMPP

Laravel es un marco de aplicación web basado en PHP, proporciona herramientas para construir aplicaciones potentes y robustas, es un marco de código abierto, que proporciona una estructura que ahorra mucho tiempo para construir y planificar aplicaciones grandes. Es una de las plataformas más seguras que utiliza una base PHP. Proporciona funciones integradas para la autorización del usuario, como inicio de sesión, registro y contraseña olvidada.

### 1. Instalar el Xampp

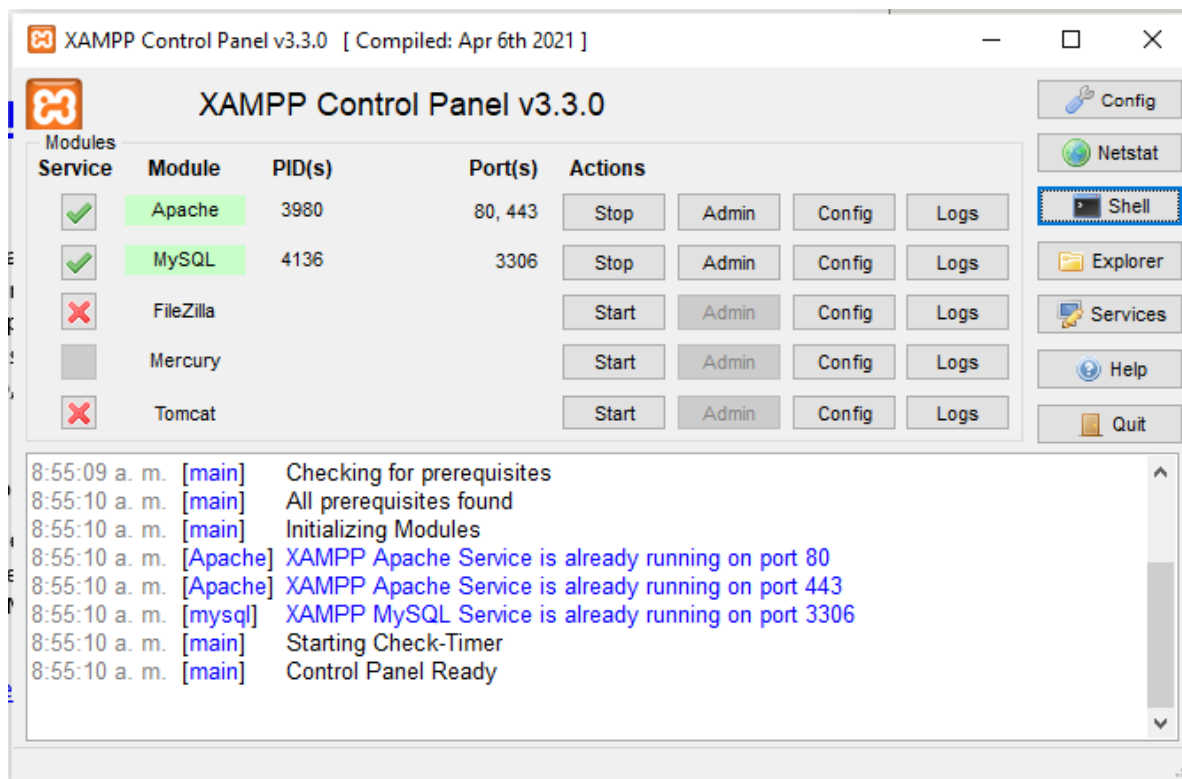
XAMPP es un paquete de software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script PHP y Perl. ... A esta fecha, XAMPP está disponible para Microsoft Windows, GNU/Linux, Solaris y Mac OS X.

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

- Instalación de Xampp
- Ejecutar Xampp
- Abrir Xampp



## 1.1 Verificar la versión de PHP



- Dar click en Shell
- Digitar `php -v`
- El sistema presenta la versión instalada del PHP

```
Administrator: XAMPP for Windows

Setting environment for using XAMPP for Windows.
Esquivel_Cesar@INTERPOL-CIA c:\xampp
# php -v
PHP 7.3.31 (cli) (built: Sep 21 2021 12:17:30) ( ZTS MSVC15 (Visual C++ 2017) x64 )
Copyright (c) 1997-2018 The PHP Group
Zend Engine v3.3.31, Copyright (c) 1998-2018 Zend Technologies
Esquivel_Cesar@INTERPOL-CIA c:\xampp
```

## 2. Configuración de Laravel en Windows 10

### 2.1 Instale el compositor en Windows:

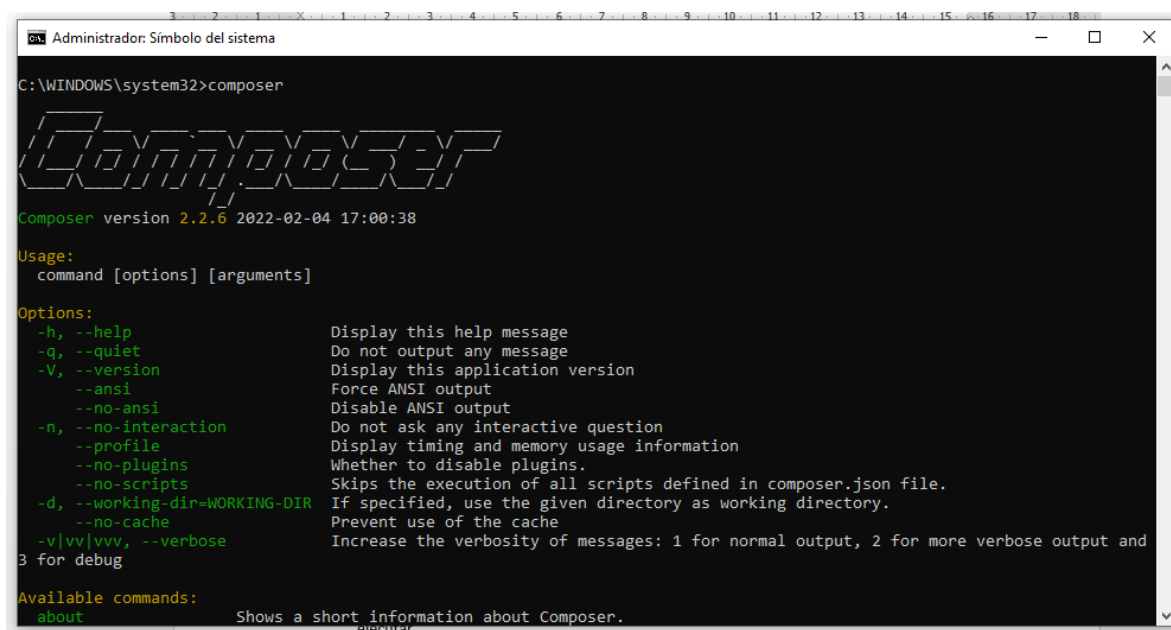
Siga el siguiente enlace para conocer los pasos de instalación de Composer.

Composer es un gestor de dependencias de PHP y básicamente se encarga de instalar las librerías que nuestros proyectos necesitan y de mantenerlas actualizadas para que funcionen correctamente.

Para instalar Composer en Windows visitamos la siguiente página:

<https://getcomposer.org/Composer-Setup.exe>

- Descargar el instalador, lo ejecutamos y le damos «Siguiente» a todos los pasos.
- Una vez terminada su instalación, abrimos una consola de Windows y ejecutamos el comando «Composer». Si todo ha salido bien, nos mostrará una lista de comandos para ejecutar.



```
C:\WINDOWS\system32>composer

Composer

Composer version 2.2.6 2022-02-04 17:00:38

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display this help message
  -q, --quiet               Do not output any message
  -V, --version             Display this application version
  --ansi                   Force ANSI output
  --no-ansi                Disable ANSI output
  -n, --no-interaction      Do not ask any interactive question
  --profile                Display timing and memory usage information
  --no-plugins              Whether to disable plugins.
  --no-scripts              Skips the execution of all scripts defined in composer.json file.
  -d, --working-dir=WORKING-DIR If specified, use the given directory as working directory.
  --no-cache                Prevent use of the cache
  -v|vv|vvv, --verbose     Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and
                             3 for debug

Available commands:
  about                    Shows a short information about Composer.
```

## **EXPLICACION DE LAS CARPETAS DEL PROYECTO**

[https://www.youtube.com/watch?v=QUiRa\\_uFTdo&ab\\_channel=MSC.GuadalupeGT](https://www.youtube.com/watch?v=QUiRa_uFTdo&ab_channel=MSC.GuadalupeGT)

**Visto el video anterior realizar un resumen indicando cuales son las carpetas claves que se deben modificar para empezar con su proyecto Web.**

Necesita aprender que es el MVC, observe el siguiente video y debata con sus compañeros

<https://www.youtube.com/watch?v=UU8AKk8Slqg>

# COMO HACER UN CRUD CON LARAVEL

## 1. APLICACIONES NECESARIAS

Xampp PHP 8.0

Composer

Node Js

Bootstrap

Visual Studio Code Extensiones

- Bootstrap Snippets

- Laravel Snippets

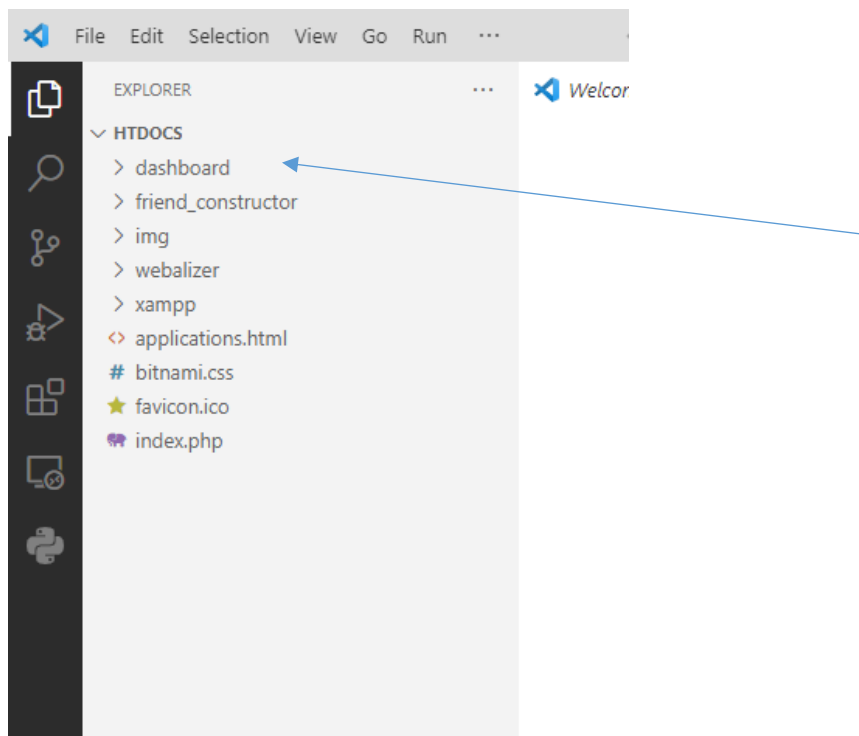
- Palenight Theme

2. Instalar las instalaciones antes mencionadas en su computadora.

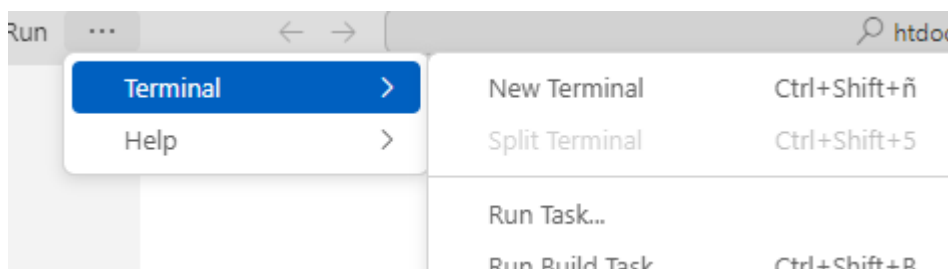
### 3. Instalar LARAVEL

Esta instalación se trabajará desde VSC.

- Desde VSC abrimos una nueva ventana, abrimos una carpeta pero este caso seleccionamos la carpeta **htdocs** de la carpeta Xampp que esta en la raíz C:



- Abrimos la terminal de VSC



- Estructura para instalar Laravel es:

**Composer create-project Laravel/Laravel nombre\_proyecto**

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\xampp\htdocs> composer create-project laravel/laravel sistema
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

- Downloading sebastian/lines-of-code (2.0.0)
- Downloading sebastian/complexity (3.0.0)
- Downloading sebastian/code-unit-reverse-lookup (3.0.0)
- Downloading phpunit/php-code-coverage (10.1.2)
- Downloading phar-io/version (3.2.1)
- Downloading phar-io/manifest (2.0.3)
- Downloading phpunit/phpunit (10.1.3)
- Downloading spatie/backtrace (1.4.0)
- Downloading spatie/flare-client-php (1.3.6)
- Downloading spatie/ignition (1.7.0)
- Downloading spatie/laravel-ignition (2.1.2)
0/93 [>-----] 0%
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

81 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 --fo

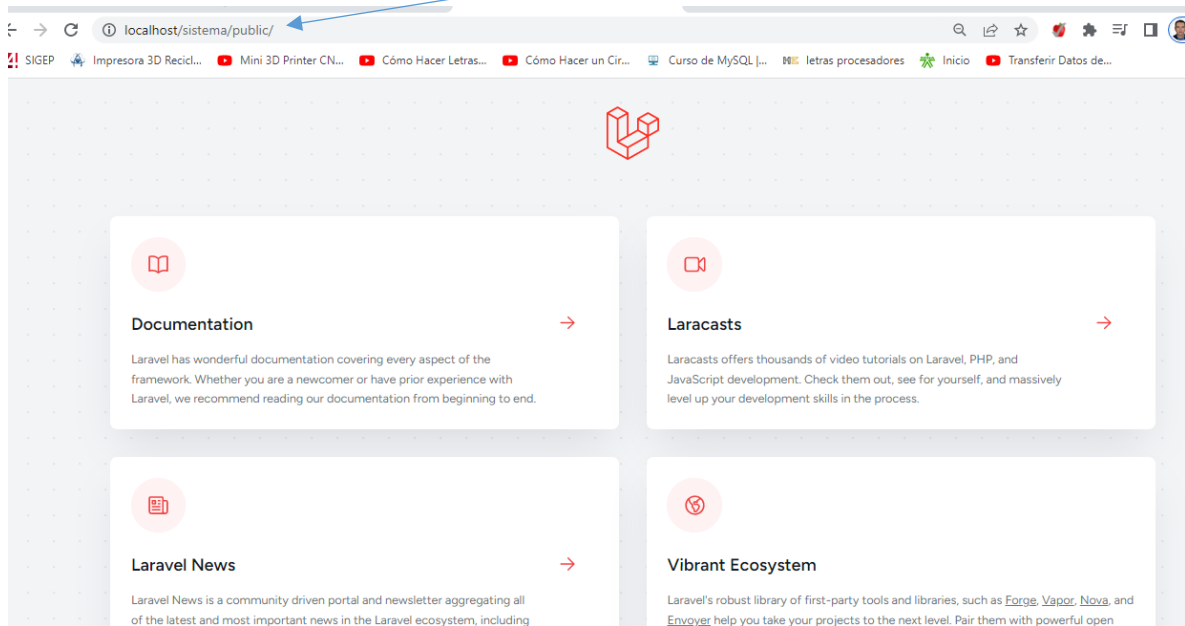
[INFO] No publishable resources for tag [laravel-assets].

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

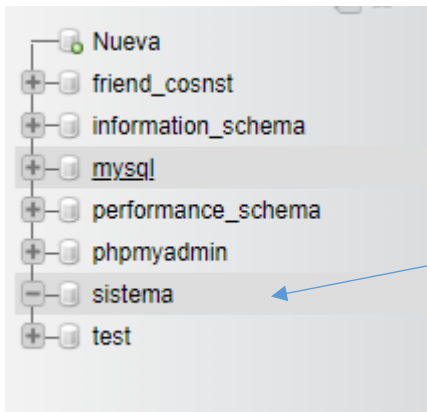
[INFO] Application key set successfully.

PS C:\xampp\htdocs>
```

- Instalada la aplicación procedemos a ejecutarlo en el navegador para verificar que el proyecto se haya creado.



#### 4. CREAR LA BASE DE DATOS DEL PROYECTO EN PHPMYADMIN





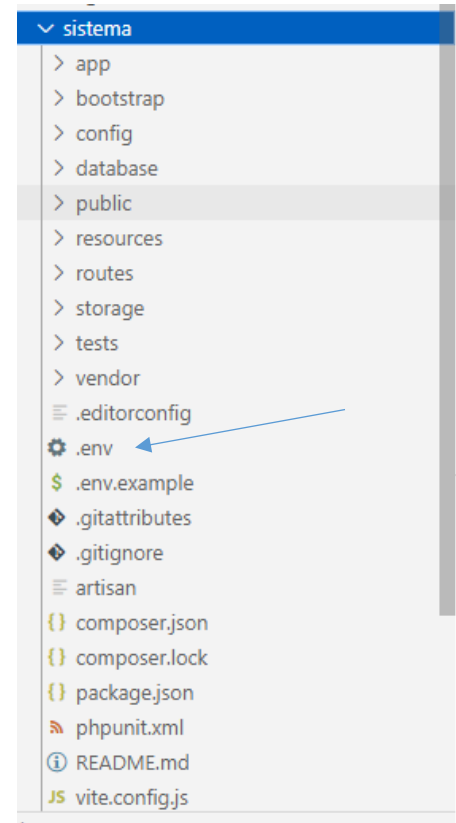
Vemos la estructura del proyecto que llamamos **sistema** con todas las carpetas que acaba de crear Laravel.

Nos dirigimos al archivo **.env**, el cual aloja todos los parámetros de conexión a las Bases de Datos.

```
LOG_CHANNEL=stack
LOG_DEPRECATIONS_CHANNEL=null
LOG_LEVEL=debug
```

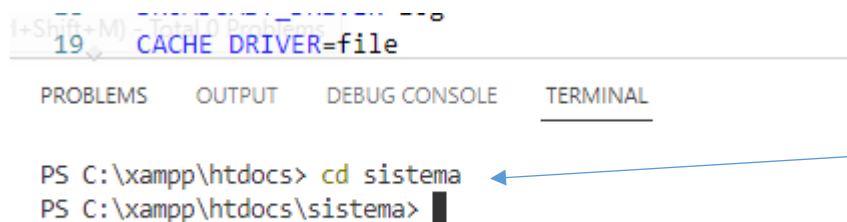
```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=laravel
DB_USERNAME=root
DB_PASSWORD=
```

```
BROADCAST_DRIVER=log
```



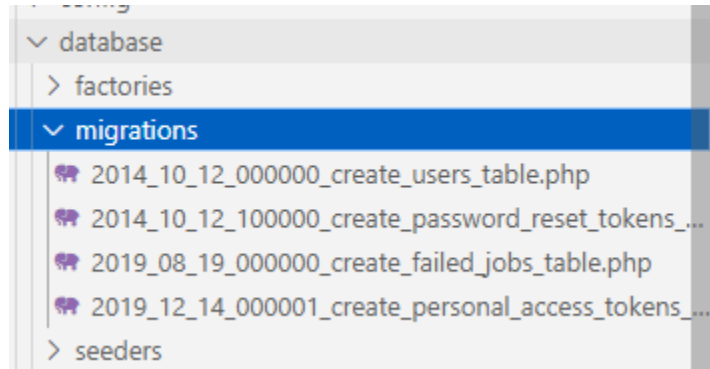
Aquí es donde debemos cambiar cada uno de los parámetros de conexión a la BD que se va a trabajar.

Para poder seguir trabajando en la carpeta de nuestro proyecto abrimos dicha carpeta desde la terminal



La carpeta **migrations** de la carpeta **Database** es la que contiene las tablas que se debe migrar hacia la BD de PhpMyadmin.

Automáticamente aparecen 4, consulte en internet que almacena cada una de ellas.



Como se dijo anteriormente estas tablas solo están en la estructura de Laravel, ahora las migraremos a nuestra BD con el comando **php artisan migrate**.

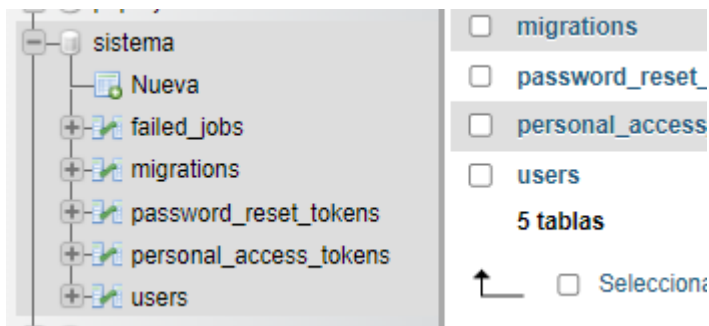
```
PS C:\xampp\htdocs\sistema> php artisan migrate
```

**INFO** Preparing database.

Creating migration table ..... 20ms **DONE**

**INFO** Running migrations.

2014_10_12_000000_create_users_table .....	39ms	<b>DONE</b>
2014_10_12_100000_create_password_reset_tokens_table .....	39ms	<b>DONE</b>
2019_08_19_000000_create_failed_jobs_table .....	33ms	<b>DONE</b>
2019_12_14_000001_create_personal_access_tokens_table .....	54ms	<b>DONE</b>



**migrate:fresh, migrate:install, migrate:refresh, migrate:reset, migrate:status**

**Consultar**

## 5. MODEL, CONTROLLER AND RESOURCE.

Crear el modelo MVC para un empleado, quiere decir que se debe crear la estructura para poder realizar el CRUD para un empleado.

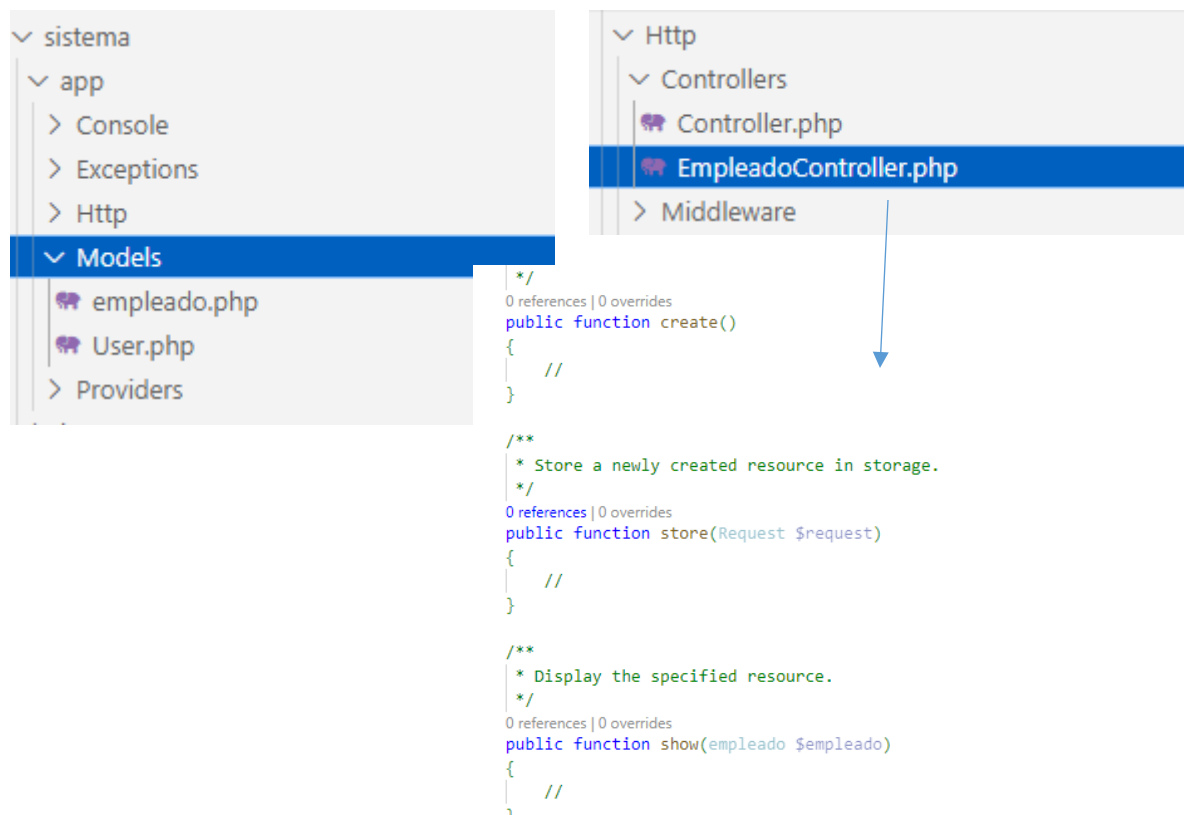
En la terminal ejecutamos el comando:

- **php artisan make:model empleado -mcr**

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\xampp\htdocs\sistema> php artisan make:model empleado -mcr

[INFO] Model [C:\xampp\htdocs\sistema\app\Models\empleado.php] created successfully.
[INFO] Migration [C:\xampp\htdocs\sistema\database\migrations\2023_05_23_010838_create_empleados_table.php] created successfully.
[INFO] Controller [C:\xampp\htdocs\sistema\app\Http\Controllers\EmpleadoController.php] created successfully.
```

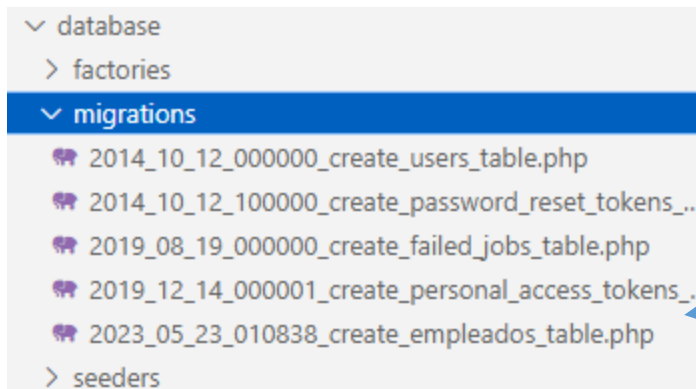
Se exploramos la carpeta sistema en VSC vemos que se creó la estructura con el nombre empleado.



## 6. MIGRACION TABLA EMPLEADO

Siguiente a esto debemos crear la estructura de la tabla para almacenar los empleados.

Recordemos que las estructuras de nuestras tablas quedan alojadas en la carpeta **migrations**



```
~/
public function up(): void
{
    Schema::create('empleados', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}
```

Dentro de **Schema** crearemos los campos de la tabla empleados que necesitamos para nuestro proyecto.

```

public function up(): void
{
    Schema::create('empleados', function (Blueprint $table) {
        $table->id();
        $table->string('Nombres');
        $table->string('PrimerApel');
        $table->string('SegundoApel');
        $table->string('Correo');
        $table->string('Foto');
        $table->timestamps();
    });
}

```

Ejecutamos el **php artisan migrate**, para migrar la estructura creada a la BD

```
PS C:\xampp\htdocs\sistema> php artisan migrate
```

```
INFO Running migrations.
```

```
2023_05_23_010838_create_empleados_table ..... 17ms DONE
```

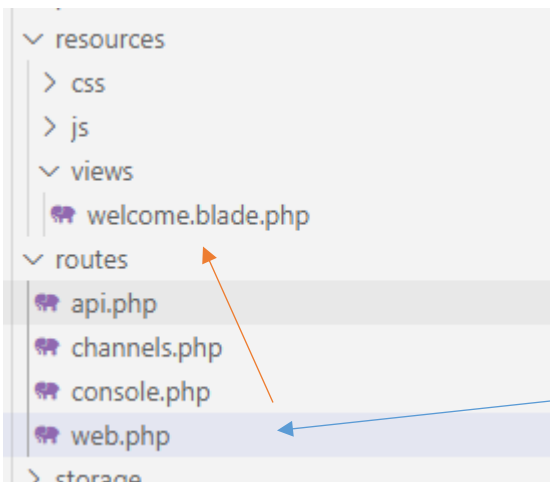
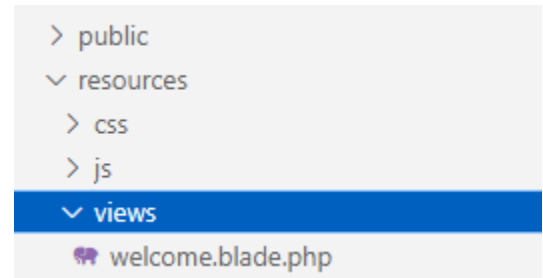
```
PS C:\xampp\htdocs\sistema> █
```

<div> <div>Examinar</div> <div>Estructura</div> <div>SQL</div> <div>Buscar</div> <div>Insertar</div> <div>Exportar</div> <div>Importar</div> <div>Privilegios</div> <div>Operaciones</div> <div>Seq</div> </div>									
Estructura de tabla		Vista de relaciones							
#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
<input type="checkbox"/> 1	id	bigint(20)		UNSIGNED	No	Ninguna		AUTO_INCREMENT	Cambiar  Eliminar  Más
<input type="checkbox"/> 2	Nombres	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar  Eliminar  Más
<input type="checkbox"/> 3	PrimerApel	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar  Eliminar  Más
<input type="checkbox"/> 4	SegundoApel	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar  Eliminar  Más
<input type="checkbox"/> 5	Correo	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar  Eliminar  Más
<input type="checkbox"/> 6	Foto	varchar(255)	utf8mb4_unicode_ci		No	Ninguna			Cambiar  Eliminar  Más
<input type="checkbox"/> 7	created_at	timestamp			Sí	NULL			Cambiar  Eliminar  Más
<input type="checkbox"/> 8	updated_at	timestamp			Sí	NULL			Cambiar  Eliminar  Más

Consultar en la página de Laravel los tipos de datos que se pueden trabajar para los campos.

## 7. CARPETA VISTAS.

Las vistas es donde encontraremos las estructuras HTML de nuestro proyecto.

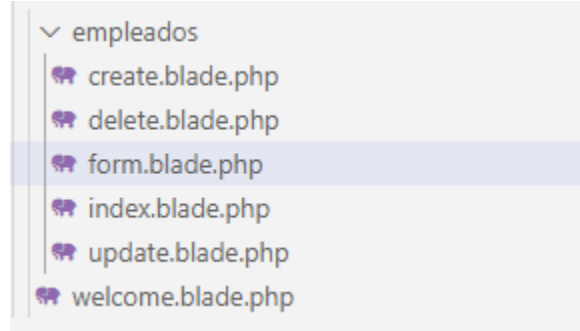


La carpeta rutas permite enlazar un archivo php con las vistas.

Si abrimos el archivo web.php vemos un código que mediante el método **get** recibe una url y luego indica que muestre la vista **welcome**, cuando ingresamos en el navegador **localhost/sistema/public**, se invoca el archivo **web.php**

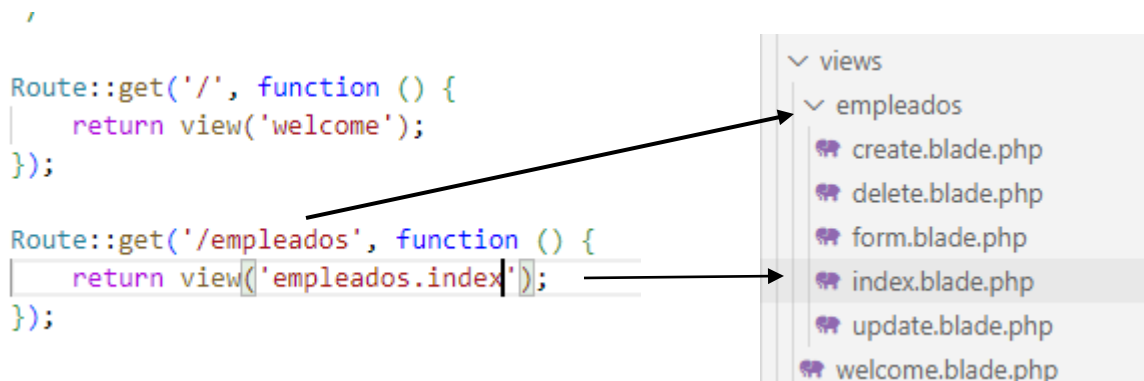
Ahora vamos a crear una carpeta para las vistas que manejaremos para los empleados.

Dentro de **views** creamos la carpeta **empleados** y dentro de *empleados* creamos el primer archivo **index.blade.php**. Este archivo será en el que presentaremos la lista con los empleados que se hayan creado.



## 8. ACCEDER A LAS VISTAS.

Ahora necesitamos acceder a las vistas del empleado que acabamos de crear. Recuerde que se accede mediante el archivo **web.php**.



Otra forma de acceder es usando las rutas de los controladores

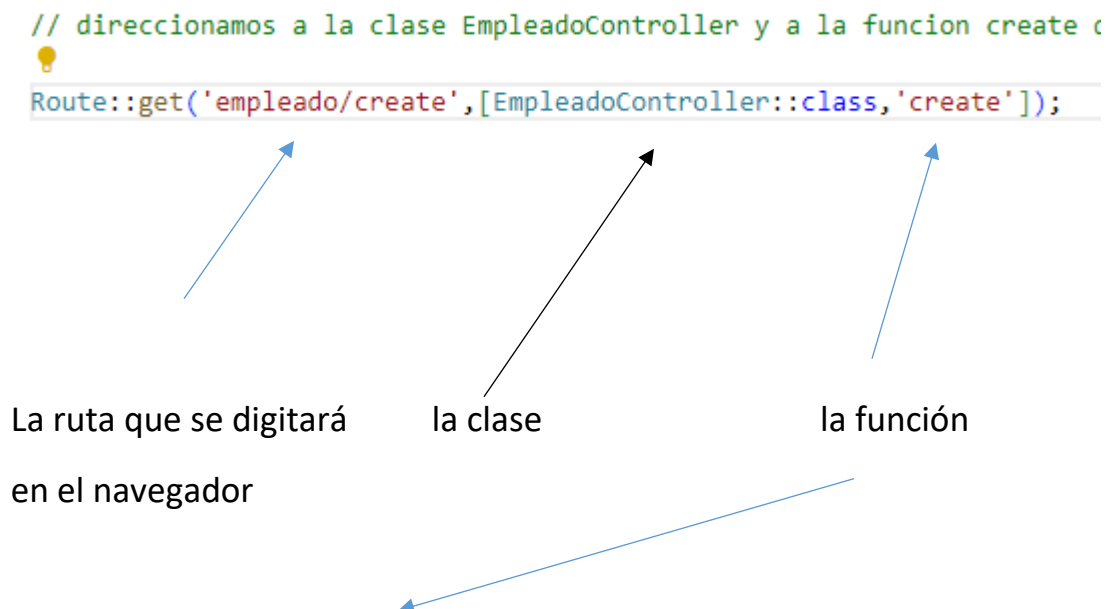
Vamos al archivo

**.php** de la carpeta APP Http Controllers

Mediante la función **Create** accederemos al archivo **create.blade.php** de las vistas.



Ahora creamos la ruta en el archivo **web.php** quien contiene las rutas y accede a las vistas.

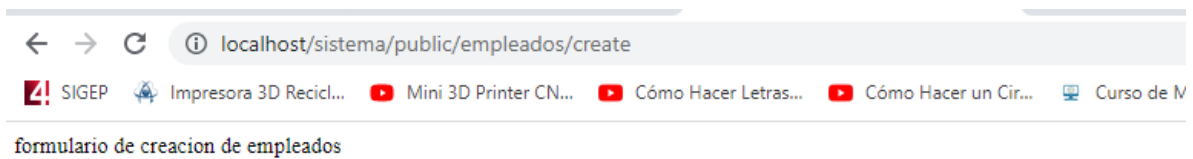




```

| */
| 1 reference | 0 overrides
| public function create()
| {
|     //Acceder a create.blade.php de la vista para crear los empleados
|     return view('empleados.create');
| }
|
| /**

```



*Existe una línea de comando que nos permite acceder a todas las funciones sin necesidad de crear una por una.*

Comentamos las líneas que acabamos de crear en el archivo web.php y ejecutamos la siguiente

```

// Route::get('/empleados', function () {
//     return view('empleados.index');
// });

// direccionamos a la clase EmpleadoController y a la funcion create que esta en e

// Route::get('/empleados/create',[EmpleadoController::class,'create']);

Route::resource('empleado', EmpleadoController::class);

```

Nos permite acceder a todas las clases del controlador empleado

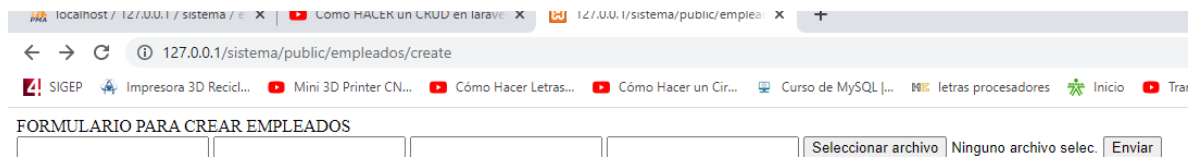
## 9. EMPEZAR A TRABAJAR EN LOS FORMULARIOS

Empezaremos creando el formulario para crear los empleados

```
<form action="" method="POST" enctype="multipart/form-data">

    <input type="text" name="Nombres" id="Nombres" placeholder="Introduzca Nombre"><br>
    <input type="text" name="PrimerApel" id="PrimerApel" placeholder="Introduzca Primer Apellido"><br>
    <input type="text" name="SegundoApel" id="SegundoApel" placeholder="Introduzca Segundo Apellido"><br>
    <input type="text" name="Correo" id="Correo" placeholder="Introduzca Email"><br>
    <input type="file" name="Foto" id="Foto"><br>
    <input type="submit" value="Guardar">

</form>
```



Quien recibe los datos que el usuario ingresa en el formulario es la clase **store** de la clase **EmpleadoController**, mediante un método de envío

```
/**
 * Store a newly created resource in storage.
 */
0 references | 0 overrides
public function store(Request $request)
{
    //
}
```

Para conocer los parámetros de envío digitamos en la consola el comando **php artisan route:list**

```
GET|HEAD      api/user .....
GET|HEAD      empleados ..... empleados.index > EmpleadoController@index
POST          empleados ..... empleados.store > EmpleadoController@store
GET|HEAD      empleados/create ..... empleados.create > EmpleadoController@create
GET|HEAD      empleados/{empleado} ..... empleados.show > EmpleadoController@show
PUT|PATCH    empleados/{empleado} ..... empleados.update > EmpleadoController@update
DELETE       empleados/{empleado} ..... empleados.destroy > EmpleadoController@destroy
GET|HEAD      empleados/{empleado}/edit ..... empleados.edit > EmpleadoController@edit
GET|HEAD      sanctum/csrf-cookie ..... sanctum.csrf-cookie > Laravel\Sanctum > CsrfCookieController@show
```

Showing [13] routes

**Método POST** direccionamos a **empleados** y al método **empleados.store** del archivo **EmpleadoController.php**.

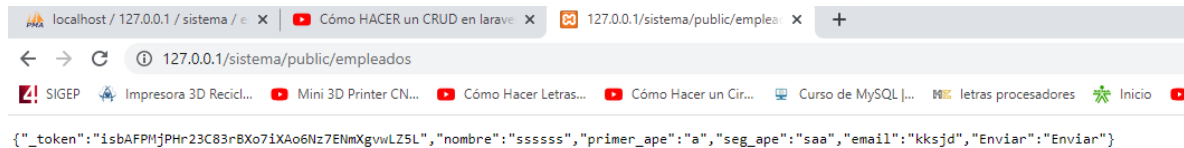
Y le enviamos una llave de seguridad al formulario que exige Laravel mediante el comando **@csrf**.

```
FORMULARIO PARA CREAR EMPLEADOS
<form action="{url('/empleados')}" method="POST" enctype="multipart/form-data">
|   @csrf
```

Luego abrimos el archivo **EmpleadoController.php**. e imprimimos los datos que viajaron por el formulario

```
0 references | 0 overrides
public function store(Request $request)
{
    //
    $datosEmpleado = request()->all();
    return response()->json($datosEmpleado);
}
```

Llenamos los datos del formulario y enviamos los datos mediante el botón enviar y obtendremos este archivo Json.




## 10. INSERTAR DATOS EN LA B.D.

Como pueden darse cuenta estamos recibiendo los datos del formulario más el token y el botón enviar, vamos a quitar estos datos.

- Dejar el botón submit solo con el value para que no nos envíe datos de él al formulario.
- Modificamos el archivo **EmpleadosController**. En el Request. No vamos a mostrar todo lo que recibimos del formulario, sino que exceptuamos el token.

```
//  
// $datosEmpleado = request()->all();  
$datosEmpleado = request()->except(['_token']);  
return response()->json($datosEmpleado);
```

Ahora utilizamos el modelo para la inserción de los datos

```
namespace App\Http\Controllers;  
  
use App\Models\Empleado;   
use Illuminate\Http\Request;
```

U references | U overrides

```
public function store(Request $request)
{
    //
    // $datosEmpleado = request()->all();
    $datosEmpleado = request()->except('_token');
    empleado::insert($datosEmpleado);
    return response()->json($datosEmpleado);
}
```

Desde el formulario en el navegador ingresamos datos y verificamos que queden guardados en la B.D.

	id	Nombres	PrimerApel	SegundoApel	Correo	Foto	created_at	updated_at
<input type="checkbox"/>	1	Cesar	esquivel	Cortes	esquivel@78.com	C:\xampp\tmp\php98A5.tmp	NULL	NULL

Vamos a cambiar el archivo de la foto porque la que acaba de ser guardada es un archivo temporal.

```
if($request->hasFile('Foto')){
    $datosEmpleado['Foto']=$request->file('Foto')->store('uploads','public');
}
```

Pregunta si hay algún archivo que venga del campo Foto y se lo pasa en la matriz de la variable \$datosEmpleado y le pide que lo cargue en un archivo y luego lo suba a la carpeta public.

	id	Nombres	PrimerApel	SegundoApel	Correo	Foto	created_at	updated_at
<input type="checkbox"/>	1	Cesar	esquivel	Cortes	esquivel@78.com	C:\xampp\tmp\php98A5.tmp	NULL	NULL
<input type="checkbox"/>	3	Cesar	esquivel	Cortes	esquivel@78.com	uploads/KO30gzZj2pA1N1247ks0CMzev8WQsWlimkG1gwYG.a...	NULL	NULL

## 11. CONSULTAR LA B.D Y MOSTRAR LOS DATOS EN LA PAGINA INDEX.

Desde el archivo EmpleadoController.php creamos una variable la cual va a pasarle los datos encontrados al index.blade.php, pero ahora trabajaremos en la función **Index**.



```
0 references | 0 overrides
public function index()
{
    //
    $listado['empleados'] = empleado::paginate(5);

    return view('empleados.index', $listado);
}
```

Luego haremos la tabla en Html en el archivo index.blade.php para mostrar los datos.

```
LISTA LOS DATOS DE LOS EMPLEADOS
<table class="table table-light">
  <thead class="thead-light">
    <tr>
      <th>#</th>
      <th>Foto</th>
      <th>Nombre</th>
      <th>P. Apellido</th>
      <th>S. Apellido</th>
      <th>Correo</th>
      <th>Accion</th>
    </tr>
  </thead>
```

```
<tbody>
  @foreach ($empleados as $datos)
    <tr>
      <td>{{ $datos->id }}</td>
      <td>{{ $datos->Foto }}</td>
      <td>{{ $datos->Nombre }}</td>
      <td>{{ $datos->PrimerApel }}</td>
      <td>{{ $datos->SegundoApel }}</td>
      <td>{{ $datos->Email }}</td>
      <td>Editar | Borrar</td>
    </tr>
  @endforeach
</tbody>
</table>
```

## 12.BORRAR REGISTROS

Ejecutamos el comando **php artisan route:list** para conocer el método

```
GET|HEAD      empleados/{empleado} ..... empleados.show > EmpleadoController@...
PUT|PATCH    empleados/{empleado} ..... empleados.update > EmpleadoController@up...
DELETE        empleados/{empleado} ..... empleados.destroy > EmpleadoController@des...
GET|HEAD      empleados/{empleado}/edit
```

Se enviara a la función **destroy**, los datos viajaran por POST pero el método que se debe cargar es el **DELETE**

```
<tbody>
  @foreach ($empleados as $datos)
    <tr>
      <td>{{ $datos->id }}</td>
      <td>{{ $datos->Foto }}</td>
      <td>{{ $datos->Nombre }}</td>
      <td>{{ $datos->PrimerApel }}</td>
      <td>{{ $datos->SegundoApel }}</td>
      <td>{{ $datos->Email }}</td>
      <td>Editar |
        <form action="{{ url('/empleados/' . $datos->id) }}" method="POST" >
          @csrf
          {{ method_field('DELETE') }}
          <input type="submit" onclick="return confirm('¿Deseas Eliminar?')" value="Eliminar">
        </form>
```

Vamos al Controlador y buscamos la función **destroy** y allí copiamos el código, le pasamos como parámetro el **\$id** que es el código del registro que hayamos seleccionado para eliminar

```
0 references | 0 overrides
public function destroy($id)
{
    //
    empleado::destroy($id);
    return redirect('empleados');
}
```

### 13. INCLUIR UN FORMULARIO

Para no repetir el código del formulario crear lo que hacemos es que pasamos el código de todas las cajas de texto junto con el botón de guardar al formulario `form.blade.php` y le hacemos un **@include** para que invocar el archivo `form.blade.php` el cual tendrá el código.

```
EmpleadoController.php  form.blade.php  create.blade.php X
sistema > resources > views > empleados > create.blade.php > form
1  FORMULARIO PARA CREAR EMPLEADOS
2  <form action="{{url('/empleados')}}" method="POST" enctype="multipart/form-data">
3      @csrf
4      @include('empleados.form');
5
6  </form>
7
```

```
EmpleadoController.php  form.blade.php  create.blade.php
sistema > resources > views > empleados > form.blade.php > input
1  Formulario que tendra los datos para crear o actualizar los empleados
2  <input type="text" name="Nombres" id="Nombres" placeholder="Introduzca Nombre"><br>
3      <input type="text" name="PrimerApel" id="PrimerApel" placeholder="Introduzca Primer Apellido"><br>
4      <input type="text" name="SegundoApel" id="SegundoApel" placeholder="Introduzca Segundo Apellido"><br>
5      <input type="text" name="Correo" id="Correo" placeholder="Introduzca Email"><br>
6      <input type="file" name="Foto" id="Foto"><br>
7      <input type="submit" value="Guardar">
```

← → ↻ ⓘ localhost/sistema/public/empleados/create

SIGEP Impresora 3D Recicl... Mini 3D Printer CN... Cómo Hacer Letras...

---

**FORMULARIO PARA CREAR EMPLEADOS**  
Formulario que tendra los datos para crear o actualizar los empleados

Introduzca Nombre
Introduzca Primer Apellido
Introduzca Segundo Apellido
Introduzca Email
Seleccionar archivo
Ninguno archivo selec.
Guardar ;



Ahora en el archivo **Update.blade.php** también ingresamos el **@include** para incluir los campos para editar los datos del empleado.

```
update.blade.php × EmpleadoController.php form.blade.php
sistema > resources > views > empleados > update.blade.php
1  FORMULARIO PARA ACTUALIZAR LOS DATOS DE LOS EMPLEADOS
2
3  @include('empleados.form');
4
```

Ahora creamos un enlace para la opción **editar** del Index que al darle click nos lleve al archivo **Update.blade.php** llevándose el registro seleccionado.

```
<tbody>
  @foreach ($empleados as $datos)
    <tr>
      <td>{{ $datos->id }}</td>
      <td>{{ $datos->Foto }}</td>
      <td>{{ $datos->Nombre }}</td>
      <td>{{ $datos->PrimerApel }}</td>
      <td>{{ $datos->SegundoApel }}</td>
      <td>{{ $datos->Email }}</td>
      <td>
        <a href="{{ url('/empleados/' . $datos->id . '/edit') }}" >
          Editar </a> |
        <form action="{{ url('/empleados/' . $datos->id )" method="POST" >
```

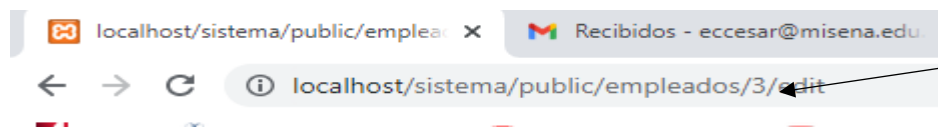


localhost/sistema/public/empleados

SIGEP Impresora 3D Recicl... Mini 3D Printer CN... Cómo Hacer Letras... Cómo Hacer un Cir... Curso de MySQL [... letras procesadores

LISTA LOS DATOS DE LOS EMPLEADOS

#	Foto	Nombre	P. Apellido	S. Apellido	Correo	Accion
3	uploads/KO30gzZj2pA1N1247ks0CMzev8WQsWlimgG1gwYG.avif	esquivel	Cortes			<a href="#">Editar</a> <a href="#">Eliminar</a>
4	uploads/e3qeUKetaIadIHqpyTQuqmRY9aNMIYqlnabEjIsi.avif	esquivel	Cortes			<a href="#">Editar</a> <a href="#">Eliminar</a>
5	uploads/qaXo9Du3SXvXXRZUvERVhdJBMku3lzZ8m5918UtC.png	sss	sss			<a href="#">Editar</a> <a href="#">Eliminar</a>
6	uploads/SWD4bdrjdvWQnuvdrRKjeENoO7AlrnizJMKEhtM8.png	sss	sss			<a href="#">Editar</a> <a href="#">Eliminar</a>
7	uploads/dHcaLzzbyDdkAiePXwesn3yUO9giKjvNBZSr6ToQ.png	aaa	aa			<a href="#">Editar</a> <a href="#">Eliminar</a>



Como vemos en la url se está llevando el **id** hacia el archivo **Update**

Luego creamos la vista en la función edit del controlador.

```
0 references | 0 overrides
public function edit(Empleado $Empleado)
{
    //
    return view('empleados.update');
}
```

localhost/sistema/public/empleados/3/edit

SIGEP Impresora 3D Recicl... Mini 3D Printer CN... Cómo Hacer Letras... Cómo Hacer un Cir... Curso de MySQL [... letras procesadores Inicio

FORMULARIO PARA ACTUALIZAR LOS DATOS DE LOS EMPLEADOS Formulario que tendra los datos para crear o actualizar los empleado

Introduzca Nombre

Introduzca Primer Apellido

Introduzca Segundo Apellido

Introduzca Email

Seleccionar archivo Ninguno archivo selec.

Guardar ;

Recibimos el id en el parámetro de la función, buscamos los datos del id y lo almacenamos en la variable `$empleado`. Luego retornamos los datos encontrados con el comando **compact**.

```
| /  
0 references | 0 overrides  
public function edit($id)  
{  
    //  
    $empleado = empleado::findOrFail($id);  
    return view('empleados.update', compact('empleado'));  
}
```

Luego pasamos los valores de cada campo al formulario en el archivo **form.blade.php**.

Ahora en el **Update.blade.php** pasamos el id seleccionado al un formulario para poder actualizar los datos de ese registro seleccionado. El método que utiliza es el PATCH

GET HEAD	empleados/{empleado} .....	empleados.show > EmpleadoController@:
PUT PATCH	empleados/{empleado} .....	empleados.update > EmpleadoController@up:
DELETE	empleados/{empleado} .....	empleados.destroy > EmpleadoController@des:
GET HEAD	empleados/{empleado}/edit	empleados.edit > EmpleadoController@:

update.blade.php X EmpleadoController.php form.blade.php create.blade.php index.blade.php

sistema > resources > views > empleados > update.blade.php > form

```
1 FORMULARIO PARA ACTUALIZAR LOS DATOS DE LOS EMPLEADOS  
2 <form action="{{url('/empleados/' . $empleado->id)}}" method="POST" enctype="multipart/form-data">  
3     @csrf  
4     {{method_field('PATCH')}}  
5     @include('empleados.form')  
6  
7 </form>  
~
```

Ahora en el método **update** del controlador recibimos el id y pasamos todos los datos menos el token de seguridad y el método patch. Luego comparamos el id recibido con id de la base de datos y si lo encuentra actualizamos, luego direccionamos.

```

0 references | 0 overrides
public function update(Request $request, $id)
{
    //
    $datos = request()->except(['_token', '_method']);
    empleado::where('id', '=', $id)->update($datos);

    $empleado = empleado::findOrFail($id);
    return view('empleados.update', compact('empleado'));
}

/**

```

Actualizamos y guardamos

←
→
↻
📄 localhost/sistema/public/empleados/3

SIGEP
 Impresora 3D Recicl...
 Mini 3D Printer CN...
 Cómo Hacer Letras...
 Cómo

## FORMULARIO PARA ACTUALIZAR LOS DATOS DE LOS EMPLEADOS

Formulario que tendra los datos para crear o actualizar los empleados

Cesar
esquivela
Cortes
esqui@78.com
<div> <div>Seleccionar archivo</div> <div>Ninguno archivo selec.</div> </div>
<div>Guardar</div>

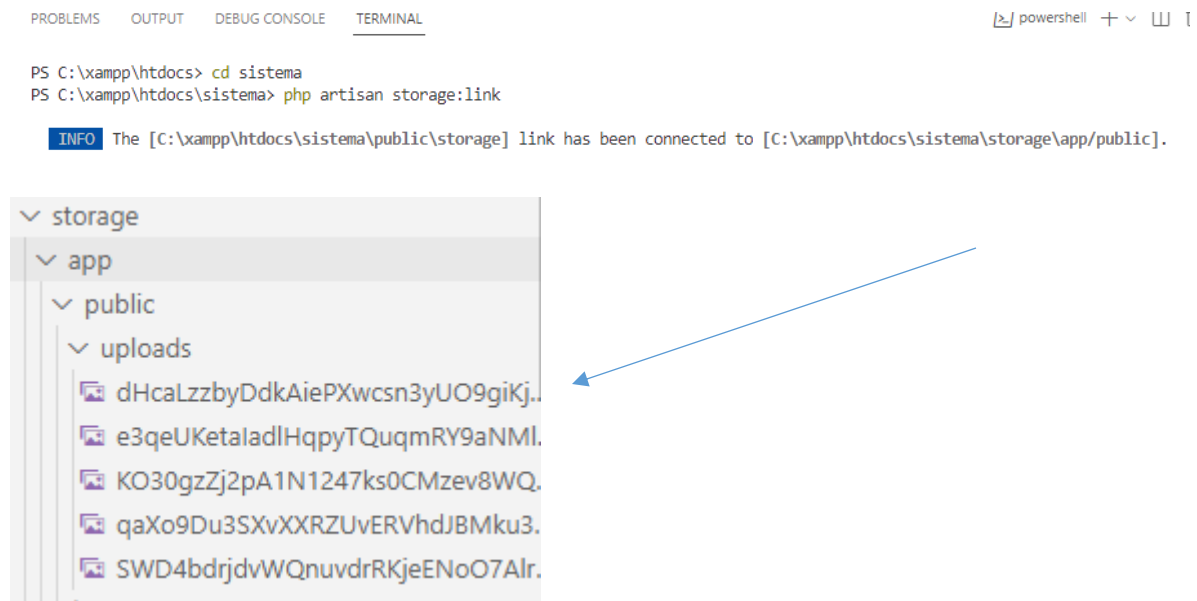
## 14.MOSTRAR LA FOTOGRAFIA

Ahora trabajaremos desde el **index** para mostrar la imagen del usuario.

Recordar que las imágenes quedan guardadas en la carpeta storage.

```
<tbody>
  @foreach ($empleados as $datos)
    <tr>
      <td>{{ $datos->id }}</td>
      <td></td>
```


Luego ejecutamos en la terminal el php artisan **storage** para que carguen las imágenes.



← → ↻ ⓘ localhost/sistema/public/empleados

SIGEP
 Impresora 3D Recicl...
 Mini 3D Printer CN...
 Cómo Hacer Letras...
 Cómo Hacer un Cir...
 Curso de MySQL |...

LISTA LOS DATOS DE LOS EMPLEADOS

#	Foto	Nombre	P. Apellido	S. Apellido	Correo	Accion
3		Cesar	esquivela	Cortes		<a href="#">Editar</a> <a href="#">Eliminar</a>

Ahora debemos hacer lo mismo para el formulario Editar. **Form.blade.php**

```

update.blade.php  EmpleadoController.php  form.blade.php x  create.blade.php
tema > resources > views > empleados > form.blade.php > br
1  Formulario que tendra los datos para crear o actualizar los empleados
2  <br>
3  <input type="text" value="{{ $empleado->Nombres }}" name="Nombres" id="Nombres"
4      <input type="text" value="{{ $empleado->PrimerApel }}" name="PrimerApel" id
5      <input type="text" value="{{ $empleado->SegundoApel }}" name="SegundoApel" i
6      <input type="text" value="{{ $empleado->Correo }}" name="Correo" id="Correo"
7      <input type="file" name="Foto" id="Foto"><br>
8      
9
10     <input type="submit" value="Guardar">
11
  
```

← → ↻ ⓘ localhost/sistema/public/empleados/3/edit

! SIGEP 3D Impresora 3D Recicl... Mini 3D Printer CN... Cómo Hacer Letras...

---

FORMULARIO PARA ACTUALIZAR LOS DATOS DE LOS EMPLEADOS

Formulario que tendra los datos para crear o actualizar los empleados

Cesar
esquivela
Cortes
esquiell@78.com
Seleccionar archivo Ninguno archivo selec.



## 15.ACTUALIZAR LA FOTO EXISTENTE.

Si el usuario selecciona una nueva foto debemos actualizarla en el storage y en la Base de Datos. Vamos al controlador, función **update**.

Para borrar la anterior imagen y poder cargar la nueva debemos la clase **illuminate** al comienzo del **controlador**.

```
namespace App\Http\Controllers;  
  
use App\Models\empleado;  
use Illuminate\Http\Request;  
use Illuminate\Support\Facades\Storage;  
|
```



0 references | 0 overrides

```
public function update(Request $request, $id)
{
    //
    $datos = request()->except(['_token', '_method']);

    //verificar si el usuario ha seleccionado una nueva foto, en caso verdadero la carga
    if($request->hasFile('Foto')){
        $datos['Foto']=$request->file('Foto')->store('uploads','public');
    }

    //sino selecciono una nueva foto sigue con la que ya tenia anteriormente
    empleado::where('id','=',$id)->update($datos);
    $empleado = empleado::findOrFail($id);
    return view('empleados.update', compact('empleado'));
}
```

← → ↻ ⓘ localhost/sistema/public/empleados/4/edit

🔍 SIGEP 🖨️ Impresora 3D Recicl... 📺 Mini 3D Printer CN... 📺 Cómo Hacer Letras...

## FORMULARIO PARA ACTUALIZAR LOS DATOS DE LOS EMPLEADOS

Formulario que tendra los datos para crear o actualizar los empleados

Cesar  
esquivel  
Cortes  
esquivel@78.com

Seleccionar archivo Ninguno archivo selec.



Guardar



## 16.AJUSTAR LOS FORMULARIOS.

Para evitar el error de que no reconozca la variable \$empleado cuando se ingresa desde la url al formulario **create**, le pasamos el **isset** a cada campo. Recordar que este mismo formulario sirve para crear o para actualizar, el **isset** pregunta si viene algún dato, en caso de ser verdadero lo muestra; sino deja el campo vacio.

```
<input type="text" value="{{isset($empleado->Nombres)?$empleado->Nombres:''}}" name="Nombres" id="Nombres" placeholder="Introduzca Nombre" />
<input type="text" value="{{isset($empleado->PrimerApel)?$empleado->PrimerApel:''}}" name="PrimerApel" id="PrimerApel" placeholder="Introduzca Primer Apellido" />
<input type="text" value="{{isset($empleado->SegundoApel)?$empleado->SegundoApel:''}}" name="SegundoApel" id="SegundoApel" placeholder="Introduzca Segundo Apellido" />
<input type="text" value="{{isset($empleado->Correo)?$empleado->Correo:''}}" name="Correo" id="Correo" placeholder="Introduzca Email" /><br>
<input type="file" name="Foto" id="Foto"><br>
@if(isset($empleado->Foto))
    
@endif
<input type="submit" value="Guardar">
```

## 17.CREAR ENLACES

Se necesita pasar con la etiqueta **<a href="">** la url que direccionara el enlace.

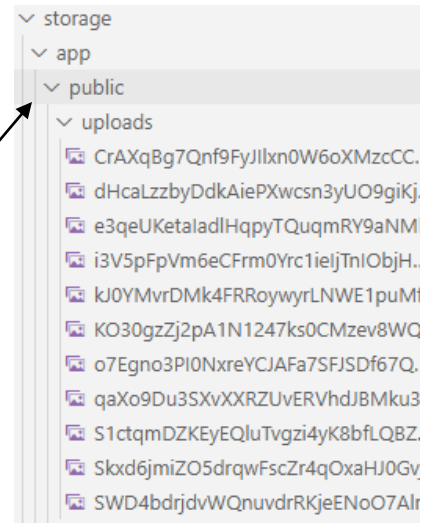
```
<a href="{{url('/empleados/create')}}">Registrar Nuevo Empleado </a>
```

## 18. BORRAR LAS FOTOS DE LA CARPETA STORAGE

Recordar que las imágenes que se suben a la BD quedan en la carpeta storage de Laravel. Lo que haremos ahora es recibir el **id** del registro al cual le queremos borrar la imagen, esto en el controlador, función **destroy**.

```
public function destroy($id)
{
    //
    $empleado = empleado::findOrFail($id);
    if(Storage::delete('public/'.$empleado->Foto)){
        empleado::destroy($id);
    }

    return redirect('empleados');
}
```



## 19.MOSTRAR MENSAJES DE CONFIRMACION

Vamos a cargar una función llamada **mensaje** al archivo **index**, para que cuando se inserte un nuevo registro le redireccione al index y presente un mensaje de confirmación.

### index

```
2
3 {{-- //recibe la funcion mensaje desde el controler para mostrar un mensaje de confirmacion --}}
4 @if(Session::has('mensaje'))
5     {{Session::get('mensaje')}}
6 @endif
```

### Controlador función Store

```
    empleado::insert($datosEmpleado);
    // return response()->json($datosEmpleado);
    return redirect('empleados')->with('mensaje','Registro ingresado con exito');
```

**Controlador función destroy.** Para que envíe msj al borrar un registro

```
public function destroy($id)
{
    //busca la imagen que viene del id seleccionado y si la encuentra la borra
    $empleado = empleado::findOrFail($id);
    if(Storage::delete('public/'.$empleado->Foto)){
        empleado::destroy($id);
    }


    return redirect('empleados')->with('mensaje','Registro eliminado exitosamente');
```

## 20.MOSTRAR EL NOMBRE DEL BOTON DE ACUERDO A LA FUNCION DONDE SE ENCUENTRE EL USUARIO.

Vamos a mostrar cambiar el nombre del botón para que cuando el usuario este en el formulario de crear muestre guardar, cuando pase a eliminar muestre Borrar, etc.

Debemos ir a cada archivo php y al final del formulario copiamos el siguiente código cambiando el nombre que queremos mostrar en cada botón.

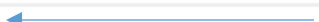
### Form.blade.php



```
@endit  
<br><br>  
<input type="submit" value="{{ $modo }}" Registro">
```

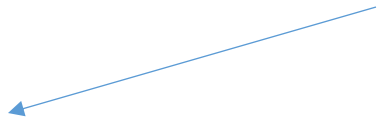
### Create.blade.php

```
<form action="{{ url('/empleados') }}" method="POST" enctype="multipart/form-data">  
    @csrf  
    @include('empleados.form', ['modo' => 'Guardar'])  
</form>
```



### Update.blade.php

```
@csrf  
{{ method_field('PATCH') }}  
@include('empleados.form', ['modo' => 'Actualizar']);
```



## FORMULARIO PARA CREAR EMPLEADOS [Listar Empleados](#)

Formulario que tendra los datos para crear o actualizar los empleados

Introduzca Nombre
Introduzca Primer Apellido
Introduzca Segundo Apellido
Introduzca Email
Seleccionar archivo

Ninguno archivo selec.

Guardar Registro

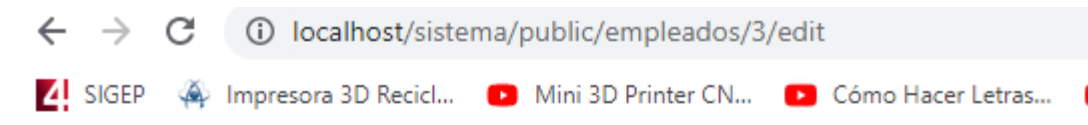


Actualizar Registro ;

También le podemos pasar el modo a un título para saber en qué formulario nos encontramos.

### Form.blade.php

```
sistema > resources > views > empleados > form.blade.php > ...  
1  Formulario que tendra los datos para crear o actualizar los empleados  
2  <br>  
3  <h1>{{ $modo }} Empleados</h1>
```



FORMULARIO PARA ACTUALIZAR LOS DATOS DE LOS EMPLEADOS  
Formulario que tendra los datos para crear o actualizar los empleados

## Actualizar Empleados

Cesar	
esquivela	
Cortes	
esqui@78.com	
Seleccionar archivo	Ninguno archivo selec.

## 21.APLICAR BOOTSTRAP Y LOGIN

Abrir la terminal y ubicar la carpeta del proyecto a trabajar, para crear la instancia del login

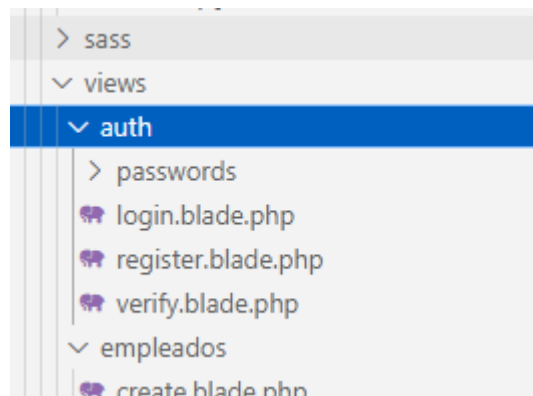
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\xampp\htdocs\sistema> composer require laravel/ui
Info from https://repo.packagist.org: #StandWithUkraine
./composer.json has been updated
Running composer update laravel/ui
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking laravel/ui (v4.2.2)
Writing lock file
```

Ahora integramos Bootstrap con el formulario de autenticación

```
12 | <div><div>
13 | <input type="submit" value="{{ $modo }}" Registro">
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\xampp\htdocs\sistema> php artisan ui bootstrap --auth
[INFO] Authentication scaffolding generated successfully.
[INFO] Bootstrap scaffolding installed successfully.
[WARN] Please run [npm install && npm run dev] to compile your fresh scaffolding.
PS C:\xampp\htdocs\sistema>
```

Se creará la carpeta **auth** en las vistas



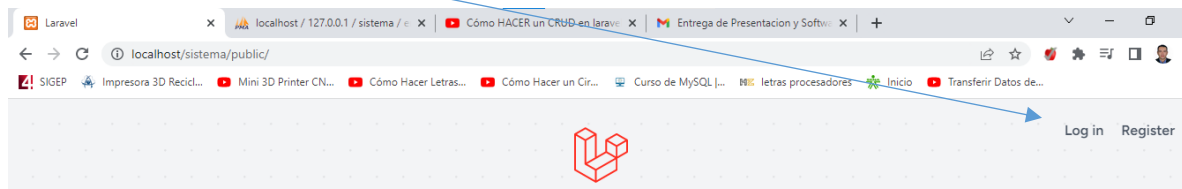
Recordemos que debemos tener instalado **Nodejs** para integrar con el Bootstrap. Ejecutamos en la consola el **npm**

```
13 | <input type= submit value= {{{modo}}} Registro /
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\xampp\htdocs\sistema> php artisan ui bootstrap --auth
INFO Authentication scaffolding generated successfully.
INFO Bootstrap scaffolding installed successfully.
WARN Please run [npm install && npm run dev] to compile your fresh scaffolding.
PS C:\xampp\htdocs\sistema> npm install
[.....] | idealTree:sistema: sill idealTree buildDeps
```

Y luego ejecutamos **npm run dev**

```
13 | <input type="submit" value="{{ $modo }}" Registro">
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
VITE v4.3.9 ready in 505 ms
→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h to show help
LARAVEL v10.11.0 plugin v0.7.8
→ APP_URL: http://localhost
```

Vamos al navegador y dejamos la ruta hasta el public y vemos que se crea la opción **login y registro**. De esta forma queda integrado el login de Bootstrap.





## 22.CONFIGURAR LA AUTENTICACION DEL USUARIO.

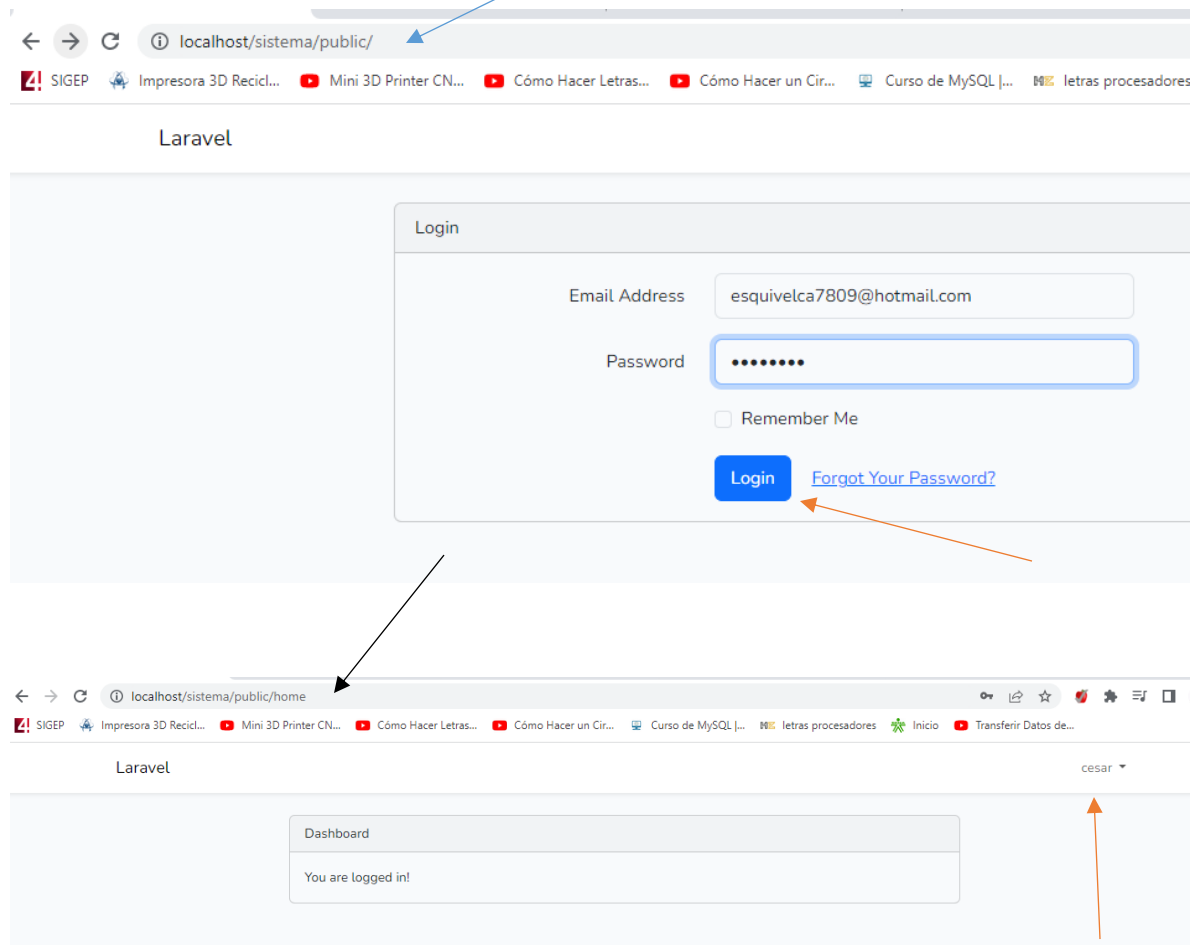
Ingresamos a la opción **register** y registramos un usuario. Podemos ver que inmediatamente el usuario puede loguearse. Esos datos quedan en la base de datos en la tabla **user**.

The image shows two screenshots of a web application running on a browser. The top screenshot displays the registration page at `localhost/sistema/public/register`. It features a form with the following fields: "Name" (highlighted with a blue border), "Email Address", "Password", and "Confirm Password". A blue "Register" button is located at the bottom of the form. The bottom screenshot shows the home page at `localhost/sistema/public/home`. It displays a "Dashboard" section with the message "You are logged in!". A blue arrow points from the "Register" button in the top screenshot to the "cesar" dropdown menu in the bottom screenshot, indicating the user's login status after registration.

Ahora debemos integrar las opciones de registro y login a nuestro proyecto.

Necesitamos que cuando el usuario escriba la url del proyecto lo direcciona al formulario de autenticación, esto se hace en el archivo web.php

```
13 |  
14 */  
15  
16 Route::get('/', function () {  
17     return view('auth.login');  
18 });  
19
```

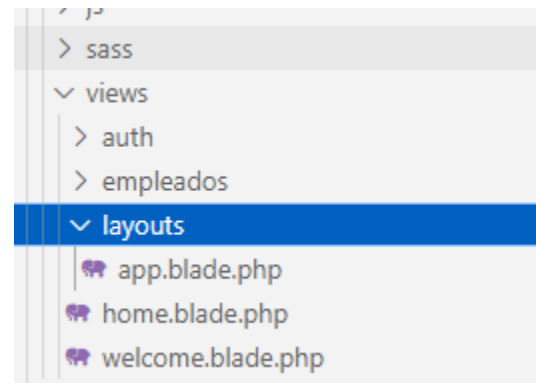


Luego hacemos unos cambios en las rutas ya creadas para que nos dirija al CRUD del proyecto, desde el EmpleadoController.

```
27
28 Route::resource('empleados', EmpleadoController::class);
29
30 Auth::routes();
31
32 Route::get('/home', [EmpleadoController::class, 'index'])->name('home');
33
34 // cuando el usuario se loguee busca (keyword) function el controlador y busca la clase index para ejecutarla
35 Route::group(['middleware'=>'auth'], function(){
36     Route::get('/', [EmpleadoController::class, 'index'])->name('home');
37 });
38
```

## 23.UTILIZAR LOS TEMPLATES.

Dentro de la carpeta Views encontramos la carpeta **layouts** que es la estructura Web con todas las plantillas que podemos ejecutar con Bootstrap para que se vea más presentados nuestros formularios. Debemos invocar este archivo en cada uno de los formularios.



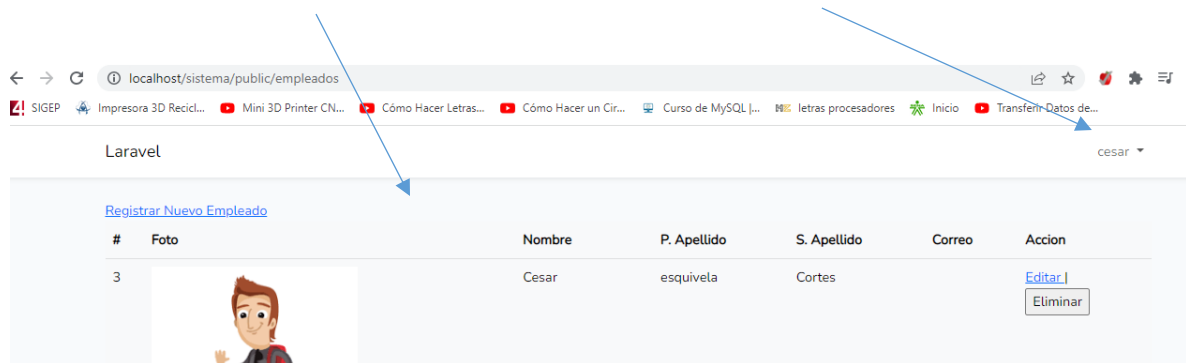
## Index.blade.php

```
web.php 1 index.blade.php X
sistema > resources > views > empleados > index.blade.php > ...
1 @extends('layouts.app')
2 @section('content')
3 <div class="container">
4
5
6 {{-- //recibe la funcion mensaje desde el controler para mostrar un m
7 @if(Session::has('mensaje'))
8 | {{Session::get('mensaje')}}
9 @endif
```

## Fin del index

```
51 </table>
52 </div>
53 @endsection
54 |
```

Al actualizar el navegador vemos que ya se integró el Bootstrap dando una mejor apariencia visual



Integrar el Bootstrap a nuestros formularios de crear y editar.

```
web.php 1 x index.blade.php create.blade.php update.blade.php x form.blade.php
sistema > resources > views > empleados > update.blade.php > div.container > a
1 @extends('layouts.app')
2 @section('content')
3 <div class="container">
4     <a href="{{url('/empleados')}}">Regresar </a>
5     <form action="{{url('/empleados/'.$empleado->id)}}" method="POST" enctype="multipart/form-data">
6         @csrf
7         {{method_field('PATCH')}}
8         @include('empleados.form',['modo'=>'Actualizar'])
9     </form>
10 </div>
11 @endsection
12
13
```

```
web.php 1 index.blade.php create.blade.php x
sistema > resources > views > empleados > create.blade.php > div.container
1 @extends('layouts.app')
2 @section('content')
3 <div class="container">
4     <a href="{{url('/empleados')}}">Listar Empleados </a>
5     <form action="{{url('/empleados')}}" method="POST" enctype="multipart/form-data">
6         @csrf
7         @include('empleados.form',['modo'=>'Crear'])
8     </form>
9
10 </div>
11 @endsection
12
13
```

## 24.AJUSTAR LAS OPCIONES DE SEGURIDAD

Como estamos utilizando un proyecto con formulario de login, necesitamos validar que el usuario ingrese los datos de validación para entrar al sistema, en caso que copie la URL y la pegue en el navegador no debemos permitir que entre al sistema.


Desde el archivo **web.php** ingresamos el comando **middleware**, para que se respete la autenticación del usuario.

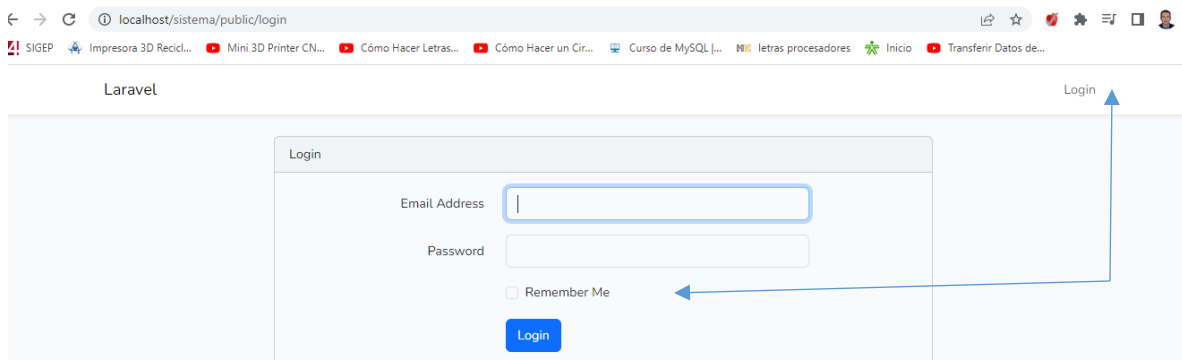
```
27 // Route::get('/empleados/create',[EmpleadoController::class,'create']);
28
29 Route::resource('empleados', EmpleadoController::class)->middleware('auth');
30
```

Ahora si pegamos una url sin loguearnos, nos redireccionará al formulario del login.

Si desea quitar la opción de Registrar y recordar contraseña, lo configuramos desde el **web.php**

```
25
26 // Route::get('/empleados/create',[EmpleadoController::class,'create']);
27
28 Route::resource('empleados', EmpleadoController::class)->middleware('auth');
29
30 Auth::routes(['register'=>false, 'reset'=>false]);
31
```



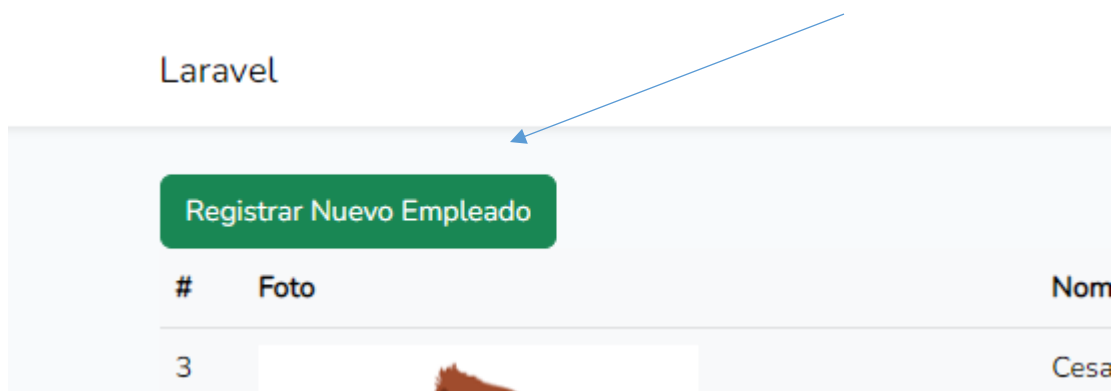


## 25. APLICAR ALGUNOS ESTILOS A LOS FORMULARIOS.

```


10
11 <a href="{{url('/empleados/create')}}" class="btn btn-success">Registrar Nuevo Empleado </a>
12
13 <table class="table table-light">

```



En adelante podrá aplicar los estilos Bootstrap que desee a los formularios.

Registrar Nuevo Empleado

#	Foto	Nombre	P. Apellido	S. Apellido	Correo	Accion
3		Cesar	esquivela	Cortes		<button>Editar</button>   <button>Eliminar</button>
4		Cesar	esquivel	Cortes		<button>Editar</button>   <button>Eliminar</button>

## 26. VALIDACION DE CAMPOS.

Illuminate \ Database \ QueryException

SQLSTATE[HY000]: General error: 1364 Field 'Foto' doesn't have a default value

```
INSERT INTO `empleados` (`Nombres`, `PrimerApel`, `SegundoApel`, `Correo`) VALUES (?, ?, ?, ?)
```

Si intentamos guardar un registro sin agregarle datos vemos que nos arroja un error en la consulta SQL del Insert porque no está recibiendo valores.

Ubíquese en el controlador luego en la función **store**, cree una variable que almacene un **array** y dentro asigne los campos que desea validar con el tipo de dato y la cantidad de caracteres que aceptará el campo que está validando.

```
0 references | 0 overrides
public function store(Request $request)
{
    // validacion de los campos al crear un registro

    $validacion = [
        'Nombres'=>'required|string|max:90',
```



En un nuevo arreglo ingresamos los mensajes que queremos presentar al momento de validar. El primero nos va a mostrar uno a uno los campos que estén vacíos y el ultimo nos valida solo el de la foto. Al final los llamamos.

```
$msj=[
    'required'=>'El :attribute es requerido',
    'Foto.required'=>'La Foto es requerida'
];

$this-> validate($request, $validacion, $msj);
```

Ahora en el **form.blade.php** hacemos un recorrido con un ciclo de los campos que estén vacíos y presentamos los errores.

```
@if(count($errors)>0)
    <div class="alert alert-danger" role="alert">
        <ul>
            @foreach($errors->all() as $error)
                <li>{{$error}}</li>
            @endforeach
        </ul>
    </div>
@endif
```

## Crear Empleados

- El nombres es requerido
- El primer apel es requerido
- El segundo apel es requerido
- El correo es requerido
- La Foto es requerida

## 27.RECUPERAR LOS DATOS QUE EL USUARIO HAYA DIGITADO AL MOMENTO DE VALIDAR CAMPOS VACIOS.

Este ejercicio se aplica en el formulario crear en caso que el usuario solo digite el nombre vamos a guardar el dato que haya ingresado y que no desaparezca del formulario cuando se validen campos vacíos.

```
<div class="form-group">  
  <input class="form-control" type="text" value="{{isset($empleado->Nombres)?$empleado->Nombres:old('Nombres')}}"  
    name="Nombres" id="Nombres" placeholder="Introduzca Nombre"><br>  
</div>
```

### Crear Empleados

- El primer apel es requerido
- El segundo apel es requerido
- El correo es requerido
- La Foto es requerida

carlos andres

Introduzca Primer Apellido

Introduzca Segundo Apellido

Introduzca Email

Seleccionar archivo Ninguno archivo selec.

Crear Registro

Como notaran se escribió Carlos Andrés pero cuando valido los otros campos no desapareció el nombre digitado.


Validar los campos en el formulario Actualizar. Es esta ocasión nos ubicamos en la función **update**. La validación de campos la hacemos igual que en el formulario crear, con la diferencia debemos verificar si ya existe una imagen o no.

```
U references | U overrides
public function update(Request $request, $id)
{
    //
    $validacion = [
        'Nombres'=>'required|string|max:90',
    ];

    $msj=['required'=>'El :attribute es requerido'];

    if($request->hasFile('Foto')){
        $validacion = ['Foto'=>'required|max:10000|mimes:jpg,png,jpeg'];
        $msj=['Foto.required'=>'La Foto es requerida'];
    }

    $this-> validate($request, $validacion, $msj);
}
```



## 28. REDIRECCIONAR Y OCULTAR MENSAJES

Empecemos con el formulario de Edición. En el controlador buscamos la función **update** y al final enviamos el mensaje.

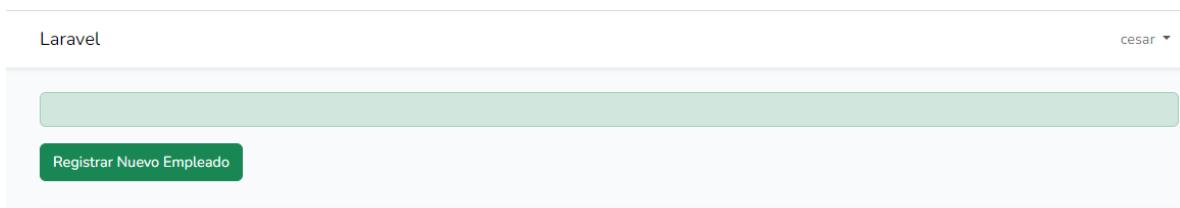
```
//cuando se edite enviamos un mensaje y redireccionamos al index  
  
return redirect('empleados')->with('mensaje','Registro eliminado exitosamente');
```

Luego en el **index** insertamos Bootstrap para que el mensaje enviado se vea de una mejor forma.

```
{{-- //recibe la funcion mensaje desde el controler para mostrar un mensaje de confirmacion --}}  
@if(Session::has('mensaje'))  
    <div class="alert alert-success alert-dismissible" role="alert">  
        {{Session::get('mensaje')}}  
    </div>  
@endif
```

## 29. AJUSTAR EL MENU DE NAVEGACION

Vamos a modificar el **Template** de Laravel que como se mencionó anteriormente lo encontramos en la carpeta **layouts**, **app.blade.php** y ubicamos la clase *navbar*, esto con el fin de poder modificar la apariencia del menú superior.





## 30. PAGINACION

Nos permite mostrar una cantidad de registros en un formulario y si se pasa de esta cantidad mostrar otra vista.

Ubíquese en la carpeta **Providers** archivo **AppServiceProvider.php**

```
2
3 namespace App\Providers;
4
5 use Illuminate\Support\ServiceProvider;
6 use Illuminate\Pagination\Paginator;
7
```

```
21 public function boot(): void
22 {
23     //instanciamos el metodo Bootstrap para poder paginar
24     Paginator::useBootstrap();
25 }
26
27
```

Ahora vamos al **index** al final del archivo y llamamos la función de paginación

```
55 |         @endforeach
56 |     </tbody>
57 | </table>
58 | {!! $empleados->links() !!}
59 | </div>
60 | @endsection
61 |
```

Laravel Empleado

Registrar Nuevo Empleado

#	Foto	Nombre	P. Apellido	S. Apellido	Correo	Accion
4		Cesar	esquivel	Cortes		<a href="#">Editar</a>   <a href="#">Eliminar</a>

[<](#) [1](#) [2](#) [>](#)

