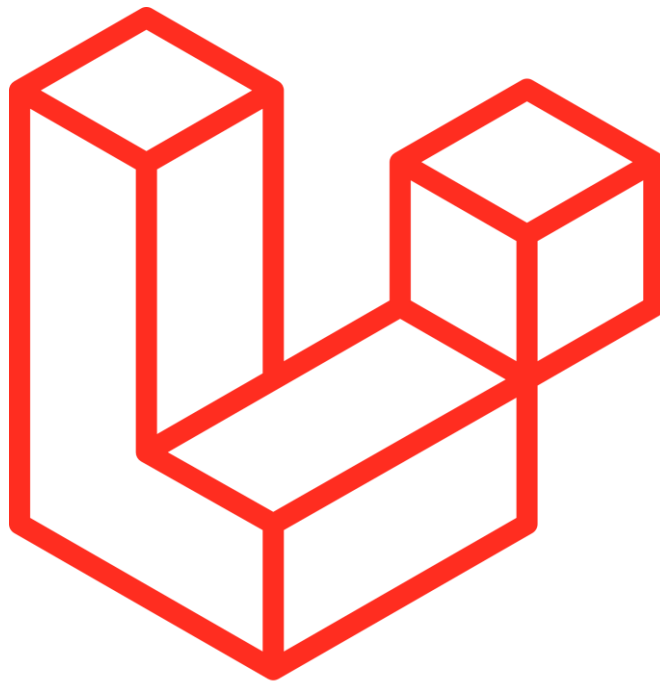


# Dossier Laravel y ficheros



## Índice

Práctica 1 .....	3
Practica 2 .....	4
Práctica 3 .....	5
Práctica 4 .....	5
Práctica 5 .....	6
Práctica 6 .....	8
Práctica 7 .....	8
Práctica 8 .....	9
Práctica 9 .....	9
Práctica 10 .....	10
Práctica 11 .....	11
Práctica 12 .....	13
Práctica 13 .....	14
Práctica 14 .....	15
Práctica 15 .....	16
Práctica 16 .....	16
Práctica 17 .....	18
Práctica 18 .....	20
Práctica 19 .....	22
Práctica 20 .....	24

**Práctica 1**

Modificar el fichero web.php para que las peticiones GET (parecido al ejemplo anterior) a la raíz de la aplicación: “/” muestren un mensaje que diga: “Under construction”

```
Route::get('/', function () {  
    echo "Under construction";  
    die();  
});
```



127.0.0.1:8000

Under construction

**Practica 2**

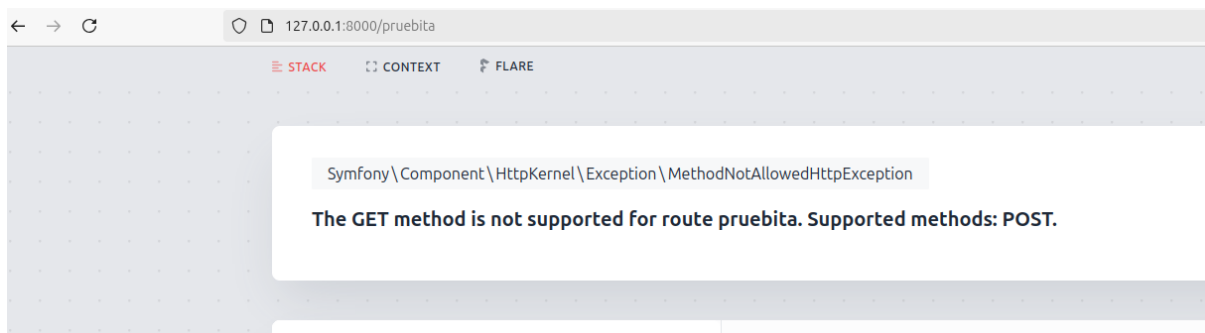
Modificar el fichero web.php para que las peticiones POST a: /pruebita muestren el mensaje: “se ha ejecutado una petición POST a la dirección: /pruebita”  
Probar a hacer la petición POST ¿muestra lo solicitado? ¿qué ocurre si se hace mediante

una petición GET? Volver a reestablecer la protección CSRF y hacer de nuevo la petición

POST ¿qué muestra ahora?

Nota: Desde Rested se puede enviar la petición POST, así como Postman o cualquier otra utilidad. Incluso desde el inspector de Firefox permite modificar una petición GET a POST y reenviar. En el inspector de firefox al editar la petición y pasar a post te da la respuesta en el “preview” no en la página renderizada del navegador

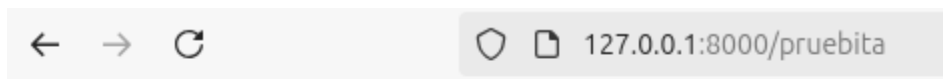
```
Route::post('/pruebita', function () {  
    echo "se ha ejecutado peticion a /pruebita";  
    die();  
});
```



No me permite hacer post, voy a probar poniendo get.

Si hago get sí me lo muestra

```
Route::get('/pruebita', function () {  
    echo "se ha ejecutado peticion a /pruebita";  
    die();  
});
```



### Práctica 3

Crear una ruta para TODA petición (ya sea GET, POST, ...) hacia /relatos/numero (recordar que hemos visto una opción para recoger todo tipo de petición). De tal forma que numero deba ser un número y muestre el mensaje: “petición recibida para el parámetro: numero”

```
Route::any('/relatos/{numero}', function ($numero) {  
    echo "peticion recibido para el parametro: $numero";  
    die();  
})-> where('numero', '[0-9]+');
```



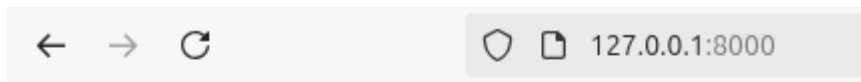
Como podemos observar ponemos en {numero} de la ruta el número 1, por ejemplo, y nos muestra en la función el parámetro recibido en ruta **\$numero**-> se le asigna **1**

### Práctica 4

Crear una ruta para el raíz: “/” En una primera implementación mostrará el mensaje: “página raíz de nuestra aplicación” que se resolverá en el propio web.php Haremos una segunda versión de esta actividad en la que redireccionará hacia el controlador y la función pertinente y allí se mostrará un mensaje que indique adicionalmente que se ha respondido desde el controlador

Desde el controlador:

```
Route::get('/', function () {  
    echo "página raíz de nuestra aplicación";  
    die();  
});
```



página raíz de nuestra aplicación

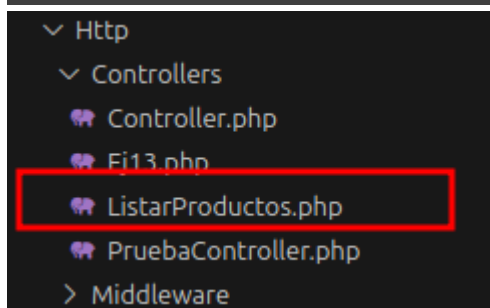
### Práctica 5

Crear un controlador llamado: ListarProductos que sea redireccionado en web.php cuando se acceda a la raíz: "/" y muestre un mensaje que diga: "Ejecutando el controlador ListarProductos mediante get". (si la llamada fue get. En el caso de que la llamada fuera post deberá decirlo)

Usamos php artisan make:controller ListarProductos en terminal para crear el controlador.

```
aed@aed:~/Melissa-AED/proyecto-laravel/nombreproyecto$ php artisan make:controller ListarProductos
PHP Deprecated:  Illuminate\Log\Logger::__construct(): Implicitly marking parameter $dispatcher as nullable is deprecated, the explicit nullable type must be used instead in /home/aed/Melissa-AED/proyecto-laravel/nombreproyecto/vendor/laravel/framework/src/Illuminate/Log/Logger.php on line 46

INFO Controller [app/Http/Controllers/ListarProductos.php] created successfully.
```

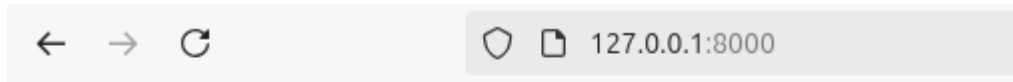


```
class ListarProductos extends Controller
{
    public function mostrarget() {
        echo "Ejecutando el controlador ListarProductos mediante get";
    }

    public function mostrarpost() {
        echo "Ejecutando el controlador ListarProductos mediante POST";
    }
}
```

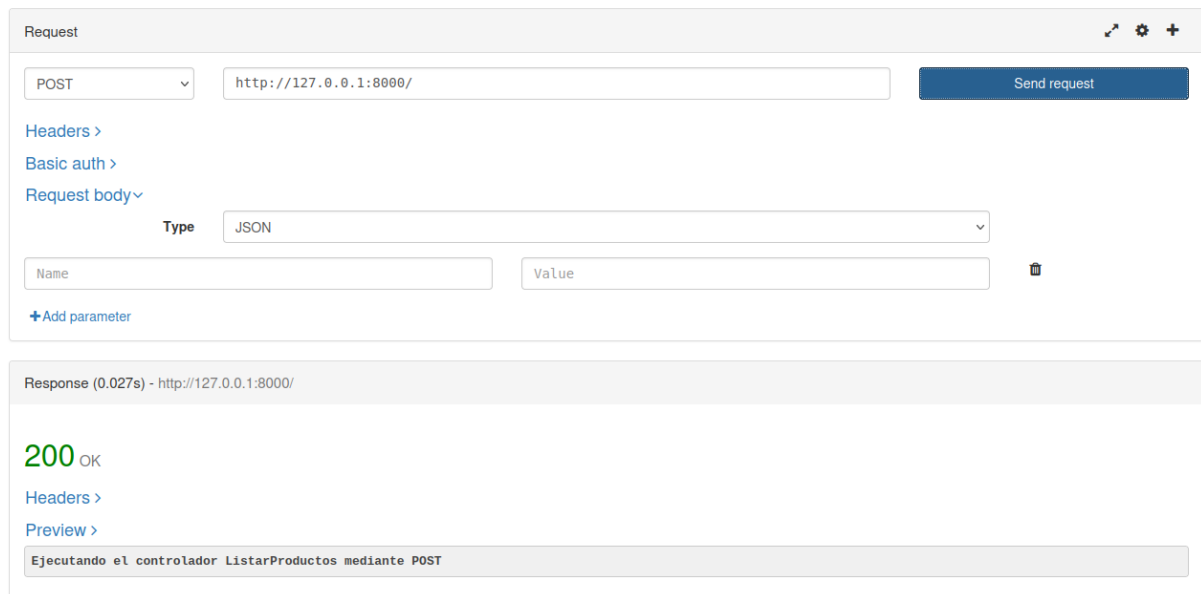
```
Route::get('/', [ListarProductos::class, 'mostrarget']);  
Route::post('/', [ListarProductos::class, 'mostrarpost']);
```

Si hacemos get nos indica el mensaje



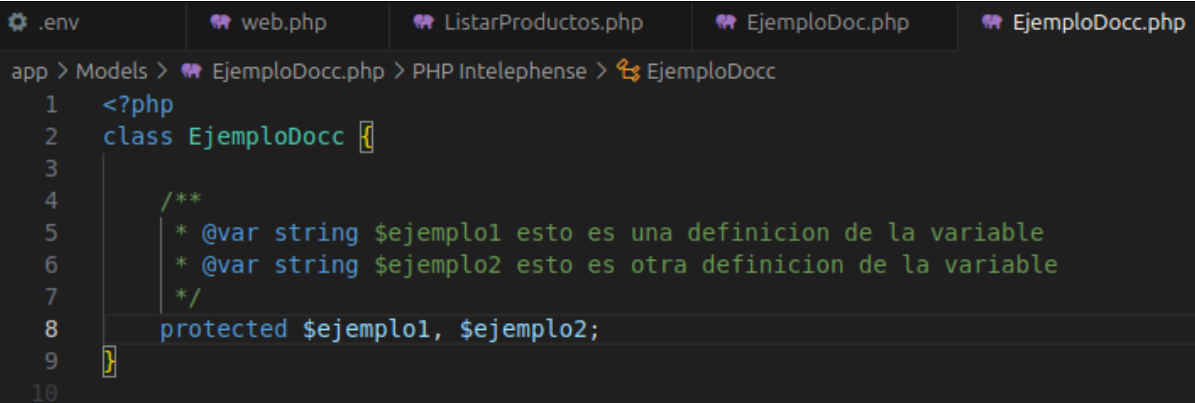
Ejecutando el controlador ListarProductos mediante get

Si hacemos post nos indica que hacemos post mediante rested por ejemplo al lanzar peticion post



**Práctica 6**

Comprobar que la anotación @var para un objeto permite que el ide con inteliphense nos ayude con los atributos y los métodos



The screenshot shows an IDE with several tabs: .env, web.php, ListarProductos.php, EjemploDoc.php, and EjemploDocc.php. The active file is EjemploDocc.php, which is open in the 'PHP Intelephense' view. The code defines a class 'EjemploDocc' with two protected attributes, '\$ejemplo1' and '\$ejemplo2', both annotated with '@var string'. The code is as follows:

```
1 <?php
2 class EjemploDocc {
3
4     /**
5      * @var string $ejemplo1 esto es una definicion de la variable
6      * @var string $ejemplo2 esto es otra definicion de la variable
7      */
8     protected $ejemplo1, $ejemplo2;
9 }
10
```

**Práctica 7**

Reproducir la vista descrita. Crear una tabla html por cada primo con un encabezado en la tabla que nos diga que campo estamos visualizando.



The screenshot shows a web browser address bar with the URL '127.0.0.1:8000/primos'. The browser interface includes back, forward, and refresh buttons.

primo: 1

primo: 2

primo: 3

primo: 5

primo: 7

primo: 11

primo: 13

primo: 17

primo: 19



**Práctica 8**

Agregar al comienzo de la vista el mensaje(sustituye por la hora/día actual):

Son las 17:53 del día: 29-11-2020

**Nota:** buscar información y usar la función PHP date()

```
<h2>Numeros primos:</h2>
<p>Son las {{ date('H:i') }} del día {{date('d/m')}} </p>
@foreach ($coleccion as $primo)
<p> primo: {{ $primo }} </p>
@endforeach
```



127.0.0.1:8000/primos

## Numeros primos:

Son las 14:33 del día 27/10

primo: 1

primo: 2

primo: 3

**Práctica 9**

El comando **sleep()** en php permite pausar la ejecución la cantidad de segundo especificada como parámetro. Modificar el ejemplo anterior para que lo muestre 3 veces con una espera de **1 segundo** entre una iteración y la siguiente, mostrando de forma actualizada la información de los segundos desde 1970

```
<h2>Tiempo desde 1970:</h2>

@for ($i = 0; $i < 3; $i++)
<p>Iteración {{ $i + 1 }}: Segundos desde 1970: {{ time() }}</p>
@php sleep(1); @endphp
@endfor
```



## Tiempo desde 1970:

Iteración 1: Segundos desde 1970: 1730040175

Iteración 2: Segundos desde 1970: 1730040176

Iteración 3: Segundos desde 1970: 1730040177

### Práctica 10

Generar una lista de números aleatorios de 0 a 100 en el controlador. Desde nuestra plantilla blade mostraremos primero la lista de números obtenidos menores de 50 y un poco más abajo en la página los mayores que 50. Hacer uso de las directivas @if para que al mostrar aquellos que sean mayores de 50

```
<h2>Números menores a 50:</h2>
@foreach ($array as $dato)
@if ($dato < 50)
<p>Dato par: {{ $dato }}</p>
@endif
@endforeach

<h2>Números mayores a 50:</h2>
@foreach ($array as $dato)
@if ($dato > 50)
<p>Dato impar: {{ $dato }}</p>
@endif
@endforeach
```



## Números menores a 50:

Dato par: 13

Dato par: 17

Dato par: 27

Dato par: 42

## Números mayores a 50:

Dato impar: 81

Dato impar: 81

Dato impar: 97

Dato impar: 94

Dato impar: 72

### Práctica 11

Enviar en un textarea una lista de palabras separadas por comas. Mostrar en una lista html esas palabras recibidas (una palabra por cada <li> de la lista ) convertidas todas a mayúsculas. Para ello se usará el bucle: @for (cuidado! No el foreach ) Observar que eso implicará “contar” el número de elementos que tiene la colección de palabras

```
public function showForm()
{
    return view('form');
}

public function processForm(Request $request)
{
    $input = $request->input('palabras');
```

```
$palabras = explode(',', $input);  
$palabras = array_map('trim', $palabras);  
$palabras = array_map('strtoupper', $palabras);  
  
return view('lista', compact('palabras'));  
}
```



## Ingresar Palabras

Ingresa palabras separadas por comas:

hola, que, tal

Enviar



## Lista de Palabras en Mayúsculas

- HOLA
- QUE
- TAL

**Práctica 12**

Ubicar imágenes en la carpeta descrita para las imágenes que quieras mostrar (mínimo 5).

Hacer que se visualicen en el navegador las imágenes en nuestra vista

**Nota:** Poner el renderizado que se considere más apropiado.

```
<body>
<p>Imágenes de la practica de ejemplo: </p>





</body>
```

← → ↺ 127.0.0.1:8000/practica12

Imágenes de la practica de ejemplo:



**Práctica 13**

Crear un formulario que envíe nombres de colores en cada ejecución del usuario. Obtendrá por respuesta una página con la lista de colores que ha ido introduciendo (usar session() para almacenar la lista de colores) (es un formulario post deberemos tener en cuenta @csrf leer más abajo)

```
<form action="{{ route('colores.store') }}" method="POST">
@csrf
<label for="color">Nombre del Color:</label>
<input type="text" id="color" name="color" required>
<button type="submit">Añadir Color</button>
</form>
```

```
<h2>Colores introducidos:</h2>
<ul>
@forelse ($colores as $color)
<li>{{ $color }}</li>
@empty
<p>No has introducido colores aún.</p>
@endforelse
</ul>
```

```
public function index()
{
$colores = session('colores', []);
return view('colores.index', compact('colores'));
}

public function store(Request $request)
{
$color = $request->input('color');

$colores = session('colores', []);
$colores[] = $color;
session(['colores' => $colores]);

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

```
Route::get('/colores', [Ej13::class, 'index']->name('colores.index'));
Route::post('/colores', [Ej13::class, 'store']->name('colores.store'));
```



# Lista de Colores

Nombre del Color:

## Colores introducidos:

- rojo

### Práctica 14

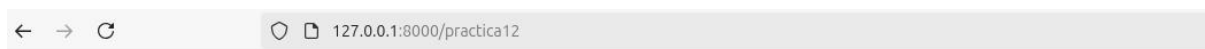
Una forma fácil de visualizar el token csrf es mediante: {{ csrf\_token() }}  
Introducir en la práctica 12 ese código y comprobar que está activo.

```
<p>Imágenes de la practica de ejemplo: </p>
<p>Token CSRF: {{ csrf_token() }}</p>





```



Imágenes de la practica de ejemplo:

Token CSRF: XryOsP5CLyaWCPBGHFOxKKqyvf7dNbVCxQdOPgK



**Práctica 15**

Crear un formulario POST Con los datos de un posible usuario (nombre, edad, gustos, etc. ) En cada ejecución de este formulario se le muestra al usuario la información almacenada del usuario en session() Observar que si se envía el formulario sin rellenar algún campo, se mantendrá la información anterior respecto a ese campo

```
<h1>Información del Usuario</h1>

<form action="{{ route('usuario.store') }}" method="POST">
@csrf

<label for="nombre">Nombre:</label>
<input type="text" id="nombre" name="nombre" value="{{ old('nombre', $usuario['nombre']) }}">
<label for="edad">Edad:</label>
<input type="number" id="edad" name="edad" value="{{ old('edad', $usuario['edad']) }}">
<label for="gustos">Gustos:</label>
<input type="text" id="gustos" name="gustos" value="{{ old('gustos', $usuario['gustos']) }}">
<button type="submit">Guardar Información</button>
</form>

<h2>Datos Guardados:</h2>
<p><strong>Nombre:</strong> {{ $usuario['nombre'] ?: 'No especificado' }}</p>
<p><strong>Edad:</strong> {{ $usuario['edad'] ?: 'No especificado' }}</p>
<p><strong>Gustos:</strong> {{ $usuario['gustos'] ?: 'No especificado' }}</p>
```

## Información del Usuario

Nombre:  Edad:  Gustos:

### Datos Guardados:

**Nombre:** Melissa

**Edad:** 27

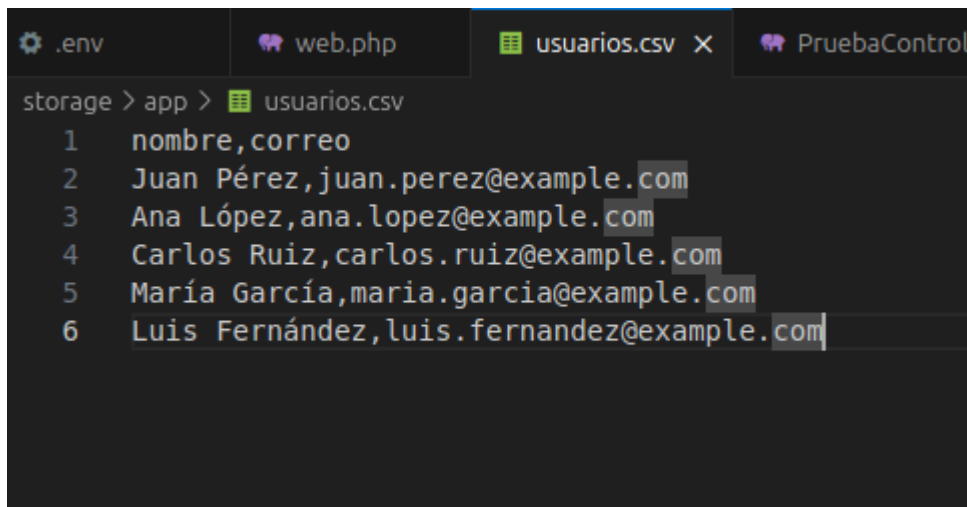
**Gustos:** Musica

**Práctica 16**

Crear un fichero con nombre y dirección de correo por fila ( en formato csv ) almacenado en Storage Leer el fichero y mostrarlo en pantalla



Creo un fichero de ejemplo:



```
.env web.php usuarios.csv X PruebaControl
storage > app > usuarios.csv
1 nombre,correo
2 Juan Pérez,juan.perez@example.com
3 Ana López,ana.lopez@example.com
4 Carlos Ruiz,carlos.ruiz@example.com
5 María García,maria.garcia@example.com
6 Luis Fernández,luis.fernandez@example.com
```

Creo funciones para lectura

```
public function store(Request $request)
{
    // Validar los campos, permitiendo valores vacíos
    $data = $request->only(['nombre', 'edad', 'gustos']);

    // Obtener los datos actuales de la sesión
    $usuario = session('usuario', [
        'nombre' => "",
        'edad' => "",
        'gustos' => "",
    ]);

    // Mantener los valores anteriores si los campos están vacíos
    $usuario['nombre'] = $data['nombre'] ?: $usuario['nombre'];
    $usuario['edad'] = $data['edad'] ?: $usuario['edad'];
    $usuario['gustos'] = $data['gustos'] ?: $usuario['gustos'];

    // Almacenar los datos actualizados en la sesión
    session(['usuario' => $usuario]);

    // Redirigir de nuevo a la página de usuario
    return redirect()->route('usuario.index');
}

public function showCSV()
{
    // Leer el contenido del archivo CSV
    $path = 'usuarios.csv';
    $file = Storage::get($path);
}
```

```
// Convertir el contenido en un array de líneas
$lines = explode("\n", trim($file));
$data = [];
// Procesar cada línea para extraer el nombre y el correo
foreach ($lines as $line) {
    $data[] = str_getcsv($line);
}
// Pasar los datos a la vista
return view('csv.show', compact('data'));
}
```



## Lista de Usuarios

Nombre	Correo
Juan Pérez	juan.perez@example.com
Ana López	ana.lopez@example.com
Carlos Ruiz	carlos.ruiz@example.com
María García	maria.garcia@example.com
Luis Fernández	luis.fernandez@example.com

### Práctica 17

Crear un formulario que se introduzca un nombre y cree un directorio en storage con ese nombre

```
public function index()
{
    return view('directorio.create');
}

public function store(Request $request)
{
    // Validar que el campo 'nombre' esté presente y no esté vacío
    $request->validate([
        'nombre' => 'required|string|max:255',
    ]);
}
```

```
// Obtener el nombre del directorio desde el formulario
$nombreDirectorio = $request->input('nombre');

// Crear el directorio en storage si no existe
Storage::makeDirectory($nombreDirectorio);

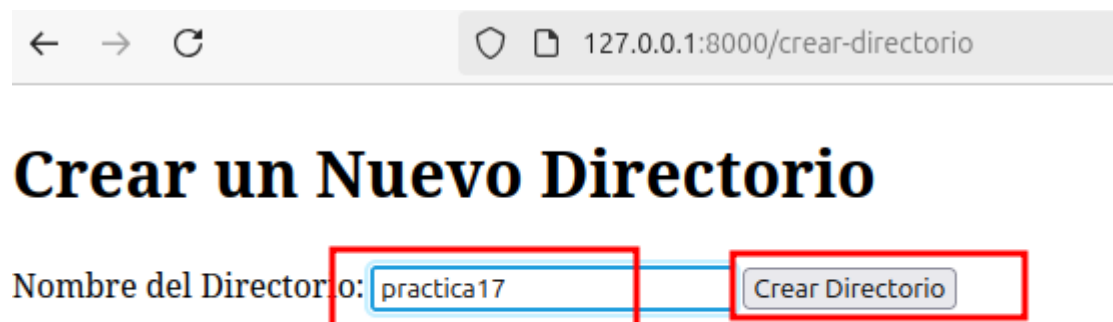
return redirect()->route('directorio.index')->with('success', "El directorio '$nombreDirectorio' ha
sido creado.");
}
<h1>Crear un Nuevo Directorio</h1>

@if (session('success'))
<p style="color: green;">{{ session('success') }}</p>
@endif

<form action="{{ route('directorio.store') }}" method="POST">
@csrf

<label for="nombre">Nombre del Directorio:</label>
<input type="text" id="nombre" name="nombre" required>

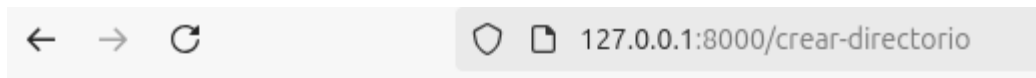
<button type="submit">Crear Directorio</button>
</form>
```



← → ↻ 127.0.0.1:8000/crear-directorio

## Crear un Nuevo Directorio

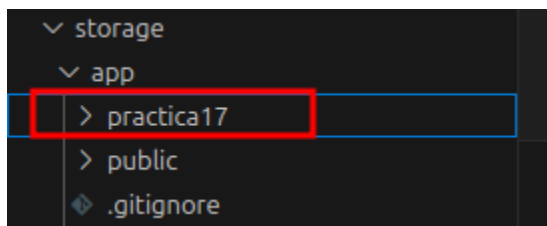
Nombre del Directorio:



# Crear un Nuevo Directorio

El directorio 'practica17' ha sido creado.

Nombre del Directorio:



## Práctica 18

Crear un fichero con nombre y dirección de correo por fila (en formato csv) almacenado en Storage Leer el fichero y mostrarlo en pantalla

```
<h1>Contenido del Archivo CSV</h1>

@if ($contenido && count($contenido) > 0)
<table>
<tr>
<th>Nombre</th>
<th>Correo</th>
</tr>
@foreach ($contenido as $row)
<tr>
<td>{{ $row[0] }}</td>
<td>{{ $row[1] }}</td>
</tr>
@endforeach
</table>
@else
<p>No hay datos en el archivo CSV.</p>
@endif

<a href="{{ route('csv.create') }}">Regresar</a>
```

# Crear Archivo CSV

Nombre:  Correo:

```
public function create()
{
    return view('create_csv');
}

public function store(Request $request)
{
    // Validación de los datos del formulario
    $request->validate([
        'name' => 'required|string',
        'email' => 'required|email',
    ]);

    // Nombre del archivo CSV
    $filename = 'usuarios.csv';

    // Crear o abrir el archivo CSV en modo de añadir
    $handle = fopen(storage_path("app/$filename"), 'a');

    // Escribir la línea de datos como un array
    fputcsv($handle, [$request->name, $request->email]);

    // Cerrar el archivo
    fclose($handle);

    return redirect()->route('csv.create')->with(['success' => 'Archivo CSV creado correctamente.']);
}

// Método para leer el archivo CSV
private function leerCsv($nombrefichero)
{
    $contenido = [];
    // Leer el archivo CSV
    if (($open = fopen(storage_path("app/$nombrefichero"), "r")) !== FALSE) {
        while (($data = fgetcsv($open, 1000, ",")) !== FALSE) {
            $contenido[] = $data; // Cada fila se convierte en un array
        }
    }
}
```

```
fclose($open);  
return $contenido;  
}  
return null;  
}
```

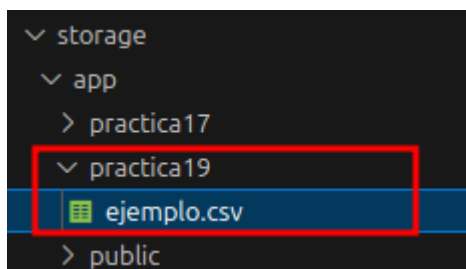
Vista

```
<h1>Contenido del Archivo CSV</h1>  
  
@if ($contenido && count($contenido) > 0)  
<table>  
<tr>  
<th>Nombre</th>  
<th>Correo</th>  
</tr>  
@foreach ($contenido as $row)  
<tr>  
<td>{{ $row[0] }}</td>  
<td>{{ $row[1] }}</td>  
</tr>  
@endforeach  
</table>  
@else  
<p>No hay datos en el archivo CSV.</p>  
@endif  
  
<a href="{{ route('csv.create') }}">Regresar</a>
```

### Práctica 19

Mostrar en una página una lista de ficheros de una carpeta en storage.  
Cuando se pulse en el nombre del fichero se descargará

Me creo un directorio de ejemplo dentro de una carpeta llamada practica 19



```
<h1>Lista de Archivos en 'practica19'</h1>  
@if (count($archivos) > 0)
```

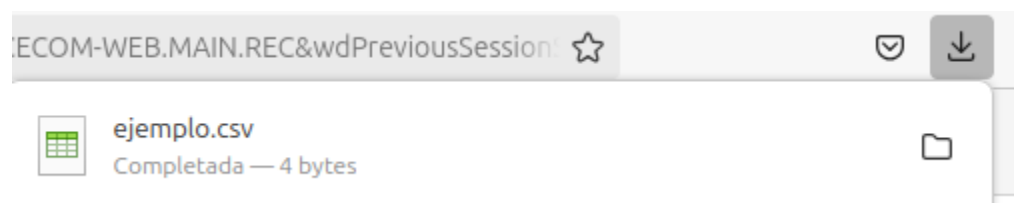
```
<ul>
@foreach ($archivos as $archivo)
<li>
<a href="{{ route('archivos.descargar', basename($archivo)) }}">
{{ basename($archivo) }}
</a>
</li>
@endforeach
</ul>
@else
<p>No hay archivos disponibles en la carpeta.</p>
@endif
```



## Lista de Archivos en 'practica19'

- [ejemplo.csv](#)

Al pulsar se descarga el archivo



Funciones:

```
public function listarArchivos()
{
// Verificar si la carpeta existe y listar los archivos
if (Storage::exists('practica19')) {
$archivos = Storage::files('practica19');
} else {
$archivos = []; // Si no existen archivos, se asigna un array vacío
}

return view('lista_archivos', compact('archivos'));
}
```

```
// Método para descargar un archivo
public function descargar($nombre)
{
    // Descarga el archivo especificado
    return response()->download(storage_path("app/practica19/$nombre"));
}
```

Rutas:

```
//PRACTICA 19
Route::get('/archivos', [FileController::class, 'listarArchivos'])->name('archivos.listar');
Route::get('/descargar/{nombre}', [FileController::class, 'descargar'])->name('archivos.descargar');
```

### Práctica 20

Continuando con el ejemplo anterior crear una opción para borrar los ficheros listados

Simplemente añadimos una función al controlador de archivos para borrar

```
public function borrar($nombre)
{
    $ruta = storage_path("app/practica19/$nombre");

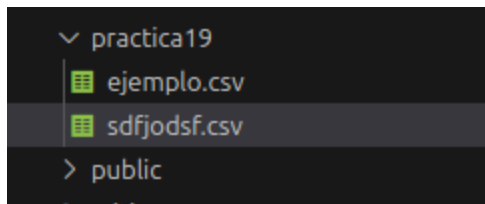
    if (file_exists($ruta)) {
        unlink($ruta); // Elimina el archivo
        return redirect()->back()->with('success', 'Archivo eliminado correctamente.');
```

Ruta:

```
//PRACTICA 20
Route::delete('/archivos/borrar/{nombre}', [FileController::class, 'borrar'])->name('archivos.borrar');
```

Si por ejemplo creamos un nuevo archivo como este:





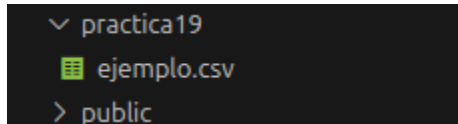
Y le damos a eliminar en nuestro form:



## Lista de Archivos

Nombre del Archivo	Acciones
ejemplo.csv	<input type="button" value="Eliminar"/> <a href="#">Descargar</a>
sdfjodsf.csv	<input type="button" value="Eliminar"/> <a href="#">Descargar</a>

Ya no nos aparece:



Vista:

```
<table>
<tr>
<th>Nombre del Archivo</th>
<th>Acciones</th>
</tr>
@foreach ($archivos as $archivo)
<tr>
<td>{{ basename($archivo) }}</td>
<td>
<form action="{{ route('archivos.borrar', basename($archivo)) }}" method="POST"
style="display:inline;">
@csrf
@method('DELETE')
<button type="submit" onclick="return confirm('¿Estás seguro de que deseas eliminar este
archivo?');">Eliminar</button>
</form>
```

```
<a href="{{ route('archivos.descargar', basename($archivo)) }}">Descargar</a>
</td>
</tr>
@endforeach
</table>
```