

---

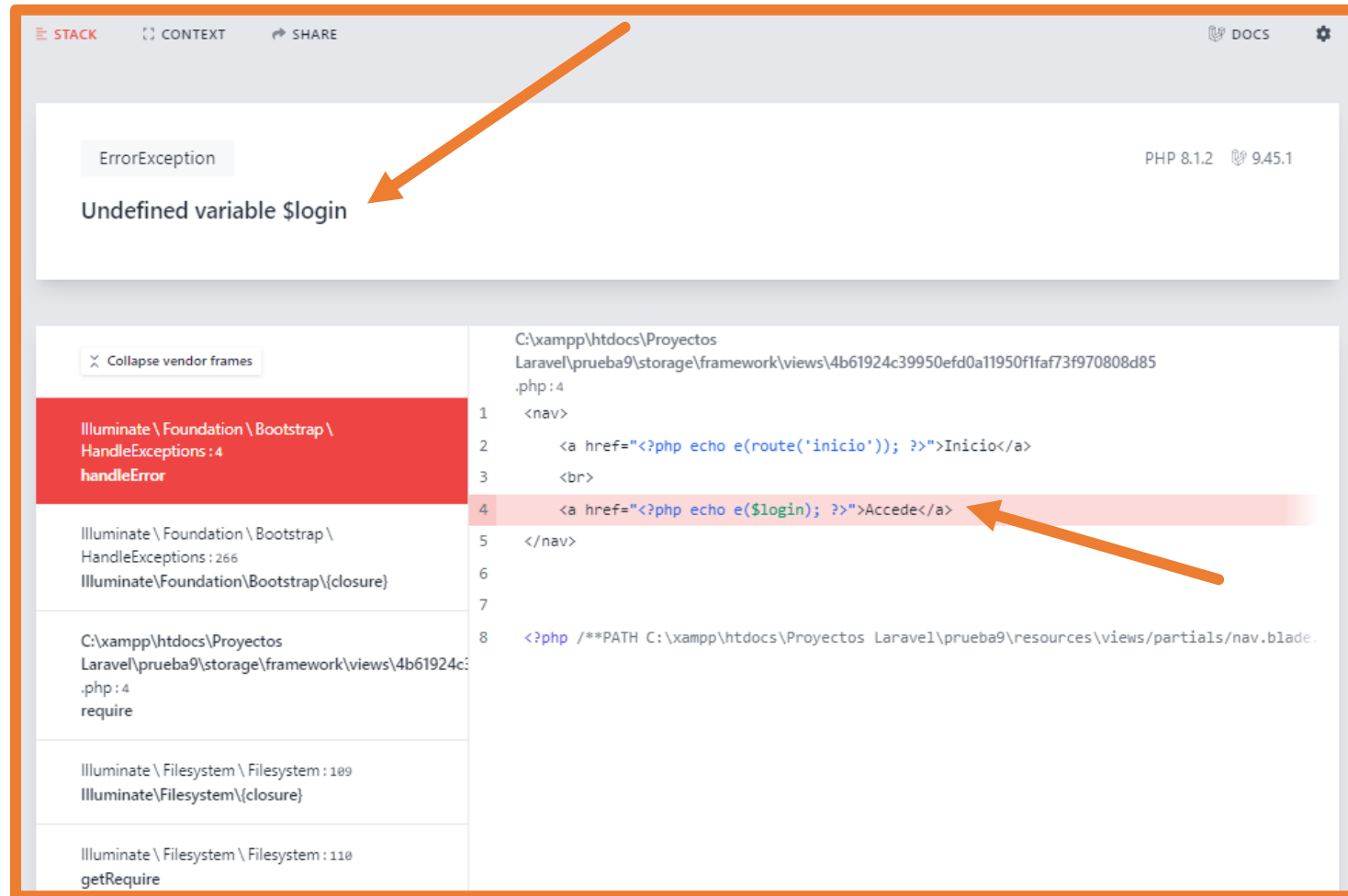
# UD10.2 – Laravel

## Rutas y Vistas

2º CFGS  
Desarrollo de Aplicaciones Web  
2022-23

# 0.- Errores en Laravel

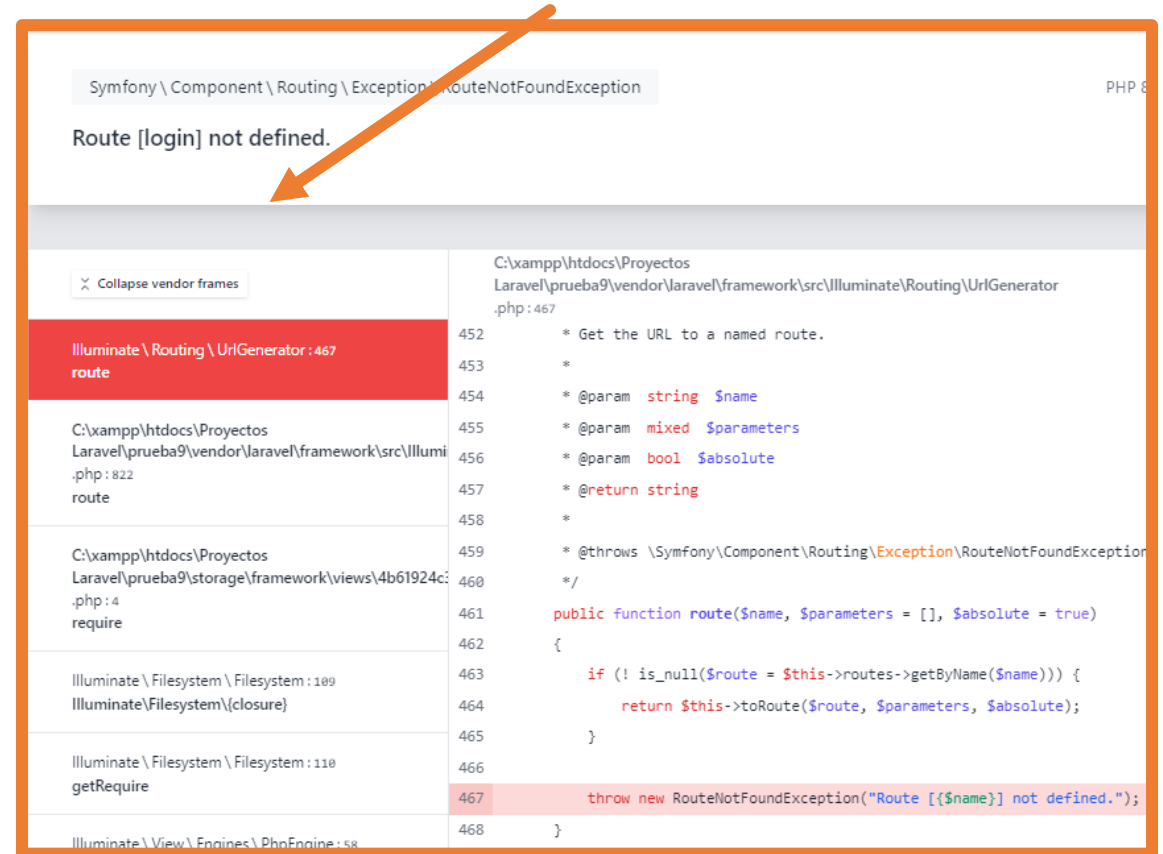
Laravel incorpora un sistema de gestión de errores: **ignition**.



# 0.- Errores en Laravel

Cuando se produce algún tipo de error en la ejecución de Laravel se mostrará una ventana con información del error.

En ocasiones es difícil encontrar el error pero esta pantalla ofrece mucha información para poder localizarlo navegando por el **stack** (pila de propagación del error).



The screenshot displays a PHP error stack trace for a `RouteNotFoundException`. At the top, the error message reads: `Symfony \ Component \ Routing \ Exception \ RouteNotFoundException` and `Route [login] not defined.`. An orange arrow points from this message to the first entry in the stack trace. The stack trace is a list of frames, each showing the file path and line number where the error occurred. The first frame is highlighted in red and points to the `route` method in `C:\xampp\htdocs\Proyectos\Laravel\prueba9\vendor\laravel\framework\src\Illuminate\Routing\UrlGenerator.php:467`. The code snippet for this frame shows the `route` method, which throws a `RouteNotFoundException` if a route is not found. The stack trace continues with other frames, including `Illuminate \ Filesystem \ Filesystem` and `Illuminate \ View \ Engines \ PhpEngine`.

```
Symfony \ Component \ Routing \ Exception \ RouteNotFoundException
Route [login] not defined.

Collapse vendor frames

Illuminate \ Routing \ UrlGenerator : 467
route

C:\xampp\htdocs\Proyectos
Laravel\prueba9\vendor\laravel\framework\src\Illuminate
.php: 822
route

C:\xampp\htdocs\Proyectos
Laravel\prueba9\storage\framework\views\4b61924c
.php: 4
require

Illuminate \ Filesystem \ Filesystem : 109
Illuminate \ Filesystem \ Filesystem \ (closure)

Illuminate \ Filesystem \ Filesystem : 110
getRequire

Illuminate \ View \ Engines \ PhpEngine : 58
```

```

452      * Get the URL to a named route.
453      *
454      * @param string $name
455      * @param mixed $parameters
456      * @param bool $absolute
457      * @return string
458      *
459      * @throws \Symfony\Component\Routing\Exception\RouteNotFoundException
460      */
461      public function route($name, $parameters = [], $absolute = true)
462      {
463          if (! is_null($route = $this->routes->getByName($name))) {
464              return $this->toRoute($route, $parameters, $absolute);
465          }
466
467          throw new RouteNotFoundException("Route [{ $name }] not defined.");
468      }


```

# 0.- Errores en Laravel

Cuando la aplicación web ya está publicada se debe desactivar el modo depuración para que no aparezcan estos errores.

Para ello en el archivo `.env` se debe cambiar la variable `APP_DEBUG` a falso

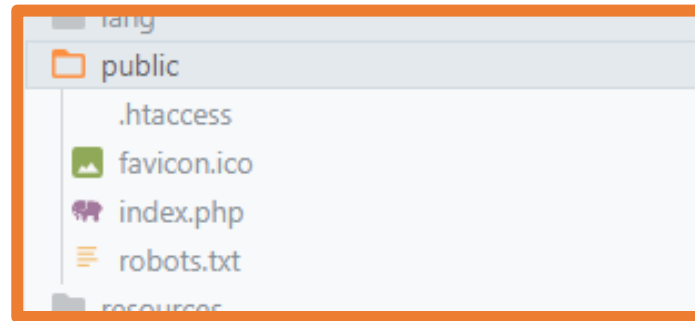
```
.env
1 APP_NAME=Laravel
2 APP_ENV=local
3 APP_KEY=base64:v1521T/a8h7f9/yr3LzKJNz1CENmx0DE2ED8VfHjT1A=
4 APP_DEBUG=true
5 APP_URL=http://localhost
```



500 | SERVER ERROR

# 1.- Rutas

Laravel implementa MVC y como se ha visto anteriormente, protege los archivos del servidor permitiendo solo el acceso a la carpeta public.



Así, de una manera parecida a cómo se estudió en la unidad "Modelo Vista Controlador", en un proyecto Laravel se tendrán que definir las **rutas (acciones)** disponibles y cómo tratar dichas rutas.

Las **rutas** en Laravel son todas las URL's a las que el servidor va a responder.

El sistema de rutas implementado en Laravel permite desarrollar la aplicación web usando directamente **URL's amigables**.

# 1.- Rutas

Existen dos tipos principales de rutas.

- Rutas de la **aplicación web**: permiten al cliente obtener las diferentes secciones de la aplicación web. Se definen en el archivo:

**routes/web.php**


- Rutas de la **API**: permiten definir servicios REST. Se definen en el archivo:

**routes/api.php**

# 1.- Rutas

Archivo: **routes/web.php**

```
routes > web.php
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  /*
6   |-----
7   | Web Routes
8   |-----
9   |
10  | Here is where you can register web routes for your application. These
11  | routes are loaded by the RouteServiceProvider within a group which
12  | contains the "web" middleware group. Now create something great!
13  |
14  */
15
16  Route::get('/', function () {
17      return view('inicio');
18  }->name('inicio');
19
```



Por defecto solo contiene una ruta, se deberán añadir todas las necesarias para el funcionamiento de la aplicación web.

# 1.- Rutas

Archivo: **routes/web.php**

Al principio se puede ver la instrucción **use** que permite "importar" elementos de un **namespace** para utilizarlos (parecido a include o require).

Los **namespaces** se usan frecuentemente en los frameworks ya que permiten encapsular elementos manteniéndolos agrupados y separados de otros con funcionalidad diferente.

```
routes > web.php
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  /*
6  |-----
7  | Web Routes
8  |-----
9  |
10 | Here is where you can register web routes for your application. These
```



## 2.- Rutas simples

Para definir una ruta en Laravel se necesita indicar tres aspectos:

- **Método:** get, post, put, patch, delete, options
- **URL:** patrón aceptado en la petición.
- **Cómo se va a responder a la petición.**

```
Route::get('fecha', function() {  
    return date('d/m/y h:i:s');  
});
```

## 2.- Rutas simples

Para definir una ruta en Laravel se necesita indicar tres aspectos:

- **Método:** get, post, put, patch, delete
- **URL:** patrón aceptado en la petición.
- **Cómo se va a responder a la petición.**

```
Route::get('fecha', function() {  
    return date('d/m/y h:i:s');  
});
```

## 2.- Rutas simples

### URL

```
Route::get('fecha', function() {  
    return date('d/m/y h:i:s');  
});
```

- Para aceptar la petición la URL tiene que ser **exacta**.
- Es **case sensitive**: distingue entre mayúsculas y minúsculas.
- Se usan **URL's amigables**.

### 3.- Redirigir rutas

Para actualizar una ruta simplemente se tiene que modificar la URL en su definición.

En ocasiones es interesante mantener durante un tiempo una ruta antigua pero que redirija a una nueva:

```
Route::redirect('date', 'fecha');
```

Por defecto redirect envía el código de estado 302 que indica una redirección temporal.

Si se quiere indicar que es una redirección permanente se puede indicar de dos modos:

```
Route::redirect('date', 'fecha', 301);
```

```
Route::permanentRedirect('date', 'fecha');
```

## 4.- Rutas con parámetros

En las rutas de Laravel se pueden pasar **parámetros** igual que en las URL's amigables.

```
Route::get('saludo/{nombre}', function($nombre) {  
    return 'Hola '. $nombre;  
});
```

```
Route::get('saludo/{nombre}/{apellido}', function($nombre, $apellido) {  
    return 'Hola '. $nombre . ' '. $apellido;  
});
```

```
Route::get('producto/{nombre}/id/{id}', function($nombre, $id) {  
    return 'Producto: '. $nombre . '(id:'. $id .)';  
});
```

```
Route::get('user/{id}/publicaciones', function($nombre) {  
    // Lógica de la ruta  
});
```

## 4.- Rutas con parámetros

### Parámetros opcionales.

Si se quieren poner parámetros opcionales se deben poner al final de la URL seguidos del carácter interrogante **?** y poner un **valor por defecto** a esos parámetros opcionales.

```
Route::get('saludo/{nombre}/{apellido?}', function($nombre, $apellido = null) {  
    return 'Hola '. $nombre .' '. $apellido;  
});
```

```
Route::get('pagar/{descuento?} ', function($descuento = 0) {  
    // lógica de la ruta.  
    return 'Compra realizada';  
});
```

## 4.- Rutas con parámetros

### Validación de parámetros.

En las rutas vistas anteriormente se puede observar que los parámetros pueden estar formados por cualquier carácter excepto /.

En ocasiones interesa que los parámetros cumplan unas determinadas condiciones.

```
Route::get('saludo/{nombre}', function() {  
    return 'Hola, ' . $nombre;  
})->where('nombre', 'ExpresiónRegular');
```

Si el parámetro no coincide con la expresión regular se devolverá un error 404.

## 4.- Rutas con parámetros

### Validación de parámetros.

```
Route::get('saludo/{nombre}', function() {  
    return 'Hola, ' . $nombre;  
})->where('nombre', '[A-Za-z\s]+');
```

```
Route::get('producto/{id}', function ($id) {  
    // lógica de la ruta  
})->where('id', '[0-9]+');
```

```
Route::get('producto/{id}/{nombre}', function ($id, $nombre) {  
    // lógica de la ruta  
})->where(['id' => '[0-9]+', 'nombre' => '[a-z]+']);
```

```
Route::get('producto/{id}/{nombre}', function ($id, $nombre) {  
    // lógica de la ruta  
})->where('id', '[0-9]+')->where('nombre ', '[a-z]+');
```



## 4.- Rutas con parámetros

### Validación de parámetros.

Algunos patrones son tan habituales que hay métodos para comprobarlos:

```
Route::get('producto/{id}/{nombre}', function ($id, $nombre) {  
    //  
})->whereNumber('id')->whereAlpha('nombre');
```

```
Route::get('producto/{nombre}', function ($nombre) {  
    //  
})->whereAlphaNumeric(nombre');
```

## 5.- Rutas con nombres

Es conveniente asignar **un nombre a una ruta** para usarla posteriormente tanto desde código PHP como desde HTML.

De esta manera, si se cambia la URL de la ruta, el código PHP y HTML seguirán siendo válidos.

```
Route::get('contacto', function () {  
    return 'Página de contacto';  
})->name('ruta_contacto');
```

Uso desde PHP:

```
return redirect()->route('ruta_contacto');
```

```
return to_route('ruta_contacto');
```

Uso desde HTML (al estudiar las plantillas Blade usadas por Laravel se explicará más este punto):

```
echo '<a href="/contacto">Contacto</a>';
```

```
<a href="<?= route('ruta_contacto') ?>">Contacto</a>
```

## 5.- Rutas con nombres

La ruta tiene un parámetro: href="/**producto/123**"

```
<a href="<?= route('producto', ['id' => 123]) ?>">Comprar</a>
```

La ruta tiene varios parámetros: href="/**producto/123/20**"

```
<a href="<?= route('producto', ['id' => 123, 'descuento' => '20']) ?>">Comprar</a>
```

La ruta tiene un parámetro pero se le pasan parámetros extra: href="/**producto/123**"

```
<a href="<?= route('producto', ['id' => 123, 'ofertas' => 'si']) ?>">Comprar</a>
```

Se convertirá en: /product/123?ofertas=si

## 6.- Rutas: combinación de opciones

Se pueden enlazar todas las opciones vistas anteriormente:

```
Route::get('saludo/{nombre?}/{id?}', function($nombre='Invitado', $id=0) {  
    return 'Hola '. $nombre .', tu código es el '. $id;  
})->where('nombre', '[A-Za-z]+')  
    ->whereNumber('id')  
    ->name('saludo');
```

## 7.- Listado de rutas

Mediante el siguiente comando se puede comprobar que las rutas están bien formadas y consultar la lista de rutas disponibles en la aplicación web.

```
# php artisan route:list
```

```
PS C:\xampp\htdocs\Proyectos Laravel\prueba9> php artisan route:list
```

```
GET|HEAD / .....  
POST _ignition/execute-solution ..... ignition.executeSolution > Spatie\LaravelIgnition > ExecuteSolutionController  
GET|HEAD _ignition/health-check ..... ignition.healthCheck > Spatie\LaravelIgnition > HealthCheckController  
POST _ignition/update-config ..... ignition.updateConfig > Spatie\LaravelIgnition > UpdateConfigController  
GET|HEAD api/user .....  
GET|HEAD contacto ..... ruta_contacto  
GET|HEAD saludo/{nombre?}/{id?} ..... saludo  
GET|HEAD sanctum/csrf-cookie ..... sanctum.csrf-cookie > Laravel\Sanctum > CsrfCookieController@show
```

Showing [8] routes

Si alguna ruta está mal formada el comando mostrará un error.

# Práctica

## **Actividad 2:** Primeras rutas.

## 8.- Grupos de rutas

Laravel ofrece la posibilidad de agrupar rutas con la finalidad de:

- **Agrupar tipos de petición HTTP**
- **Aplicar middleware**
- **Capturar subdominios**
- **Añadir prefijos**

## 8.- Grupos de rutas

### Agrupar tipos de petición HTTP

```
Route::match(['get', 'post'], '/', function () {  
    // La misma ruta tanto si es petición GET o POST  
});
```

```
Route::any('/', function () {  
    // La misma ruta para todo tipo de petición HTTP  
});
```



## 8.- Grupos de rutas

### Aplicar middleware

```
Route::middleware(['m1', 'm2'])->group(function () {  
    Route::get('', function () {  
        // Se aplican los middleware mid1 y mid2  
    });  
  
    Route::get('user/profile', function () {  
        // Se aplican los middleware mid1 y mid2  
    });  
});
```

Los middleware se explicarán más adelante.

## 8.- Grupos de rutas

### Capturar subdominios

```
Route::domain('admin.blog.com')->group(function () {  
    Route::get('', function () {  
        // Ruta: admin.blog.com  
    });  
  
    Route::get('user/{id}', function ($id) {  
        // Ruta: admin.blog.com/user/id_usuario  
    });  
});
```

Las rutas para los subdominios deben indicarse antes que las del dominio.

## 8.- Grupos de rutas

### Capturar subdominios

```
Route::domain('{subdominio}.blog.com')->group(function () {  
    Route::get('', function ($subdominio) {  
        //  
    });  
  
    Route::get('user/{id}', function ($subdominio, $id) {  
        //  
    });  
});
```

## 8.- Grupos de rutas

### Añadir prefijos

```
Route::prefix('admin')->group(function () {  
    Route::get('users', function () {  
        // Ruta: /admin/users  
    });  
  
    Route::get('users/{id}', function ($id) {  
        // Ruta: /admin/users/id_usuario  
    });  
  
    Route::get('posts', function () {  
        // Ruta: /admin/posts  
    });  
});
```

## 9.- Vistas

Se puede decir una vista es una página web dentro de todo el conjunto de páginas web de una aplicación web.

Hasta el momento se ha visto como una ruta puede devolver el contenido que se mostrará en una página web (return ...).

Esta práctica no tiene mucho sentido para mostrar páginas web completas:

- El archivo routes/web.php se haría muy largo e insostenible.
- No cumple con el patrón MVC.

## 9.- Vistas

Para separar la parte de visualización tal y como se indica en el patrón MVC Laravel ofrece la posibilidad de devolver una vista.

Así, las **vistas** serán **archivos con contenido HTML** que se devolverán cuando se reciba una petición a una ruta.

Al crear un proyecto Laravel en el archivo routes/web.php aparece una única ruta que devuelve una vista:

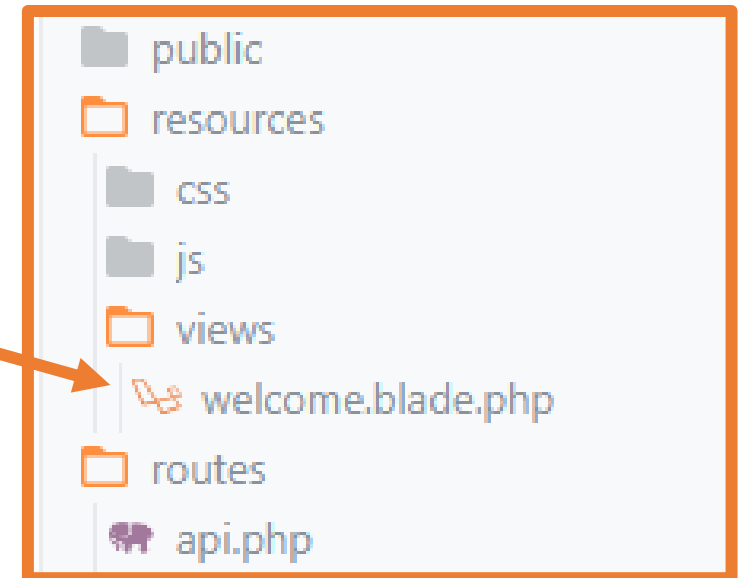
```
Route::get('', function () {  
    return view('welcome');  
});
```

## 9.- Vistas

La única ruta que tiene un proyecto Laravel nuevo indica que se ha de devolver la vista **welcome**.

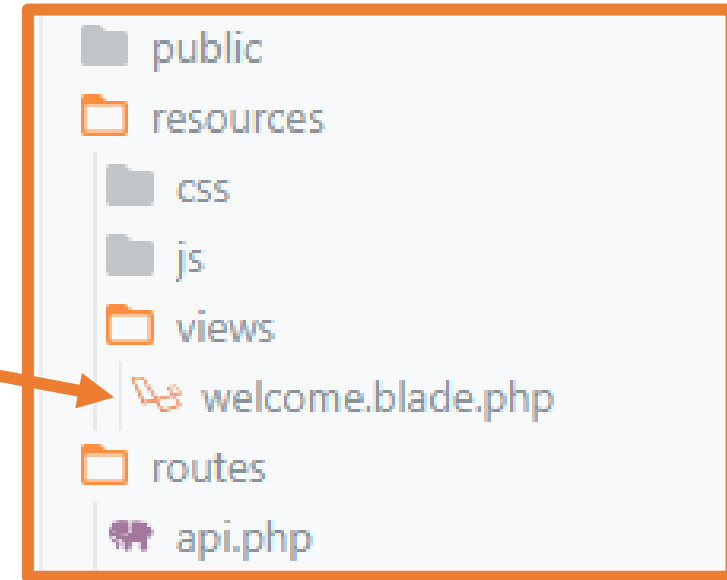
Esa vista se puede encontrar en el directorio **resources/views** que es donde se han de almacenar todas las vistas

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



## 9.- Vistas

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



No es necesario indicar la ruta hacia el archivo ni la extensión ya que Laravel asume que **las vistas se encuentran en resources/views**.

En Laravel todo archivo .php dentro de resources/views es una vista:

- .php → vista simple (únicamente código HTML y PHP).
- .blade.php → plantilla Blade (código HTML, Blade y PHP).



## 9.- Vistas

Para crear una vista se debe añadir un archivo nuevo

resources/views/inicio.blade.php

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Mi blog</title>
</head>
<body>
    Soy Rick, bienvenido a mi blog
    <br>
    <?php
        echo date('d/m/Y H:i:s');
    ?>
</body>
</html>
```

routes/web.php

```
Route::get('/', function () {
    return view('inicio');
})->name('inicio');
```

## 10.- Pasar valores a las vistas

Las vistas son como plantillas con huecos que rellenar.

Es habitual rellenar esos huecos con datos conseguidos desde la base de datos en base a los parámetros pasados a las rutas.

Como aún no se ha visto el acceso a BBDD, simplemente se verá cómo pasar valores a las vistas desde las rutas que es similar a hacerlo con datos recuperados de la BBDD.

Existen varias maneras de pasar valores a las vistas.

# 10.- Pasar valores a las vistas

La mejor opción es usando la función **compact** de PHP, pero las variables ya deben existir:

```
Route::get('saludo', function() {  
    $nombre = 'Rick';  
    return view('inicio', compact('nombre'));  
});
```

```
Route::get('saludo', function() {  
    $nombre = 'Rick';  
    $apellido = 'Sanchez';  
    return view('inicio', compact('nombre', 'apellido'));  
});
```

```
Route::get('saludo/{nombre}', function($nombre) {  
    return view('inicio', compact('nombre'));  
});
```

```
Route::get('saludo/{nombre}', function($nombre) {  
    $saludo = 'Hola buenas tardes';  
    return view('inicio', compact('nombre', 'saludo'));  
});
```

# 10.- Pasar valores a las vistas

Mediante un **array**:

```
Route::get('saludo', function() {  
    return view('inicio', ['nombre' => 'Rick']);  
});
```

```
Route::get('saludo', function() {  
    return view('inicio', ['nombre' => 'Rick', 'apellido' => 'Sanchez']);  
});
```

Mediante el método **with**:

```
Route::get('saludo', function() {  
    return view('inicio')->with('nombre', 'Rick');  
});
```

```
Route::get('saludo', function() {  
    return view('inicio')->with('nombre', 'Rick')  
        ->with('apellido', 'Sanchez');  
});
```

# 10.- Pasar valores a las vistas

Mediante el método **with** y un **array**:

```
Route::get('saludo', function() {  
    return view('inicio')->with(['nombre' => 'Rick']);  
});
```

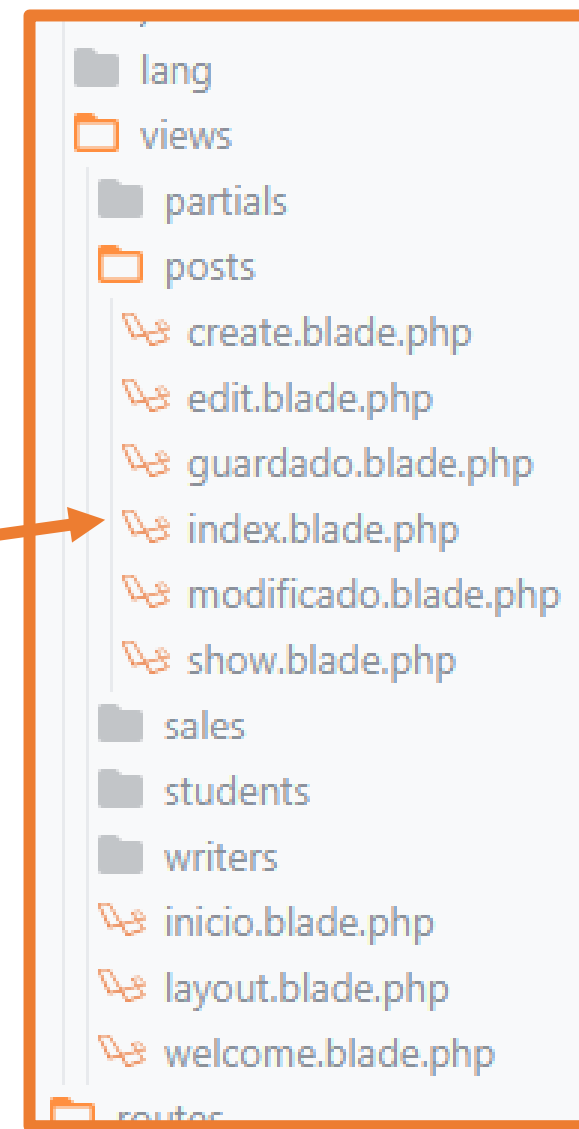
```
Route::get('saludo', function() {  
    return view('inicio')->with(['nombre' => 'Rick', 'apellido' => 'Sanchez']);  
});
```

## 11.- Organizar las vistas

Laravel permite organizar las vistas dentro de directorios.

Para devolver una vista que se encuentra dentro de un directorio simplemente se debe añadir el nombre del directorio y un punto delante del nombre de la vista.

```
return view('posts.index');
```



## 12.- Plantillas Blade

**Blade** es el sistema de plantillas que integra Laravel. Con Blade se **simplifica** la inclusión de variables y código no HTML en las vistas.

Las plantillas Blade son archivos con "doble" extensión **.blade.php** lo que significa que tienen las características tanto de un archivo PHP como de un archivo Blade:

- PHP: permite código HTML y código PHP.
- Blade: permite la sintaxis Blade.

## 12.- Plantillas Blade

Laravel **renderiza** las vistas y las almacena en storage/framework/views, así, volver a llamar a una vista renderizada no afecta negativamente al rendimiento de la aplicación.

Las vistas renderizadas solo se regeneran cuando se cambia el código en ellas.

Además al usar vistas renderizadas se evitan ataques **XSS** (Cross Site Scripting) que podrían **inyectar código JavaScript malicioso** en la aplicación web.

Las vistas se **renderizan automáticamente bajo demanda**, pero se puede **forzar la renderización** de todas las vistas de la aplicación web para mejorar el rendimiento:

```
# php artisan view:cache    // limpia la caché y compila todas las vistas  
  
# php artisan view:clear    // limpia la caché
```



## 12.- Plantillas Blade

A continuación, se van a estudiar los aspectos más importantes de la sintaxis de Blade.

En la documentación de Laravel se pueden encontrar muchas más funcionalidades.

<https://laravel.com/docs/9.x/blade>

## 12.- Plantillas Blade

### Comentarios

Aunque se pueden usar los comentarios HTML, dentro de las plantillas Blade es habitual usar los comentarios con sintaxis Blade.

#### Comentario HTML

```
<!-- Esto es un comentario HTML -->
```

#### Comentario Blade:

```
{{-- Esto es un comentario Blade --}}
```

## 12.- Plantillas Blade

### Mostrar variables

Mostrar el valor de una variable con php:

```
Hola <?php echo $nombre; ?>
```

```
Hola <?=$nombre?>
```

Mostrar el valor de una variable con Blade:

```
Hola {!!$nombre!!}
```

## 12.- Plantillas Blade

### Cargar subvistas

En ocasiones se pueden crear vistas parciales para incluir en otras vistas:

```
@include('cuenta.favoritos')
```

Las subvistas tendrán acceso a todas las variables de la vista principal, pero si se quiere pasar valores extra:

```
@include('cuenta.favoritos', ['estado' => 'oferta'])
```

Si se intenta incluir una vista que no existe Laravel lanza un error, para evitar esto:

```
@includeIf('cuenta.favoritos')
```

## 12.- Plantillas Blade

### Rutas para enlaces

Se pueden añadir rutas de manera directa:

```
<a href="/contacto">Contáctanos</a>
```

Como Laravel trabaja con rutas a las que se les asigna nombre, para crear enlaces a esas rutas se usa la función **route** vista al principio del tema:

```
<a href="{{route('ruta_contacto')}}">Contáctanos</a>
```

## 12.- Plantillas Blade

### Estructuras de control

```
@if ($edad >= 18)
    Puedes votar.
@endif
```

```
@if ($edad >= 18)
    Puedes votar.
@else
    No puedes votar
@endif
```

```
@if ($numero > 0)
    Número positivo
@else if ($numero < 0)
    Número negative
@else
    El número es cero
@endif
```

## 12.- Plantillas Blade

### Estructuras de control

```
@isset ($edad)
    {{$edad}}
@endisset
```

```
@isset ($edad)
    {{$edad}}
@else
    No se puede mostrar la edad
@endisset
```

```
@empty (productos)
    No hay productos
@endempty
```

```
@empty (productos)
    No hay productos
@else
    Hay productos
@endempty
```

## 12.- Plantillas Blade

### Estructuras de control

```
@switch($i)
    @case(1)
        La variable vale 1
    @break

    @case(2)
        La variable vale 2
    @break

    @default
        La variable no vale ni 1 ni 2
@endswitch
```



## 12.- Plantillas Blade

### Estructuras de control

```
@while ($numero < 100)
    <p>{{$numero++}}</p>
@endwhile
```

```
@for ($i = 0; $i < 10; $i++)
    <p>{{$i}}.-</p>
@endfor
```

## 12.- Plantillas Blade

### Estructuras de control

```
@foreach ($productos as $producto)
    <p>Artículo: {{$producto->nombre}} ( {{$product->precio}} €)</p>
@endforeach
```

Con **foreach** si el array no contiene ningún elemento se producirá un error por eso se recomienda usar **forelse**.

```
@forelse ($productos as $producto)
    <p>Artículo: {{$producto->nombre}} ( {{$product->precio}} €)</p>
@empty
    <p>No hay artículos.</p>
@endforelse
```

## 12.- Plantillas Blade

### Estructuras de control

En los bucles se puede usar **@continue** y **@break**.

```
@foreach ($usuarios as $usuario)
    @if ($usuario->rol == 1) {{-- Usuario administrador, no se debe mostrar --}}
        @continue
    @endif

    <li>{{ $usuario->nombre }}</li>
@endforeach
```

## 12.- Plantillas Blade

### Variable \$loop

Dentro de los bucles, Laravel ofrece la variable **\$loop** que ofrece información muy útil.

```
@foreach ($usuarios as $usuario)
    @if ($loop->first)
        Este es el primer usuario.
    @endif

    @if ($loop->last)
        Este es el ultimo usuario.
    @endif

    <p>Usuario: {{ $usuario->nombre }}</p>
@endforeach
```

## 12.- Plantillas Blade

### Variable \$loop

Dentro de los bucles, Laravel ofrece la variable **\$loop** que ofrece información muy útil.

```
@foreach ($categorias as $categoria)
    @foreach ($categoria->productos as $producto)
        @if ($loop->parent->first)
            Este es el primer product de la categoría: {{$categoria->nombre}}.
        @endif
    @endforeach
@endforeach
```

## 12.- Plantillas Blade

### Variable \$loop

Propiedad	Descripción
<code>\$loop-&gt;index</code>	El índice de la iteración actual (empieza en cero).
<code>\$loop-&gt;iteration</code>	El número de iteración actual (empieza en uno).
<code>\$loop-&gt;remaining</code>	Las iteraciones que faltan para acabar.
<code>\$loop-&gt;count</code>	La cantidad de elementos del array.
<code>\$loop-&gt;first</code>	Verdadero si se está en la primera iteración.
<code>\$loop-&gt;last</code>	Verdadero si se está en la última iteración.
<code>\$loop-&gt;even</code>	Verdadero si se está en una iteración par.
<code>\$loop-&gt;odd</code>	Verdadero si se está en una iteración impar.
<code>\$loop-&gt;depth</code>	Nivel de profundidad del bucle en bucles anidados (empieza en uno).
<code>\$loop-&gt;parent</code>	Si es un bucle anidado referencia a la variable \$loop del padre.

## 12.- Plantillas Blade

### Combinación de bucles y subvistas

En ocasiones es necesario mostrar diferentes elementos con el mismo estilo, como podría ser todos los productos de una categoría (dentro de un div, con un enlace, precio...).

En este caso se puede crear una subvista para mostrar un solo elemento y desde la vista principal aplicar esa vista a un conjunto de elementos:

subvista	array	variable en la subvista
<code>@each('producto.catalogo', \$productos, 'producto')</code>		

Se puede indicar un parámetro extra que será la vista a mostrar si el conjunto de elementos se encuentra vacío:

```
@each('producto.catalogo', $productos, 'producto', 'product.vacio')
```

## 12.- Plantillas Blade

### PHP directo

En ocasiones es necesario usar código PHP de manera directa, como por ejemplo para inicializar una variable.

Se pueden usar las etiquetas para delimitar código PHP: `<?php ?>`, pero es mejor usar directivas Blade:

```
@php
    $contador = 1;
@endphp
```



# Práctica

## **Actividad 3:**

Plantilla listando elementos de un array.

## 12.- Plantillas Blade

### Layout

Las aplicaciones web suelen tener páginas homogéneas, compartiendo así el estilo, la cabecera, el menú, el pie de página...

Se pueden crear **layouts** con ese contenido común y con "**huecos**" que podrán ser rellenados desde otros archivos Blade.

Para ello se usa las directivas **@yield**, **@extends**, **@section** y **@endsection**.

La mejor manera de entenderlo es mediante un ejemplo:

# 12.- Plantillas Blade

## Layout

layout.blade.php

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>
        @yield('titulo')
    </title>
</head>
<body>
    <nav>
        {{-- Menú de navegación --}}
    </nav>

    @yield('contenido')
</body>
</html>
```

inicio.blade.php

```
@extends('layout')

@section('titulo', 'Inicio')

@section('contenido')
    Soy Rick, bienvenido a mi blog.
    <br>
    {{date('d/m/Y H:i:s')}}
@endsection
```

# 12.- Plantillas Blade

## Layout

Anteriormente se ha visto que se pueden incluir vistas dentro de otras vistas con **@include**.

Estos archivos comunes a un conjunto de vistas se suelen incluir en una subcarpeta llamada **partials** dentro de resources/views o dentro de una subcarpeta de resources/views si esa vista solo se usa en las plantillas de esa subcarpeta.

partials/nav.blade.php

```
<nav>
    <a href="{{ route('inicio') }}">Inicio</a>
    <br>
    <a href="{{ route('login') }}">Accede</a>
</nav>
```

layout.blade.php

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>
        @yield('titulo')
    </title>
</head>
<body>
    @include('partials.nav')

    @yield('contenido')
</body>
</html>
```

## 12.- Plantillas Blade

### Páginas de error personalizadas

Laravel permite añadir páginas de error personalizadas, para ello solo hay que crear el directorio **errors** dentro de `resources/views` y en ese directorio añadir una página en Blade para cada error.

Por ejemplo: **404.blade.php**.

# Práctica

## **Actividad 4:**

Layout y página de error.