UT8: LARAVEL

¿Qué es Laravel?

Laravel es un framework de PHP diseñado para crear aplicaciones web de manera rápida, sencilla y estructurada. Utiliza una arquitectura **MVC (Modelo-Vista-Controlador)** que separa la lógica de negocio (Modelos), las vistas (HTML) y el control de las peticiones (Controladores).

- **Modelo**: Contiene la lógica de negocio y maneja los datos.
- **Controlador**: Actúa como intermediario entre la Vista y el Modelo.
- **Vista**: Muestra los datos al usuario.

Ventajas principales de Laravel:

- **Fácil de usar:** Ofrece herramientas para tareas comunes como autenticación, enrutamiento y manejo de bases de datos.
- **ORM Eloquent:** Simplifica el acceso a bases de datos.
- **Motor de plantillas Blade:** Permite diseñar vistas de forma rápida y limpia.
- **Comunidad amplia:** Excelente documentación y muchos recursos disponibles.
- **PHP moderno:** Compatible con las últimas versiones de PHP y buenas prácticas.

Requisitos previos

- **PHP 8+**: Laravel requiere una versión moderna de PHP.
- **Composer**: Administrador de dependencias de PHP.
- **Servidor local:** Puedes usar XAMPP, WAMP o Laravel Valet (si estás en Mac).

Instalación de Laravel

Se supone que ya tienes una copia de PHP instalada en tu sistema local.

Composer es un gestor de paquetes y dependencias. Para ejecuta este comando:

```
curl -Ss getcomposer.org/installer | php
```

Verás que se descarga y compila el script composer.phar, que es el que usamos para instalar Laravel. Aunque hay muchas formas de configurar una nueva aplicación Laravel, lo haremos a través del script composer.phar de Laravel. Para instalar este script, ejecuta:

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

Esto descargará e instalará todos los archivos del framework, así como todas las dependencias que requiere. Los paquetes se guardarán dentro del directorio del proveedor.

Para crear una nueva aplicación ejecutar el siguiente comando:

```
laravel new nombre_Aplicacion
```

VSCode extensiones¶

- Recomendable instalar los siguientes plugins para Visual Studio Code

```
Laravel Blade Snippets
```

Carpetas en Laravel¶

Al crear un nuevo proyecto con este framework, Laravel crea una serie de carpetas por defecto. Esta estructura de carpetas es la recomendada para utilizar Laravel.

Public¶

Esta es la carpeta más importante ya que es donde se ponen todos los archivos que el cliente va a mostrar al usuario cuando introduzcamos la URL de nuestro sitio web. Normalmente se carga el archivo index.php por defecto.

Routes¶

Otra de las carpetas que más vamos a usar. En ella se albergan todas las rutas (redirecciones web) de nuestro proyecto, pero más concretamente en el archivo web.php

```
Dada una ruta → se cargará una vista
```

Resources¶

Esta es nuestra carpeta de recursos donde guardaremos los siguientes archivos, que también, están separados por sus carpetas... como cada nombre indica:

- *css* Archivos CSS
- *js* Archivos JS (JavaScript)
- *lang* Archivos relacionados con el idioma del sitio (variables & strings)
- *views* Archivos de nuestras vistas (Vistas Blade (HTML dinámico)), lo que las rutas cargan

Rutas¶

Las rutas en Laravel (y en casi cualquier Framework) sirven para redireccionar al cliente (o navegador) a las vistas que nosotros queramos.

Estas rutas se configuran en el archivo *routes/web.php* donde se define la ruta que el usuario pone en la URL después del dominio y se retorna la vista que se quiere cargar al introducir dicha dirección en el navegador.

Por ejemplo, la ruta:

```php
<?php

// Ruta por defecto para cargar la vista welcome cuando el usuario introduce simplemente el dominio

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

Indica que, al acceder a la URL raíz (/), Laravel mostrará la vista llamada welcome, ubicada en resources/views/welcome.blade.php.

Los archivos Blade (que veremos un poco más adelante) tienen la extensión *.blade.php* y se almacenan en el directorio:

resources/views

Un archivo llamado *welcome.blade.php* estará ubicado en *resources/views/welcome.blade.php*.

Alias¶

Es interesante darle un alias o un nombre a nuestras rutas para poder utilizar dichos alias en nuestras plantillas de Laravel que veremos más adelante.

Para ello, basta con utilizar la palabra *name* al final de la estructura de la ruta y darle un nombre que queramos; normalmente descriptivo y asociado a la vista que tiene que cargar el enroutador de Laravel.

```php
<?php

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

Después veremos que es muy útil ya que a la hora de refactorizar o hacer un cambio, si tenemos enlaces o menús de navegación que apuntan a esta ruta, sólo tendríamos que cambiar el parámetro dentro del get() y no tener que ir archivo por archivo.

Laravel nos proporciona una manera más cómoda a la hora de cargar una vista si no queremos parámetros ni condiciones. Tan sólo definiremos la siguiente línea que hace referencia la ruta datos en la URL y va a cargar el archivo usuarios.php de nuestra carpeta views como le hemos indicado en el segundo parámetro.

```php
<?php

/* http://localhost/datos/ */

Route::view('datos', 'usuarios');
```

- *Route::view:* Es un método que conecta una URL directamente con una vista sin necesidad de definir una función.
- Primer parámetro: *'datos'*: Define la URL a la que el usuario accede, en este caso: http://localhost/datos.
- Segundo parámetro: *'usuarios'*: Es el nombre de la vista que se cargará, ubicada en el directorio resources/views/usuarios.blade.php.

Pero no sólo podemos retornar una vista sino, desde un simple string a módulos propios de Laravel.

Parámetros¶

Ya hemos visto que con PHP podemos pasar parámetros a través de la URL, como si fueran variables, que las recuperábamos a través del método GET o POST.

Con Laravel también podemos introducir parámetros pero de una forma más vistosa y ordenada, de tal manera que sea visualmente más cómodo de recordar y de indexar por los motores de búsqueda como Google.

```
http://localhost/cliente/324
```

Para configurar este tipo de rutas en nuestro archivo de rutas `public/routes/web.php` haremos lo siguiente.

```php
<?php

Route::get('cliente/{id}', function($id) {
    return('Cliente con el id: ' . $id);
});
```

- *Route::get('cliente/{id}', ...):*
    - Define una ruta que acepta un parámetro dinámico {id} después de /cliente/.
    - Cuando alguien visita una URL como http://localhost/cliente/324, Laravel reconoce 324 como el valor del parámetro {id}.
- *function($id):*
    - Laravel pasa automáticamente el valor del parámetro {id} a esta función como la variable $id.
- *return 'Cliente con el id: ' . $id;:*
    - Devuelve una respuesta de texto que incluye el valor del parámetro.
    - Por ejemplo, si accedes a http://localhost/cliente/324, la respuesta será:

      Cliente con el id: 324

¿Qué ocurre si no pasas el parámetro?

- Si accedes a http://localhost/cliente/ sin un parámetro {id}, Laravel mostrará un error 404 (Not Found), porque la estructura de la URL no coincide con la ruta definida.

Para resolver ésto, podemos definir una ruta por defecto en caso de que el id (o parámetro) no sea pasado. Para ello usaremos el símbolo ? en nuestro nombre de ruta e inicializaremos la variable con el valor que queramos.

```php
<?php

Route::get('cliente/{id?}', function($id = 1) {
    return ('Cliente con el id: ' . $id);
});
```

- *Route::get('cliente/{id?}', ...):*
    - Define una ruta que acepta un parámetro dinámico {id}.
    - El signo de interrogación ? después de {id} indica que este parámetro es opcional. Es decir, la URL puede incluirlo o no.
- *function($id = 1):*
    - Si el parámetro id no es proporcionado en la URL, Laravel asignará automáticamente el valor predeterminado 1 a la variable $id.
- *return 'Cliente con el id: ' . $id;:*
    - Devuelve una respuesta que incluye el valor de $id. Dependiendo de si el parámetro fue pasado o no, el texto cambiará.

Ahora tenemos otro problema, porque estamos filtrando por id del cliente que, normalmente es un número, pero si metemos un parámetro que no sea un número, vamos a obtener un resultado no deseado.

Para resolver este caso haremos uso de la cláusula where junto con una expresión regular númerica.

```php
<?php

Route::get('cliente/{id?}', function($id = 1) {
    return ('Cliente con el id: ' . $id);
}) -> where('id', '[0-9]+');
```

- *Route::get('cliente/{id?}', ...):*
    - Define una ruta que acepta un parámetro opcional {id}.
    - Si el parámetro no se proporciona, tomará el valor por defecto 1.
- *->where('id', '[0-9]+'):*
    - Usa la cláusula **where** para validar el parámetro id con una expresión regular.

- *[0-9]+:*
    - **[0-9]**: Acepta solo números del 0 al 9.
    - **+**: Indica que debe haber al menos un dígito (uno o más números).

Comportamiento esperado

1. Cuando el parámetro es un número:
    - URL: http://localhost/cliente/324
      Respuesta: Cliente con el id: 324
2. Cuando no se proporciona el parámetro:
    - URL: http://localhost/cliente/
      Respuesta: Cliente con el id: 1
3. Cuando el parámetro no es un número:
    - URL: http://localhost/cliente/abc
      Resultado: Laravel devuelve un error 404 (Not Found).

Además, podemos pasarle variables a nuestra URL para luego utilzarlas en nuestros archivos de plantillas o en archivos .php haciendo uso de un array asociativo. Veamos un ejemplo con la forma reducida para ahorrarnos código

```php
<?php

Route::view('datos', 'usuarios', ['id' => 5446]);
```

- *Route::view('datos', 'usuarios'):*
    - datos: Define la URL que el usuario visitará, en este caso: http://localhost/datos.
    - usuarios: Indica la vista que se cargará, ubicada en el archivo: resources/views/usuarios.blade.php.
- *['id' => 5446]:*
    - Es un array asociativo que contiene las variables que se pasan a la vista.
    - En este caso, estamos pasando la variable id con el valor 5446 para que esté disponible en la plantilla Blade.

... y el archivo `resources/views/usuarios.php` debe tener algo parecido a esto

```php
<!-- Estructura típica de un archivo HTML5 -->

<p>Usuario con id: <?= $id ?></p>

<!-- ... -->
```

Puedes pasar múltiples variables a la vista utilizando un array asociativo.

```
Route::view('datos', 'usuarios', ['id' => 5446, 'nombre' => 'Manolo']);
```

Con las plantillas de Laravel blade.php veremos cómo simplificar aún más nuestro código.

Para más información acerca de las rutas, parámetros y expresiones regulares en las rutas puedes echar un vistazo a la documentación oficial de rutas que contiene numerosos ejemplos.

Actividad 1:

Haz un ejemplo de cada caso que hemos visto y alguno nuevo que encuentres en la documentación oficial y pruebalos.

Plantillas o Templates¶

A través de las plantillas de Laravel vamos a escribir menos código PHP y vamos a tener nuestros archivos mejor organizados.

Blade es el sistema de plantillas que trae Laravel, por eso los archivos de plantillas que guardamos en el directorio de views llevan la extensión blade.php.

De esta manera sabemos inmediatamente que se trata de una plantilla de Laravel y que forma parte de una vista que se mostrará en el navegador.

Directivas¶

- Laravel tiene un gran número de directivas que podemos utilizar para ahorrarnos mucho código repetitivo entre otras funciones.
- Digamos que las directivas son pequeñas funciones ya escritas que aceptan parámetros y que cada una de ellas hace una función diferente dentro de Laravel.
- *@yield* Marca una sección en la plantilla principal donde se cargará contenido dinámico. Se usa conjuntamente con *@section*.
- *@section* y *@endsection*. Usados para definir bloques de contenido dinámico en las vistas que extienden la plantilla.
- *@extends* importa el contenido de una plantilla ya creada.

Separando código¶

Veamos un ejemplo de cómo hacer uso del poder de Laravel para crear plantillas y no repetir código.

Supongamos que tenemos ciertas estructuras HTML repetidas como puede ser una cabecera header, un menú de navegación nav y un par de secciones que hacen uso de este mismo código.

Supongamos que tenemos 2 apartados en la web:

- Blog
- Fotos

Primero de todo tendremos que generar un archivo que haga de plantilla de nuestro sitio web.

Para ello creamos el archivo *plantilla.blade.php* dentro de nuestro directorio de plantillas *resources/views*.

Dicho archivo va a contener el típico código de una página simple de HTML y en el body añadiremos nuestros contenido estático y dinámico.

```
<body>
    <!-- CONTENIDO ESTÁTICO PARA TODAS LAS SECCIONES -->
    <h1>Bienvenid@s a Laravel</h1>
    <hr>

    <!-- MENÚ -->
    <nav>
        <a href={{ route('noticias') }}>Blog</a> |
        <a href={{ route('galeria') }}>Fotos</a>
    </nav>

    <!-- CONTENIDO DINÁMICO EN FUNCIÓN DE LA SECCIÓN QUE SE VISITA -->
    <header>
        @yield('apartado')
    </header>
</body>
```

Cada sección que haga uso de esta plantilla contendrá el texto estático *Bienvenid@s a Laravel* seguido de un menú de navegación con enlaces a cada una de las secciones y el contenido dinámico de cada sección.

Ahora crearemos los archivos dinámicos de cada una de las secciones, en nuestro caso *blog.blade.php* y *fotos.blade.php*

```
<?php

// blog.blade.php

@extends('plantilla')

@section('apartado')
  <h2>Estás en BLOG</h2>
@endsection
```

Importamos el contenido de plantilla bajo la directiva @extends para que cargue los elementos estáticos que hemos declarado y con la directiva @section y @endsection definimos el bloque de código dinámico, en función de la sección que estemos visitando.

Ahora casi lo mismo para la sección de fotos

```php
<?php

// fotos.blade.html

@extends('plantilla')

@section('apartado')
  <h2>Estás en FOTOS</h2>
@endsection
```

El último paso que nos queda es configurar el archivo de rutas *routes/web.php*

```php
<?php

// web.php

Route::view('blog', 'blog') -> name('noticias');
Route::view('fotos', 'fotos') -> name('galeria');
```

Esto enlaza las URLs /blog y /fotos con las vistas blog.blade.php y fotos.blade.php, respectivamente. Los nombres de las rutas (noticias y galeria) se usan en el menú de navegación para hacerlas más claras.

De esta manera podremos hacer uso del menú de navegación que hemos puesto en nuestra plantilla (*plantilla.blade.php*) y gracias a los alias noticias y galeria, la URL será más amigable.

Nota:

- *route('noticias')* genera automáticamente la URL asociada al nombre de ruta noticias, que es */blog*.
- *route('galeria')* genera automáticamente la URL asociada al nombre de ruta galeria, que es */fotos*.

Estructuras de control¶

Como en todo buen lenguaje de programación, en Laravel también tenemos estructuras de control.

En Blade (plantillas de Laravel) siempre que iniciemos un bloque de estructura de control DEBEMOS cerrarla

- *@foreach* ~ *@endforeach* lo usamos para recorrer arrays
- *@if* ~ *@endif* para comprobar condiciones lógicas
- *@switch* ~ *@endswitch* en función del valor de una variable ejecutar un código
- *@case* define la casuística del switch
- *@break* rompe la ejecución del código en curso
- *@default* si ninguna casuística se cumple

```php
<?php

$equipo = ['María', 'Alfredo', 'William', 'Verónica'];

@foreach ($equipo as $nombre)
  <p> {{ $nombre }} </p>
@endforeach
```

Acordaros que podemos pasar variables a través de las rutas como si fueran parámetros. Pero en este caso, vamos a ver otra directiva más; el uso de @compact.

```php
<?php

// Uso de @compact
$equipo = ['María', 'Alfredo', 'William', 'Verónica'];

// Route::view('nosotros', ['equipo' => 'equipo']);
Route::view('nosotros', @compact('equipo'));
```

Las dos últimas líneas son equivalentes.

Actividad 1:

Crea un sitio web con Laravel que contenga el título "Bienvenidos a Laravel", un texto de bienvenida (puede ser un poco de Lorem Ipsum) y a continuación un menú de navegación con sus correspondientes alias y los siguientes enlaces:

- *Inicio* enlace a la página principal donde se visualizará el texto de Lorem Ipsum además de los elementos estáticos (Título y menú de navegación).

- *Nosotros* enlace que vaya a la página "nosotros" y muestre, además de los elementos estáticos de todo el sitio, un h2 que diga "Estás en la sección Nosotros"

- *Proyecto* enlace que cargue una vista con el siguiente texto "Estás en el proyecto numero: X" donde X es un número entero que podamos introducirlo en la propia ruta. Si no se mete ningún número en la ruta, por defecto tiene que ser 1; por ejemplo

http://localhost:8000s/proyecto/210937

- Recuerda que el título y el menú de navegación han de aparecer en todas las vistas que cargues.