Unidad 5. Introducción a Laravel. Rutas

1. Laravel

1.1. El mando artesano

Cuando creamos un proyecto Laravel, en la raíz del proyecto se instala una herramienta llamada 'artisan'. Se trata de una interfaz de línea de comandos (CLI, Command Line Interface), que nos proporciona una serie de opciones adicionales para gestionar nuestros proyectos Laravel. Por ejemplo, crear controladores, migrar datos a una base de datos, etc.

Para verificar que está instalado y las opciones que ofrece, podemos escribir el siguiente comando en una terminal desde la carpeta del proyecto que hemos creado:

```
docker exec -it laravel-myapp-1 bash
```

```
lista de artesanos php
```

Y obtendremos una lista de las opciones que ofrece artisan. Más adelante utilizaremos algunas opciones. Para empezar, podemos ejecutar este comando para comprobar la versión de Laravel del proyecto en el que nos encontramos:

```
php artesano --versión
```

1.2. Estructura de un proyecto Laravel

Cuando creamos un proyecto Laravel, se crea una estructura de carpetas y archivos predefinida. A continuación, explicamos brevemente las principales carpetas y archivos que se generan:

- 'app': contiene el código fuente de la aplicación. La mayoría de las clases que definamos estarán en esta carpeta. Inicialmente, se incluyen algunas subcarpetas dentro de:
  - 'Modelos'
  - 'Http': contiene los controladores y middleware
  - 'Proveedores': contiene los proveedores de servicios de la aplicación, más aquellos que podamos definir.
  - Además, esto incluye, o se pueden incluir, carpetas adicionales para nuestra aplicación, como por ejemplo la carpeta 'Events' para definir los eventos que ocurren, o diferentes carpetas para almacenar el modelo de datos o clases de nuestra aplicación. 'bootstrap':
- contiene el archivo 'app.php', que es el que lanza la aplicación. Además, contiene la carpeta 'cache', donde se almacenan los archivos ya subidos por Laravel para agilizar su acceso en futuras peticiones. 'config': contiene los archivos de configuración de la aplicación, donde se tienen
- las variables de entorno, o si nuestra aplicación está en desarrollo o producción, o los parámetros de conexión a la base de datos, entre otras cosas. No obstante, los cambios de configuración es preferible hacerlos en el archivo '.env', ubicado en la raíz del proyecto Laravel, de manera que en este último archivo almacenaremos los datos privados (usuario y contraseña de

la base de datos, por ejemplo), y en los archivos 'config' correspondientes accederemos a estas variables de entorno definidas en 'env'.

- 'database':almacena los elementos de gestión de la base de datos, como generadores de objetos, migraciones, etc.
- 'public':contenido visible de la web. Contiene el fichero 'index.php', punto de entrada de todas las peticiones a la web, así como carpetas donde ubicar el contenido estático del cliente (imágenes, hojas de estilos CSS, ficheros

- JavaScript...). 'resources':contiene, por un lado, las vistas de nuestra aplicación. Por otro lado, también contiene ficheros CSS (archivos sass) y JavaScript (archivos unmiplayed) no compilados. Además, también almacena los ficheros de traducción, por si queremos hacer sitios multiidioma.
- 'routes':almacena las rutas de la aplicación 'web.php'
- 'storage':contiene las vistas compiladas, y otros ficheros generados por Laravel, como logs o sesiones. 'tests':para
- almacenar los tests sobre los componentes de nuestra aplicación
- 'vendor':donde se almacenan las dependencias o librerías adicionales que se requieran en nuestro proyecto Laravel. Nuevamente, Git debe ignorar esta carpeta y regenerarla cada vez que se clone el repositorio remoto, para evitar almacenar demasiada información innecesaria.

Aunque algunos de los conceptos vistos aquí quizás no estén claros aún, los iremos viendo paso a paso durante el curso.

1.3. Configuración general del proyecto

A partir de la estructura de carpetas de un proyecto Laravel vista arriba, ahora echaremos un vistazo rápido a dónde se encuentran las configuraciones generales del proyecto.

Por un lado, tenemos un archivo '.env' en la raíz del proyecto, que básicamente contiene una serie de variables de entorno generales. Por ejemplo, tienes la variable 'APP_NAME' con el nombre que quieres que tenga la aplicación, o un conjunto de variables que utilizaremos más adelante para conectarnos a la base de datos, entre otras cosas:

Por ejemplo, para el proyecto que hemos creado anteriormente (biblioteca):

```
APP_NAME=biblioteca
...
CONEXIÓN_BD=mysql
HOST_BD=127.21.0.2
PUERTO_BD=3306
BASE_DATOS_BD=DB_myapp
NOMBRE_USUARIO_BD=marta
CONTRASEÑA_BD=
...
```

Por otro lado, la carpeta 'config' contiene algunos archivos de configuración general.

Laravel 11 usa por defecto sqlite, pero vamos a trabajar con mysql. Debemos cambiar en config/ database.php de sqlite a mysql

```
'predeterminado' => env('DB_CONNECTION', 'mysql'),
```

2. Rutas con Laravel

Las rutas son un mecanismo que permite a Laravel establecer qué respuesta enviar a una petición que intenta acceder a una URL determinada. Estas rutas se especifican en distintos archivos dentro de la carpeta 'routes' de nuestro proyecto Laravel.

Podríamos decir que existen dos tipos principales de rutas (almacenadas en el archivo 'routes/web.php' de la aplicación):

- Las rutas web , que nos permitirán cargar diferentes vistas en función de la URL indicada por el cliente.
- Las rutas API , a través de las cuales definiremos diferentes servicios REST, como también veremos más adelante.

En esta unidad nos vamos a centrar en el primer grupo. Veamos qué tipos de rutas podemos definir y qué características tienen.

En el archivo 'routes/web.php', inicialmente ya existe una ruta predefinida a la raíz del proyecto, que carga la página de bienvenida en él.

```
<?php

utilizar Iluminar\Soporte\Fachadas\Ruta;

Ruta::get('/', función() { return
        view('bienvenido');
});
```

Además de utilizar el método 'get', desde la clase 'Route' también podemos acceder a otros métodos estáticos útiles, como 'Route::post' (útil para recoger datos de formularios, por ejemplo), o también 'Route::put', 'Route::delete'...
Los veremos con más detalle en unidades posteriores.

2.1 Rutas sencillas

Las rutas simples tienen un nombre de ruta fijo, y una función que responde a ese nombre emitiendo una respuesta. Podemos añadir, por ejemplo, una segunda ruta mediante la cual, si accedemos a la URL http://localhost:8010/hello nos muestre el mensaje "Hola mundo". Añadimos para ello la siguiente ruta debajo de la anterior que ya estaba definida:

```
Ruta::get('hola', función() { return "Hola mundo";

});
```

Si ahora accedemos a http://localhost:8010/hello, deberíamos ver el mensaje, en texto simple.

2.2 Rutas con parámetros

También es posible pasar parámetros en la URL de la ruta. Para ello, incluimos el nombre del parámetro entre llaves, y se lo pasamos también a la función del segundo parámetro. Por ejemplo, si definimos una ruta para saludar al nombre que nos llega como parámetro, el código quedaría así:

```
Ruta::get('saludo/{nombre}', función($nombre){ return "Hola, " .
$nombre; });
```

En este caso, si el parámetro es obligatorio y no lo indicamos en la URL, nos redirigirá a una página de error 404. Para indicar que un parámetro no es obligatorio se termina su nombre con un signo de interrogación, y también es conveniente darle un valor por defecto en la función de respuesta de PHP. De esta forma modificaríamos la ruta anterior para que el nombre del usuario sea opcional y, en caso de no ponerlo, se asigne el nombre “Invitado”.

```
Ruta::get('saludo/{nombre?}' , función($nombre = "Invitado"){ return "Hola, " });
                                . $nombre;
```

2.2.1 Validación de parámetros

Algunos parámetros deberán seguir un patrón determinado. Por ejemplo, un identificador numérico solo contendrá dígitos. Para asegurarnos de ello, podemos utilizar el método 'where' al definir la ruta. A este método le pasamos dos parámetros: el nombre del parámetro a validar y la expresión regular que debe cumplir. En el caso del nombre anterior, si queremos que solo contenga letras (mayúsculas o minúsculas), podemos hacer algo como esto:

```
Ruta::get('saludo/{nombre?}' , función($nombre = "Invitado"){ return "Hola, " . $nombre; })-
>where('nombre', "[A-Za-z]+");
```

En caso de que la ruta no cumpla con el patrón se obtendrá una página de error. Más adelante veremos cómo podemos personalizar estas páginas de error.

2.3 Rutas con nombre

En ocasiones puede resultar conveniente asociar un nombre a una ruta, sobre todo cuando dicha ruta va a formar parte de un enlace en alguna página de nuestro sitio, ya que en un futuro la ruta podría cambiar y de esta forma evitamos tener que actualizar los enlaces con el nuevo nombre.

Para ello, al definir la ruta, asociamos a la función 'nombre' el nombre que queramos. Por ejemplo:

```
Ruta::get('contacto', función() { return "Página de
contacto";
```

```
})->nombre('ruta_contacto');
```

Ahora, si queremos definir un enlace a esta ruta en cualquier lugar, basta con utilizar la función 'route' de Laravel, indicando el nombre que le hemos asignado a esta ruta. Así, en lugar de poner esto:

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

Podemos hacer algo como esto, como veremos a continuación cuando usemos el motor de plantillas Blade:

```
<a href="{{ route('path_contact') }}"> contacto</a>
```

De esta manera, ante futuros cambios en las rutas, solo tendremos que cambiar la URL en 'Route::get'.

2.4. Combinación de elementos en rutas

Podemos combinar varias cláusulas 'where' en una ruta para validar los diferentes parámetros que pueda tener. También podemos combinar varias cláusulas 'where' con una cláusula 'name'. Por ejemplo, la siguiente ruta espera recibir un nombre con caracteres y un id numérico, ambos con valores predeterminados:

```
Ruta::get('greeting/{name?}/{id?}' ,
function($name="Invitado", $id=0) { return

"Hola $name,tu código es el $id"; })->where('name', "[A-Za-z]+")
->where('id', "[0-9]+") -> name('greeting');
```

Si accedemos a cada una de las siguientes URLs obtendremos cada una de las respuestas indicadas:

URL	Respuesta
/saludo	Hola Invitado, tu código es 0
/saludo/Nacho	Hola Nacho, tu código es 0
/saludo/Nacho/3	Hola Nacho, tu código es 3
/saludo/3	Error 404 (URL incorrecta)

Tenga en cuenta que el último caso es incorrecto. No podemos especificar un id sin especificar un nombre antes, porque viola el patrón de URL. Se puede dejar un parámetro omitido, siempre y cuando también se omitan los siguientes.