PROGRAMACIÓN WEB EN ENTORNO SERVIDOR UD-16

FRAMEWORK LARAVEL

ÍNDICE

[COMPOSER 3](#_Toc158751691)

[A. INSTALACIÓN DE COMPOSER 4](#_Toc158751692)

[LARAVEL 4](#_Toc158751693)

[A. RUTAS 5](#_Toc158751694)

[B. ALIAS 6](#_Toc158751695)

# COMPOSER

Logotipo

Descripción generada automáticamenteComposer es un gestor de dependencias en proyectos, para programación en PHP. Eso quiere decir que nos permite gestionar (declarar, descargar y mantener actualizados) los paquetes de software en los que se basa nuestro proyecto PHP.

Generalmente siempre usamos alguna que otra librería de terceros desarrolladores, que nos permite evitar empezar todo desde cero. Ya sea un framework o algo más acotado como un sistema para debug o envío de email, validación de formularios Composer es una herramienta imprescindible para el desarrollo con PHP, ya que permite:

* Uso de librerías de terceros
* Integración de proyectos basados en un framework
* Mantenimiento a lo largo de toda la vida de un proyecto PHP.

Los gestores de paquetes nos ayudan para resumir las tareas de descarga y mantenimiento de las versiones del proyecto para que estén siempre actualizadas.

Composer nos permite declarar las librerías que queremos usar en un proyecto. Su uso es extremadamente simple, lo que anima a cualquier persona a usarlo, sea cual sea su nivel técnico.

El archivo donde vamos a especificar que debe cargar composer se llama **composer.json**, y es un simple JSON en el que indicamos cosas como el autor del proyecto, las dependencias, etc.

Texto

Descripción generada automáticamente

Una vez tenemos definidas las dependencias en nuestro proyecto debemos instalarlas. Esto lo conseguimos con un simple comando en el terminal en el que le pedimos a Composer que las instale:

composer install

Con este comando, se creará dentro de nuestro proyecto un directorio vendor donde estarán las librerías declaradas. Tendremos que hacer un único include o require en nuestro código y todas laslibrerías estarán disponibles para usar.

require 'vendor/autoload.php’;

https://packagist.org/ es el repositorio de paquetes que son instalables por medio de Composer, y se pueden buscar por cualquier concepto que nos interese.

## INSTALACIÓN DE COMPOSER

La instalación es muy simple:

* Descarga el programa de instalación de composer ComposerSetup.exe en el enlace <https://getcomposer.org/download/>
* Indica la ubicación de php.

Si te da problemas la instalación sobre Xampp, prueba a cambiar la configuración de php en el archivo php.ini, quitando el ; de la línea ;extension=php\_openssl.dll.

# LARAVEL

Forma

Descripción generada automáticamenteLaravel es un potente y elegante framework de aplicaciones web PHP que agiliza el desarrollo de aplicaciones web mediante una arquitectura MVC (Modelo-Vista-Controlador). Creado por Taylor Otwell, Laravel ha ganado una inmensa popularidad debido a su simplicidad y facilidad de uso, lo que lo ha convertido en una de las principales opciones para los desarrolladores web de todo el mundo.

Puedes utilizar Composer, que ya está instalado, para instalar Laravel globalmente en tu sistema. Para ello, abre el símbolo del sistema y ejecuta el comando indicado dentro de:

composer create-project laravel/laravel app-name

Esto descargará automáticamente todos los archivos relevantes de Laravel para crear un nuevo proyecto.

Con tu nuevo proyecto de aplicación creado, necesitarás iniciar un servidor. Para ello, escribe lo siguiente:

cd app-name

php artisan serve

Una vez iniciado el servidor, deberías poder acceder a tu proyecto de aplicación a través de tu navegador web. Para ello, abre tu navegador y ve a la siguiente dirección: https://localhost:8000

Con esto, ya puedes empezar a desarrollar aplicaciones web utilizando Laravel en tu máquina.

Si algo te ha dado problemas y no ves la página inicial de Laravel, instala de nuevo Xampp, Composer y Laravel…

Al crear un nuevo proyecto, 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. En función de la dirección web, cargaremos una vista u otra.
* **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, 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 public/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.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Cuando alguien acceda a las rutas /, /about o /contact, se devolverán las vistas correspondientes welcome.blade.php, about.blade.php y contact.blade.php, respectivamente.

## ALIAS

Es interesante darles un alias o un nombre a nuestras rutas para poder utilizar dichos alias en nuestras plantillas de Laravel.

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.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

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.



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

## PARÁMETROS

Con Laravel también podemos introducir parámetros 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.

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

Texto

Descripción generada automáticamente

¿Qué pasa si no introducimos un id y sólo navegamos hasta cliente/? Nos va a devolver un 404.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para resolver esto, 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.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

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, podemos obtener un resultado no deseado.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Para resolver este caso haremos uso de la cláusula where junto con una expresión regular numérica. Si el parámetro es incorrecto nos dará también un error 404.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Teams

Descripción generada automáticamente

Además, podemos pasarle variables a nuestra URL para luego utilizarlas 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.

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

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

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

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

Si ahora accedemos a la web:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Para acabar con este apartado, comentar un comando de Artisan que sirve para mostrar todas las rutas disponibles en un proyecto o aplicación web creada con Laravel. Es un comando muy útil para saber qué tipos de rutas puedes tener y qué valores te van a permitir, parámetros, etc.

php artisan route:list

Imagen que contiene Texto

Descripción generada automáticamente

## 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.

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 Define el contenido dinámico que se va a cargar. Se usa conjuntamente con @section.
* @section y @endsection bloque de código dinámico.
* @extends importa el contenido de una plantilla ya creada

Supongamos que tenemos ciertas estructuras HTML repetidas como puede ser 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 y 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.

Texto

Descripción generada automáticamente

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:

Captura de pantalla con letras

Descripción generada automáticamente

## 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

Dentro de una plantilla podremos usar el bucle @foreach igual que lo hacíamos en php:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente con confianza media

Se pueden usar otras estructuras de control similares a las de PHP:

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

## CONTROLADORES

Los controladores son el lugar perfecto para definir la lógica de negocio de nuestra aplicación o sitio web.

Hace de intermediario entre la vista (lo que vemos con nuestro navegador o cliente) y el servidor donde la app está alojada.

Por defecto, los controladores se guardan en una carpeta específica situada en app/Http/Controllers y tienen extensión .php.

Para crear un controlador nuevo debemos hacer uso de nuestro querido autómata artisan donde le diremos que cree un controlador con el nombre que nosotros queramos.

Abrimos la consola y nos situamos en la raíz de nuestro proyecto:

php artisan make:controller PagesController

Captura de pantalla de computadora

Descripción generada automáticamente

Si todo ha salido bien, recibiremos un mensaje por consola con que todo ha ido bien y podremos comprobar que, efectivamente se ha creado el archivo PagesController.php con una estructura básica de controlador, dentro de la carpeta Controllers que hemos descrito anteriormente.

Ahora podemos modificar nuestro archivo de rutas web.php para dejarlo limpio de lógica y trasladar ésta a nuestro nuevo controlador.

La idea de esto es dejar el archivo web.php tan limpio como podamos para que, de un vistazo, se entienda todo perfectamente.

RECUERDA que sólo movemos la lógica, mientras que las cláusulas como where y name las seguimos dejando en el archivo de rutas web.php

Veamos cómo quedaría un refactor del archivo de rutas utilizando un Controller como el que acabamos de crear.

Ahora nos quedaría de la siguiente manera:

Texto

Descripción generada automáticamente

El controlador que acabamos de crear PagesControler.php nos quedaría de la siguiente manera:

Texto

Descripción generada automáticamente

## MIGRACIONES & ELOQUENT

Con las migraciones vamos a gestionar la base de datos de nuestro sitio web; tanto crear nuevas BBDD como editarlas desde Laravel.

Las migraciones de un sitio hecho con Laravel se alojan en la ruta database/migrations y tienen extensión .php.

Si te fijas, nada más instalar Laravel con la imagen de Bitnami, se han creado varios archivos de migraciones que podemos usar, pero de momento los dejamos ahí y NO LOS BORRAMOS porque son muy útiles.

1. Archivos .env.

Es de uso común trabajar con archivos de entorno llamados también archivos .env. Normalmente, en un proyecto real puedes encontrarte con varios archivos de este tipo en función del despliegue que se quiera hacer. Por ejemplo:

* **test.env config.** 🡪 para desplegar en entorno seguro de pruebas.
* **release.env config.** 🡪 para desplegar cambios de versión que se envía a los beta testers.
* **production.envconfig.** 🡪 para desplegar el código ya probado al resto del mundo.

En nuestro caso, como no vamos a desplegar nada, sólo vamos a usar un único archivo .env.

En del directorio raíz de nuestro proyecto, existe el fichero .env que es donde debemos configurar los datos de nuestro servidor MySQL y rellenarlo con la información correspondiente a nuestra base de datos ya creada:

Texto

Descripción generada automáticamente

Para crear la BD laravel o como se llame, recuerda tener arrancado MySQL y si quieres hacerlo desde phpMyAdmin, tendrás que arrancar también el servidor web.

Una vez tengamos ésto, lo que nos queda es ejecutar el comando de las migraciones a través del CLI artisan:

php artisan migrate

Si todo ha salido bien obtendremos el siguiente resultado donde podremos observar que todas las migraciones se han insertado correctamente en la base de datos.

Texto

Descripción generada automáticamente

Si nos vamos al cliente que utilicemos para manejar la base de datos (phpMyAdmin por ejemplo) veremos que en nuestra base de datos se han creado todas las tablas de la migración que hemos ejecutado y además una tabla que se llama migrations.

La tabla migrations es simplemente un registro de todas las migraciones llevadas a cabo. Además, podemos hacer un rollback en caso de que queramos reestablecer nuestra base de datos.

php artisan migrate:rollback

O si preferimos hacer un reset para dejarla limpia

php artisan migrate:reset

## MIGRACIONES Y ELOQUENT (MODELOS)

Gracias a Eloquent y su integración con Laravel, podremos crear modelos de datos de una manera automatizada a través de artisan.

Ahora que ya sabemos manejar las migraciones es hora de crear nuestras propias migraciones pero a través de Eloquent. A través de la instrucción make:model creamos un nuevo modelo de datos, a continuación ponemos el nombre siempre empezando en Mayúsucla y en SINGULAR y pasamos el parámetro relacionado con las migraciones -m.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Si todo ha salido bien, veremos en nuestro directorio de migraciones database/migrations un nuevo archivo que se llama fecha\_identificador\_create\_books\_table.php.

Texto

Descripción generada automáticamente

El siguiente paso es ver nuestro archivo de migraciones y editarlo para que contenga las tablas que nosotros queramos. Si lo visualizamos tan sólo tendrá la estructura básica con un par de tablas. Vamos a añadir unos atributos más.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Una vez que estamos satisfechos con nuestro esquema debemos volver a ejecutar el código de migración a través de artisan para que se introduzca esta nueva información en la base de datos. Si nos dice que no hay nada que migrar, podemos ejecutar php artisan migrate:rollback. Si nos da cualquier tipo de problema borramos las tablas de la base de datos y volvemos a empezar.

Para ver todos los tipos de datos que maneja Eloquent puedes visitar la documentación.

Con cada tabla nos habrá creado un modelo, que se almacena en app/Models.

Vamos a recuperar los datos y mostrarlos con una vista ya creada o creamos una nueva y solicitamos los datos desde el HTML.

En web.php vamos a crear una nueva ruta dentro del controlador PagesController, que se llamará notas, para ver las notas de la BD.

Texto

Descripción generada automáticamente

Dentro del controlador deberemos importar el modelo.

También en el controlador vamos a crear la acción para las notas:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Como podemos ver, llamamos al modelo indicando que nos devuelva todos los registros con all( ). Ya solo nos queda definir la vista:

Texto

Descripción generada automáticamente