Guia para la versión **Laravel 5.0 LTS**

ACTUALIZACIÓN (24/02/2017): Actualizado a **Laravel 5.1**

ACTUALIZACIÓN (01/03/2017): Actualizado a **Laravel 5.3 PARCIALMENTE**

Framework Laravel

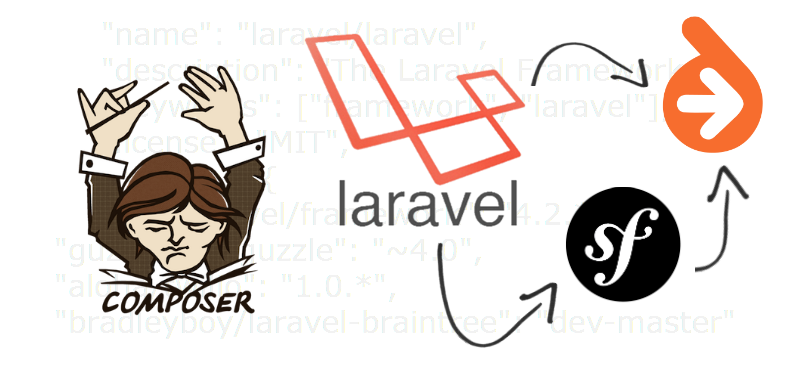
¿Que es Laravel?

**Laravel** es un framework de código abierto para desarrollar aplicaciones y servicios web con PHP 5. Su filosofía es desarrollar código PHP de forma elegante y simple, evitando el "código espagueti".

¿Que necesitamos para poder correr Laravel?

* Composer [(Enlace)](https://getcomposer.org/)
* Servidor Web, por ejemplo Xampp [(Enlace)](https://www.apachefriends.org/es/index.html)

# Composer



¿Que es Composer?

Composer es un manejador de dependencias para PHP. Antes php ha tenido algunos problemas en el mundo de la descarga de paquetes/librerías, para hacer eso tenías que ir a la web de alguna librería y buscar la sección de “Descargas” para poder utilizarla; sin contar con el hecho que para hacer eso en algunas ocasiones necesitabas registrarte a la página; programadores PHP se quejaban de que no existía algo como **npm** para *Node.js* o **bundler** para *Ruby*. Bien, ahora existe, se llama **Composer** y podría considerarse una de las maravillas del mundo de PHP.

Descargar Composer

La instalación de Composer es bastante sencilla, basta con entrar en la sección [Download Composer](https://getcomposer.org/download/) y descargar e instalar el ejecutable.

Verificar Instalación

Para poder verificar que el Composer se instaló de manera correcta, tenemos que ir a los Comandos del Sistema y escribir “composer”. En caso de funcionar nos mostrará una ventana similar:



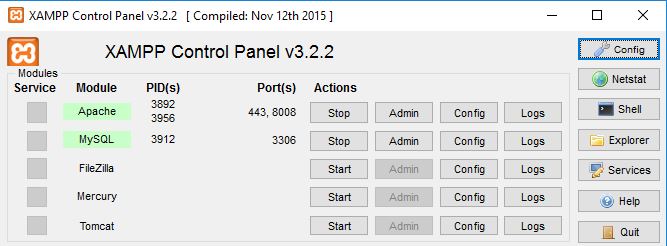
# Servidor Web Xampp



¿Qué es Xampp?

XAMPP es un servidor independiente de plataforma de código libre. Te permite instalar de forma sencilla Apache en tu propio ordenador, sin importar tu sistema operativo. Incluye además servidores de bases de datos como MySQL con su respectivo gestor phpMyAdmin. Incorpora también el intérprete de PHP, el intérprete de Perl, servidores de FTP (FileZilla por ejemplo), entre muchas cosas más.  
Una de las ventajas de usar XAMPP es que su instalación es de lo más sencilla, basta descargarlo, instalarlo y comenzar a usarlo.

Verificar correcto funcionamiento de Xampp



Para saber si Xampp está corriendo correctamente, bastará con iniciar, verificar que tanto Apache como MySQL estén en verde y luego entrar a la pantalla de bienvenida de Xampp desde el link: http//:localhost:PUERTO mostrándonos una pantalla como esta:



# Laravel desde Cero

Proyecto Base de Laravel

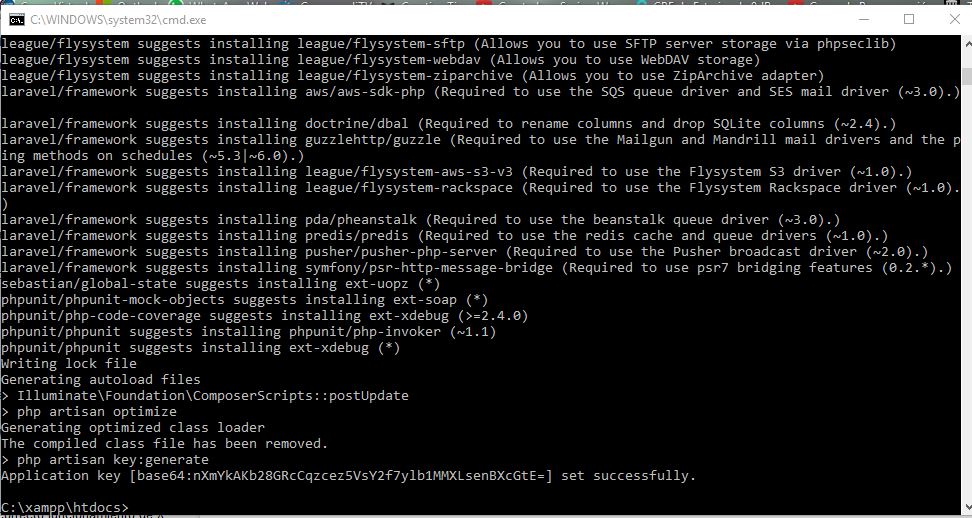
Para crear el proyecto base tenemos que ir, mediante los Comandos del Sistema, dentro de nuestra carpeta “htdocs” ubicada dentro de nuestro servidor web xampp.

*Por ejemplo: “ C:\xampp\htdocs”*

|  |
| --- |
| Para movernos dentro del CMD utilizamos el comando “cd..” para regresar un directorio hacia atrás y “cd directorio” para ingresar al directorio, por ejemplo “cd htdocs” |

Una vez parado sobre el directorio “htdocs” tipeamos la siguiente línea:

*“composer create-project laravel/laravel --prefer-dist nombreProyecto”*

De está manera le estamos indicando a composer que queremos crear un projecto nuevo (create-project), que el tipo de proyecto va a ser de Laravel (laravel/laravel), que la instalación sea desde el repositorio de distribución de laravel y que se llame de la siguiente manera (nombreProyecto). Una vez damos enter, comenzará a descargar todas las dependencias que necesita laravel, dándonos un mensaje similar al siguiente al finalizar 

Visualizar proyecto

Una vez descargadas todas las dependencias podremos ver nuestro proyecto corriendo en nuestro servidor local. **Tenemos 2 maneras de hacerlo**, la primera es ingresando desde la barra de direcciones la ubicación de nuestra carpeta, generalmente es: *“http://localhost:8080/unlam”*

En este caso nuestro puerto es 8080 y nuestra carpeta del proyecto se llama “unlam”

Si utilizan está ruta, van a notar que no les muestra una hoja de bienvenida, sino que les muestra una estructura de carpetas que componen el proyecto. Esto se debe a que laravel solo permite que el usuario pueda ingresar a la carpeta public, que el resto de las carpetas no sean de su acceso. Entonces, para ver de manera correcta nuestra Hoja de bienvenida debemos ingresar en el siguiente link

*“http://localhost:8080/unlam/public”*

La segunda manera de ingresar es creando un puerto especial para el proyecto, esto se realiza de manera muy sencilla utilizando los comandos de Artisan. Para ello, ingresamos en nuestro CMD (Comandos del Sistema), nos paramos dentro de la carpeta de nuestro proyecto de laravel (C:\xampp\htdocs\unlam) e ingresamos el siguiente comando “php artisan serve” el cual nos devolverá una dirección con un puerto determinado



Solo bastará entrar en “localhost:8000” y podrán visualizar exactamente lo mismo que si estuvieran ingresando en *“http://localhost:8080/unlam/public”*

|  |
| --- |
| El puerto 8000 solo estará disponible para nuestro proyecto de laravel mientras la ventana del cmd se encuentre abierta. |

|  |
| --- |
| Entre las herramientas que Laravel nos proporciona para el desarrollo de aplicaciones se encuentra Artisan, la interfaz de línea de comandos, la cual es un medio para la interacción con la aplicación donde los usuarios (en este caso los desarrolladores) dan instrucciones en forma de línea de línea de comando. |

Carpetas de Laravel

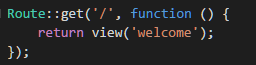
* **app/** Donde se encuentran las clases de modelos, controladores, rutas, middlewares, entre otras.
* **bootstrap/** Permite el sistema de arranque de laravel. No debemos tocar esta carpeta, No confundir con el Framework Bootstrap.
* **config/** Donde se ubican los archivos de configuración del proyecto.//
* **database/** Dentro de esta carpeta encontraremos las migraciones, seeder y factores de nuestra base de datos. (Esto será explicado más adelante)
* **public/** Única carpeta visible en el servidor web, en ella encontraremos las imágenes, los estilos css, los archivos js, el favicon, etc...
* **resources/** Contiene todos los archivos que están disponibles para los usuarios, como los asset, la carpeta lang (En donde colocaremos las traducciones) y la carpeta views (que contiene las vistas, lo que ve el usuario).
* **routes/** En esta carpeta se almacenan las rutas de laravel, las cuales indican de qué manera procesar las peticiones http entrantes.
* **storage/** Es la carpeta que utiliza el framework para almacenar cosas como la caché, las sesiones o las vistas, logs, etc. Está carpeta no debemos tocarla a menos que queramos borrar todo su contenido para que se vuelva a generar.
* **test/** Aqui escribiremos todas las pruebas que tendrá la aplicación.
* **vendor/** Lugar en donde se guardar las librerías de terceros. Esto no lo debemos tocar ya que el Composer se encarga de manejarlas.

## Rutas

Las rutas sirven para mapear las URLs de la aplicación a sus correspondientes controladores, es decir, las rutas escuchan la petición http, si la petición no encuentra una ruta que la escuche arroja error, en caso de encontrarla realiza lo que esa ruta indique, por ejemplo, actuar en una función anónima o ir a un método de un controlador.

Ruta con función anónima

Son las rutas que impactan contra una función anónima, que funciona solo dentro de ese router y en ese momento.



*Ruta que recibe la petición “/” y la desarrolla en la función anónima trayendo la vista “welcome”*

* **Route::get** → Estamos indicando que es una ruta que espera una petición http del tipo get.
* **‘ / ‘** → Es el parámetro esperado en esa petición
* **function(){ return view(‘welcome’);}** → Es la función anónima en la cual se resuelve que se hace con la petición.
* **return view(‘welcome’)** → Se indica que esa función tiene un retorno del tipo “view()” el cual retorna una vista llamada “welcome”

Rutas con Controladores

Generalmente estas son las más usadas. Imagínense que tenemos todas las vistas para hacer un ABM de usuarios, tenemos la vista Alta, Modificacion y Eliminacion, las cuales llevan una lógica de más de una línea. En lugar de estar armando toda la lógica con una función anónima haciendo un código espagueti, lo que se utiliza es apuntarlo hacia un método de un controlador, por ejemplo un controlador “UserController” que se encargue de manejar todo el ABM de usuarios. La manera de crearla es la siguiente:

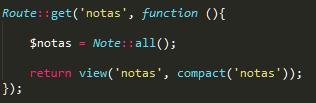


*Tenemos una ruta que espera un parámetro del tipo get llamado alumnos y que apunta al método index del controlador “AlumnosController*”

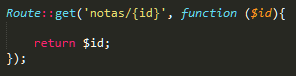
Dentro del método del controlador se llevará a cabo toda la lógica que requiera y se retorna una vista que será mostrada al usuario.

|  |
| --- |
| Para poder entender cómo utilizar un Controller y sus métodos es necesario leer el apartado de controladores |

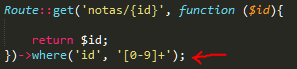
Pendiente **Laravel 5.4**



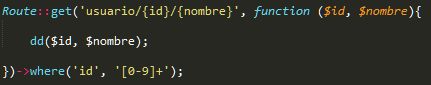
Con está ruta obtenemos la página notes, vamos a obtener todas las notas de la tabla Note y las vamos a mostrar en la vista notas, el comando compact guarda todas las notas en un array.



Le paso una variable desde la url, luego se la pasó a la función y la puedo utilizar.

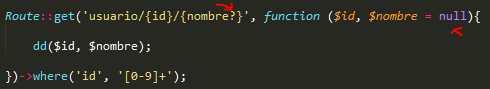


Le digo que el parámetro “id” solamente puede contener números del 0 al 9 y con el signo + le indico que pueden ser todos los que sean necesarios.



Le pasó por url dos parámetros, de los cuales el parámetro id sólo puede ser numérico y los muestro utilizando dd().

NOTA: dd() es equivalente a var\_dump($var)



Le indico que el segundo parámetro es opcional y le doy un valor por defecto de null

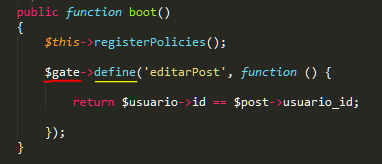
## Grupo de Rutas

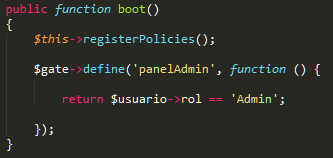
Explicado en el aparato de Middleware

## Rutas para Formularios

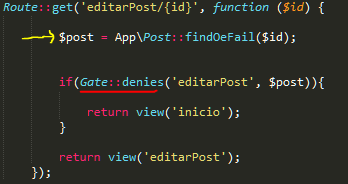
# Restricciones de Acceso

Aveces necesitamos que determinada hoja sea vista únicamente por un determinado usuario, para ello, podemos aplicar restricciones de acceso.

1. Ingresamos en: Providers → AuthServiceProvider.php
2. Creamos toda nuestra lógica dentro de la función **boot**

Instanciamos la variable $gate y usamos define para definir las reglas que tiene que cumplir. Como primer parámetro le decis el nombre de la regla y como segundo parámetro le pasamos una función en donde compara el id del usuario conectado con el id del creador del post, devolviendo un True o False según corresponda.

Otro ejemplo sería para un panelAdmin donde devolvemos true únicamente si el usuario conectado es del rol Administrador

1. Luego nos vamos a nuestra hoja de rutas y creamos la ruta en cuestion

Cosas a tener en cuenta, al crear la variable $post lo que le estoy diciendo es que busque en la tabla Post de la Base de Datos una fila donde el id sea el mismo que yo ingrese y que en caso de no encontrarlo nos arroje un error. Seguidamente llamó a nuestra restricción con la propiedad “denies” es decir, que si el usuario tiene el acceso restringido nos retorne a la vista de inicio.

*NOTA: Pude haber usar Gate:allows en donde si el usuario tenía acceso, lo retornaba a la vista que correspondía para editar el post.*

*NOTA 2: Si observan en el gate pedimos dos parámetros, post y user, pero en la ruta solo le enviamos el post, esto es porque laravel inyecta el usuario por nosotros, por lo que no es necesario enviarlo.*

## Restricción de Vista en la Plantilla

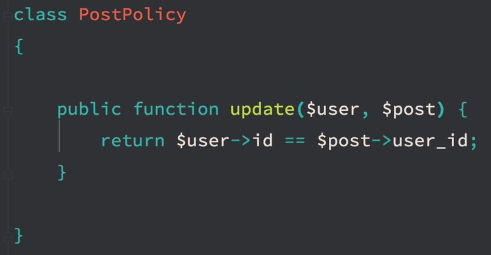
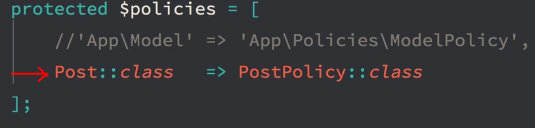
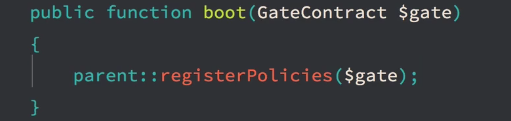
Supongamos que tenemos una lista de post creados y queremos que solamente el autor del post pueda tener acceso al boton de editar, bueno, tendríamos que hacer algo tan sencillo como esto en la plantilla de blade:



Utilizando las abreviaturas de @can y de @cannot pasando como parámetros el nombre del gate que usamos para crear la regla de acceso y en caso de que lo cumpla nos mostrará lo que entra dentro de @can @endcan

# Policy

Las policy son políticas o reglas para determinadas clases, con ella por ejemplo, podemos establecer las reglas de acceso para determinadas vistas o cosas.

1. Desde el CMD utilizar el comando **php artisan make:policy NombreModeloPolicy**
2. Dentro de nuestro archivo php de la carpeta policy aplicamos las reglas que habíamos aplicado antes para poder editar post
3. Ingresamos en el archivo Providers → AuthServiceProvider.php y referenciamos nuestra nueva clase en donde teniamos el código anteriormente
4. Luego, en la parte superior de nuestro archivo AuthServiceProvider.php debemos agregar la Clase
5. Ya no va a ser necesario declarar manualmente la función, ais que vamos a eliminarla y dejarla asi:
6. Para utilizarla dentro de la función de nuestro controlador:
7. Para utilizarla dentro de nuestra plantilla blade

# Modelos

Los Modelos son uno de los componentes principales en los patrones del tipo MVC, son los encargados de acceder a los datos, modificarlos, etc..

## Crear un nuevo Modelo

Para crear un nuevo modelo en Laravel tenemos que ingresar a nuestro CMD y teclear lo siguiente:

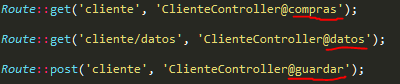
**php artisan make:model NombreModelo**

# Controladores

## Crear controlador

Para crear el controller debemos ingresar a la carpeta de nuestro proyecto desde CMD e ingresar el siguiente comando: **php artisan make:controller NombreController** creándose un nuevo controlador en la siguiente ubicacion: “/app/Http/Controllers/”

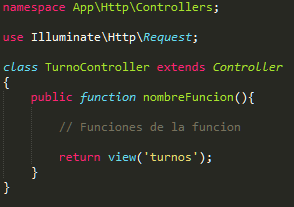
## Utilización



Es parecido a la utilización de las rutas comunes, con la diferencia de que en lugar de usar una función anónima (function(){ }) le indicamos a qué controlador apunta y a que sección de ese controlador apunta.

NOTA: Por costumbre se suele poner **Nombre**Controller

## Composición del Controller



Dentro de nuestra Class del Controller creamos funciones públicas en donde se llevarán a cabo las acciones y nos retorna la vista junto con todo lo que necesite. Pueden haber la cantidad de funciones que queramos. El nombre de la funcion sera lo que pongamos en la route, por ejemplo TurnoController@**nombreFuncion**

## Controladores Asociados a Modelos

Pendiente **Laravel 5.4**

# Plantillas Blade

Las Plantillas de Blade tiene una gran funcionalidad, algunas de las las principales son:

* Evitarnos el uso de tags php en todo nuestro código.
* Tener una Plantilla Base (Layout) en la cual poder poner nuestra cabecera y pie de página sin la necesidad de repetirla en cada hoja
* El poder escapar código html o javascript de nuestras variables.

## Mostrar una variable

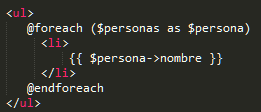
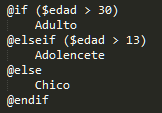
Blade nos permite mostrar variables de dos maneras:

* {{ $nombre }} la cual escapa etiquetas HTML
* {!! $nombre !!} que NO escapa etiquetas HTML

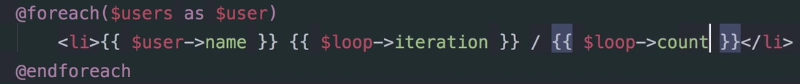
Es decir, si la variable $nombre tuviera como contenido <script> alert(“Buen Dia”) </script> utilizando la primer opción se vería como texto plano, mientras que con la segunda se activaria el alert de javascript.

## Bucles y Condicionales en Blade

Otra de las cosas que nos permite blade es ahorrar mucho codigo utilizando los clásicos bucles y condicionales (while, for, if, foreach, etc) por ejemplo:



### Variable $loop para foreach



Al foreach se le agregó la variable $loop, la cual cuenta con las siguientes opciones:

* iteration: número de iteración (comienza en 1)
* index: número de índice (comienza en 0)
* remaining: cuantos ítems quedan
* count: cuantos ítems hay
* first: verdadero sólo si es el primer ítem.
* last: verdadero sólo si es el último ítem.
* depth: número de ciclos anidados (comienza en 1)
* parent: variable loop del foreach anterior (puede ser null)

## Plantilla Base (Layout)

Para crear la plantilla base tenemos que crear una vista (view) nueva con el nombre de “layout.blade.php” (o el nombre que quieran, es opcional). En esa vista ponemos el código que queremos que se repita, por ejemplo, la cabecera con los archivos de referencia del css, nuestro logo, menú principal, pie de página, etc… En el lugar que queremos que se ingrese informacion propia de la hoja donde se encuentre ponemos: @yield('contenido') El nombre de “contenido” puede ser cualquiera que queramos.

Lo que nos quedaría por hacer es en las hojas que queremos aplicar la plantilla base, al principio de todo, poner: @extends('layout') y luego @section('contenido') todo nuestro contenido @endsection

Lo que hará blade es poner todo lo que tenemos dentro de la sección contenido dentro de la sección contenido de la plantilla layout.

TIP: Si por ejemplo queremos que el <title> de nuestra hoja varía dependiendo la hoja, podemos poner algo asi:

<title>Bienvenido a la Hoja - @yield('título')</title> **En nuestra plantilla layout**

@section('titulo', 'Compras') **En nuestra hoja**

## Incluir archivos en plantilla

Si por ejemplo tenemos una porcion de codigo que necesitamos reutilizar en otros lados, podemos hacer uso de **@include(‘ubicacion/del/archivo’)**

## Componentes y Slots

Pendiente **Laravel 5.4**

# Colecciones

## Mensaje o Función de Orden Superior

Pendiente **Laravel 5.4**

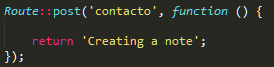
## Método Where

A partir de la versión de Laravel 5.3 se pueden hacer búsquedas más exactas en una colección

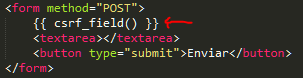


Los valores soportados son: = == === <= >= < > <> != !==

# Formularios



Para recibir un formulario se utiliza la ruta con post



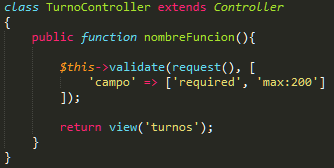
Al crear el formulario hay que agregarle el campo {{ csrf\_field() }} que es usado para proteger a los formularios de la aplicación de ataques de tipo CSRF. Si el formulario apunta a la misma ruta no hace falta indicarle el action.

# 

# 

# Validaciones en Laravel

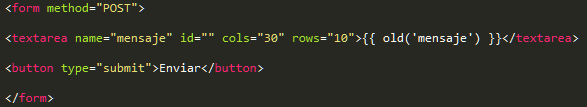
Laravel tiene una manera muy sencilla para realizar las validaciones, nos ubicamos en nuestro controller:



De está manera le estamos diciendo al controlador que valide los datos enviados por el formulario (request) y a continuación le indicamos cuales son los campos a validar. En este caso validamos el campo “Campo” diciendo que es requerido (required) y que puede tener un tamaño máximo de 200 caracteres (max:200)

## Evitar perder datos de formulario

Les puede suceder que al enviar el formulario y pasar por la validación este no sea aprobado, por lo que laravel no deja enviarlo y a su vez recarga la página, perdiendo todos los datos que el usuario haya cargado, para evitar que esto sucede, bastará con poner el Helper {{ old() }} en nuestro campo.



NOTA: El Helper **{{ old() }}** sirve para mantener los datos del formulario en caso de que lo enviado no supere la validación. Este Helper pide como primer parámetro el nombre del campo que tiene que recuperar.

# Facades

## Facades Automáticos

Pendiente **Laravel 5.4**

# Helpers Laravel

## Miscellaneous

### request()

El Helper **request()** devuelve un objeto de tipo *Illuminate\Http\Request*.

**request()->all()** → En este caso el método all() devuelve el array de datos enviado con el formulario.

**request()->get(‘note’)** → Trae solo el dato note del request

**request()->only([‘note’])** → Trae un array solo con esos datos.

#### Usando Facade Request

Request::all()

#### Usando Inyección de dependencias

public function store(Request $request)  
{  
 $request->all();  
}

# 

# Migraciones en Laravel

## ¿Qué son las Migraciones?

Las migraciones son como un control de versiones del estado de nuestras tablas. Trabajan en doble sentido, tanto para crear las tablas de la base de datos, como para destruirlas. Una de sus grandes ventajas es que te abstrae del motor de SQL que utilices, ya que crea automáticamente las tablas y al usar el ORM eloquent para las consultas sql podrías perfectamente cambiar tu motor sql de un dia para el otro sin mucho problema.

## Crear Tabla de Migraciones (Install)

Para poder utilizar el sistema de migraciones de Laravel tenemos que crear antes que nada la tabla “migrations” para ello, desde nuestro CMD, parados dentro de la carpeta de nuestro proyecto, ejecutamos la siguiente línea de comando:

**php artisan migrate:install**

Generandonos automáticamente una nueva tabla llamada “migrations”

## Crear una Migración

**nombre\_migration**: es el nombre de la migración

**--create=nombreTabla:** es el nombre que va a tener esa tabla

**php artisan make:migration nombre\_migration --create=nombreTabla**

## Eliminar tablas y volverlas a crear (Refresh)

Elimina todas las tablas de tu base de datos y las vuelve a crear, guiándose de las migraciones.

**php artisan migrate:refresh**

## Eliminar todas las tablas (Reset)

Elimina todas las tablas de tu base de datos

**php artisan migrate:reset**

## Deshacer última migración (Rollback)

Deshace el último grupo de migraciones. (Cada vez que enviamos una tanda de migraciones estas son marcadas en un número de lote, por lo cual si teníamos 5 migraciones en el lote 1, y 2 en el lote 2, cuando hagas el RollBack solo va a deshacer las del lote 2 y si volves a hacer rollback las del lote 1

**php artisan migrate:rollback**

## Ver estado de las migraciones (Status)

Con este comando puedes saber en qué estado están tus migraciones, si están cargadas o no

**php artisan migrate:status**

## Migrar a Base de Datos (Migrate)

Carga todas las migraciones a la Base de Datos

**php artisan migrate**

# Seeds (Generar datos de prueba)

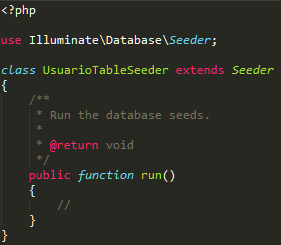
La función de los seeders es inicializar las tablas con datos, ahorrandonos el tener que estar cargando datos a mano.

## Generar nuevo Seeder

Para generar un nuevo seeder tenemos que ir a nuestro CMD, dentro de la carpeta de nuestro proyecto y tipear el siguiente comando:

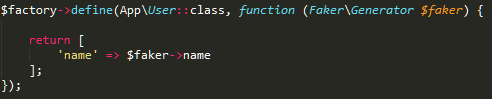
**php artisan make:seeder NombreTableSeeder**

Como siempre, el nombre es a gusto, generalmente yo pongo el nombre de la tabla, seguido por la palabra Table o Tabla y Seeder.



## Generar datos con Factory y Faker

Nos vamos a la carpeta database->factories->ModelFactory.php y creamos una nueva $factory en el cual indicamos la Clase a usar y dentro del return los campos que queremos generar datos falsos junto con el tipo a generar

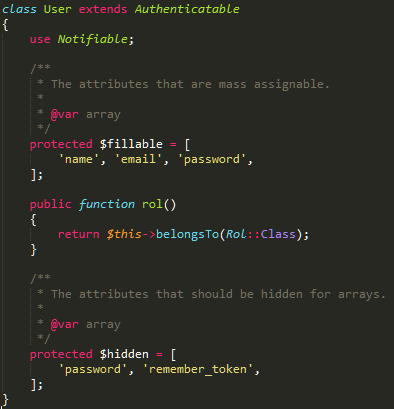


Algunos ejemplos son:

* name: genera nombres falsos
* email: genera emails falsos
* paragraph: genera un párrafo falso.
* Lista de Formatos: <https://github.com/fzaninotto/Faker#formatters>

## Generar datos con Tablas Relacionales

En caso de que tengamos una segunda tabla que se relaciona con la tabla que queremos generarle datos, tendríamos que hacer lo siguiente:

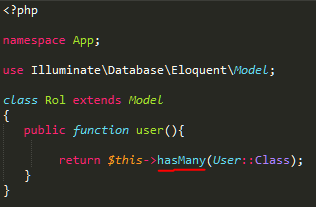


public function nombre(){

return $this->belongsTo(::Class): }

Ir a nuestro modelo en cuestión y agregarle la función, en nuestro caso rol, retornando una relación con Eloquent, con una relación del tipo belongTo ya que un Usuario pertenece a un Rol de Usuario

Ahora lo que nos faltaria es en el Modelo de Rol aclarar la relación con notas, para ello vamos al model Rol y lo agregamos:



Es muy similar al anterior, con la diferencia de que un Rol tiene muchos usuarios, por eso utilizamos “hasMany”

public function raza(){

return $this->hasMany(Raza::Class);

}

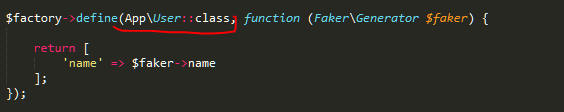
**NOTA: Para que esto funcione no hay que olvidarse de crear el Seeder para Rol**

## Darle funcionalidad al Seeder

Una vez que tengamos nuestro seed creador y nuestro factory armado, solo nos bastará unirlos para que el seed funcione

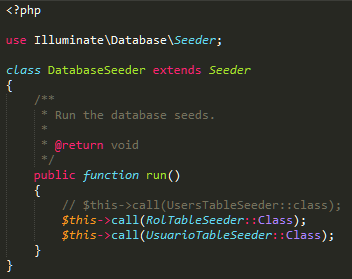
Se agrega la línea factory, indicando la clase, con “times” indicamos la cantidad de datos falsos que se van a generar y con “create” le decimos que lo cree.

NOTA: Recuerden que en el ModelFactory le indicamos a través del define a que tabla correspondía

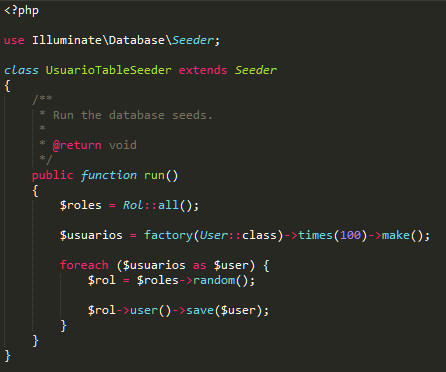


## Relacionar los Seeders con DatabaseSeeder

Para que todo esto funcione, no hay que olvidarse de relacionarlos con dicho archivo, para ello ingresamos, hacemos la llamada y listo, teniendo en consideración que primero van las tablas fuertes y luego las débiles.



Por último, volvemos a nuestro tabla UsuarioTablesSeeder y la modificamos para incluir los roles



Lo que hacemos en este caso es:

* En la variable $roles guardamos todos los roles que tenemos
* En la variable $usuarios generamos 100 usuarios fakes
* Recorremos los 100 usuarios que tenemos usando un foreach

## Ejecutar los Seeders (Seed)

Para poner en funcionamiento todos los Seeders que tengamos creados bastará con utilizar el siguiente comando para ejecutar DatabaseSeeder: **php artisan db:seed**

## Ejecutar seeders con Rollback (Refresh)

Otra manera es ejecutar los Seed pero realizando un Rollback a las migraciones, evitando asi algun tipo de error, con el comando: **php artisan migrate:refresh --seed**

# Traducción de Módulos de Laravel

Para traducir y lograr que nuestra web sea multilenguaje, en donde pongamos textos como “login” “remember me” etc.. lo podemos reemplazar por **{{ trans('auth.login\_title') }}** o sino **@lang('auth.login\_title')** Entonces, Laravel detectará el idioma del usuario y utiliza el idoma que corresponda.

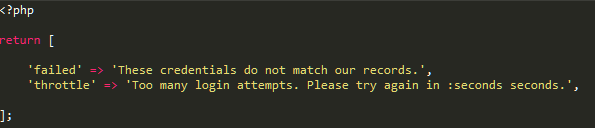
## Traducciones basadas en JSON

Pendiente **Laravel 5.4**

## Archivos de Traducción

El archivo de traducción se encuentra ubicado en: resources→ lang → en (o es, dependiendo el idioma, si queres multilenguaje o si queres pisar el inglés sin modificar otro archivo)

Aquí creamos el archivo auth.php, Dentro del archivo creamos un array por ejemplo



## Modificar idioma de la aplicación

Para modificar el idioma de nuestra aplicación para que sea de manera nativa español vamos a:

**Config→ app.php** y cambiamos la variable ‘locale’ => ‘es’

*NOTA 1: Recordar que si ponemos “es” como idioma base, necesitamos tener la carpeta “es” dentro de “lang” para el idioma*

*NOTA 2: la variable ‘fallback\_locale’ sirve para que si Laravel no encuentra traducción al español de una palabra use su versión en inglés.*

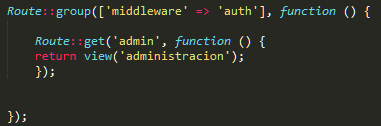
# Middleware

## ¿Que son?

Son un mecanismo que se utiliza para filtrar las peticiones HTTP en una aplicación. Por ejemplo si queremos saber si un usuario está logueado o no y si puede ver determinada página.

## Control de Sesión con Middleware y Route Group

Si tenemos varias vistas que piden que si o si haya un usuario conectado, en lugar de pasarle el middleware a cada una de ellas podemos usar los Grupos de Rutas los cuales les indican a Laravel que las siguientes rutas tienen que cumplir con un determinado requisito.



De está manera, cuando quiera ingresar a la ruta de admin, primero va a verificar a través del middleware de autenticación que haya un usuario conectado. (En este ejemplo hay un solo usuario en el sistema, pero si hubiera mas usuarios habría que aclarar que además tiene que tener un determinado rol).

## Middleware Personalizado

Alguna veces los Middleware que trae por defecto Laravel no son suficientes para nuestro sistema, por tal motivo, nos da la opción de crear los nuestros propios.

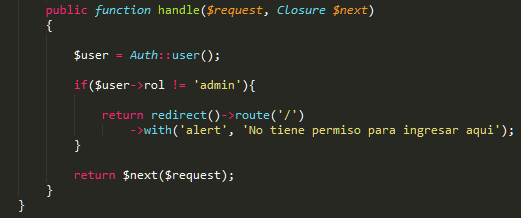
### Crear Middleware

Para crearlo bastará con ingresar en nuestro CMD y usar el siguiente comando:

**php artisan make:middleware nombreMiddleware**

#### Implementar la Lógica al Middleware

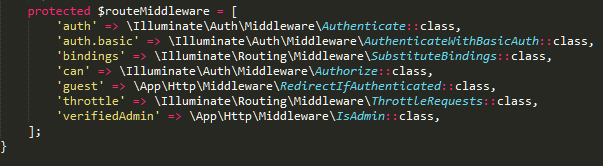
Una vez creado lo que nos compete ahora es darle nuestra lógica de negocio.



En este caso usando la variable $user traigo los datos del usuario, luego con el if consultó si el rol de ese usuario es admin, en caso de no serlo, lo redirijo a la vista index y le envió un mensaje de alerta.

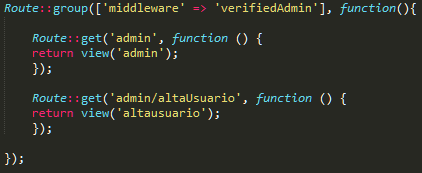
#### Registrarlo en el Kernel

Una vez creado lo tenemos que registrar dentro del archivo Kernel, ubicado en la carpeta HTTP.



Lo registramos dentro de la variable $routeMiddleware

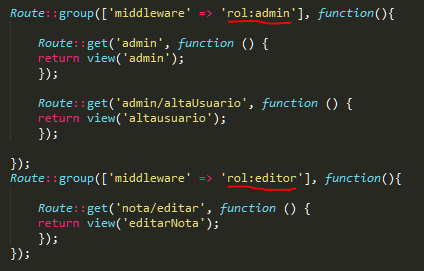
#### Crear el Grupo de Rutas



#### Middleware con Jerarquías

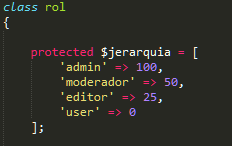
Una manera muy buena de verificar si un usuario tiene permiso de ingresar a una determinada pantalla es a través de los roles de usuario por niveles. Puede suceder que tengamos una pantalla que por ejemplo, sea la de edición de una nota de una revista, la cual solo tendrán acceso los Editores y Administradores. En lugar de estar aclarando que esos dos roles podrán ingresar, solamente vamos a aclarar a través de las jerarquías, que a partir de la jerarquía más pequeña aceptada en adelante podrán ingresar a esa seccion (si asi se lo necesitase)

##### Modificando el Grupo de Rutas



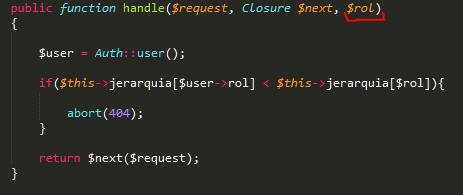
Lo primero que haremos será modificar el grupo de las rutas, agregando la variable admin y editor, que ese sería el rol que pedimos como mínimo.

##### Modificamos nuestro MiddleWare



Al comienzo de nuestro middleware agregamos la variable de jerarquía y le indicamos cuales son las jerarquías disponibles, asignándoles un valor de importancia.

##### Lógica del Middleware



Lo primero que hacemos es agregar el parámetro a nuestra función **handle**, en nuestro caso el parámetro sería $rol.

Luego lo que hacemos es con $user traer el usuario que está activo ahora y en el if preguntar si el rol del usuario activo es menor al rol que nosotros le pasamos proveniente del grupo de rutas… en caso de ser menor no lo vamos a redireccionar ni avisar del error, solamente le vamos a dar un mensaje de error 404 (página no existente) porque el usuario no tiene porque saber que esa ruta existe porque no es algo que lo afecte, además damos un poco de seguridad extra.

## Middleware para Formulario

### TrimString

Pendiente **Laravel 5.4**

### ConvertEmpetyStringToNull

Pendiente **Laravel 5.4**

# Login y Registro

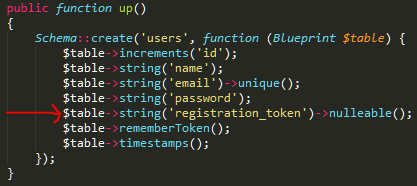
((MEJORAS DE **LARAVEL 5.3** PENDIENTE))

## ((PENDIENTE)) Restricción de Acceso por Login Fallido

## ((PENDIENTE)) Email de confirmación de registro

Generalmente necesitamos que los usuarios que se registran en nuestra web reciban un email con la confirmación de la misma, la forma en que lo hace es guardar un token que luego es comparado con el token que recibe el usuario. Para lograr esto debemos seguir estos pasos:

1. Debemos guardar dicho token en algún lado, por ejemplo, la tabla de usuarios.



Marcamos que va a ser nullable ya que una vez el usuario se registre, vaciaremos ese campo. Obviamente en las tablas de Factory y Seeders no agregaremos este campo, ya que no es necesario verificar a los usuarios generados aleatoriamente.

1. s

## Recuperar Contraseña

Para poder realizar el envío del reset de contraseña es necesario manejarse con un SMTP (Servidor de Correos Electrónicos), para ello, hay 3 formas:

1. Usando un SMTP
2. Usando un SMTP Fake
3. Usando Logs

Las últimas dos formas solo sirven para corroborar el correcto funcionamiento, la única manera correcta de hacer el envio es con un servidor SMTP.

# Envío de Emails en Laravel

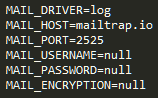
## Loguearse con Usuario en lugar de Email

Dentro de nuestro AuthController agregamos está linea, siempre considerando de que exista el campo en nuestra tabla:

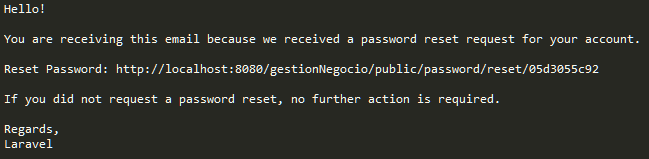
**protected $username = 'username';**

## Envio de mail vía log

Una manera de corroborar que la parte de programación de nuestro envio de mail funcione correctamente, es a través del sistema de Logs, la cual se configura de la siguiente manera:



Ingresamos al archivo .env y modificamos MAIL\_DRIVER= dándole como valor “log”, de está manera cuando alguien pida una recuperación de clave nos va a llegar a nosotros en forma de log. Esto lo podemos ver en: storage→ logs → laravel.log

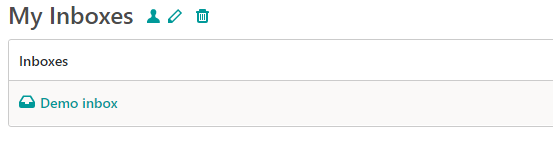


## Envio de mail vía Fake SMTP

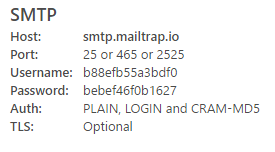
Una manera de poder enviar mails de prueba sin estar llenando nuestro correo con basura es usando los llamados “Fake SMTP”. Estos simulan todo el proceso de envío de mails, pero sin enviarle el mail al usuario final.

#### PASOS PARA CONFIGURARLO

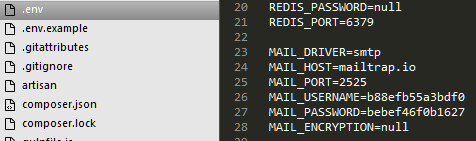
1. Ingresamos en <https://mailtrap.io>
2. Nos registramos y logueamos



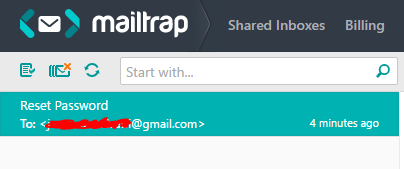
1. Nos va a mostrar una pantalla como esa, entramos en “Demo Inbox”



1. Nos aparecerán muchos datos, lo importante aquí son los datos: Host, Port, Username y Password
2. Regresamos a nuestro archivo .env de Laravel y modificamos los datos correspondientes para el mail



1. Una vez modificados, intentamos recuperar la contraseña y por ultimo ingresamos a la web de mailtrap y corroboramos la llegada correcta del mail



## Mailables

Pendiente **Laravel 5.3**

### Fakes para comprobar envio

Pendiente **Laravel 5.3**

### Markdown

Pendiente **Laravel 5.4**

# Notificaciones

## Notificaciones vía Email

Pendiente **Laravel 5.3**

## Notificaciones vía SMS

Pendiente **Laravel 5.3**

## Notificaciones vía Slack

Pendiente **Laravel 5.3**

## Notificaciones con PHPUnit

Pendiente **Laravel 5.3**

# Laravel Scout

## ¿Que es Algolia?

Algolia permite indexar nuestra base de datos y utilizarla para hacer busquedas rapidas.

Pendiente **Laravel 5.3**

# Paginación

Crear una paginación en Laravel es muy sencillo. Lo que nosotros usualmente haciamos si, por ejemplo, queríamos traer los nombres de los usuarios, era usar: **$users = Usuario::all();** y luego guardarlo como vector y manejarlo. En este caso se modifica un poco y se utiliza: **$users = Usuario::paginate();**

*NOTA: El método paginate() puede ser usado con cualquier consulta de eloquent permitiendo recorrer el resultado como un array.*

Luego, a la hora de mostrarlo con el método all, creabamos un foreach que recorre el vector y muestre los datos de nombre, eso queda tal cual está, solamente hay que agregar al final de todo, en el lugar que queramos que se vea la paginación la siguiente línea: **{!! $users->render() !!}** y con estos simples pasos ya tenemos disponible la paginación de datos.

*NOTA: el método render() nos genera un HTML que es compatible con el css de bootstrap y es el que nos genera la paginación*

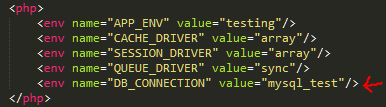
## Personalización

Pendiente **Laravel 5.3**

# Test (PHPUnit)

## Utilizar una DB para Testing

Se recomienda tener una segunda base de datos local para realizar las pruebas de test sin afectar a la base de datos principal, para ello, hay que seguir los siguientes pasos:

1. Crear una Database nueva, preferentemente con el mismo nombre que la principal, agregando al final “\_test”
2. Ingresamos en config/database.php al apartado de conexiones
3. Eliminamos las bases de datos que no vamos a utilizar, en este caso, como nuestro proyecto usa MySQL no vamos a necesitar pgsql ni sqlite.
4. Realizamos una copia del modelo que vamos a usar agregando el sufijo “\_test” tanto al nombre del motor como al de la database:
5. Desde nuestro CMD realizamos la migración para la DB de test, utilizando el comando: **php artisan migrate --database=mysql\_test**
6. Ahora nos toca configurar el archivo “phpunit.xml” agregando el último apartado para indicarle que base de datos tiene que usar para las pruebas

## Evitar generar datos en DB

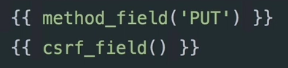
A la hora de hacer test puede que tengamos uno que cree datos para ver si determinada herramienta funciona correctamente, trayendo consigo el problema de generar datos basura de manera constante. La solución a este problema es utilizar la propiedad **Transactions** la cual nos permite hacer transacciones con la base de datos y cuando termina el test automáticamente hace un Rollback y borra los datos que hayan sido guardados, es decir, evita la persistencia de datos. Basta únicamente en poner la línea **use DatabaseTransactions;**  dentro de nuestra clase de Test

## Métodos para Pruebas

* visit: Para visitar una ruta. Parámetro: nombre de la ruta.
* see: Para indicar un texto que esperamos ver. Parámetro: texto.
* dontSee: Para indicar un texto que **no** esperamos ver. Parámetro: texto.
* seeLink: Para indicar un link que esperamos ver. Parámetro: nombre del link
* click: Para hacer clic en un elemento. Parámetro: nombre del elemento
* seeInElement: Para ver un texto en un elemento. Parámetro: tipo de elemento, texto que queremos ver.

# Almacenamiento de Archivos (Store)

Rápido instructivo de como poder subir el avatar del usuario a Laravel y poder visualizarlo:

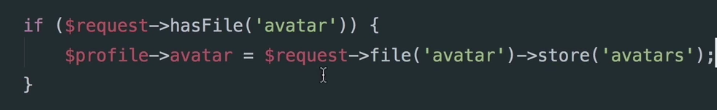
1. Primero que nada tenemos que tener en nuestra tabla el campo necesario
2. No olvidarse de hacer la relacion de las tablas avatar y usuario, en el caso de que tengas dos tablas distintas
3. Dentro del formulario donde haremos la carga, no hay que olvidarse estas dos cosas, , la primera sirve para indicarle a laravel que es un formulario del tipo put

NOTA: Put significa que modificaremos un recurso.

1. A la etiqueta form agregarle el campo:

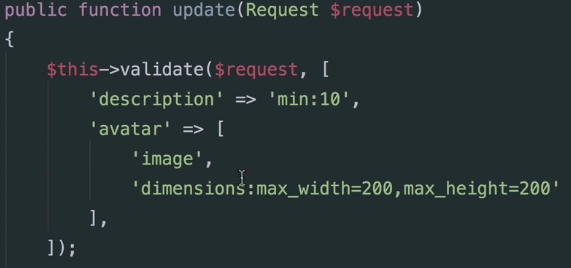


1. Ahora bien, los archivos que suban se colocaran en la carpeta storage, la cual **no es publica** para los usuarios, por lo cual tendremos que ir a ph artisan y escribir lo siguiente: **php artisan storage:link** lo que estamos haciendo en este paso es vincular la carpeta privada de storage con la publica, de manera que los usuarios puedan ver esas imagenes. Otra forma sería directamente cambiar la ruta y que se suba todo en la carpeta publica, pero por cuestiones de ordne no conviene.
2. Para que lo subido pueda verse en el php, tiene que tener está tipo de ruta
3. En el controlador, para evitar que nos de error si el usuario no subio una avatar pero si modifico datos del formulario



## Restricción de Subida y validación dimensión de Imagen

Para evitar que nos suban una imagen gigante o un archivo que no es una imagen, en el controlador que se encarga de todo este agregamos:



De está manera, le estamos indicando que el campo descripcion (un posible campo de nuestro formulario) debe tener mínimo 10 caracteres, mientras que el campo avatar debe ser de extension tipo imagen y que debe tener esas dimenciones maximas.

Algunas de las opciones son:

* **image :** con la que podemos validar que solo sean imágenes el tipo de archivo que se suba.
* **dimensions :** que nos permite restringir las dimensiones de una imagen especificando las opciones: altura y anchura mínima (min\_height y min\_width), altura y anchura máxima (max\_height y max\_width), dimensiones específicas (height y width) o ratio.
* **file :** para validar que un archivo fue subido exitosamente.
* **max:value :** para restringir el tamaño máximo en Kb que puede tener la imagen o archivo.
* **min:value :** para validar el tamaño mínimo en Kb del archivo o imagen.
* **mimes:jpg,png :** para restringir por extensión según su MIME types.
* **mimetypes:text/plain :**para validar el MIME type del archivo.
* **size :**value para restringir a un tamaño específico expresado en Kb la imagen o archivo.



## Unique



Se da la aclaracion de ignorar el id por que sino cuando quieras actualizar otros datos que no sean el nick de usuario va a dar error por que va a considerar que queres modificarlo

## in

Pendiente **Laravel 5.3**

## dimensions

Pendiente **Laravel 5.3**

## exists

Pendiente **Laravel 5.3**

## not int

Pendiente **Laravel 5.3**

## Descarga de Archivos

Pendiente **Laravel 5.3**

## Protección de Archivos

Pendiente **Laravel 5.3**

# Eloquent ORM

## firsOrCreate

Pendiente **Laravel 5.3**

## Optimizar consultas con Eager Loading N:1

Pendiente **Laravel 5.3**

## Método toggle N:N

Pendiente **Laravel 5.3**

## Método witch Default para 1:1

Pendiente **Laravel 5.3**

## Eventos Basados en clases

Pendiente **Laravel 5.4**

# Laravel Mix

Pendiente **Laravel 5.4**

# Laravel Dusk

Pendiente **Laravel 5.4**

# Otros detalles

## Error en Database

Todo lo que está en la carpeta database no trabaja con el standard psr-4 sino que con el classmap para el autoload de las clases, así que si creas nuevas clases o cambias el nombre de alguna tienes que hacer:

**composer dump-autoload**

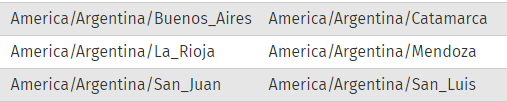
## Foreign key constraint is incorrectly formed

Para que la migración funcione de manera correcta primero debes crear la migracion Paises y luego Usuarios, de esa manera la migracion de Usuarios encuentra la clave foranea de la tabla Paises y así sucesivamente.

<https://laraveles.com/foro/viewtopic.php?id=6034>

## Zona Horaria



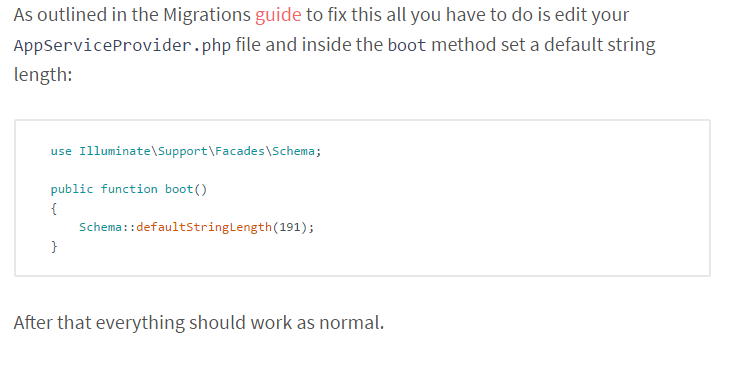


[Noticia Original](http://pathros.blogspot.com.ar/2014/10/laravel-4-configurar-la-zona-horaria.html) | [Todas las Zonas Horarias](http://php.net/manual/es/timezones.america.php)

## Manejador de Fechas (Carbón)

Link a la documentación: <http://carbon.nesbot.com/docs/#api-getters>

## Specified key was too long error



use Illuminate\Support\Facades\Schema;

Schema::defaultStringLength(191);

## Instalar proyecto bajado de repositorio

1. composer install
2. Crear archivo .env basandonos en .env.example
3. php artisan key:generate
4. crear DB a mano
5. php artisan migrate:install
6. php artisan migrate
7. php artisan db:seed

<https://styde.net/como-instalar-proyectos-existentes-de-laravel/>