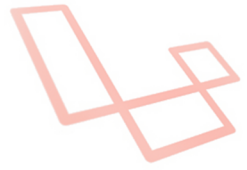


# Curso de Laravel 5 en español



## Curso de Laravel 5

[duilio.me](http://duilio.me)

16/03/2015

```
1  <?php
2
3
4  class Idea extends Eloquent {
5
6      /**
7       * Dreaming of something more?
8       *
9       * @with  Laravel
10      */
11     public function create()
12     {
13         // Have a fresh start...
14     }
15
16 }
17
```



# Presentación

## Cree en tí

Terminó la entrevista y me informaron que otro candidato y yo habíamos pasado, **pero sólo tenían un puesto**, así que nos iban a poner a prueba durante una semana.

Era mi primera entrevista para mi primer trabajo, y aunque no dudaba de mis habilidades, sentía que iba a enfrentar a programadores con mucha más experiencia que yo.

Mis sospechas parecieron comprobarse cuando me enteré que el otro candidato era un **hacker** de un chat IRC que yo frecuentaba, de hecho él había creado su propio cliente de IRC cuando yo apenas sabía deletrear “programación”.

Pensé que estaba perdido, sin embargo no podía rendirme y fui el Lunes a trabajar a primera hora.

Comenzó la jornada, estábamos en escritorios separados, así que no podía ver su monitor, pero era indudable que **tecleaba con una velocidad increíble**, yo no podía ni siquiera escribir en el chat a esa velocidad, menos aún programar.

Aún así decidí hacer mi mejor esfuerzo, nos asignaron 7 módulos y el viernes había terminado 5. Al final la empresa decidió contratarnos a ambos. El esfuerzo había valido la pena.

A la siguiente semana decidí preguntarle a mi colega cuantos módulos había terminado él: **uno y medio, me dijo**. ¿Qué? No podía creerlo, así que la siguiente vez que lo escuché teclear tuve que voltear y ver su pantalla. **Estaba chateando en el MSN**. Con la novia. Hizo lo mismo la semana previa. Increíble.

Con mi framework sucedió una historia similar, la versión 2 traía autenticación integrada, generadores y otros features que no se incluyeron hasta Laravel 5. Pero nunca pensé que fuera lo suficientemente bueno, y quizás no lo era, pero ¿Y eso qué? Publicarlo me habría permitido dar a conocer mi trabajo, recibir feedback, sugerencias y aportes de otros.

Con este blog hice lo contrario, instalé un WordPress en 5 minutos y comencé a grabar videos con el micrófono del celular y el software que trae mi Macbook, pero no dude en mi capacidad de transmitir conocimientos, e incluso contar alguna anécdota o dar algún consejo de vez en cuando.

Diariamente leo a programadores que dicen cometer errores estúpidos o tener dudas tontas pero ¿Saben qué? Yo también me equivoco y bastante, y he tenido las mismas dudas que Uds. y aún tengo muchas, pero ya no temo preguntar, tampoco temo equivocarme, es la única manera de aprender y crecer.

**Cree en ti**, si hay una tecnología o **idioma** que quieras aprender, **comienza hoy mismo**, trabaja en lo que quieres alcanzar aunque sea un poco cada día y tarde o temprano vas a conseguirlo.

# Índice general

<b>I</b>	<b>Conceptos básicos</b>	<b>5</b>
1.	Aprende Laravel 5: Instalación y uso de Composer	7
2.	Qué es PSR-4 y uso de los namespaces	10
<b>II</b>	<b>Curso de Laravel 5</b>	<b>12</b>
3.	Vagrant y Homestead	14
4.	La estructura de Laravel 5.	17
5.	Primeros pasos con Laravel 5	19
6.	Usando la autenticación de usuario integrada	21
7.	Creando Migraciones en Laravel 5	24
8.	Seeders y el componente Faker	28
9.	Fluent y Eloquent - Parte 1	33
10.	Fluent y Eloquent – Parte 2	40
11.	Cambios al motor de plantillas Blade en Laravel 5	43
12.	Componentes Html y Form a Laravel	48
13.	Listado y Paginación	52
14.	Creación de usuarios con Laravel y Eloquent	56
15.	Editar Registros con Laravel	60
16.	Validación - Parte 1	64

<b>17. Validación – Parte 2</b>	<b>68</b>
<b>18. Crear una página web o aplicación en español con Laravel (i18n)</b>	<b>70</b>
<b>19. Eliminar Registros</b>	<b>73</b>
<b>20. Aplicando el patrón de diseño de software DRY en Laravel 5</b>	<b>76</b>
<b>21. AJAX y Laravel usando jQuery</b>	<b>79</b>
<b>22. Búsquedas y filtros con Laravel y Eloquent (Query scopes) - Parte 1</b>	<b>83</b>
<b>23. Filtros de búsqueda y paginación con Laravel 5 - Parte 2</b>	<b>87</b>
<b>24. Crea tus propios Middleware con Laravel 5</b>	<b>91</b>
<b>25. Protege el acceso a tu aplicación con los Middleware de Laravel 5</b>	<b>92</b>
<b>III Tips</b>	<b>94</b>
<b>26. Cambiar contraseña con Laravel (validando clave actual)</b>	<b>96</b>
<b>27. Agregar rutas adicionales a un controlador de tipo resource en Laravel</b>	<b>100</b>
<b>IV Agradecimientos</b>	<b>102</b>

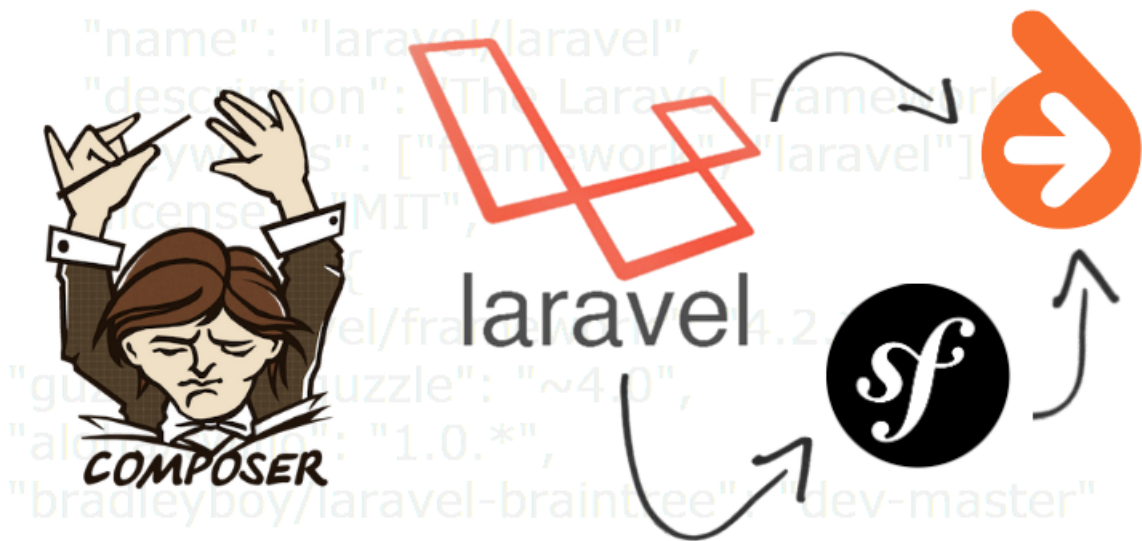
**Parte I**

**Conceptos básicos**



## Chapter 1

# Aprende Laravel 5: Instalación y uso de Composer



**Composer** es un manejador de dependencias de PHP, similar a NPM en node.js y a los bundles de Ruby. Es una de las mejores herramientas disponibles para el desarrollo en PHP hoy en día, nos permite manejar las dependencias de nuestros proyectos, ya sea la descarga de un framework completo como Laravel o Symfony, o componentes más sencillos como un generador de PDFs, esto lo podemos hacer a través de Composer.

En el siguiente video aprenderemos cómo instalar Composer en nuestro computador y luego cómo encontrar paquetes usando Packagist e instalarlos finalmente con Composer.

Ir a lección: [Instalación y uso de composer](#).



## Lecturas recomendadas

- [Qué es Composer y cómo usarlo](#)
- [Cómo instalar Composer en Windows](#)
- [Cómo instalar Composer en Ubuntu](#)

## Actividades

- Instala Composer de manera global en tu equipo Busca e instala un paquete utilizando Packagist y Composer
  - Ir a la página oficial y seguir los pasos según tu sistema operativo. [Sitio oficial de Composer](#)
- Aprende más sobre el archivo composer.json ([Tutorial en inglés](#))

## Preguntas

- ¿Cuál es la relación entre Composer y Laravel?
  - Composer es un manejador de dependencias, similar a lo que **npm** es a Javascript y a lo que **Bundler** es a Ruby, con unas simples líneas de código puedes descargar y actualizar desde un framework completo hasta un simple componente que tu proyecto necesite. Laravel entonces es un “paquete más” que instalas con Composer. Cuando creas un proyecto con Laravel lo haces a través de Composer y Laravel es una dependencia de tu proyecto: dicho de otra manera, tu proyecto va a depender de Laravel. Laravel a su vez depende de varios componentes de terceros, entre ellos Symfony, Doctrine, todo lo cual es manejado automáticamente por Composer.
- ¿Es una ventaja o desventaja que los frameworks usen Composer (ej: Laravel vs Codeigniter)
  - Es una total ventaja, de esta forma se evita tener que estar bajando plugins y sus dependencias de forma manual, lo cual puede causar incompatibilidades, futuros errores, problemas de seguridad (al tener que buscar y descargar parches de forma manual, lo más probable es que nos olvidemos de ellos). Además se pueden conseguir muchos **paquetes**, los cuales podemos instalar y re utilizar en cualquier proyecto.



## Chapter 2

# Qué es PSR-4 y uso de los namespaces

En este segundo video les voy a hablar sobre la especificación PSR-4

PSR-4 fue creada por el grupo de [interoperabilidad de PHP](#)

Este grupo ha trabajado en la creación de especificaciones que nos permitan a nosotros, desarrolladores de PHP, estandarizar muchos procesos, como, en este caso, la manera como nombramos a las clases.

El uso de especificaciones PSR puede ser opcional o parcial, pero particularmente el patrón PSR-4 lo encuentro necesario para estructurar nuestras clases.

La ventaja de usar estándares como el PSR-4 es que le permite a Composer cargar nuestras clases automáticamente.

También aprenderemos qué son los namespaces (espacios de nombres) en PHP y cómo combinarlos con la especificación PSR-4 para ordenar y estructurar mejor nuestras clases.

Ir a lección: [PSR4 y uso de los namespaces](#).

## Tutoriales recomendados

- [Instala y usa Eloquent \(el ORM de Laravel\) con Composer \(sin instalar Laravel\)](#)

## Actividades

- Crea un proyecto con Composer y configura el `composer.json` para que utilice PSR-4 y agrega varias clases al proyecto siguiendo la estructura que vimos en clase, luego usa dichas clases en otro archivo y comprueba que son cargadas automáticamente.
- Busca proyectos open source en [Packagist](#) y [GitHub](#) y revisa los archivos `composer.json` ¿Qué patrón utilizan para cargar clases?

## Preguntas

- ¿Cuál es la ventaja del uso de estándares en PHP?
  - Los **estándares** ayudan a asegurar que el código tenga una alta calidad, menos errores, y pueda ser mantenido más fácilmente por varios desarrolladores.
- ¿Cuál es la diferencia entre classmap y PSR-4?
  - El primero no provee un estándar como tal, por lo que podemos nombrar los archivos y clases como queramos, entonces para habilitar las capacidades de “autoload” Composer necesita escanear todos los archivos de la carpeta para crear así un “class map” en otras palabras: determinar qué clases están en qué archivos. Esto se logra con “composer dump-autoload” pero tiene la desventaja que tendremos que ejecutar el comando cada vez que agreguemos o modifiquemos una clase. Mientras que con PSR-4 creamos un estándar que nos permite crear más archivos y clases sin necesidad de actualizar. El estándar explicado de manera sencilla es: Los nombres de espacio (namespace) reproducen la estructura de carpetas, mientras que el nombre de la clase será idéntico al nombre del archivo. La manera más conveniente y más fácil de localizar clases tanto para Composer como para nosotros los programadores, por supuesto es usando el estándar PSR-4
- ¿Conoces otros estándares de estructurar y autocargar clases?
  - Anteriormente se usaba PSR-0, pero este está ahora obsoleto

## **Parte II**

# **Curso de Laravel 5**



## Capítulo 3

# Vagrant y Homestead

Cuando comenzamos a desarrollar aplicaciones, lo primero que necesitamos es un entorno que pueda ejecutarlas.

En el caso de PHP usamos herramientas como XAMPP, WAMP o MAMP que nos permiten instalar y configurar rápidamente Apache, MySQL y PHP. Pero esta facilidad viene con un costo: la dificultad de personalizar o agregar componentes extras a nuestro entorno de desarrollo.

Es por esto que crearon Laravel Homestead, un paquete para Vagrant que nos permite tener lo mejor de dos mundos: facilidad de instalación y configuración, pero la posibilidad de personalizar completamente nuestro entorno.



Además usar Vagrant y Homestead garantiza que tendrás un ambiente más similar al ambiente de producción y tus colegas podrían trabajar con el mismo ambiente también.

La excusa: es que solo funciona en mi computadora ya no es válida en el 2015.

Quizás si nunca has usado Laravel u otro framework de PHP una herramienta como XAMPP es más adecuada para ti, pero si buscas o quieres conocer entornos más profesionales, en el siguiente videotutorial de Laravel te explicaré cómo instalar Homestead: [Vagrant y Homestead](#).

**Nota:** Si estás usando Windows, quizás necesites habilitar la virtualización de hardware (VT-x) desde tu BIOS.

## Tutoriales recomendados

Dimitri Acosta creó dos excelentes tutoriales sobre Laravel Homestead 2 donde encontrarán los comandos utilizados en el video, paso a paso, se los dejo a continuación:

- [Cómo instalar y configurar Laravel Homestead 2](#)
- [Instala PHPMyAdmin en Laravel Homestead 2](#)
- [Documentación oficial de Laravel Homestead \(en inglés\)](#)
- [Otras alternativas para configurar tu entorno de desarrollo](#)

## Actividades

- Instala [VirtualBox](#), [Vagrant](#) y [Laravel Homestead](#) en tu computador
- Crea tu primer proyecto de Laravel tal como se demuestra en el [video](#)

## Preguntas

- ¿Cuáles son las ventajas de Homestead sobre herramientas como XAMPP o WAMP?
  - Con Homestead, todos los desarrolladores cuentan con un entorno de desarrollo idéntico y así se evitan conflictos de versiones, de igual manera es muy fácil descargar nuevas extensiones de PHP y configurar el entorno, sin tocar la configuración de nuestro propio sistema operativo (dado que todos los paquetes los instalaremos en la máquina virtual)
  - Sin embargo, para proyectos sencillos, plataformas como WAMP, XAMPP o MAMP tienen como principal característica que son de muy fácil instalación y nos permite tener un entorno local de desarrollo con sólo ejecutar un instalador
- ¿En qué casos es mejor usar una herramienta como XAMPP y en cuáles Homestead?
  - Homestead, se aprovecha más en situaciones donde existe un equipo de trabajo y se desea evitar conflicto de versiones, también cuando se requiere personalizar el entorno de desarrollo local (instalar nuevas extensiones, etc.)
  - WAMP o XAMPP, son de gran ayuda para quienes trabajan solos o en proyectos simples





## Capítulo 4

# La estructura de Laravel 5.

La carpeta app/ ahora está encargada exclusivamente de alojar las clases de nuestra aplicación y usa el esquema PSR-4 que vimos en una lección anterior. Aquí es donde pasaremos la mayor parte del tiempo de desarrollo de nuestra aplicación.

Luego tenemos otras carpetas como bootstrap/ y la carpeta de configuración o config/ que ha sido simplificada en Laravel 5 y ahora usa variables de entorno.

Si quieres aprender más de la nueva estructura de Laravel 5 te recomiendo que veas el siguiente video: [Estructura Laravel 5](#).

### Artículos recomendados

[Estructura de tu aplicación en Laravel 5](#)



## Capítulo 5

# Primeros pasos con Laravel 5

Laravel 5

Simplicity is an acquired taste. - Katharine Gerould

En este capítulo explicaré cómo funciona la página de inicio Laravel 5, paso a paso, es ideal para principiantes o usuarios más avanzados que se preguntan cómo funciona el framework Laravel.

Ir a la lección: [Primero pasos Laravel 5](#).

### Tutoriales recomendados

- [Escribir rutas con Laravel](#)
- [Asociar rutas a controladores y Acciones con Laravel](#)



## Capítulo 6

# Usando la autenticación de usuario integrada

Avanzando en el estudio del código que viene por defecto en Laravel 5, vamos a abrir de nuevo el archivo de rutas en `app/Http/routes.php`

Veremos la siguiente ruta:

```
Route::get('home', 'HomeController@index');
```

Según lo aprendido en la clase anterior, para acceder a esta ruta debemos escribir `/home` en el navegador, pero si intentamos acceder a ella veremos que somos “redireccionados” a la ruta `/auth/login` ¿Por qué? La página “home” está protegida por un middleware.

En Laravel 5 los middleware (que en Laravel 4 se conocían como “Filtros”) están en el medio de la petición de usuario y el código para ejecutar dicha petición, y nos permiten proteger rutas entre otras utilidades.

Ir a lección: [Autenticación de usuarios - Laravel 5](#).

## Tutoriales recomendados

- [Migraciones en Laravel](#)
- [Autenticación de usuarios en Laravel 4](#)
- [Traits en PHP](#)
- [Middleware \(documentación oficial en inglés\)](#)
- [Ogres are like Middleware \(Laracasts, en inglés\)](#)

## Actividad

### Prueba la autenticación y registro de usuarios en Laravel 5

Podemos examinar el comportamiento del módulo autenticación escribiendo en el navegador: nombre\_de\_tu\_aplicación.app/home. Notarás que se te redigirá a la ruta auth/login, esto pasa porque la petición pasa primero por el middleware Authenticate.php y en él se indica que no tienes autorización para ir a esa ruta. Si en cambio en el archivo Http/HomeController comentamos la línea de código que hace el llamado al middleware:

```
// $this->middleware('auth');
```

e intentas ingresar a la ruta /home, notarás que ahora puedes acceder.

## Preguntas

- ¿Cuál es la ventaja de usar middlewares?
  - Ayuda a restringir acceso a rutas protegidas, con un código muy fácil de entender.
  - Evalúan las peticiones antes de que lleguen al controlador.
- ¿Cuál es la ventaja del uso de migraciones?
  - Facilitan la creación y modificación de las tablas en nuestro proyecto
  - Mejoran la portabilidad de la base de datos
  - Puedes “migrar” tus tablas en cualquiera de los motores de base de datos que Laravel soporta.
  - Fácil control de versiones de la base de datos cuando se está trabajando en equipo
  - Su uso es opcional, puedes implementarlas en tu proyecto o no
- ¿Encuentras otro caso en el que sea útil usar un middleware?
  - Restringir que solo los usuarios de determinado grupo pueden acceder a determinada sección





## Capítulo 7

# Creando Migraciones en Laravel 5

Las migraciones son una serie de archivos y clases en PHP que actúan como un control de versiones de base de datos, tanto para crear las tablas al inicio del desarrollo de nuestra aplicación, como para realizar cambios como agregar o eliminar una tabla, agregar o eliminar una columna y así sucesivamente, a medida que desarrollamos nuestro proyecto.

Las migraciones permiten definir las tablas con programación orientada a objetos en vez de SQL, lo cual nos da portabilidad a los diferentes motores que soporta Laravel: MySQL, Postgres, SQLite, y SQL Server.

También son ideales para que nuestro equipo de trabajo mantenga los cambios y pueda ejecutarlos con un comando en vez de estar adivinando que SQLs cargar y cuales no.

A pesar de su ventaja son totalmente opcionales, así que puedes prescindir de ellas y crear tus tablas con herramientas como PHPMyAdmin o SequelPro pero yo recomiendo usar las migraciones cada vez que se pueda.

Ir a lección: [Creando migraciones en Laravel 5](#).

### Material recomendado

- [Videotutorial: migraciones en Laravel 4](#)
- [Cómo encadenar métodos con PHP](#)
- [Laravel Fundamental: Migrations](#)
- [Documentación oficial: Migrations & Seedings](#)
- [Documentación oficial: Schema Builder](#)
- [Tutorial: configurar base de datos y migraciones en Laravel 4](#)

### Actividades

- Crea una nueva tabla usando una migración

- Modifica una tabla usando una migración

Para crear la migración para la tabla de **users** lo hacemos con la siguiente línea de comando:

```
php artisan make:migration create_users_table
```

Luego en la función **up** del archivo generado en **database/migrations/**, crear los campos de la tabla:

```
Schema::create('users', function(Blueprint $table)
{
    $table->increments('id');
    $table->string('first_name');
    $table->string('last_name');
    $table->string('email')->unique();
    $table->string('password', 60);
    $table->enum('type', ['admin', 'user']);
    $table->rememberToken();
    $table->timestamps();
});
```

Para crear la migración para la tabla de **user profiles** lo hacemos con la siguiente línea de comando:

```
php artisan make:migration create_user_profiles_table
```

Luego en la función **up** del archivo generado en **database/migrations/**, crear los campos de la tabla:

```
Schema::create('user_profiles', function(Blueprint $table)
{
    $table->increments('id');
    $table->mediumText('bio')->nullable();
    $table->string('twitter')->nullable();
    $table->string('website')->nullable();
    $table->integer('user_id')->unsigned();

    $table->foreign('user_id')
        ->references('id')
        ->on('users')
        ->onDelete('cascade');
    $table->timestamps();
});
```

En la consola instalamos nuestras migraciones con la siguiente línea de código:

```
php artisan migrate:install
```

## Preguntas

- ¿Cuáles es la ventaja de usar migraciones?
  - Podemos manipular las tablas de nuestra base de datos con programación orientada a objetos
  - Permite que la aplicación pueda ser instalada en los distintos motores de base de datos aceptados por Laravel
- ¿Cuando no se usan migraciones en un proyecto?
  - uando el proyecto es sencillo, con pocos o ningún cambio en la base de datos a futuro, la aplicación solo va a ser instalada en un solo motor de base de datos y no existe un grupo de trabajo que necesite tener el orden de las versiones de la base de datos.
  - Cuando las tablas del proyecto ya fueron creadas por un tercero o provistas por el cliente



## Capítulo 8

# Seeders y el componente Faker

Los seeders, son un recurso que nos permite cargar información a nuestras tablas para probar de manera sencilla y rápida el funcionamiento de nuestra aplicación (paginación, filtros, entre otros). Ya basta de estar insertando datos uno por uno, inventando nombres o colocando el típico y horrible “asdffasd” o “123”.

Los seeders se ubican en la carpeta database/seeds y puedes crear tantos como desees.

Cuando queremos crear un usuario tipo “admin” podemos hacerlo de forma manual en un seed, pero cuando queremos insertar muchos usuarios debe de existir una manera más agradable de lograrlo, ahí entra Faker.

Faker es un componente que genera datos para ti. De manera que con solo agregar el componente, hacer el llamado del mismo en el seed más unas simples líneas de códigos, podrás tener tantos datos como sean necesarios para evaluar tu aplicación correctamente.

Ir a lección: [Seeders y el complemento Faker](#).

## Tutoriales recomendados

- [Videotutorial: listar registros con Laravel y Codeception](#)
- [Documentación de Faker](#)
- [Documentación oficial: migrations and seedings](#)

## Actividades

### Crea un seed e inserta un registro en una tabla.

Para ello debemos de crear una clase que extienda de Seeder en la carpeta **database/seeds**

Ejemplo: clase AdminUserSeeder.php

```

<?php

use Illuminate\Database\Seeder;

Class AdminTableSeeder extends Seeder{

    public function run() {

        \DB::table('users')->insert(array(

            "first_name" => "Rafael",
            "last_name"  => "Torrealba",
            "email"      => "rafaeltorrealba6@gmail.com",
            "password"   => \Hash::make("secret"),
            "type"       => "admin",

        ));

    }

}
?>

```

En la función run del archivo **database/seeds/DatabaseSeeds** agregar el llamado al seed que creamos:

```
$this->call('AdminTableSeeder');
```

Luego instalamos de nuevo la migración pero con el parámetro **-seed** para que lea los seeders:

```
php artisan migrate:refresh --seed
```

## Instala Faker e inserta varios registros en una tabla.

Lo primero que debemos de hacer es instalar el complemento, para ello debemos de colocar la siguiente línea de código en nuestro archivo **composer.json** en la parte "require-dev":

```
"fzaninotto/faker": "1.5.*@dev"
```

Luego actualizamos composer en la consola con el siguiente comando:

```
composer update
```

Luego agregar la siguiente línea para llamar el complemento en nuestro controlador y colocarle un alias Faker:

```
use Faker\Factory as Faker;
```

Como siguiente paso, tenemos: crear un nuevo objeto y utilizar las funciones de complemento para que genere los datos, además utilizar un ciclo for para repetir el proceso tantas veces como usuarios querramos crear.

```
$faker = Faker::create();

for($i = 0; $i < 30; $i ++){

    $id = \DB::table('users')->insert(array(

        'first_name' => $faker->firstName(),
        'last_name' => $faker->lastName(),
        'email' => $faker->unique()->email(),
        'password' => \Hash::make('secret'),
        'type' => 'user'

    ));
}
```

### Inserta un registro en una tabla relacionada a otra.

Guardamos el «id» de usuario en una variable, para ello se utiliza la función insertGetId y luego insertamos los valores en el profile del mismo usuario.

```
$id = \DB::table('users')->insertGetId(array(
    'first_name' => $faker->firstName($gender),
    'last_name' => $faker->lastName(),
    'email' => $faker->unique()->email(),
    'password' => \Hash::make('secret'),
    'type' => 'user'
));

\DB::table('user_profiles')->insert(array(
    'user_id' => $id,
    'bio' => $faker->paragraph(rand(2,5)),
    'website' => 'http://www.'. $faker->domainName(),
    'twitter' => 'http://www.twiter.com/'. $faker->userName(),
```

```
));
```

## Preguntas

- ¿Cuál es la ventaja de los seeders?
  - Nos permite cargar información (generalmente de prueba) a nuestras tablas de forma agradable para probar de forma rápida el funcionamiento de nuestra aplicación.
- ¿Cuál es la ventaja de Faker?
  - Faker genera información de prueba por nosotros, además de que nos ahorra tiempo, los datos de prueba van a tener relación con lo que se desee mostrar en las vistas.
- ¿En qué otros escenarios podemos usar Faker?
  - Para generar datos de prueba para las vistas.





## Capítulo 9

# Fluent y Eloquent - Parte 1

En Laravel existen dos opciones para realizar nuestras consultas SQL.

En primer lugar Fluent, que no es nada más que un constructor de consultas SQL o “query builder” (basado en PDO) y está encargado de generar cualquier consulta a la base de datos, ya sea para traer, insertar, actualizar, o eliminar datos. Cabe destacar que además de facilitar la interacción con nuestra base de datos, las consultas generadas vienen por defecto con los niveles de seguridad para evitar inyecciones SQL en nuestras consultas.

Ir a la lección: [Fluent y Eloquent - Parte 1](#).

### Tutoriales recomendados

- Cómo encadenar métodos con PHP (POO)
- Documentación oficial: Query Builder

### Actividades

**Crea un query para imprimir todos los usuarios de tipo user.**

```
$result = \DB::table('users')
    ->where('type', '=', 'user')
    ->get();

dd($result);
```

**Crea un query para imprimir sólo el nombre y el email del usuario ordenados alfabéticamente.**

```

$result = \DB::table('users')
    ->select('first_name','email')
    ->orderBy('first_name','ASC')
    ->get();

dd($result);

```

### Genera una consulta para traer al usuario que tenga el ID 5

```

$result = \DB::table('users')
    ->where('id','=','5')
    ->get();

dd($result);

```

### Devuelve los usuarios de la DB dependiendo de su genero:

#### Agrega el campo gender a la migración

```

Schema::create('user_profiles', function(Blueprint $table)
{
    $table->increments('id');
    $table->mediumText('bio')->nullable();
    $table->string('twitter')->nullable();
    $table->string('website')->nullable();
    $table->string('gender')->nullable();
    $table->integer('user_id')->unsigned();

    $table->foreign('user_id')
        ->references('id')
        ->on('users')
        ->onDelete('cascade');
    $table->timestamps();
});

```

Luego con el seeder haz que algunos usuarios sean de género masculino y otros de género femenino

```

$gender = $faker->randomElement($array = array ('female', 'male'));

$id = \DB::table('users')->insertGetId(array(

    'first_name' => $faker->firstName($gender),
    'last_name'  => $faker->lastName,
    'email'      => $faker->unique()->email,
    'password'   => \Hash::make('secret'),
    'type'       => 'user'

));

\DB::table('user_profiles')->insert(array(

    'user_id'    => $id,
    'bio'        => $faker->paragraph(rand(2,5)),
    'website'    => 'http://www.' . $faker->domainName,
    'gender'     => $gender,
    'twitter'    => 'http://www.twitter.com/' . $faker->userName,

));

```

**Haz consultas para traer todos los usuarios de género masculino o femenino y ordenados alfabéticamente.**

```

$result = \DB::table('users')
    ->select('users.*', 'user_profiles.gender as gender')
    ->leftJoin('user_profiles', 'users.id', '=', 'user_profiles.
user_id')
    ->where('gender', '=', 'female')
    ->orderBy('first_name', 'ASC')
    ->get();

dd($result);

```

**Devuelve sólo usuarios activos:**

**Agrega el campo active en la migración.**

```
Schema::create('users', function(Blueprint $table)
{
    $table->increments('id');
    $table->string('first_name');
    $table->string('last_name');
    $table->string('email')->unique();
    $table->string('password', 60);
    $table->enum('type', ['admin', 'user']);
    $table->boolean('active')->default(true);
    $table->rememberToken();
    $table->timestamps();
});
```

Con el seeder haz que algunos usuarios están como activos (1) y otros como inactivos (0)

```
$id = \DB::table('users')->insertGetId(array(

    'first_name' => $faker->firstName($gender),
    'last_name'  => $faker->lastName,
    'email'      => $faker->unique()->email,
    'active'     => $faker->numberBetween(0, 1),
    'password'   => \Hash::make('secret'),
    'type'       => 'user'

));
```

Ahora crea una consulta para devolver el email de todos los usuarios activos.

```
$result = \DB::table('users')
    ->select('email')
    ->where('active', '=', '1')
    ->get();

dd($result);
```

**Trae sólo usuarios mayores de edad:**

**Agrega el campo birthdate a las migraciones y a los seeders**

```

Schema::create('user_profiles', function(Blueprint $table)
{
    $table->increments('id');
    $table->mediumText('bio')->nullable();
    $table->string('twitter')->nullable();
    $table->string('website')->nullable();
    $table->date('birthdate')->nullable();
    $table->string('gender')->nullable();
    $table->integer('user_id')->unsigned();

    $table->foreign('user_id')
        ->references('id')
        ->on('users')
        ->onDelete('cascade');
    $table->timestamps();
}
);

```

### Usa Faker para generar fechas aleatorias

```

\DB::table('user_profiles')->insert(array(

    'user_id'    => $id,
    'bio'        => $faker->paragraph(rand(2,5)),
    'website'    => 'http://www.' . $faker->domainName,
    'birthdate'  => $faker->date($format = 'Y-m-d'),
    'gender'     => $gender,
    'twitter'    => 'http://www.twitter.com/' . $faker->userName,
));

```

**Luego crea una consulta para traer sólo los usuarios que sean mayor de edad y ordenados del mayor al más joven.**

Para ello vamos a crear una variable «under\_date» la cual tendrá la fecha de hoy menos 18 años exactamente. De esta manera podemos mostrar los usuarios que tengan fecha de nacimiento menor a nuestra variable.

```

$under_date = \Carbon\Carbon::now()->subYears(18)->format('Y-m-d');

$result = \DB::table('users')

```

```

->select('users.*','user_profiles.birthdate as birthdate')
->leftJoin('user_profiles','users.id','=','user_profiles.user_id')
->where('birthdate','<',$under_date)
->orderBy('birthdate','ASC')->get();

```

```
dd($result);
```

## Preguntas

- ¿En qué ocasiones Fluent no nos protege de inyecciones SQL?
  - Las ocasiones en que no nos encontramos protegidos de inyecciones SQL es cuando usamos `DB::raw()`, ya que este método ingresa directamente lo escrito dentro en la sentencia SQL y no es escapada por PDO.
- ¿Cuáles son las ventajas y desventajas del uso de `DB::raw()`?
  - La ventaja de `DB::raw()` es la personalización o manejo directo de la sentencia SQL por nosotros mismos, si fuera alguna petición que no se encontrará con el uso de Fluent, este método nos da la flexibilidad de realizarla manualmente. La desventaja: nos podríamos exponer a inyección de SQL si no somos cuidados: Nunca pases datos que provengan del usuario directamente con `DB::raw()`
- ¿Cuál es la ventaja de realizar JOINS en nuestras consultas?
  - Nos ayudan a obtener datos compartidos por dos tablas con el uso de las llaves primarias y foráneas. Nos permiten escribir sentencias condicionales más complejas.
- ¿Qué error frecuente encontramos cuando realizamos un JOIN?
  - Lo que nos puede ocurrir es ver un campo duplicado, si ambas tablas cuentan con una columna del mismo nombre como usualmente es la columna "id", podemos confundirnos al no saber a qué tabla pertenece dicho dato. La solución: especificar alias (id as profile\_id, por ejemplo) al momento de crear la consulta con Fluente.





## Chapter 10

# Fluent y Eloquent – Parte 2

Hoy comenzaremos a usar Eloquent, el ORM de Laravel.

Cuando trabajamos con bases de datos SQL, usamos un sistema de datos relacional, donde un registro es representado a través de columnas de una fila de la base de datos. Cada registro tiene una llave primary (primary key, PK o ID) y se relaciona con otros registros a través de claves foráneas (foreign keys o FK).

Sin embargo nuestros proyectos los estamos construyendo con PHP y programación orientada a objetos. En la POO los objetos se relacionan con otros a través de propiedades y métodos.

Como podemos ver, las bases de datos relacionales son incompatibles con la forma con la que trabajamos en la programación orientada a objetos.

Es por esto que los ORMs como Eloquent nos permiten mapear datos de la base de datos para convertirlos a objetos y viceversa, tomar un objeto y guardarlo como un registro de la base de datos.

Ir a la lección: [Fluent y Eloquent – Parte 2](#)

## Material relacionado

- [Mapeo objeto-relacional \(Wikipedia\)](#)
- [Eloquent ORM \(Documentación oficial\)](#)
- [Eloquent 101 \(Laracasts\)](#)
- [Eloquent Relationships \(Laracasts\)](#)
- [Uso de métodos mágicos en PHP](#)

## Actividades

**Crea una tabla usando migraciones y seeders ([Creando Migraciones en Laravel 5](#)).**  
**Luego crea un modelo para dicha tabla.**

```
php artisan make:model User
```

**Haz un par de consultas utilizando el nuevo modelo de Eloquent y concatenando los métodos de Fluent como vimos en la clase.**

```
$user = User::find(2);
dd($user->toArray());
```

Otra consulta pudiera ser:

```
$users = User::where('type', '=', 'user')
    ->with('profile')
    ->get();
dd($users->toArray());
```

## Preguntas

- ¿Cuál es la principal razón por la cual se usan ORMs?
  - Manejar las peticiones a la base de datos con programación orientada a objetos, ejemplo: filas como objetos, tablas como clases.
- ¿Es verdad que al usar un ORM no necesitaremos nunca usar ni saber SQL?
  - No, en aplicaciones de mediano a gran tamaño, puede existir momentos donde se requiera de una sentencia SQL (por ejemplo un JOIN para una búsqueda más compleja, etc).
- ¿Cuál es la diferencia entre Fluent y Eloquent?
  - La diferencia es que Fluent es un generador de sentencias SQL (o query builder) mientras que Eloquent es un ORM que maneja estas peticiones con programación orientada a objetos, lo cual hace que sea posible el uso de clases y objetos para nuestros registros.



## Chapter 11

# Cambios al motor de plantillas Blade en Laravel 5

Laravel 5 continua usando Blade como motor de plantillas.

Sin embargo hay algunos cambios que hay que tener en cuenta si ya has usado Laravel 4:

Las vistas ahora se encuentran en resources/views Las etiquetas de Blade `{{ $var }}` y `{{{ $var }}}` ahora escapan los datos automáticamente (para protegernos de ataques XSS) Laravel 5 agregó una nueva etiqueta: `{!! $var !!}` que vamos a usarla cuando queremos generar HTML dentro de la etiqueta (el contenido dentro de `{!! !!}` no será escapado por Laravel) Los componentes de Form y Html fueron separados a un paquete aparte (esto lo veremos en la clase siguiente)

Ir a la lección: [Blade el motor de plantillas](#).

## Tutoriales recomendados

- [Tutorial básico de Blade en Laravel 4](#)
- [Blade \(documentación oficial\)](#)
- [Blade 101 \(Laracasts\)](#)

## Actividades

**Crea tus primeras páginas con Blade, pasando datos desde las rutas o los controladores a las vistas.**

Para traer la información de un usuario y enviarla a la vista users/index:

```
$user = User::select('id', 'first_name', 'email', 'type')
    ->first();
```

```
return view('users.index', compact('user'));
```

A continuación el código de la vista que mostrará los datos de los usuarios formateados en una tabla:

```
<html>
<head>
    <link href='//fonts.googleapis.com/css?family=Lato:100' rel='stylesheet'
    type='text/css'>
    <link href="/css/style.css" rel="stylesheet">
    <link href="/css/app.css" rel="stylesheet">
</head>
<body>
<div class="container">
    <div class="content">
        <div class="title">Laravel 5</div>
        <h1>Listado de usuarios</h1>

        <table class="table table-striped">
            <tr>
                <th>id</th>
                <th>Nombre</th>
                <th>Email</th>
                <th>Tipo</th>
            </tr>

            <tr>
                <td>{{ $user->id }}</td>
                <td>{{ $user->first_name }}</td>
                <td>{{ $user->email }}</td>
                <td>{{ $user->type }}</td>
            </tr>

        </table>
    </div>
</div>
</body>
</html>
```

## Intenta utilizar otras estructuras dentro de Blade como @foreach guiándote con la documentación oficial

Para traer la información de todos los usuarios y enviarla a la vista users/index:

```
$users = User::select('id', 'first_name', 'email', 'type')
    ->orderBy('first_name', 'ASC')
    ->get();
return view('users.index', compact('users'));
```

A continuación el código para imprimir los datos de todos los usuarios, formateados en una tabla:

```
<table class="table table-striped">
<tr>
    <th>id</th>
    <th>Nombre</th>
    <th>Email</th>
    <th>Tipo</th>
    <th>Acciones</th>
</tr>

@foreach($users as $user)
    <tr>
        <td>{{ $user->id }}</td>
        <td>{{ $user->first_name }}</td>
        <td>{{ $user->email }}</td>
        <td>{{ $user->type }}</td>
    </tr>
@endforeach
</table>
```

**Agrega una sección adicional para que puedas personalizar parte del footer (o pie de página) del layout en las plantillas que lo extiendan (similar a como hicimos con la sección title)**

Para ello debemos colocar en nuestro layout.blade.php:

```
@yield('footer')
```

La vista que requiera un footer personalizado, pasa la información a nuestro layout de esta manera:

```
@section('footer')
    Aquí el código de nuestro footer personalizado
@endsection
```

```
@stop
```

La otra forma es colocar como segundo parámetro lo que va a tener esta sección por defecto de no se sobre-escriba por cualquier vista:

```
@yield('foot', '
    </body>
</html>'
)
```

Por último la vista que requiera un footer personalizado, pasa la información a nuestro layout de la siguiente forma, recordando que ahora debemos colocar las etiquetas `</body>` y `</html>` porque estamos sobre-escribiendo la información por defecto de esta sección.

```
@section('footer')
    Aquí el código de nuestro footer personalizado
</body>
</html>
@stop
```

## Preguntas

- ¿Es Blade y otros motores de plantillas más lentos que usar PHP directamente?
  - Blade convierte nuestro código en plantilla a “PHP puro y duro” y lo almacena en su “cache” de plantillas, por lo que no requiere reinterpretar el código de Blade cada vez que exista una petición. Dando como resultado respuestas tan rápidas como si escribiéramos las vistas con PHP plano.
- Piensa en uno o más ejemplos de lógica de vista vs lógica de tu aplicación
  - Vista: marcar un botón como seleccionado, mostrar o no un menú, mostrar o no un banner, mostrar errores o no de formulario.
  - Aplicación: Verificar si el usuario está registrado o no, validación de los datos, etc..
- ¿Todos los proyectos en PHP tendrán vistas con Blade?
  - No, puedes escribirlos con PHP plano (no recomendado). Además, hay proyectos donde construyes un API y sólo necesitas retornar datos en formato JSON y es un framework Javascript el que genera las vistas.
- ¿Qué otros motores de plantillas has usado?
  - Smarty
  - Twig





## Chapter 12

# Componentes Html y Form a Laravel

Continuando la explicación sobre Blade, en el siguiente video veremos ejemplos detallados de vistas con HTML plano y la diferencia de utilizar el motor de plantilla Blade, también se mostrará el uso del helper `asset()` y el error más común a la hora de instalar Laravel 5 (Que las vistas se muestren sin estilo).

Por otro lado instalaremos el componente de Laravel Collective para utilizar las etiquetas dinámicas de Form y HTML como por ejemplo: `Html::style()` y `Form::text()`, que fueron eliminados del núcleo del framework (Illuminate) y ahora los mantiene este grupo.

Así mismo vamos a trabajar con el framework Twitter Bootstrap en Laravel y resaltaremos el uso correcto en cuanto al “escape de datos” en Laravel 5 y su diferencia con respecto a las versiones anteriores.

Ir a la lección: [Componentes Html y Form a Laravel](#).

## Tutoriales recomendados

- [Blade \(documentación oficial\)](#)
- [Laravel Collective \(documentación oficial\)](#)
- [Integrar Bootstrap con Blade](#)

## Actividad

**Instalar el componente Collective** Modificar todo el código de la vista “login” que trae por defecto Laravel con los métodos del componente Collective.

Para ello comenzaremos instalando el paquete a través de Composer. Edita el archivo `composer.json` de tu proyecto y agrega la línea de código:

```
"require": { "laravelcollective/html": "~5.0" }
```

Luego debemos actualizar Composer desde el terminal:

Next, update Composer from the Terminal:

```
composer update
```

Luego debemos agregar un nuevo proveedor en nuestro array de proveedores. en config/app.php:

```
'providers' => [ // ... 'Collective\Html\HtmlServiceProvider', // ... ],
```

Por último agregar los alias en config/app.php:

```
'aliases' => [ // ...
'Form' => 'Collective\Html\FormFacade',
'Html' => 'Collective\Html\HtmlFacade',
// ... ],
```

La vista del login modificada con los métodos del componente quedaría:

```
{!! Form::open( [ 'url' => '/auth/login',
'role' => 'form',
'class' => 'form-horizontal' ] )
!!}
<div class="form-group">
    {!! Form::label('email', 'E-Mail Address',
[ 'class' => 'col-md-4 control-label' ] )
    !!}
    <div class="col-md-6">
        {!! Form::email('email',
old('email') ,
[ 'class' => 'form-control' ])
        !!}
    </div>
</div>
<div class="form-group">
    {!! Form::label('password', 'Password',
[ 'class' => 'col-md-4 control-label' ] )
    !!}
    <div class="col-md-6">
        {!! Form::password('password',
[ 'class' => 'form-control' ] )
        !!}
    </div>
</div>
<div class="form-group">
```

```

        <div class="col-md-6 col-md-offset-4">
        {!! Form::checkbox( 'remember', null ) !!} Remember Me
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-6 col-md-offset-4">
        {!! Form::submit( 'Login',
            [ 'class' => 'btn btn-primary',
              'style' => 'margin-right: 15px' ] )
            !!}
        <a href="{{ asset('/password/email') }}">
            Forgot Your Password?
        </a>
        </div>
    </div>
    {!! Form::close() !!}

```

## Preguntas

- ¿Qué beneficios observas en el uso del motor de plantillas Blade?
  - Tener una estructura que permita reutilizar código fácilmente
  - La sintaxis es mucho más agradable que la provista por PHP básico
  - Blade incluye otras características avanzadas como la herencia de plantillas (layouts), uso de View Composers y otros.
- ¿Cuáles son las ventajas de usar Helpers para generar HTML?
  - Si quisieras generar un simple boton (<button>) quizás no veas ninguna ventaja, pero ahora imagina que necesitas generar un SELECT con opciones dinámicas.
- ¿En cuáles escenario es conveniente usar un framework de CSS como Bootstrap y en cuáles no?
  - Cuando no contamos con un diseñador o front-end developer, frameworks como Bootstrap pueden ser de mucha ayuda, para hacer paneles de administración, etc. Para hacer sitios webs con diseño personalizado y óptimos, quizás sea mejor partir desde cero (con ayuda de un front-end developer)



## Chapter 13

# Listado y Paginación

Con los conocimientos básicos de: Vistas, Modelos, Controladores, rutas, Blade y otros que hemos aprendido durante este Curso de Laravel 5, vamos a comenzar a construir nuestro primer módulo. Así que hoy aprenderemos cómo hacer un listado paginado con Laravel 5.

Usuarios			
<a href="#">Nuevo usuario</a>			
Hay 31 usuarios			
#	Nombre	Email	Acciones
38	Duilio Palacios	i@duilio.me	<a href="#">Editar</a> <a href="#">Eliminar</a>
31	Otilia Little	bKoepp@Aufderhar.net	<a href="#">Editar</a> <a href="#">Eliminar</a>
30	Berry Johnson	Rippin.Friedrich@gmail.com	<a href="#">Editar</a> <a href="#">Eliminar</a>

Para ello veremos cómo podemos generar y utilizar un controlador de tipo Resource que usaremos para el resto de nuestro módulo de usuarios con Laravel, así evitaremos las posibles colisiones en las rutas y a su vez comenzaremos a ver las asociaciones entre una acción en un controlador, una URL y el nombre de una ruta (named routes).

Por último, veremos cuan sencillo resulta hacer la paginación de nuestros registros en este nuevo videotutorial de Laravel: [Listado y paginación](#).

## Tutoriales recomendados

- [Aprende a escribir rutas con Laravel](#)
- [Cómo asociar rutas a controladores y acciones con Laravel](#)
- [Paginación de registros con Laravel 4 y Codeception](#)
- [Paginación \(Documentación oficial\)](#)

## Actividades

**Mostrar la columna “type” en el listado de usuarios.**

```
@foreach($users as $user)
  <tr>
    <td>{{ $user->id }}</td>
    <td>{{ $user->full_name }}</td>
    <td>{{ $user->email }}</td>
    <td>{{ $user->type }}</td>
    <td>
      <a href="{{ route('admin.users.show', $user->id) }}" class="btn btn-primary">Ver</a>
      <a href="{{ route('admin.users.edit', $user->id) }}" class="btn btn-success">Editar</a>
    </td>
  </tr>
@endforeach
```

**Colocar arriba de la tabla más información: número de usuarios, número de páginas, etc.**

```
<p> Hay {{ $users->total() }} usuarios </p>
<p>Usted está en la pagina {{ $users->currentPage() }}</p>
```

**Mostrar usuarios con paginación de 20 en 20 ordenados alfabéticamente.**

```
$users = User::select('id', 'first_name', 'email', 'type')
            ->orderBy('first_name', 'ASC')
            ->paginate(20);
```

## Preguntas

- ¿Cuál es la utilidad de usar `Route::group` y `Route::resource`?
  - `Route::group` nos sirve para asignar características similares a un conjunto de rutas, como pueden ser: prefijo, namespace, middleware, etc.

- Route:resource nos va a permitir generar mediante artisan un controlador con las funciones básicas necesarias para hacer un módulo CRUD RESTful.
- ¿Qué beneficio nos brinda crear sub-carpetas dentro de Controllers/ y de views/?
- Organizar mejor nuestros controladores y nuestras vistas, por ejemplo si vas a crear un panel de administrador o un servicio RSS sería conveniente crear los controladores en las subcarpetas respectivas: Admin/ y Rss/





## Chapter 14

# Creación de usuarios con Laravel y Eloquent



Ya hemos creado nuestro listado y paginación de usuarios, ahora vamos a crear nuevos usuarios. En el siguiente video, aprenderemos como crear formularios con Laravel y a usar las rutas correctas para cada una de nuestras funciones, para ello nos apoyaremos con el comando: `artisan route:list`. Por supuesto, también utilizaremos el framework de CSS Twitter Bootstrap para ir aplicando estilos a nuestra aplicación.

Por último, aprenderemos a encriptar contraseñas y a redireccionar a un usuario a una ruta específica.

Ir a lección: [Creación de usuarios](#).

## Tutoriales recomendados

- [Aprende a escribir rutas con Laravel](#)
- [Cómo asociar rutas a controladores y acciones con Laravel](#)
- [Documentación Bootstrap – CSS \(Documentación oficial\)](#)
- [Http Controllers \(Documentación oficial\)](#)
- [Forms \(Laracasts\)](#)

## Actividades

### Agregar nuevos campos al módulo de usuarios

Podemos realizarlo con una simple modificación de nuestra migración. **7**

### Hacer el formulario para agregar el profile a un usuario

```
<div class="form-group">
    {!! Form::label('bio', 'Biografía') !!}
    {!! Form::text('bio', null, ['class' => 'form-control', 'placeholder' => '']) !!}
</div>
<div class="form-group">
    {!! Form::label('twitter', 'Twitter') !!}
    {!! Form::text('twitter', null, ['class' => 'form-control', 'placeholder' => 'Twitter']) !!}
</div>
<div class="form-group">
    {!! Form::label('website', 'Website') !!}
    {!! Form::text('website', null, ['class' => 'form-control', 'placeholder' => 'Website']) !!}
</div>
<div class="form-group">
    {!! Form::label('birthdate', 'Fecha de nacimiento') !!}      {!! Form::date(
    'birthdate', \Carbon\Carbon::now(), ['class' => 'form-control datepicker', '
    placeholder' => '']) !!} </div>
<div class="form-group">
    {!! Form::label('gender', 'Género') !!}
    {!! Form::select('gender', ['' => 'Eligir género', 'female' => 'Femenino',
    'male' => 'Masculino'], ['class' => 'form-control']) !!}
</div>
```

## Preguntas


- ¿Cuál es la utilidad del comando `php artisan route::list`?
  - Esta sentencia nos permite visualizar la lista completa de las rutas declaradas en nuestra aplicación. Muy útil para saber de qué rutas disponemos, cuál es su nombre (named routes), qué tipo de método usan (GET, POST, etc.)
- ¿Para qué sirve la variable `fillable`?

- La variable `fillable` de nuestro modelo nos permite indicar cuales son las columnas que pueden ser agregadas al modelo mediante "mass assignement" es decir, cuando pasamos un array de datos al modelo con el uso del método `fillable`, o con métodos como `create()`, esta variable nos permite ser explícitos en qué columnas queremos que estén disponibles para ser pasadas a través de dichos métodos, lo contrario a `fillable` es la propiedad `guarded`



## Chapter 15

# Editar Registros con Laravel



Editar usuario: Duilio

Correo electrónico

i@duilio.me

Contraseña

Continuando con nuestro módulo de usuarios tipo CRUD, en esta lección aprenderemos a editar registros, la forma en que debemos crear nuestro formulario y las rutas correspondientes a cada método. Además utilizaremos sub-secciones o “partials” en las vistas, lo que nos va a permitir reutilizar código y simplificar las plantillas. A su vez, se mostrará la forma correcta para evitar que se modifique el atributo password si el usuario decide no cambiarlo.

Ir a lección: [Edición de usuarios](#).

## Tutoriales recomendados

- [Cómo asociar rutas a controladores y acciones con Laravel](#)
- [Edición de registros con Laravel 4 y Codeception](#)
- [Http Controllers \(Documentación oficial\)](#)
- [Forms \(Laracasts\)](#)

## Actividades

- Comienza a crear tu propio módulo usando lo aprendido en clase, no olvides compartir el código.
- Hacer el formulario para editar el profile de un usuario:

```

<div class="form-group">
    {!! Form::label('bio', 'Biografía') !!}
    {!! Form::text('bio', null, ['class' => 'form-control', 'placeholder' => '
    ']) !!}
</div>
<div class="form-group">
    {!! Form::label('twitter', 'Twitter') !!}
    {!! Form::text('twitter', null, ['class' => 'form-control', 'placeholder'
=> 'Twitter']) !!}
</div>
<div class="form-group">
    {!! Form::label('website', 'Website') !!}
    {!! Form::text('website', null, ['class' => 'form-control', 'placeholder'
=> 'Website']) !!}
</div>
<div class="form-group">
    {!! Form::label('birthdate', 'Fecha de nacimiento') !!}
    {!! Form::date('birthdate', \Carbon\Carbon::now(), ['class' => 'form-
control datepicker', 'placeholder' => '']) !!}
</div>
<div class="form-group">
    {!! Form::label('gender', 'Género') !!}
    {!! Form::select('gender', ['' => 'Eligir género', 'female' => 'Femenino'
, 'male' => 'Masculino'], ['class' => 'form-control']) !!}
</div>

```

## Preguntas

- ¿Qué hace el método `user::findOrFail()` si no encuentra el registro?
  - Además de cumplir con la función `find()`, si la búsqueda no es exitosa, Laravel lanzará un error 404.
- ¿Cuál es la diferencia entre el método POST y el métodos PUT?
  - El navegador soporta dos metodos GET y POST. Frameworks como Symfony, Laravel y Ruby emulan métodos como PUT y DELETE para crear requests mas específicos. POST se usa para publicar o crear registros mientras que PUT indica que el registro se va a editar o actualizar
- ¿Que diferencia tiene el método `form::model()` con respecto a `form::open()`?

- `form::open()` simplemente crea una nueva etiqueta de formulario pero sin datos precargados, en cambio con `form::model()` se puede atar un modelo a nuestro formulario, de tal forma que se mostrarán o seleccionarán los datos existentes en los campos (input, select) automáticamente
- ¿Qué debemos retornar al final de un método como `store()` o `update()`?
- Debemos retornar una función `redirect()`, para direccionar al usuario a una vista deseada, o por ejemplo una respuesta en formato JSON si estamos trabajando con AJAX.
  - También es conveniente retornar un mensaje de éxito si se cumplió la función correctamente. (Feedback al usuario).





## Chapter 16

# Validación - Parte 1

Laravel maneja la validación de manera muy sencilla, siguiendo un patrón para definir las reglas de validación con cadenas en un array asociativo, similar a como lo hacía Codeigniter:

```
$rules = [  
    'name' => 'required',  
    'password' => 'required|min:8',  
    'email' => 'required|email|unique:users'  
];
```

Además, si has trabajado con Laravel 3 o 4 seguramente este código te resultará familiar:

```
$validator = Validator::make($data, $rules);  
if($validator->fails()){  
    // It failed  
    Redirect::back()->withErrors($validator->messages())->withInput();  
}  
// Everything good here:  
User::create($data);  
return Redirect::route('users.index');
```

Este es típicamente el código que hace falta para validar un array de datos (que típicamente viene de un Request POST o PUT). Básicamente usamos el patrón “Factory” para crear un nuevo validador, esto es lo que hace `Validator::make` y luego tenemos algunos métodos muy fáciles de usar como “`$validator->fails`” que comprueba si la validación falló y “`$validator->messages()`” que nos devuelve los errores de validación.

Sin embargo si ya has trabajado con Laravel sabrás que este código se termina repitiendo una y otra vez dentro de tu aplicación.

Por ello en Laravel 5, Taylor Otwell incorporó 2 métodos para hacer la validación aún más fácil, que en las versiones anteriores. Estos son: el método “`validate`” dentro del controlador con el trait `Illuminate\Foundation\Validation\ValidatesRequests` y los Form Requests.

Veremos los 3 métodos: desde `Validator::make` hasta los Form Requests, así que si eres nuevo o ya tienes experiencia con Laravel este nuevo videotutorial de Laravel te será muy útil.

Ir a lección: [Validación - Parte 1](#)

## Material adicional

- [Validación con Laravel 4 y Codeception](#)
- [Documentación oficial sobre validación](#)
- [Form Requests and controller validation \(Laracasts\)](#)

## Actividades

### Impide que el usuario agregue contraseñas menores a 6 caracteres

```
'password' => 'required|min:6',
```

### Solicita que el usuario confirme su contraseña

En el formulario debemos de agregar el otro campo:

```
<div class="form-group">
    {!! Form::label('password_confirmation', 'Repita contraseña') !!}
    {!! Form::password('password_confirmation', ['class' => 'form-control']) !!}
!!}
</div>
```

y la validación del password para confirmarlo quedaría así:

```
'password' => 'required|confirmed|min:6',
```

### Crea la validación para el formulario de user profile

```
class createUserProfileRequest extends Request {
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
}
```

```

*/
public function authorize(){
    return true;
}
/**
 * Get the validation rules that apply to the request.
 *
 * @return array
 */
public function rules(){
    return [ 'bio' => 'required',
            'twitter' => 'required',
            'website' => 'required|active_url',
            'birthdate' => 'required|date',
            'gender' => 'required|in:male,female'
    ];
}
}

```

## Preguntas

- ¿Con qué criterios escogerías cuando usar validate y cuando FormRequest?
  - El método validate es conveniente si queremos validar una, dos o hasta 3 columnas. Mientras que FormRequest es conveniente para validar formularios más extensos, ya que al extraer la lógica a una clase aparte, nuestro controlador queda más limpio.
- ¿Qué sucede si validate o FormRequest fallan?
  - Laravel lanza una excepción que luego es capturada y finalmente se retorna una redirección hacia la página anterior con los errores, o se envían los errores en formato JSON (si la petición original fue AJAX)
- ¿Por qué el código siguiente no se ejecuta a pesar de que no hay condicionales (if)?
  - Porque la excepción que lanza Laravel interrumpe el flujo normal del código, entonces si no se cumplen las reglas en validate o FormRequest, somos redirigidos a la URL anterior con los respectivos errores.
- ¿Qué otras reglas agregarías antes de crear el usuario?
  - Que el email no esté repetido
  - Que la contraseña tenga un nivel de seguridad adecuada
  - El nombre y apellido sean de carácter alfabéticos



## Chapter 17

# Validación – Parte 2

En este capítulo seguimos aprendiendo cómo validar los datos que envían los usuarios, esta vez para el formulario que nos permite editar y actualizar registros.

Aprenderemos más sobre la regla “unique” de Laravel, además veremos la regla “in” y repasaremos los conocimientos aprendidos en la lección anterior sobre Form Requests, la nueva forma de validar que trae Laravel 5 y nos permite separar esta responsabilidad del controlador.

Ir a lección: [Validación - Parte 2](#)



## Chapter 18

# Crear una página web o aplicación en español con Laravel (i18n)

En este capítulo aprenderemos como traducir los mensajes de error y notificaciones de nuestra aplicación al español o cualquier idioma.

Es muy común que nuestra aplicación deba soportar varios idiomas según las necesidades de nuestros usuarios. Laravel, facilita increíblemente la internacionalización (i18n) de los mensajes de nuestro sitio o aplicación web; ya sea a través de archivos de idioma o haciendo uso de la configuración de los servicios como Autenticación, utilizando los métodos disponibles como `Lang::get` o la función `trans()`, así como traducciones según parámetros y uso de singulares o plurales.

Hoy veremos por primera vez estas características, pasaremos nuestro sitio al español con la ayuda de un paquete y también les enseñaré cómo traducir los atributos y labels de nuestros formularios (que típicamente los escribimos en inglés).

Ir a la lección: [Internacionalización](#)

## Material adicional

- [Localization \(Documentación oficial\)](#)
- [Traducciones para Laravel](#)

## Actividades

- Instala el paquete de traducciones dentro de tu proyecto Traduce el formulario de registro que viene con Laravel

## Preguntas

- ¿Es conveniente colocar cadenas de texto en nuestras vistas y controladores o debemos usar siempre `trans()`?
  - Si solo va a utilizar un solo idioma, se puede utilizar texto sin ningún problema. Pero si se necesita o existe la posibilidad de que un futuro nuestra aplicación sea multi-idioma es preferible utilizar desde el inicio `{{ trans() }}` o `@lang()`.
- ¿Cómo implementarías un sitio multi-idioma con Laravel?





## Chapter 19

# Eliminar Registros

Continuando con la creación de un módulo en nuestro curso de Laravel 5. Para ello haremos la acción restante: eliminar registros.

Con Laravel podemos eliminar registros de dos formas utilizando el ORM de Laravel, Eloquent dentro de nuestro resource controller.

Una de ellas es usando el método destroy, que acepta como parámetro una ID.

```
User::destroy($id);
```

Otra es usando el método delete (para ello necesitamos cargar antes el objeto User)

```
$user = User::find($id); $user->delete();
```

También en esta lección aprenderemos a mostrar alerts o mensajes temporales con el método Session::flash.

Ir a lección: [Eliminar Registros](#)

## Material adicional

- [Eliminar registros con Laravel 4 y Codeception](#)
- [Usar Eloquent en tu proyecto sin Laravel](#)

## Actividad

- Elimina varios registros.

## Preguntas

- ¿Cuál es la diferencia entre destroy y delete?
  - El método destroy elimina el objeto directamente a través de una sentencia SQL DELETE por lo cual es más rápido que el método delete() pero no obtendremos el objeto ni sus atributos para usarlos en nuestra acción.
  - En cambio el método delete() aunque elimina el registro de la base de datos, aún tendremos disponible el objeto y sus atributos o métodos mientras se ejecute la acción (ideal para enviar mensajes “alert” o si necesitamos hacer operaciones adicionales con dicho objeto).
- ¿Cuándo es conveniente usar un método u otro?
  - El delete() es conveniente en casos donde queremos mostrar información en la vista de lo que se ha borrado u hacer operaciones adicionales con el objeto, destroy() es conveniente si la velocidad de la operación es crucial y no necesitamos obtener los datos del registro, sólo ejecutar la consulta DELETE.



## Chapter 20

# Aplicando el patrón de diseño de software DRY en Laravel 5

Tan importante como elegir un lenguaje o un framework para crear nuestra aplicación, es aplicar patrones de diseño de software.

Dichos patrones son maneras reusables de solucionar problemas comunes, y muchos de ellos son aplicables a diferentes lenguajes y frameworks de desarrollo.

Hoy les quiero hablar del patrón DRY: no te repitas o “don’t repeat yourself” que es muy conocido y aplicado en la comunidad de Ruby on Rails pero también puede ser aplicado a Laravel y otros frameworks.

Este patrón consiste en la idea de evitar la duplicación de código tanto como sea posible, duplicar código hace a nuestra aplicación más difícil de mantener, brinda la posibilidad de crear muchas inconsistencias y conlleva a la larga a un mayor esfuerzo y tiempo de desarrollo.

“Copiar y pegar” es la opción más rápida cuando se nos presenta un problema, pero no suele ser la mejor opción.

Hoy veremos una pequeña comparación entre un controlador de Ruby on Rails y otro de Laravel 5 y cómo podemos aplicar el método DRY para nuestro UsersController.

Ir a lección: [Patrón de diseño de software DRY](#)

## Material Recomendado

- [Filtrar registros con Laravel 4](#)

## Actividad

- Si estás trabajando en un proyecto en Laravel actualmente, aplica el patrón DRY donde sea posible.

## Preguntas

- ¿Qué otras formas se te ocurren o haz hecho para aplicar el patrón DRY a tus proyectos?
  - Crear funciones reutilizables
  - Utilizar un sistema de plantillas
  - Normalización de bases de datos



## Chapter 21

# AJAX y Laravel usando jQuery



Tal como lo indica el título, hoy aprenderemos cómo interactuar desde Javascript con un backend en Laravel, tal como lo haríamos con un API; pero en este caso haremos un ejemplo sencillo dentro de nuestra aplicación, para eliminar filas utilizando jQuery y una petición POST con AJAX.

A pesar de que hoy en día existen frameworks de Javascript más potentes como AngularJS, jQuery aún es útil si sólo necesitamos agregar contenido dinámico para algún widget dentro de nuestra aplicación o sitio web, por ejemplo un campo de auto-completado o combos dependientes.

Hoy veremos por primera vez cómo podemos combinar un framework de Javascript con una aplicación de Laravel, también aprenderemos sobre el formato JSON, todo esto le va a permitir a nuestros usuarios eliminar registros sin necesidad de que la página recargue dentro del navegador.

Por último responderemos a la pregunta: ¿Es posible utilizar el framework de Javascript \_\_\_\_ con Laravel?

Ir a la lección: [AJAX y Laravel usando jQuery](#)

## Material adicional

- [Eliminar un registro con Laravel 4 y Codeception](#)



## Actividades

**Mueve el código Javascript que creamos en clase a un archivo .js y llámalo desde el layout app.blade.php**

En la carpeta resource/js crear archivo ajax-delete.js y colocar el código:

```
$(document).ready(function() {

    $('.btn-delete').click(function(e){
        e.preventDefault();

        var row = $(this).parents('tr');
        var id = row.data('id');
        var form = $('#form-delete');
        var url = form.attr('action').replace(':USER_ID', id);
        var data = form.serialize();
        $.post(url, data, function(result){
            row.fadeOut();
        }).fail(function(){
            row.fadeIn();
        });
    });
});
```

**Intenta hacer el código de Javascript genérico para que funcione con otros módulos similares**

**Busca “bootstrap notifications” en Google e intenta integrar un componente a tu módulo de manera de imprimir mensajes más elegantes que no bloqueen al usuario (como lo hace la función alert). Por ejemplo, podrías usar este [plugin](#).**

Primero [descargar](#), colocar el bootstrap-notify.css en la carpeta resources/css y colocar el archivo bootstrap-notify.js en la carpeta resources/js.

Luego puedes colocar el llamado a los archivos en tu layout app.blade.php:

```
<script src="{{ URL::asset('/js/ajax-delete.js') }}"></script>
<script src="{{ URL::asset('/js/bootstrap-notify.js') }}"></script>
```

Por último tu script ajax-delete quedaría así:

```
$(document).ready(function() {
```

```

$( '.btn-delete' ).click(function(e){
    e.preventDefault();

    var row = $(this).parents('tr');
    var id = row.data('id');
    var form = $('#form-delete');
    var url = form.attr('action').replace(':USER_ID', id);
    var data = form.serialize();
    $.post(url, data, function(result){
        row.fadeOut();

        $( '.top-left' ).notify({
            message: {
                text: result.message
            }
        }).show();

    }).fail(function(){
        row.fadeIn();

        $( '.top-left' ).notify({
            message: { text: 'El usuario no fue eliminado' },
            type: 'danger'
        }).show();
    });
});
});
});

```

## Preguntas

- ¿Cuándo es conveniente que recargue el navegador y cuándo que nuestra app tenga interacción 100% con Javascript?
  - Recargar el navegador completo, es necesario cuando la información es más unilateral, la cual esperamos una nueva vista de la aplicación, o nos dirigimos a otra sección con la acción anterior.
  - Con javascript, cuando la información es bilateral y dinámica. Como usa facebook o Twitter en el home del sitio, que la información es necesaria que se refresque sola o con un pequeño comando (click)



## Chapter 22

# Búsquedas y filtros con Laravel y Eloquent (Query scopes) - Parte 1

Usuarios			
<a href="#">Nuevo usuario</a>		Nombre de usuario	Buscar
Hay 501 usuarios			
#	Nombre	Email	Acciones
501	Martine Harber	Botsford.Ernie@gmail.com	<a href="#">Editar</a> <a href="#">Eliminar</a>

En los capítulos anteriores aprendimos cómo crear, editar, eliminar, listar y hasta paginar registros. Con esto casi terminamos el módulo de usuarios.

Digo “casi” porque imaginen que de pronto tenemos 500 usuarios o más en nuestra base de datos y queremos saber si Taylor Otwell se registró en nuestra aplicación. Tal como está el módulo ahora tendríamos que buscar registro por registro lo cual no es nada práctico.

Por ello hoy les enseñaré cómo buscar o filtrar registros en la base de datos usando el ORM Eloquent y una característica incluida en dicho ORM llamada “query scopes”. Ir a lección: [Búsquedas y filtros con Laravel](#)

## Material relacionado

- [Filtrar registros con Laravel y Codeception](#)
- [Query scopes \(documentación oficial\)](#)

## Actividades

### Agrega un segundo campo (de tipo select) para filtrar usuarios según su tipo (agrega también un nuevo query scope)

El código en la vista para el select sería:

```
{!!Form::select('type',[ '' => 'Seleccione un tipo ',
    'user' => 'user',
    'visitor' => 'visitor',
    'editor' => 'editor'],
    null,
    ['class' => 'form-control'])
!!}
```

Luego creamos el scopeType en nuestro modelo:

```
public function scopeType($query, $type){
    if(trim($type) != ''){
        $query->where('type', $type);
    }
}
```

Ya creado, lo podemos utilizar en nuestro controlador de esta manera:

```
$users = User::name($request->get('name'))
    ->type($request->get('type'))
    ->orderBy('first_name', 'ASC')
    ->paginate();
```

### Extrae todo el código para filtrar y paginar usuarios a un método “static” dentro del modelo User o a un repositorio

En nuestro modelo colocamos nuestra función que realiza el filtrado y paginación. La llamaremos filterAndPaginate(). Recibe el request y retornamos los usuarios.

```
static function filterAndPaginate($request){
    $users = User::name($request->get('name'))
        ->type($request->get('type'))
        ->orderBy('first_name', 'ASC')
        ->paginate();
    return $users;
}
```

```
}
```

Nuestro controlador solo quedaría llamando a nuestra función y pasándole como parámetro el request:

```
public function index(Request $request) {  
    $users = User::filterAndPaginate($request);  
    return view('admin.users.index', compact('users'));  
}
```

## Preguntas

- ¿En qué situaciones es necesario agregar filtros o buscadores?
  - Donde la cantidad de registros va a ir creciendo con el tiempo
  - Donde los registros de tu aplicación estén organizados por alguna categoría o tipo
- ¿Qué tipos de filtros o buscadores encuentras más útiles en un sistema?
  - Búsqueda por texto
  - Filtros por categorías



## Chapter 23

# Filtros de búsqueda y paginación con Laravel 5 - Parte 2

Este capítulo es la continuación de la lección anterior donde aprendimos a buscar registros con Laravel, y por supuesto es parte del curso de Laravel 5.

Usuarios

Nuevo usuario

Hay 501 usuarios

#	Nombre	Email	Tipo	Acciones
501	Marcelo Pacocha	Anthony68@Lubowitz.com	editor	<a href="#">Editar</a> <a href="#">Eliminar</a>

Hoy aprenderemos cómo filtrar usuarios con 2 o más campos de búsqueda, además veremos cómo combinar la paginación de Laravel con los filtros que agregamos al módulo.

También aprenderemos a mantener los valores de búsqueda dentro del formulario, y a mover parte de la lógica al modelo para simplificar nuestro controlador y aplicar nuevamente el principio DRY. Ir a lección: [Filtros de búsqueda y paginación](#)

## Material relacionado

- [Filtrar un listado de registros con Laravel 4 y Codeception](#)
- [Filtrar registros con Laravel: Refactorización \(DRY\)](#)
- [Paginación de registros con Laravel 4 y Codeception](#)
- [Implementación del patrón repositorio \(Repository Pattern\) en Laravel](#)



## Actividades

### Mueve el formulario de filtros a un partial

Creamos el archivo `views/admin/users/partials/filters.blade.php` y colocamos el código de nuestros filtros:

```
{!!Form::open(['route' => 'admin.users.index', 'method' => 'GET', 'class' =>
'navbar-form navbar-left pull-right', 'role' => 'search']) !!}
<div class="form-group">
    {!!Form::text('name', null, ['class' => 'form-control', 'placeholder' =>
'Nombre de usuario']) !!}
    {!!Form::select('type', ['' => 'Seleccione un tipo', 'user' => 'user', '
visitor' => 'visitor', 'editor' => 'editor'], null, ['class' => 'form-control
']) !!}
</div>
{!!Form::close() !!}
```

### Crea un botón para limpiar filtros o “ver todos los resultados”

```
<a href="{{ route('admin.users.index') }}" class="btn btn-succes">Mostrar
todos </a>
```

### Agrega un filtro adicional al módulo

¿Qué les parece si agregamos un nuevo select a nuestro formulario para elegir el tipo de estatus?, así quedaría:

```
{!!Form::open(['route' => 'admin.users.index', 'method' => 'GET', 'class' =>
'navbar-form navbar-left pull-right', 'role' => 'search']) !!}
<div class="form-group">
    {!!Form::text('name', null, ['class' => 'form-control', 'placeholder' =>
'Nombre de usuario']) !!}
    {!!Form::select('type', ['' => 'Seleccione un tipo', 'user' => 'user', '
visitor' => 'visitor', 'editor' => 'editor'], null, ['class' => 'form-control
']) !!}
    {!!Form::select('active', ['' => 'Estatus', 'true' => 'Activo', 'false' =>
'Inactivo'], 'true', ['class' => 'form-control']) !!}
</div>
{!!Form::close() !!}
```

Agregamos un nuevo Scope en nuestro Model Users:

```
public function scopeActive($query, $active){
    if(trim($active) != ''){
        $query->where('active', $active);
    }
}
```

Ya que habíamos creado en los capítulos anteriores un método estático que realizara la búsqueda de nuestros usuarios con los filtros, simplemente agregamos la línea para hacer el llamado a la nueva función Scope:

```
static function filterAndPaginate($request){
    $users = User::name($request->get('name'))
        ->type($request->get('type'))
        ->active($request->get('active'))
        ->orderBy('first_name', 'ASC')
        ->paginate();
    return $users;
}
```

Recordemos que estamos utilizando el siguiente llamado en nuestro controlador.

```
$users = User::filterAndPaginate($request);
```



## Chapter 24

# Crea tus propios Middleware con Laravel 5

En el capítulo anterior aprendimos qué son los **Middleware de Laravel 5** y cómo usarlos. Pero la verdadera utilidad de los Middleware llega cuando podemos generar nuevos que nos ayuden a proteger nuestra aplicación de acuerdo a la lógica que queramos implementar.

Cada aplicación es diferente, en todas la seguridad es un tema crucial, sin embargo hay aplicaciones que sólo tienen un tipo de usuario, otras que sólo tienen un par de tipos y otras que requieren sistemas de permisos complejos como **listas de control de acceso o ACL**.

Yo he encontrado que para muchos sistemas tener un par de tipos de usuarios con permisos bien definidos es suficiente, y es lo que aprenderemos en esta lección: **Crea tus propios Middleware con Laravel 5**

## Chapter 25

# Protege el acceso a tu aplicación con los Middleware de Laravel 5

Ya terminamos el módulo de usuarios, y quizás Uds. ya han hecho otros módulos también; pero no podríamos subirlos a una aplicación real si no encontramos la manera de **restringir el acceso** a ellos. Esto es un problema común y nuevamente Laravel viene al rescate.

Laravel 5 implementa **Middleware**, que tienen una utilidad similar a los filtros (filters) de Laravel 4. En el sentido que nos permiten proteger rutas y acciones de acceso no autorizado.

El Middleware, como el nombre lo indica, se sitúa en el **medio** entre la petición del usuario (Request) y las acciones del controlador que arman y envían la respuesta (Response).

Entonces si quieres crear un nuevo usuario, envías una petición, pero antes de que tu petición sea procesada, el middleware puede revisar si realmente tienes permiso para ejecutar dicha acción, y en caso de que no, mandarte a la página de login o enviar un error 401 (dependiendo del Middleware que uses o defines).

En el siguiente video veremos cómo restringir el acceso al admin solamente a usuarios que estén autenticados:

**Protege el acceso a tu aplicación con los Middleware de Laravel 5**

Por favor comparte el curso de Laravel 5, y ayúdame a erradicar mysql\_query, \$\_GET y HTML concatenado con PHP y "echo" de la faz de la tierra.

## Material adicional (en inglés)

- [Ogres are like Middleware \(Laracasts Fundamentals\)](#)
- [Middleware \(official documentation\)](#)
- [Middleware filter-style](#)

## Actividades

**Utiliza `Session::flash` para enviar un mensaje tipo: “necesitas hacer login para entrar al panel” desde el Middleware y luego imprímelo en la vista `auth/login`.**

A la función que nos redirecciona en el middleware `Http/Middleware/Authenticate.php` podemos concatenarle una variable de sesión, quedaría así:

```
return redirect()->guest('auth/login')->with('message', 'Necesitas hacer login para entrar al panel');
```

En nuestra vista `resources/views/auth/login.blade.php` llamamos a nuestra variable de sesión así:

```
@if (Session::get('message') != null)
    <div class="alert alert-danger">
        {{ Session::get('message') }}
    </div>
@endif
```

## **Parte III**

# **Tips**





## Chapter 26

# Cambiar contraseña con Laravel (validando clave actual)

Laravel 5 es genial y por eso incluye “out of the box” el módulo para recuperar contraseñas, el cual es imprescindible en la mayoría de los sistemas, pero ¿Qué pasa si el usuario no ha olvidado su contraseña, sino que simplemente quiere cambiarla?

La mayoría de los sistemas web incluyen la posibilidad de cambiar tu clave una vez que te has conectado a dicho sistema, pero por seguridad te pedirán que confirmes tu contraseña actual.

En este tutorial aprenderás cómo crear reglas personalizadas de validación con Laravel para crear la regla que te permitirá confirmar la clave actual del usuario.

Si ya has trabajado con Laravel 4, quizás estás acostumbrado a crear reglas de validación nuevas usando el método `Validator::extend` de esta manera:

```
Validator::extend('foo', function($attribute, $value, $parameters)
{
    return $value == 'foo';
});
```

En Laravel 5 esto sigue funcionando, y puedes colocarlo dentro del método “boot” de cualquier Service Provider. Pero la manera que me parece más correcta de agregar la regla extra, es extender la clase de Validación por completo.

Para ello crea esta clase dentro de tu carpeta `app/`. Puedes dejarla justo en `app/` o crear un subdirectorio llamado “Core”, “Support”, “Validation” o lo que quieras, sólo recuerda que el nombre del directorio y del namespace deben coincidir dado que Laravel 5 usa PSR-4 para autocargar las clases:

```
<?php namespace NombreDeTuApp\Core;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Hash;
use Illuminate\Validation\Validator as LaravelValidator;
```

```
class Validator extends LaravelValidator {

    public function validateCurrentPassword($attribute, $value, $parameters)
    {
        return Hash::check($value, Auth::user()->clave);
    }
}
```

Nota: por supuesto no olvides cambiar el namespace.

Con esto ya tienes tu clase de validación personalizada, fíjate que incluí una nueva regla de validación para comprobar que la clave actual del usuario es correcta, esto, como ya comenté, es una medida común de seguridad, más que nada para prevenir que si el usuario dejó su cuenta abierta una tercera persona maliciosa no pueda cambiar la contraseña.

Nota también que Laravel nuevamente hace uso de convenciones (cumpliendo con el principio DRY): si quieres definir una nueva regla de validación esta llevará el prefijo “validate” y luego el nombre de la regla en formato CamelCase.

Lo segundo que necesitamos es registrar nuestra clase de validación y esto lo hacemos con un Service Provider, crea esta clase dentro de app/Providers/ValidatorServiceProvider.php:

```
<?php namespace NombreDeTuApp\Providers;
use Illuminate\Support\Facades\Validator;
use Illuminate\Support\ServiceProvider;
class ValidatorServiceProvider extends ServiceProvider {

    public function boot() {

        Validator::resolver(function($translator, $data, $rules, $messages) {
            return new \NombreDeTuApp\Core\Validator($translator, $data, $rules,
$messages);
        });
    }

    public function register(){ }

}
```

Una vez hecho esto tienes que registrar el Service Provider, esto se hace en el archivo config/app.php, busca el array llamado \$providers y coloca al final el nombre completo de la clase (incluyendo el namespace):

```
'providers' => [
    // OTROS PROVIDERS VAN AQUI. NO LOS BORRES O ALGO MALO VA A PASAR
    // COLOCA TU PROVIDER AL FINAL:
```

```
// PUEDES COLOCARLE UN COMENTARIO BONITO, ALGO COMO:
/*****MI APP PROVIDERS *****/
'NombreDeTuApp\Providers\ValidatorServiceProvider'
];
```

¡Listo! Ya la nueva regla de validación está disponible. Ahora puedes definir un nuevo Form Request o validar con el método `$this->validate` dentro del controlador, como hemos visto ya en el curso de Laravel 5.

A mí me gustan los Form Requests así que yo creé el mío con lo siguiente:

```
<?php namespace NombreDeTuApp\Http\Requests;
use Illuminate\Foundation\Http\FormRequest;

class ChangePasswordRequest extends FormRequest {

    /** * Determine if the user is authorized to make this request.
    *
    * @return bool
    */

    public function authorize() {
        return true;
    }

    /** * Get the validation rules that apply to the request.
    *
    * @return array
    */

    public function rules() {
        return [
            'clave_actual' => 'required|min:6|current_password',
            'clave' => 'confirmed|min:6' ];
        }

    public function messages() {
        return [];
    }

    /** * Get the sanitized input for the request.
    *
    * @return array
    */
```

```
public function sanitize() {  
    return $this->only( 'clave' );  
}  
}
```

Acá la novedad es que tenemos una nueva regla llamada “current\_password” (nota que es lo mismo que definimos en nuestro Validador pero sin el prefijo validate y el formato es con underscore en vez de camel case).

Si has seguido el curso de Laravel 5 seguramente ya sabrás como hacer uso de los Form Requests y sabrás cómo crear formularios en tus vistas así que dejo el resto del código en tus manos.

## Chapter 27

# Agregar rutas adicionales a un controlador de tipo resource en Laravel

En el módulo de usuarios que creamos para el curso básico de Laravel 5, creamos un controlador de tipo resource, el cual resulta muy práctico si necesitamos las acciones básicas de cualquier módulo: crear, editar, listar, eliminar y ver, pero ¿Qué sucede si necesitamos una acción extra? Por ejemplo, si quisiéramos una nueva acción para crear un reporte de usuarios ¿Cómo incluimos la nueva ruta al resource controller?

Quizás lo primero que se te ocurra es hacer esto:

```
Route::resource('users', 'UserController');  
Route::get('users/report', 'UserController@report');
```

Pero si haces eso, obtendrás un error como el siguiente:

Whoops, looks like something went wrong.

1/1 ModelNotFoundException in Builder.php line 125:  
No query results for model [Course\User].

No query results for model [Course\User]. Esto sucede porque la ruta asignada para el método `UserController@show` tiene el siguiente formato: `admin/users/{users}` y el parámetro `users` por defecto no tiene restricciones, es decir, puede ser un número o una cadena de texto...

¿Entonces cuando tipeo `users/report` en el navegador se está llamando al método `show` con el parámetro "report" en vez del método `report`?

¡Exacto! Puedes solucionar esto de dos formas:

La primera y más fácil es invertir el orden de las rutas:

```
Route::get('users/report', 'UserController@report');  
Route::resource('users', 'UserController');
```

Esto soluciona el problema porque **el orden de las rutas es importante para Laravel**, entonces ahora la ruta `users/report` tendrá precedencia sobre la ruta `show` y las demás rutas del resource controller. La segunda forma, sería definir un “patrón” para el parámetro “users”, y esto se hace con esta función:

```
Route::pattern('users', '[0-9]+');
```

Por supuesto, tienes que colocarla antes de definir tu resource controller, pero ahora no importa si colocas el resource controller antes de la ruta para `users/report`

Esta función limita al parámetro `users` a sólo números, por lo tanto “report” ya no será un parámetro `users` válido y `users/report` no coincidirá con la ruta “`users.show`”

Este tutorial, aunque básico, te mostró un par de tips interesantes para escribir **rutas en Laravel**, no olvides revisar los otros tutoriales de **Laravel 5** y **Laravel 4** para aprender más sobre rutas y otros temas de este maravilloso framework.

**¿Te sabes otros tips de Laravel?** Para compartirlos escríbeme a [i@duilio.me](mailto:i@duilio.me) y los publicaremos en el sitio.

Ir al post: [Agregar rutas adicionales a un controlador](#)

## **Parte IV**

# **Agradecimientos**

# ¡Gracias por aprender con nosotros!

Muchas gracias a todos por seguir este curso básico, sobretodo a quienes se han tomado unos momentos para comentar y compartir el contenido, y a quienes han hecho aportes al sitio, gracias por la confianza y por permitirme dedicar varias semanas a enseñarles Laravel y también a aprender de Ustedes.

Gracias a: [Juan Gonzalez @juanDJGL](#) por mejorar el audio del [curso básico de Laravel 5](#).

Gracias a: [Ronald Coello @roncreative](#), [Manuel Gonzalez @manuelitox](#) y [Jessica Aristimuño @Ni-nAiskel](#) por el apoyo con el diseño de contenido y del sitio.

Gracias a: [Rafael Torrealba @reta110](#) por crear la guía del curso de Laravel 5.

## Gracias por compartir tu conocimiento con nosotros:

- Manuel González - [@manuelitox](#) - ¡6 tutoriales!
- Sergio Márquez - [@smarquezs](#) - ¡5 tutoriales!
- Yair Rodríguez - [@imyairodriguez](#) - ¡5 tutoriales!
- Dimitri Acosta - [@Dimitri\\_Acosta](#) - ¡4 tutoriales!
- Guillermo Rodas - [@Garethderioth](#) - ¡2 tutoriales!
- Ariel Crippa - [@armandoweb](#) -
- Fernando Montoya - [@montogeek](#) -
- Jesús David Barranco Velasco - [@jdbv1982](#) -
- Jose Trezza - [@jtrezza](#) -
- Marcos Luna - [@marcoslunah](#) -
- Ricky Cruz - [@kldbug](#) -
- Yoel Monzon - [@yoelfme](#) -



## Muchísimas gracias por otorgarme tu confianza y permitirme crear un curso de Laravel libre y gratuito:

- Abel Ponce – [@vnponce](#)
- Alberto Sosa – [@albertodsgreen](#)
- Aldo Franco Zumaran Orefice
- Alejandro Candelas Tirado
- Alfonso Hernandez Montoya
- Alfonso Rios Perez
- Alfonso Serrano Albert – [@alfonso\\_serrano](#)
- Alvaro Yanes Basanta – [@al\\_yanes](#)
- Andres Felipe Vásquez – [@anvarstudios](#)
- Andres Gonzalo Riveros Düllmann – [@847ard](#)
- Andres Velasquez
- Andros Fenollosa Hurtado – [@androsfenollosa](#)
- Angeles Fernandez Regatillo Ruiz
- Antonio Gonzalez Pelaez
- Anwar Sarmiento
- Armando Lazarte – [@armandolazarte](#)
- Carlos Castaño Guillen – [@carloscasen](#)
- Casen Xu – [@CasenXu](#)
- César Cruz Cáceres
- Clemir Rondón Pérez – [@clemir](#)
- Daniel Muñoz Morales
- Daniel Saavedra – [@mantrax314](#)
- David Sánchez
- Dimitri Acosta – [@Dimitri\\_Acosta](#)
- Edgar Morales – [@egarmorales](#)
- Eduardo Cifuentes López – [@edcilo](#)
- Enrique Ruiz-chena Fuentes
- Erick Daniel Brito Arroyo
- Ernesto Aides – [@enaides](#)
- Fabio Enrique Sanchez Rosado – [@sauware](#)
- Federico Castañeda Ortiz
- Federico Ramato
- Fernando Fernandez Sanchez
- Filemon Linares Carbajal
- Fortunato Fernandez – [@EyTato](#)
- Francisco Alba Ponce – [@F\\_albaponce](#)
- Francisco René Sandoval Pérez – [@resand91](#)
- Gerardo Andres Arcos Celis
- Giovanni M. López – [@gml\\_89](#)
- Gonzalo Prudente – [@gonzaloprudente](#)
- Intelvid Limitada – [@cvallejo](#)
- Isaac Aldalur Errazkin
- Israel Elizarraraz
- Iván Abascal Lozano – [@abalozz](#)
- Jair Andres Galvis Tellez
- Javier Ocampo Bernasconi – [@job73](#)
- Jesús García Valadez – [@\\_Chucho\\_](#)
- Jesús Márquez Racero
- Jose Alegret Garcia
- Jose Ayram
- Jose Caballero Garcia – [@j82caballero](#)
- José Manuel Cerrejón González
- José Miguel García De León – [@menatoric59](#)

- Juan Carlos Valdés
- Juan Gonzalez – @juanDJGL
- Juan Ignacio De Los Cobos Islas – @nachodlc
- Juan Ignacio Navarro Guardado – @aerofun
- Juan Torres – @juanintorres
- Leyvi Silvan Silvan – @Leyvisilvan
- Luis Coronel – @380AC
- Luis Lagardera Salazar – @feltoxXx
- Manuel Antonio Vargas Medina – @mVargasMedina
- Manuel Cabrera Vivas
- Manuel Camacho – @maedca
- Manuel Villagordo Vera – @manuelvillagordo
- Marc Miralles – @MarcMirallesBio
- Mario Mendoza – @mamc8408
- Miguel Leon Fernandez
- Milton Gómez – @milthongomez
- Montalvo Miguel – @imontalvomiguel
- Nacho Alvarez Fernandez – @xDVeGax
- Narcisa Ludeña
- Omar Zurita – @kikez\_ec
- Oscar-mauricio Ortiz-forero
- Osledy Jocelyn Bazo P – @uokesita
- Rafael Agostini – @initsogar Rafael
- Raúl Larico Ramos
- Raul Soto Velasquez
- Richard Flores Palomo
- Roberto Espinosa Gómez
- Rubén Darío Gutiérrez Sánchez – @\_RubenGutierrez
- Sergio Valente – @Plus\_rt
- Shirley Concepcion Candelaria – @Shirley-Concep
- Snheidert Smalbach Moreno – @iamSmalbach
- Varela Perez – @RafaVarelaCol