**EBook Gratis**

# APRENDIZAJE Laravel

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#laravel

# Tabla de contenido

# Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: laravel

It is an unofficial and free Laravel ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Laravel.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Capítulo 1: Empezando con Laravel

## Observaciones

# Laravel StackOverflow Slack Community

*Próximamente*

# Tutorial Destacado

Empezando con Laravel

# Pautas de contribución

*Próximamente*

# Guía de estilo de contribución

*Próximamente*

# Acerca de Laravel

Creada por Taylor Otwell como un marco web PHP gratuito de código abierto , Laravel está destinada a facilitar y acelerar el proceso de desarrollo de aplicaciones web con un gran gusto por la simplicidad.

Sigue el patrón arquitectónico modelo-vista-controlador ( MVC ), así como el estándar de codificación PSR-2 y el estándar de carga automática PSR-4 .

Ejecutar un desarrollo guiado por pruebas ( TDD ) en Laravel es divertido y fácil de implementar.

Alojado en GitHub y disponible en https://github.com/laravel/laravel , laravel cuenta de un micro-servicios de arquitectura, por lo que es tremendamente extensible y esto, con facilidad, con el uso de encargo y o de terceros existente paquetes

## Principales características

### MVC

Laravel usa el modelo MVC, por lo tanto, hay tres partes centrales del marco que trabajan juntas: modelos, vistas y controladores. Los controladores son la parte principal donde se realiza la mayor parte del trabajo. Se conectan a modelos para obtener, crear o actualizar datos y mostrar los resultados en las vistas, que contienen la estructura HTML real de la aplicación.

### Motor de plantilla de hoja

Laravel se envía con un motor de plantillas conocido como Blade. Blade es bastante fácil de usar, sin embargo, potente. Una característica que el motor de plantillas Blade no comparte con otras populares es su permisividad; permitiendo el uso de código PHP plano en los archivos del motor de plantillas Blade.

Es importante tener en cuenta que los archivos del motor de plantillas Blade tienen `.blade` anexado a los nombres de los archivos justo antes del habitual `.php` que no es otra cosa que la extensión real del archivo. Como tal, `.blade.php` es la extensión de archivo resultante para los archivos de plantilla Blade. Los archivos del motor de plantillas Blade se almacenan en el directorio resources / views.

### Enrutamiento y middleware

Puede definir las URL de su aplicación con la ayuda de rutas. Estas rutas pueden contener datos variables, conectarse a controladores o pueden envolverse en middlewares. Middelware es un mecanismo para filtrar solicitudes HTTP. Se pueden utilizar para interactuar con las solicitudes antes de que lleguen a los controladores y, por lo tanto, se pueden modificar o rechazar solicitudes.

### Artesano

Artisan es la herramienta de línea de comandos que puede utilizar para controlar partes de Laravel. Hay muchos comandos disponibles para crear modelos, controladores y otros recursos necesarios para el desarrollo. También puede escribir sus propios comandos para extender la herramienta de línea de comandos Artisan.

### ORM elocuente

Para conectar sus modelos a varios tipos de bases de datos, Laravel ofrece su propio ORM con un amplio conjunto de funciones para trabajar. El marco también proporciona migración y siembra y también cuenta con reversiones.

### Manejo de eventos

El marco es capaz de manejar eventos a través de la aplicación. Puede crear detectores de eventos y controladores de eventos similares a los de NodeJs.

## Versiones

| Versión | Fecha de lanzamiento |
|---|---|
| 1.0 | 2011-06-09 |
| 2.0 | 2011-11-24 |
| 3.0 | 2012-02-22 |
| 3.1 | 2012-03-27 |
| 3.2 | 2012-05-22 |
| 4.0 | 2013-05-28 |
| 4.1 | 2013-12-12 |
| 4.2 | 2014-06-01 |
| 5.0 | 2015-02-04 |
| 5.1 (LTS) | 2015-06-09 |
| 5.2 | 2015-12-21 |
| 5.3 | 2016-08-24 |
| 5.4 | 2017-01-24 |

# Examples

## Bienvenido a la documentación de la etiqueta Laravel!

**Laravel** es un Framework PHP muy conocido. Aquí, aprenderás todo sobre Laravel. Comenzando desde *tan simple como* saber qué es la Programación Orientada a Objetos hasta el tema avanzado de desarrollo de paquetes Laravel.

Esta, como todas las demás etiquetas de documentación de Stackoverflow, es una documentación impulsada por la comunidad, por lo que si ya tiene experiencia en Laravel, ¡comparta sus conocimientos agregando sus propios temas o ejemplos! No se olvide de consultar nuestra **guía de estilo de Contribución** sobre este tema para obtener más información sobre cómo contribuir y la guía de estilo que creamos para asegurarnos de poder brindar la mejor experiencia a las personas que desean aprender más sobre Laravel.

Más que eso, estamos muy contentos de que venga, ¡espero que podamos verlo a menudo aquí!

## Guia de inicio

La guía de inicio es una navegación personalizada que ordenamos por nosotros mismos para facilitar la búsqueda de temas, especialmente para principiantes. Esta navegación está ordenada

por nivel de dificultad.

# Empezando

Instalación

# Vistas Laravel

Blade: Introducción

Blade: Variables y Estructuras de Control

O

**Instalación desde aquí**

1. Consigue el compositor desde aquí e instálalo.
2. Obtén Wamp desde aquí , instálalo y configura la variable de entorno de PHP
3. Obtener ruta a `www` y escriba el comando:

```
composer create-project --prefer-dist laravel/laravel projectname
```

Para instalar una versión específica de Laravel, obtenga la ruta a `www` y escriba el comando:

```
composer create-project --prefer-dist laravel/laravel=DESIRED_VERSION projectname
```

O

**Via Laravel Installer**

Primero, descargue el instalador de Laravel usando Composer:

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

Asegúrese de colocar el directorio `$HOME/.composer/vendor/bin` (o el directorio equivalente para su sistema operativo) en su $ PATH para que el `laravel` pueda `laravel` ejecutable `laravel` .

Una vez instalado, el `laravel new` comando `laravel new` creará una nueva instalación de Laravel en el directorio que especifique. Por ejemplo, `laravel new blog` creará un directorio llamado `blog` contiene una nueva instalación de Laravel con todas las dependencias de Laravel ya instaladas:

```
laravel new blog
```

Lea Empezando con Laravel en línea: https://riptutorial.com/es/laravel/topic/794/empezando-con-laravel

# Capítulo 2: Artesano

## Sintaxis

- php artisan [comando] [opciones] [argumentos]

## Parámetros

| Mando | Descripción |
|---|---|
| claro compilado | Eliminar el archivo de clase compilado |
| abajo | Poner la aplicación en modo mantenimiento. |
| env | Mostrar el entorno marco actual |
| ayuda | Muestra ayuda para un comando |
| lista | Listas de comandos |
| emigrar | Ejecutar las migraciones de base de datos. |
| optimizar | Optimizar el marco para un mejor rendimiento. |
| servir | Servir la aplicación en el servidor de desarrollo PHP |
| gitano | Interactúa con tu aplicación |
| arriba | Llevar la aplicación fuera del modo de mantenimiento. |
| nombre de la aplicación | Establecer el espacio de nombres de la aplicación |
| auth: borrar-restablece | Tokens de restablecimiento de contraseña caducados |
| caché: claro | Vaciar el caché de la aplicación |
| caché: tabla | Crear una migración para la tabla de base de datos de caché. |
| config: caché | Crear un archivo de caché para una carga de configuración más rápida |
| config: borrar | Eliminar el archivo de caché de configuración |
| db: semilla | Sembrar la base de datos con registros. |
| evento: generar | Genera los eventos que faltan y los oyentes basados en el registro. |

| Mando | Descripción |
|---|---|
| clave: generar | Establecer la clave de aplicación |
| hacer: auth | Scaffold inicio de sesión y registro vistas y rutas. |
| hacer: consola | Crear un nuevo comando artesanal. |
| marca: controlador | Crear una nueva clase de controlador |
| hacer: evento | Crear una nueva clase de evento |
| hacer: trabajo | Crear una nueva clase de trabajo |
| hacer: oyente | Crear una nueva clase de escucha de eventos |
| hacer: middleware | Crear una nueva clase de middleware |
| hacer: migración | Crear un nuevo archivo de migración |
| Haz un modelo | Crear una nueva clase de modelo elocuente. |
| hacer: política | Crear una nueva clase de política |
| marca: proveedor | Crear una nueva clase de proveedor de servicios |
| hacer un pedido | Crear una nueva clase de solicitud de formulario |
| hacer: sembradora | Crear una nueva clase de sembradora. |
| hacer: prueba | Crear una nueva clase de prueba |
| migrar: instalar | Crear el repositorio de migración. |
| migrar: actualizar | Restablecer y volver a ejecutar todas las migraciones. |
| migrar: restablecer | Deshacer todas las migraciones de base de datos |
| migrar: deshacer | Deshacer la última migración de la base de datos |
| migrar: estado | Mostrar el estado de cada migración. |
| cola: fallado | Listar todos los trabajos en cola fallidos |
| cola: tabla fallida | Crear una migración para la tabla de base de datos de trabajos de cola fallidos |
| cola: flush | Vacíe todos los trabajos de cola fallidos |
| cola: olvidar | Eliminar un trabajo de cola fallido |
| cola: escuchar | Escuchar una cola dada |

| Mando | Descripción |
|---|---|
| cola: reiniciar | Reinicie los demonios del trabajador de cola después de su trabajo actual |
| cola: reintentar | Reintentar un trabajo de cola fallido |
| cola: tabla | Crear una migración para la tabla de base de datos de trabajos en cola. |
| cola: trabajo | Procesar el siguiente trabajo en una cola |
| ruta: caché | Cree un archivo de caché de ruta para un registro de ruta más rápido |
| ruta: claro | Eliminar el archivo de caché de ruta |
| ruta: lista | Listar todas las rutas registradas |
| horario: correr | Ejecutar los comandos programados |
| sesion: mesa | Crear una migración para la tabla de base de datos de sesión. |
| vendedor: publicar | Publicar cualquier activo publicable desde paquetes de proveedores |
| vista: claro | Borrar todos los archivos de vista compilados |

# Examples

## Introducción

Artisan es una utilidad que puede ayudarte a realizar tareas repetitivas específicas con los comandos de bash. Cubre muchas tareas, entre las que se incluyen: trabajar con **migraciones de** bases de datos y **sembrar** , borrar el **caché** , crear los archivos necesarios para la configuración de **autenticación** , **crear** nuevos *controladores, modelos, clases de eventos* y mucho más.

> Artesano es el nombre de la interfaz de línea de comandos incluida con Laravel. Proporciona una serie de comandos útiles para su uso mientras desarrolla su aplicación.

Para ver una lista de todos los comandos de Artisan disponibles, puede usar el comando list:

```
php artisan list
```

Para saber más sobre cualquier comando disponible, solo preceda su nombre con la palabra clave de **ayuda** :

```
php artisan help [command-name]
```

## Listar todas las rutas registradas filtradas por múltiples métodos

```
php artisan route:list --method=GET --method=POST
```

Esto incluirá todas las rutas que acepten métodos `GET` y `POST` simultáneamente.

## Ejecutando comandos Laravel Artisan usando código PHP

También puede usar comandos de Laravel Artisan desde sus rutas o controladores.

Para ejecutar un comando usando código PHP:

```
Artisan::call('command-name');
```

Por ejemplo,

```
Artisan::call('db:seed');
```

## Creando y registrando nuevo comando artesanal.

Puede crear nuevos comandos a través de

`php artisan make:command [commandName]`

Así que esto creará la clase de comando [commandName] dentro del directorio `app/Console/Commands` .

Dentro de esta clase encontrará variables de `protected $signature` `protected $description` y `protected $description` , que representan el nombre y la `protected $description` de su comando que se usará para describir su comando.

después de crear el comando, puede registrar su comando dentro de la clase `app/Console/Kernel.php` donde encontrará la propiedad de los `commands` .

para que pueda agregar su comando dentro de la matriz $ command como:

```
protected $commands = [
    Commands\[commandName]::class
];
```

y luego puedo usar mi comando a través de la consola.

así como ejemplo he nombrado mi comando como

```
protected $signature = 'test:command';
```

Así que cada vez que corro

```
php artisan test:command
```

Llamará al método de `handle` dentro de la clase que tiene `test:command` firma `test:command` .

Lea Artesano en línea: https://riptutorial.com/es/laravel/topic/1140/artesano

# Capítulo 3: Autenticación

## Examples

### Autenticación múltiple

Laravel te permite usar múltiples tipos de autenticación con guardias específicos.

En laravel 5.3 la autenticación múltiple es un poco diferente de Laravel 5.2

Explicaré cómo implementar la característica de autenticación múltiple en 5.3.

Primero necesitas dos modelos de usuario diferentes

```
cp App/User.php App/Admin.php
```

cambie el nombre de la clase a Admin y establezca el espacio de nombres si utiliza modelos diferentes. debería parecerse a

**App \ Admin.php**

```
<?php

namespace App;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;

class Admin extends Authenticatable
{
    use Notifiable;

    protected $fillable = ['name', 'email', 'password'];
    protected $hidden   = ['password', 'remember_token'];

}
```

También necesitas crear una migración para admin.

```
php artisan make:migration create_admins_table
```

A continuación, edite el archivo de migración con el contenido de la migración de usuarios predeterminada. Se ve como esto

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

```
class CreateAdminsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('admins', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();

            $table->softDeletes();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('admins');
    }
}
```

Editar **config / auth.php**

```
'guards'    => [
        'web'   => [
            'driver'   => 'session',
            'provider' => 'users',
        ],

        'api'   => [
            'driver'   => 'token',
            'provider' => 'users',
        ],
        //Add Admin Guard
        'admin' => [
            'driver'   => 'session',
            'provider' => 'admins',
        ],
    ],
```

y

```
'providers' => [
        'users'  => [
            'driver' => 'eloquent',
            'model'  => App\User::class,
        ],
        //Add Admins Provider
```

```
        'admins' => [
            'driver' => 'eloquent',
            'model'  => App\Admin::class,
        ],
    ],
```

Tenga en cuenta que añadimos dos entradas. uno en **guardias** variable uno en **proveedores** variable.

Y así es como usas el otro guardia luego "web".

Mi aplicación \ Http \ Controllers \ Admin \ LoginController

```
<?php

namespace App\Http\Controllers\Admin;

use App\Http\Controllers\Controller;
use Illuminate\Foundation\Auth\AuthenticatesUsers;
use Illuminate\Support\Facades\Auth;

class AuthController extends Controller
{

    use AuthenticatesUsers;

    protected $guard = 'admin';

    protected $redirectTo = '/admin/';

    public function showLoginForm()
    {
        return view('admin.login');
    }

    protected function guard()
    {
        return Auth::guard($this->guard);
    }

}
```

esto necesita poca explicación.

en pocas palabras, **Auth :: guard ('admin')** le permitirá utilizar métodos de autenticación (como inicio de sesión, cierre de sesión, registro, etc.) con su administrador admin.

Por ejemplo

```
Auth::guard('admin')->login($user)
```

buscará $ usuario en la tabla de administradores e iniciará sesión con el usuario mientras

```
Auth::login($user)
```

Trabajará normalmente con la tabla de usuarios. La protección predeterminada se especifica en **config / auth.php** con la matriz *predeterminada* . En fresco laravel es "web".

En el controlador, debe implementar métodos de AuthenticatesUsers para mostrar sus rutas de visualización personalizadas. Y necesita implementar otras funciones como guardia para usar sus nuevas protecciones de usuario.

En este ejemplo, mi inicio de sesión de **administrador** es **admin / login.blade**

Y al implementar la función guard () para devolver **Auth :: guard ('admin'),** todos los métodos de rasgos de AuthenticatesUsers funcionan con "admin" guard.

En versiones anteriores de laravel, esto es un poco diferente de la versión 5.3.

en 5.2 la función getGuard devuelve la variable $ guard de la clase y la función principal (inicio de sesión) úsala en

```
Auth::guard($guard)->attempt(...)
```

en 5.3 la función de guarda devuelve todo Auth :: guard () y la función principal la usa como

```
$this->guard()->attempt(...)
```

Lea Autenticación en línea: https://riptutorial.com/es/laravel/topic/7051/autenticacion

# Capítulo 4: Autorización

## Introducción

Laravel proporciona una forma sencilla de autorizar acciones de usuario en recursos específicos. Con la Autorización, puede permitir selectivamente que los usuarios accedan a ciertos recursos y denegar el acceso a otros. Laravel proporciona una API simple para administrar las autorizaciones de los usuarios mediante el uso de `Gates` y `Policies` . `Gates` proporciona un enfoque simple basado en el cierre de la autorización mediante el `AuthServiceProvider` mientras que las `Policies` permiten organizar la lógica de autorización en torno a modelos que usan clases.

## Examples

### Usando puertas

`Gates` son cierres que determinan si un usuario puede realizar una determinada acción en un recurso. `Gates` se definen normalmente en el método de arranque de `AuthServiceProvider` y se nombran de forma sucinta para reflejar lo que está haciendo. Un ejemplo de una puerta que solo permite a los usuarios premium ver algunos contenidos se verá así:

```
Gate::define('view-content', function ($user, $content){
    return $user->isSubscribedTo($content->id);
});
```

Una `Gate` siempre recibe una instancia de usuario como primer argumento, no necesita pasarla al usar la puerta y, opcionalmente, puede recibir argumentos adicionales, como el modelo elocuente en cuestión.

### Autorizando acciones con puertas

Para usar el ejemplo anterior en una plantilla blade para ocultar el contenido del usuario, normalmente haría algo como esto:

```
@can('view-content', $content)
 <! -- content here -->
@endcan
```

Para evitar completamente la navegación al contenido, puede hacer lo siguiente en su controlador:

```
if(Gate::allows('view-content', $content)){
    /* user can view the content */
}

        OR

if(Gate::denies('view-content', $content)){
```

```
    /* user cannot view content */
}
```

Nota: No es necesario que pase el usuario autenticado actualmente a estos métodos, Laravel se encarga de eso por usted.

## Políticas

Las políticas son clases que lo ayudan a organizar la lógica de autorización en torno a un recurso modelo. Usando nuestro ejemplo anterior, podríamos tener una `ContentPolicy` que administra el acceso de los usuarios al modelo de `Content` .

Para hacer `ContentPolicy` , laravel proporciona un comando artesanal. Simplemente correr

```
php artisan make:policy ContentPolicy
```

Esto creará una clase de política vacía y la colocará en la `app/Policies` carpeta de `app/Policies` . Si la carpeta no existe, Laravel la creará y colocará la clase dentro.

Una vez creadas, las políticas deben registrarse para ayudar a Laravel a saber qué políticas usar cuando se autorizan acciones en modelos. El `AuthServiceProvider` de Laravel, que viene con todas las instalaciones nuevas de Laravel, tiene una propiedad de políticas que asigna sus modelos elocuentes a sus políticas de autorización. Todo lo que necesitas hacer agrega la asignación a la matriz.

```
protected $policies = [
    Content::class => ContentPolicy::class,
 ];
```

## Políticas de escritura

`Policies` escritura sigue casi el mismo patrón que escribir `Gates` . La puerta de permisos de contenido puede reescribirse como una Política como esta:

```
function view($user, $content)
{
    return $user->isSubscribedTo($content->id);
}
```

Las políticas pueden contener más métodos según sea necesario para atender todos los casos de autorización de un modelo.

## Autorizar acciones con políticas

### A través del modelo de usuario

El modelo de usuario de Laravel contiene dos métodos que ayudan con las autorizaciones utilizando `Policies` ; `can` y `can't` Estos dos pueden usarse para determinar si un usuario tiene autorización o no en un modelo, respectivamente.

Para verificar si un usuario puede ver un contenido o no, puede hacer lo siguiente:

```
if($user->can('view', $content)){
    /* user can view content */
}

        OR

if($user->cant('view', $content)){
    /* user cannot view content */
}
```

### Via Middleware

```
Route::get('/contents/{id}, function(Content $content){
    /* user can view content */
})->middleware('can:view,content');
```

### A través de los controladores

Laravel proporciona un método auxiliar, denominado `authorize` que toma el nombre de la política y el modelo asociado como argumentos, y autoriza la acción según su lógica de autorización o niega la acción y lanza una excepción de `AuthorizationException` que el controlador de excepciones de Laravel convierte a un `403 HTTP response`

```
pubic function show($id)
{
    $content = Content::find($id);

    $this->authorize('view', $content);

    /* user can view content */
}
```

Lea Autorización en línea: https://riptutorial.com/es/laravel/topic/9360/autorizacion

# Capítulo 5: Ayudante de cámara

## Introducción

Valet es un entorno de desarrollo hecho a medida para macOS. Se abstrae la necesidad de máquinas virtuales, Homestead o Vagrant. Ya no es necesario actualizar constantemente su `/etc/hosts` . Incluso puedes compartir tus sitios públicamente usando túneles locales.

Laravel Valet hace que todos los sitios estén disponibles en un dominio `*.dev` vinculando nombres de carpetas a nombres de dominio.

## Sintaxis

- comando de valet [opciones] [argumentos]

## Parámetros

| Parámetro | Conjunto de valores |
|---|---|
| mando | dominio , fetch-share-url, olvidar, ayuda, instalar, enlace , enlaces , listas, registros, en la última versión, abrir, estacionar , rutas, reiniciar, asegurar, iniciar, detener, desinstalar, desvincular, no seguro, lo que |
| opciones | -h, --help, -q, --quiet, -V, --version, --ansi, --no-ansi, -n, --no-interaccion, -v, -vv, -vvv, - -verboso |
| argumentos | *(Opcional)* |

## Observaciones

Debido a que Valet para Linux y Windows no son oficiales, no habrá soporte fuera de sus respectivos repositorios Github.

## Examples

### Enlace de valet

Este comando es útil si desea servir un solo sitio en un directorio y no en todo el directorio.

```
cd ~/Projects/my-blog/
valet link awesome-blog
```

Valet creará un enlace simbólico en `~/.valet/Sites` que apunta a su directorio de trabajo actual.

Después de ejecutar el comando de enlace, puede acceder al sitio en su navegador en `http://awesome-blog.dev` .

Para ver una lista de todos sus directorios vinculados, ejecute el comando valet links . Puedes usar `valet unlink awesome-blog` para destruir el enlace simbólico.

## Parque de valet

```
cd ~/Projects
valet park
```

Este comando registrará su directorio de trabajo actual como una ruta que Valet debe buscar sitios. Ahora, cualquier proyecto Laravel que cree dentro de su directorio "estacionado" se servirá automáticamente utilizando la convención `http://folder-name.dev` .

## Enlaces de valet

Este comando mostrará todos los enlaces de Valet registrados que ha creado y sus rutas de archivo correspondientes en su computadora.

**Mando:**

```
valet links
```

**Salida de muestra:**

```
...
site1 -> /path/to/site/one
site2 -> /path/to/site/two
...
```

**Nota 1:** puede ejecutar este comando desde cualquier lugar, no solo desde una carpeta vinculada.

**Nota 2: los** sitios se incluirán en la lista sin el **.dev** final, pero seguirá usando **site1.dev** para acceder a su aplicación desde el navegador.

## Instalación

***¡¡IMPORTANTE!!* Valet es una herramienta diseñada solo para macOS.**

**Prerrequisitos**

- El valet utiliza el puerto HTTP de su máquina local (puerto 80), por lo tanto, no podrá usarlo si *Apache* o *Nginx* están instalados y ejecutándose en la misma máquina.
- Se requiere que Homebrew, el administrador de paquetes no oficial de macOS, use Valet correctamente.
- Asegúrese de que Homebrew esté actualizado a la última versión ejecutando `brew update` en

el terminal.

**Instalación**

- Instale PHP 7.1 utilizando Homebrew a través de `brew install homebrew/php/php71` .
- Instalar Valet con Composer a través de `composer global require laravel/valet` .
- `~/.composer/vendor/bin` directorio `~/.composer/vendor/bin` al "PATH" de su sistema si aún no está allí.
- Ejecute el comando de `valet install` .

**Posterior a la instalación** Durante el proceso de instalación, Servicio de *Dnsmasq* instalado. También registró el demonio de Valet para que se inicie automáticamente cuando se inicie el sistema, por lo que no necesita ejecutar el `valet start` o la `valet install` cada vez que reinicie su máquina.

## Dominio valet

Este comando le permite cambiar o ver el TLD *(dominio de nivel superior)* utilizado para enlazar dominios a su máquina local.

**Obtener el TLD actual**

```
$ valet domain
> dev
```

**Establecer el TLD**

```
$ valet domain local
> Your Valet domain has been updated to [local].
```

## Instalación (Linux)

***¡¡IMPORTANTE!! Valet es una herramienta diseñada para macOS, la versión a continuación está adaptada para el sistema operativo Linux.***

**Prerrequisitos**

- **No** instale el valet como `root` o usando el comando `sudo` .
- El valet utiliza el puerto HTTP de su máquina local (puerto 80), por lo tanto, no podrá usarlo si Apache o Nginx están instalados y ejecutándose en la misma máquina.
- Se requiere una versión actualizada del `composer` para instalar y ejecutar Valet.

**Instalación**

- Ejecutar `composer global require cpriego/valet-linux` para instalar Valet globalmente.
- Ejecute el comando de `valet install` para finalizar la instalación.

**Instalación posterior**

Durante el proceso de instalación, Valet instaló DnsMasq. También registró el demonio de Valet para que se inicie automáticamente cuando se inicie el sistema, por lo que no necesita ejecutar el `valet start` o la `valet install` cada vez que reinicie su máquina.

La Documentación Oficial se puede encontrar aquí .

Lea Ayudante de cámara en línea: https://riptutorial.com/es/laravel/topic/1906/ayudante-de-camara

# Capítulo 6: Ayudantes

## Introducción

Los ayudantes de Laravel son las funciones accesibles globalmente definidas por el marco. Puede llamarse directamente y usarse independientemente en cualquier lugar dentro de la aplicación sin necesidad de crear una instancia de un objeto o clase de importación.

Hay ayudantes para manipular *Arrays* , *Paths* , *Strings* , *URLs* , etc.

## Examples

### Métodos de matriz

**array_add ()**

Este método se utiliza para agregar nuevos pares de valores clave a una matriz.

```
$array = ['username' => 'testuser'];

$array = array_add($array, 'age', 18);
```

resultado

```
['username' => 'testuser', 'age' => 18]
```

### Metodos de cuerda

**el caso de Carmel()**

Este método cambia una cadena a camel case

```
camel_case('hello_world');
```

resultado

```
HelloWorld
```

### Camino mehods

Los métodos de ruta ayudan a acceder fácilmente a las rutas relacionadas con la aplicación fácilmente desde cualquier lugar.

**public_path ()**

Este método devuelve la ruta pública completa de la aplicación. Que es el directorio público.

```
$path = public_path();
```

## Urls

### url ()

La función url genera una URL completa para la ruta dada.

si tu sitio es `hello.com`

```
echo url('my/dashboard');
```

volvería

```
hello.com/my/dashboard
```

si no se pasa nada al método url, se devolverá una instancia de `Illuminate\Routing\UrlGenerator` , y se podría usar de esta manera

devolvería la url actual

```
echo url()->current();
```

regresaría url completo

```
echo url()->full();
```

volvería url anterior

```
echo url()->previous();
```

Lea Ayudantes en línea: https://riptutorial.com/es/laravel/topic/8827/ayudantes

# Capítulo 7: Base de datos

## Examples

### Conexiones de base de datos múltiples

Laravel permite al usuario trabajar en múltiples conexiones de base de datos. Si necesita conectarse a varias bases de datos y hacer que funcionen juntos, tenga cuidado con la configuración de la conexión.

También permite el uso de diferentes tipos de bases de datos en la misma aplicación si así lo requiere.

**Conexión predeterminada** En `config/database.php` , puede ver la llamada al elemento de configuración:

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

Este nombre hace referencia al nombre `mysql` las conexiones a continuación:

```
'connections' => [

    'sqlite' => [
        'driver' => 'sqlite',
        'database' =>  database_path('database.sqlite'),
        'prefix' => '',
    ],

    'mysql' => [
        'driver' => 'mysql',
        'host' => env('DB_HOST', 'localhost'),
        'port' => env('DB_PORT', '3306'),
        'database' => env('DB_DATABASE', 'forge'),
        'username' => env('DB_USERNAME', 'forge'),
        'password' => env('DB_PASSWORD', ''),
        'charset' => 'utf8',
        'collation' => 'utf8_unicode_ci',
        'prefix' => '',
        'strict' => false,
        'engine' => null,
    ],
],
```

Si no mencionó el nombre de la conexión de la base de datos en otros códigos o comandos, Laravel seleccionará el nombre de la conexión de la base de datos predeterminada. sin embargo, en varias conexiones de base de datos, incluso si configura la conexión predeterminada, es mejor que configure en todas partes qué conexión de base de datos utilizó.

**Archivo de migración**

En el archivo de migración, si se trata de una conexión de base de datos única, puede utilizar:

```
 Schema::create("table",function(Blueprint $table){
     $table->increments('id');
});
```

En la conexión de varias bases de datos, utilizará el método `connection()` para indicar a Laravel qué conexión de base de datos usa:

```
 Schema::connection("sqlite")->create("table",function(Blueprint $table){
     $table->increments('id');
});
```

**Migrar Artesanalmente**

Si usa una conexión de base de datos única, ejecutará:

```
php artisan migrate
```

Sin embargo, para la conexión de varias bases de datos, es mejor que sepa qué conexión de base de datos mantiene los datos de migración. por lo que ejecutará el siguiente comando:

```
php artisan migrate:install --database=sqlite
```

Este comando instalará la tabla de migración en la base de datos de destino para preparar la migración.

```
php artisan migrate --database=sqlite
```

Este comando ejecutará la migración y guardará los datos de la migración en la base de datos de destino

```
php artisan migrate:rollback --database=sqlite
```

Este comando revertirá la migración y guardará los datos de la migración en la base de datos de destino

**Modelo elocuente**

Para especificar una conexión de base de datos utilizando Eloquent, debe definir la propiedad `$connection` :

```
namespace App\Model\Sqlite;
class Table extends Model
{
    protected $table="table";
    protected $connection = 'sqlite';
}
```

Para especificar otra (segunda) conexión de base de datos usando Eloquent:

```
namespace App\Model\MySql;
class Table extends Model
{
    protected $table="table";
    protected $connection = 'mysql';
}
```

Laravel utilizará la propiedad `$connection` definida en un modelo para utilizar la conexión especificada definida en `config/database.php` . Si la propiedad `$connection` no está definida en un modelo, se utilizará el valor predeterminado.

También puede especificar otra conexión utilizando el método estático `on` :

```
// Using the sqlite connection
Table::on('sqlite')->select(...)->get()
// Using the mysql connection
Table::on('mysql')->select(...)->get()
```

### Base de datos / Generador de consultas

También puede especificar otra conexión utilizando el generador de consultas:

```
// Using the sqlite connection
DB::connection('sqlite')->table('table')->select(...)->get()
// Using the mysql connection
DB::connection('mysql')->table('table')->select(...)->get()
```

### Prueba de unidad

Laravel proporciona `seeInDatabase($table,$fielsArray,$connection)` para probar el código de conexión de la base de datos. En el archivo de prueba de unidad, debe hacer como:

```
$this
    ->json(
        'GET',
        'result1/2015-05-08/2015-08-08/a/123'
    )
     ->seeInDatabase("log", ["field"=>"value"], 'sqlite');
```

De esta manera, Laravel sabrá qué conexión de base de datos probar.

### Transacciones de base de datos en prueba unitaria

Laravel permite a la base de datos revertir todos los cambios durante las pruebas. Para probar múltiples conexiones de base de datos, necesita establecer propiedades de `$connectionsToTransact`

```
use Illuminate\Foundation\Testing\DatabaseMigrations;

class ExampleTest extends TestCase
{
     use DatabaseTransactions;
```

```
     $connectionsToTransact =["mysql","sqlite"] //tell Laravel which database need to rollBack

    public function testExampleIndex()
    {
        $this->visit('/action/parameter')
         ->see('items');
    }
}
```

Lea Base de datos en línea: https://riptutorial.com/es/laravel/topic/1093/base-de-datos

# Capítulo 8: Cajero

## Observaciones

Laravel Cashier se puede utilizar para la facturación de suscripción proporcionando una interfaz en los servicios de suscripción de Braintree y Stripe. Además de la administración básica de suscripciones, se puede utilizar para gestionar cupones, intercambiar suscripciones, cantidades, períodos de gracia de cancelación y generación de facturas en PDF.

## Examples

### Configuración de la raya

#### Configuración inicial

Para usar Stripe para manejar los pagos, debemos agregar lo siguiente a `composer.json` luego ejecutar la `composer update` :

```
"laravel/cashier": "~6.0"
```

La siguiente línea debe agregarse a `config/app.php` , el proveedor de servicios:

```
Laravel\Cashier\CashierServiceProvider
```

---

#### Configuración de base de datos

Para utilizar el cajero necesitamos configurar las bases de datos, si una tabla de usuarios no existe, necesitamos crear una y también debemos crear una tabla de suscripciones. El siguiente ejemplo modifica una tabla de `users` existente. Ver Modelos Elocuentes para más información sobre modelos.

Para utilizar el cajero, cree una nueva migración y agregue lo siguiente que logrará lo anterior:

```
// Adjust users table

Schema::table('users', function ($table) {
    $table->string('stripe_id')->nullable();
    $table->string('card_brand')->nullable();
    $table->string('card_last_four')->nullable();
    $table->timestamp('trial_ends_at')->nullable();
});

//Create subscriptions table

Schema::create('subscriptions', function ($table) {
    $table->increments('id');
    $table->integer('user_id');
    $table->string('name');
```

```
    $table->string('stripe_id');
    $table->string('stripe_plan');
    $table->integer('quantity');
    $table->timestamp('trial_ends_at')->nullable();
    $table->timestamp('ends_at')->nullable();
    $table->timestamps();
});
```

Luego necesitamos ejecutar `php artisan migrate` para actualizar nuestra base de datos.

---

### Configuración del modelo

Luego tenemos que agregar el rasgo facturable al modelo de usuario que se encuentra en `app/User.php` y cambiarlo a lo siguiente:

```
use Laravel\Cashier\Billable;

class User extends Authenticatable
{
    use Billable;
}
```

---

### Teclas de rayas

Para asegurarnos de que finalizamos el dinero en nuestra propia cuenta de Stripe, debemos configurarlo en el archivo `config/services.php` agregando la siguiente línea:

```
'stripe' => [
    'model'  => App\User::class,
    'secret' => env('STRIPE_SECRET'),
],
```

Reemplazando el `STRIPE_SECRET` con su propia clave secreta de banda.

---

Después de completar este Cajero, Strip se configura para que pueda continuar con la configuración de las suscripciones.

Lea Cajero en línea: https://riptutorial.com/es/laravel/topic/7474/cajero

# Capítulo 9: Cambiar el comportamiento de enrutamiento predeterminado en Laravel 5.2.31 +

## Sintaxis

- mapa de funciones públicas (Router $ router) // Defina las rutas para la aplicación.
- función protegida mapWebRoutes (Router $ router) // Defina las rutas "web" para la aplicación.

## Parámetros

| Parámetro | Encabezamiento |
| --- | --- |
| Enrutador $ enrutador | \ Illuminate \ Routing \ Router $ router |

## Observaciones

Middleware significa que cada llamada a una ruta pasará por el middleware antes de que realmente llegue a su código específico de ruta. En Laravel, el middleware web se utiliza para garantizar el manejo de la sesión o la comprobación de token csrf, por ejemplo.

Hay otros middlewares como auth o api por defecto. También puedes crear fácilmente tu propio middleware.

## Examples

### Agregar api-route con otro middleware y mantener el middleware web predeterminado

Desde la versión 5.2.31 de Laravel, el middleware web se aplica de forma predeterminada dentro de RouteServiceProvider ( https://github.com/laravel/laravel/commit/5c30c98db96459b4cc878d085490e4677b0b67ed)

En app / Providers / RouteServiceProvider.php encontrará las siguientes funciones que aplican el middleware en cada ruta dentro de su app / Http / route.php

```
public function map(Router $router)
{
    $this->mapWebRoutes($router);
}
```

```
// ...

protected function mapWebRoutes(Router $router)
{
    $router->group([
        'namespace' => $this->namespace, 'middleware' => 'web',
    ], function ($router) {
        require app_path('Http/routes.php');
    });
}
```

Como se puede ver se aplica la web de **middleware** . Podrías cambiar esto aquí. Sin embargo, también puede agregar fácilmente otra entrada para poder poner las rutas de api, por ejemplo, en otro archivo (por ejemplo, route-api.php)

```
public function map(Router $router)
{
    $this->mapWebRoutes($router);
    $this->mapApiRoutes($router);
}

protected function mapWebRoutes(Router $router)
{
    $router->group([
        'namespace' => $this->namespace, 'middleware' => 'web',
    ], function ($router) {
        require app_path('Http/routes.php');
    });
}

protected function mapApiRoutes(Router $router)
{
    $router->group([
        'namespace' => $this->namespace, 'middleware' => 'api',
    ], function ($router) {
        require app_path('Http/routes-api.php');
    });
}
```

Con esto, puede separar fácilmente las rutas de api de las rutas de su aplicación sin el envoltorio del grupo desordenado dentro de sus rutas.

Lea Cambiar el comportamiento de enrutamiento predeterminado en Laravel 5.2.31 + en línea: https://riptutorial.com/es/laravel/topic/4285/cambiar-el-comportamiento-de-enrutamiento-predeterminado-en-laravel-5-2-31-plus

# Capítulo 10: Clase CustomException en Laravel

## Introducción

Se lanzan excepciones de PHP cuando ocurre un evento o error sin precedentes.

Como regla general, no debe usarse una excepción para controlar la lógica de la aplicación, como las sentencias if, y debe ser una subclase de la clase Exception.

Una de las principales ventajas de que una sola clase detecte todas las excepciones es que podemos crear controladores de excepciones personalizados que devuelven mensajes de respuesta diferentes según la excepción.

## Examples

### Clase CustomException en laravel

todos los errores y excepciones, tanto personalizados como predeterminados, son manejados por la clase Handler en app / Exceptions / Handler.php con la ayuda de dos métodos.

- informe()
- hacer()

```
public function render($request, Exception $e)
{
 //check if exception is an instance of ModelNotFoundException.
 if ($e instanceof ModelNotFoundException)
 {
     // ajax 404 json feedback
     if ($request->ajax())
     {
         return response()->json(['error' => 'Not Found'], 404);
     }
     // normal 404 view page feedback
     return response()->view('errors.missing', [], 404);
  }
   return parent::render($request, $e);
}
```

luego cree la vista relacionada con el error en la carpeta de errores llamada 404.blade.php

Usuario no encontrado.

Rompiste el balance de internet.

Lea Clase CustomException en Laravel en línea:

https://riptutorial.com/es/laravel/topic/9550/clase-customexception-en-laravel

# Capítulo 11: Colas

## Introducción

Las colas permiten que su aplicación reserve partes del trabajo que requieren mucho tiempo para ser manejadas por un proceso en segundo plano.

## Examples

### Casos de uso

Por ejemplo, si está enviando un correo electrónico a un cliente después de comenzar una tarea, es mejor redirigir inmediatamente al usuario a la página siguiente mientras se pone en cola el correo electrónico que se enviará en segundo plano. Esto acelerará el tiempo de carga para la página siguiente, ya que enviar un correo electrónico a veces puede llevar varios segundos o más.

Otro ejemplo sería actualizar un sistema de inventario después de que un cliente verifique con su pedido. En lugar de esperar a que se completen las llamadas a la API, lo que puede demorar varios segundos, puede redirigir inmediatamente al usuario a la página de éxito de pago mientras se ponen en cola las llamadas a la API para que ocurran en segundo plano.

### Configuración del controlador de cola

Cada uno de los controladores de cola de Laravel se configura desde el archivo `config/queue.php` . Un controlador de cola es el controlador para administrar cómo ejecutar un trabajo en cola, identificar si los trabajos tuvieron éxito o no, e intentar el trabajo nuevamente si está configurado para hacerlo.

Fuera de la caja, Laravel admite los siguientes controladores de cola:

#### sync

Sincronización, o síncrono, es el controlador de cola predeterminado que ejecuta un trabajo en cola dentro de su proceso existente. Con este controlador habilitado, efectivamente no tiene cola ya que el trabajo en cola se ejecuta inmediatamente. Esto es útil para propósitos locales o de prueba, pero claramente no se recomienda para producción ya que elimina el beneficio de rendimiento de la configuración de su cola.

#### database

Este controlador almacena trabajos en cola en la base de datos. Antes de habilitar este controlador, deberá crear tablas de base de datos para almacenar sus trabajos en cola y fallidos:

```
php artisan queue:table
php artisan migrate
```

#### sqs

Este controlador de cola utiliza el Servicio de cola simple de Amazon para administrar trabajos en cola. Antes de habilitar este trabajo, debe instalar el siguiente paquete compositor: `aws/aws-sdk-php ~3.0`

También tenga en cuenta que si planea usar demoras para trabajos en cola, Amazon SQS solo admite una demora máxima de 15 minutos.

#### iron

Estos controladores de cola utilizan Iron para administrar trabajos en cola.

#### redis

Este controlador de cola utiliza una instancia de Redis para administrar trabajos en cola. Antes de usar este controlador de cola, deberá configurar una copia de Redis e instalar la siguiente dependencia del `predis/predis ~1.0` : `predis/predis ~1.0`

#### beanstalkd

Este controlador de cola utiliza una instancia de Beanstalk para administrar trabajos en cola. Antes de usar este controlador de cola, deberá configurar una copia de Beanstalk e instalar la siguiente dependencia del compositor: `pda/pheanstalk ~3.0`

#### null

Especificar nulo como su controlador de cola descartará cualquier trabajo en cola.

Lea Colas en línea: https://riptutorial.com/es/laravel/topic/2651/colas

# Capítulo 12: Colecciones

## Sintaxis

- $ collection = collect (['Value1', 'Value2', 'Value3']); // Las teclas predeterminadas son 0, 1, 2, ...,

## Observaciones

`Illuminate\Support\Collection` proporciona una interfaz fluida y conveniente para tratar con matrices de datos. Puede que los haya utilizado sin saberlo, por ejemplo, las consultas de modelos que recuperan varios registros devuelven una instancia de `Illuminate\Support\Collection` .

Para obtener documentación actualizada sobre Colecciones, puede encontrar la documentación oficial aquí.

## Examples

### Creando Colecciones

Al usar el ayudante de `collect()` , puede crear fácilmente nuevas instancias de colección pasando una matriz como:

```
$fruits = collect(['oranges', 'peaches', 'pears']);
```

Si no desea usar las funciones de ayuda, puede crear una nueva Colección utilizando la clase directamente:

```
$fruits = new Illuminate\Support\Collection(['oranges', 'peaches', 'pears']);
```

Como se mencionó en los comentarios, los Modelos devuelven por defecto una instancia de `Collection` , sin embargo, usted es libre de crear sus propias colecciones según sea necesario. Si no se especifica ninguna matriz en la creación, se creará una Colección vacía.

### dónde()

Puede seleccionar ciertos elementos de una colección utilizando el método `where()` .

```
$data = [
    ['name' => 'Taylor',  'coffee_drinker' => true],
    ['name' => 'Matt', 'coffee_drinker' => true]
];

$matt = collect($data)->where('name', 'Matt');
```

Este bit de código seleccionará todos los elementos de la colección donde el nombre es 'Matt'. En este caso, solo se devuelve el segundo artículo.

# Anidamiento

Al igual que la mayoría de los métodos de matriz en Laravel, `where()` admite la búsqueda de elementos anidados. Extendamos el ejemplo anterior agregando una segunda matriz:

```
$data = [
    ['name' => 'Taylor',  'coffee_drinker' => ['at_work' => true, 'at_home' => true]],
    ['name' => 'Matt', 'coffee_drinker' => ['at_work' => true, 'at_home' => false]]
];

$coffeeDrinkerAtHome = collect($data)->where('coffee_drinker.at_home', true);
```

Esto solo lo devolverá Taylor, mientras él toma café en casa. Como puede ver, el anidamiento se admite mediante la notación de puntos.

# Adiciones

Al crear una colección de objetos en lugar de matrices, estos también se pueden filtrar usando `where()` también. La Colección intentará recibir todas las propiedades deseadas.

5.3

Tenga en cuenta que, desde Laravel 5.3, el método `where()` intentará comparar los valores de forma predeterminada. Eso significa que al buscar `(int)1` , todas las entradas que contengan `'1'` también se devolverán. Si no te gusta ese comportamiento, puedes usar el método `whereStrict()` .

## Usando Obtener valor de búsqueda o Devolver predeterminado

A menudo se encuentra en una situación en la que necesita encontrar un valor correspondiente a las variables, y las colecciones lo cubrieron.

En el siguiente ejemplo, tenemos tres configuraciones regionales diferentes en una matriz con un código de llamada correspondiente asignado. Queremos poder proporcionar una configuración regional y, a cambio, obtener el código de llamada asociado. El segundo parámetro en `get` es un parámetro predeterminado si no se encuentra el primer parámetro.

```
function lookupCallingCode($locale)
{
    return collect([
        'de_DE' => 49,
        'en_GB' => 44,
        'en_US' => 1,
    ])->get($locale, 44);
}
```

En el ejemplo anterior podemos hacer lo siguiente

```
lookupCallingCode('de_DE'); // Will return 49
lookupCallingCode('sv_SE'); // Will return 44
```

Incluso puede pasar una devolución de llamada como el valor predeterminado. El resultado de la devolución de llamada se devolverá si la clave especificada no existe:

```
    return collect([
        'de_DE' => 49,
        'en_GB' => 44,
        'en_US' => 1,
    ])->get($locale, function() {
        return 44;
    });
```

## Usando Contiene para verificar si una colección cumple cierta condición

Un problema común es tener una colección de elementos que todos deben cumplir con ciertos criterios. En el siguiente ejemplo, hemos recopilado dos artículos para un plan de dieta y queremos verificar que la dieta no contenga ningún alimento no saludable.

```
// First we create a collection
$diet = collect([
    ['name' => 'Banana', 'calories' => '89'],
    ['name' => 'Chocolate', 'calories' => '546']
]);

// Then we check the collection for items with more than 100 calories
$isUnhealthy = $diet->contains(function ($i, $snack) {
    return $snack["calories"] >= 100;
});
```

En el caso anterior, la variable `$isUnhealthy` se establecerá en `true` cuando el Chocolate cumpla con la condición, y la dieta no sea saludable.

## Usando Pluck para extraer ciertos valores de una colección

A menudo se encontrará con una colección de datos en los que solo está interesado en partes de los datos.

En el siguiente ejemplo, obtuvimos una lista de participantes en un evento y queremos proporcionarle a la guía turística una lista simple de nombres.

```
// First we collect the participants
$participants = collect([
    ['name' => 'John', 'age' => 55],
    ['name' => 'Melissa', 'age' => 18],
    ['name' => 'Bob', 'age' => 43],
    ['name' => 'Sara', 'age' => 18],
]);
```

```
// Then we ask the collection to fetch all the names
$namesList = $partcipants->pluck('name')
// ['John', 'Melissa', 'Bob', 'Sara'];
```

También puede usar el `pluck` para colecciones de objetos o matrices / objetos anidados con notación de puntos.

```
$users = User::all(); // Returns Eloquent Collection of all users
$usernames = $users->pluck('username'); // Collection contains only user names

$users->load('profile'); // Load a relationship for all models in collection

// Using dot notation, we can traverse nested properties
$names = $users->pluck('profile.first_name'); // Get all first names from all user profiles
```

## Usando Map para manipular cada elemento en una colección

A menudo es necesario cambiar la forma en que se estructura un conjunto de datos y manipular ciertos valores.

En el siguiente ejemplo, tenemos una colección de libros con un descuento adjunto. Pero preferimos tener una lista de libros con un precio que ya está descontado.

```
$books = [
    ['title' => 'The Pragmatic Programmer', 'price' => 20, 'discount' => 0.5],
    ['title' => 'Continuous Delivery', 'price' => 25, 'discount' => 0.1],
    ['title' => 'The Clean Coder', 'price' => 10, 'discount' => 0.75],
];

$discountedItems =  collect($books)->map(function ($book) {
   return ['title' => $book["title"], 'price' => $book["price"] * $book["discount"]];
});

//[
//    ['title' => 'The Pragmatic Programmer', 'price' => 10],
//    ['title' => 'Continuous Delivery', 'price' => 12.5],
//    ['title' => 'The Clean Coder', 'price' => 5],
//]
```

Esto también podría usarse para cambiar las claves, digamos que queremos cambiar el `title` la clave para `name` esta sería una solución adecuada.

## Uso de sum, avg, min o max en una colección para cálculos estadísticos

Las colecciones también le proporcionan una manera fácil de hacer cálculos estadísticos simples.

```
$books = [
    ['title' => 'The Pragmatic Programmer', 'price' => 20],
    ['title' => 'Continuous Delivery', 'price' => 30],
    ['title' => 'The Clean Coder', 'price' => 10],
]

$min = collect($books)->min('price'); // 10
```

```
$max = collect($books)->max('price'); // 30
$avg = collect($books)->avg('price'); // 20
$sum = collect($books)->sum('price'); // 60
```

### Ordenar una colección

Hay varias maneras diferentes de clasificar una colección.

# Ordenar()

El método de `sort` ordena la colección:

```
$collection = collect([5, 3, 1, 2, 4]);

$sorted = $collection->sort();

echo $sorted->values()->all();

returns : [1, 2, 3, 4, 5]
```

El método de `sort` también permite pasar una devolución de llamada personalizada con su propio algoritmo. Bajo el capó de tipo usa el `usort` de php.

```
$collection = $collection->sort(function ($a, $b) {
    if ($a == $b) {
        return 0;
    }
    return ($a < $b) ? -1 : 1;
});
```

# Ordenar por()

El método `sortBy` ordena la colección por la clave dada:

```
$collection = collect([
['name' => 'Desk', 'price' => 200],
['name' => 'Chair', 'price' => 100],
['name' => 'Bookcase', 'price' => 150],
]);

$sorted = $collection->sortBy('price');

echo $sorted->values()->all();

returns:  [
        ['name' => 'Chair', 'price' => 100],
        ['name' => 'Bookcase', 'price' => 150],
        ['name' => 'Desk', 'price' => 200],
    ]
```

El método `sortBy` permite usar el formato de notación de puntos para acceder a una clave más profunda con el fin de ordenar una matriz multidimensional.

```
$collection = collect([
    ["id"=>1,"product"=>['name' => 'Desk', 'price' => 200]],
    ["id"=>2, "product"=>['name' => 'Chair', 'price' => 100]],
    ["id"=>3, "product"=>['name' => 'Bookcase', 'price' => 150]],
    ]);

$sorted = $collection->sortBy("product.price")->toArray();

return: [
  ["id"=>2, "product"=>['name' => 'Chair', 'price' => 100]],
  ["id"=>3, "product"=>['name' => 'Bookcase', 'price' => 150]],
  ["id"=>1,"product"=>['name' => 'Desk', 'price' => 200]],
]
```

# SortByDesc ()

Este método tiene la misma firma que el método `sortBy` , pero ordenará la colección en el orden opuesto.

## Utilizando reducir ()

El método de `reduce` reduce la recopilación a un solo valor, pasando el resultado de cada iteración a la iteración posterior. Por favor vea el método de reducción .

El método de `reduce` recorre cada elemento con una colección y produce un nuevo resultado para la siguiente iteración. Cada resultado de la última iteración se pasa a través del primer parámetro (en los siguientes ejemplos, como `$carry` ).

Este método puede hacer mucho procesamiento en grandes conjuntos de datos. Por ejemplo, en los siguientes ejemplos, usaremos los siguientes datos de estudiantes de ejemplo:

```
 $student = [
    ['class' => 'Math', 'score' => 60],
    ['class' => 'English', 'score' => 61],
    ['class' => 'Chemistry', 'score' => 50],
    ['class' => 'Physics', 'score' => 49],
];
```

**Puntuación total del alumno**

```
$sum = collect($student)
    ->reduce(function($carry, $item){
        return $carry + $item["score"];
    }, 0);
```

Resultado: `220`

Explicación:

- `$carry` es el resultado de la última iteración.
- El segundo parámetro es el valor predeterminado para $ carry en la primera ronda de iteración. En este caso, el valor predeterminado es 0.

**Pasar un estudiante si todas sus calificaciones son> = 50**

```
$isPass = collect($student)
    ->reduce(function($carry, $item){
        return $carry && $item["score"] >= 50;
    }, true);
```

Resultado: `false`

Explicación:

- El valor predeterminado de $ carry es verdadero
- Si toda la puntuación es mayor que 50, el resultado devolverá verdadero; Si hay menos de 50, devuelve falso.

**Reprobar a un estudiante si cualquier puntaje es <50**

```
$isFail = collect($student)
    ->reduce(function($carry, $item){
        return $carry || $item["score"] < 50;
    }, false);
```

Resultado: `true`

Explique:

- el valor predeterminado de $ carry es falso
- si cualquier puntaje es menor a 50, devuelva verdadero; Si todas las puntuaciones son mayores que 50, devuelva falso.

**Devuelve el tema con la puntuación más alta.**

```
$highestSubject = collect($student)
    ->reduce(function($carry, $item){
        return $carry === null || $item["score"] > $carry["score"] ? $item : $carry;
    });
```

resultado: `[ "subject" => "English", "score" => 61 ]`

Explique:

- El segundo parámetro no se proporciona en este caso.
- El valor predeterminado de $ carry es nulo, por lo tanto, lo verificamos en nuestro condicional.

## Usando macro () para extender colecciones

La función `macro()` permite agregar una nueva funcionalidad a `Illuminate\Support\Collection` objetos

Uso:

```
Collection::macro("macro_name", function ($parameters) {
    // Your macro
});
```

Por ejemplo:

```
Collection::macro('uppercase', function () {
    return $this->map(function ($item) {
        return strtoupper($item);
    });
});

collect(["hello", "world"])->uppercase();
```

Resultado: `["HELLO", "WORLD"]`

## Usando Sintaxis de Array

El `Collection` objeto implementa la `ArrayAccess` y `IteratorAggregate` interfaz, lo que permite que sea utilizado como una matriz.

**Acceder al elemento de recogida:**

```
 $collection = collect([1, 2, 3]);
 $result = $collection[1];
```

Resultado: `2`

**Asignar nuevo elemento:**

```
$collection = collect([1, 2, 3]);
$collection[] = 4;
```

Resultado: `$collection` es `[1, 2, 3, 4]`

**Colección de bucles:**

```
$collection = collect(["a" => "one", "b" => "two"]);
$result = "";
foreach($collection as $key => $value){
    $result .= "(".$key.": ".$value.") ";
}
```

Resultado: `$result` es `(a: one) (b: two)`

**Array a la conversión de la colección:**

Para convertir una colección a una matriz nativa de PHP, use:

```
$array = $collection->all();
//or
$array = $collection->toArray()
```

Para convertir una matriz en una colección, use:

```
$collection = collect($array);
```

### Uso de colecciones con funciones de matriz

Tenga en cuenta que las colecciones son objetos normales que no se convertirán correctamente cuando sean utilizadas por funciones que requieren explícitamente arreglos, como `array_map($callback)` .

Asegúrese de convertir la colección primero o, si está disponible, use el método proporcionado por la clase `Collection` lugar: `$collection->map($callback)`

Lea Colecciones en línea: https://riptutorial.com/es/laravel/topic/2358/colecciones

# Capítulo 13: Conexiones DB múltiples en Laravel

## Examples

### Pasos iniciales

Se pueden definir múltiples conexiones de base de datos, de cualquier tipo, dentro del archivo de configuración de la base de datos (probable `app/config/database.php` ). Por ejemplo, para extraer datos de 2 bases de datos MySQL, defina ambos por separado:

```
<?php
return array(

    'default' => 'mysql',

    'connections' => array(

        # Our primary database connection
        'mysql' => array(
            'driver'    => 'mysql',
            'host'      => 'host1',
            'database'  => 'database1',
            'username'  => 'user1',
            'password'  => 'pass1'
            'charset'   => 'utf8',
            'collation' => 'utf8_unicode_ci',
            'prefix'    => '',
        ),

        # Our secondary database connection
        'mysql2' => array(
            'driver'    => 'mysql',
            'host'      => 'host2',
            'database'  => 'database2',
            'username'  => 'user2',
            'password'  => 'pass2'
            'charset'   => 'utf8',
            'collation' => 'utf8_unicode_ci',
            'prefix'    => '',
        ),
    ),
);
```

La conexión por defecto todavía se establece en `mysql` . Esto significa que, a menos que se especifique lo contrario, la aplicación utiliza la conexión `mysql` .

### Usando el constructor de esquemas

Dentro de Schema Builder, use la fachada de esquema con cualquier conexión. Ejecute el método `connection()` para especificar qué conexión usar:

```
Schema::connection('mysql2')->create('some_table', function($table)
{
    $table->increments('id');
});
```

## Usando el generador de consultas DB

Al igual que Schema Builder, define una conexión en el Query Builder:

```
$users = DB::connection('mysql2')->select(...);
```

## Usando Eloquent

Hay varias formas de definir qué conexión usar en los modelos Eloquent. Una forma es establecer la variable $ connection en el modelo:

```
<?php

class SomeModel extends Eloquent {

    protected $connection = 'mysql2';

}
```

La conexión también se puede definir en tiempo de ejecución a través del método `setConnection` .

```
<?php

class SomeController extends BaseController {

    public function someMethod()
    {
        $someModel = new SomeModel;

        $someModel->setConnection('mysql2');

        $something = $someModel->find(1);

        return $something;
    }
}
```

## De la documentación de Laravel.

Se puede acceder a cada conexión individual a través del método de conexión en la fachada `DB` , incluso cuando hay múltiples conexiones definidas. El `name` pasado al método de `connection` debe corresponder a una de las conexiones enumeradas en el archivo de configuración `config/database.php` :

```
$users = DB::connection('foo')->select(...);
```

También se puede acceder al raw, instancia de PDO subyacente utilizando el método getPdo en una instancia de conexión:

```
$pdo = DB::connection()->getPdo();
```

https://laravel.com/docs/5.4/database#using-multiple-database-connections

Lea Conexiones DB múltiples en Laravel en línea:
https://riptutorial.com/es/laravel/topic/9605/conexiones-db-multiples-en-laravel

# Capítulo 14: Constantes

## Examples

### Ejemplo

Primero debe crear un archivo constants.php y es una buena práctica crear este archivo dentro de app / config / folder. También puede agregar el archivo constants.php en el archivo compose.json.

Archivo de ejemplo:

> app / config / constants.php

Constantes basadas en matrices dentro del archivo:

```
return [
    'CONSTANT' => 'This is my first constant.'
];
```

Y puedes obtener esta constante incluyendo la `Config` fachada:

```
use Illuminate\Support\Facades\Config;
```

Luego obtenga el valor con el nombre constante `CONSTANT` como abajo:

```
echo Config::get('constants.CONSTANT');
```

Y el resultado sería el valor:

> Esta es mi primera constante.

Lea Constantes en línea: https://riptutorial.com/es/laravel/topic/9192/constantes

# Capítulo 15: Controladores

## Introducción

En lugar de definir toda la lógica de manejo de su solicitud como cierres en archivos de ruta, es posible que desee organizar este comportamiento usando clases de controlador. Los controladores pueden agrupar la lógica de manejo de solicitudes relacionadas en una sola clase. Los controladores se almacenan en el directorio `app/Http/Controllers` de forma predeterminada.

## Examples

### Controladores básicos

```
<?php

namespace App\Http\Controllers;

use App\User;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * Show the profile for the given user.
     *
     * @param  int  $id
     * @return Response
     */
    public function show($id)
    {
        return view('user.profile', ['user' => User::findOrFail($id)]);
    }
}
```

Puede definir una ruta a esta acción del controlador así:

```
Route::get('user/{id}', 'UserController@show');
```

Ahora, cuando una solicitud coincide con el URI de la ruta especificada, se ejecutará el método `show` en la clase `UserController` . Por supuesto, los parámetros de la ruta también se pasarán al método.

### Controlador Middleware

Middleware puede asignarse a las rutas del controlador en sus archivos de ruta:

```
Route::get('profile', 'UserController@show')->middleware('auth');
```

Sin embargo, es más conveniente especificar middleware dentro del constructor de su controlador. Usando el método de middleware del constructor de su controlador, puede asignar

fácilmente el middleware a la acción del controlador.

```
class UserController extends Controller
{
    /**
     * Instantiate a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');

        $this->middleware('log')->only('index');

        $this->middleware('subscribed')->except('store');
    }
}
```

### Controlador de recursos

El enrutamiento de recursos de Laravel asigna las rutas "CRUD" típicas a un controlador con una sola línea de código. Por ejemplo, es posible que desee crear un controlador que maneje todas las solicitudes HTTP de "fotos" almacenadas por su aplicación. Con el comando `make:controller` Artisan, podemos crear rápidamente dicho controlador:

```
php artisan make:controller PhotoController --resource
```

Este comando generará un controlador en `app/Http/Controllers/PhotoController.php` . El controlador contendrá un método para cada una de las operaciones de recursos disponibles.

## Ejemplo de cómo se ve un Controlador de Recursos

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PhotoController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
    }

    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
```

```
     */
    public function create()
    {
        //
    }

    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        //
    }

    /**
     * Display the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
    }

    /**
     * Update the specified resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }

    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy($id)
    {
        //
    }
```

```
}
```

El ejemplo del controlador de recursos comparte el nombre del método de aquellos en la tabla a continuación.

A continuación, puede registrar una ruta con recursos para el controlador:

```
Route::resource('photos', 'PhotoController');
```

Esta declaración de ruta única crea múltiples rutas para manejar una variedad de acciones en el recurso. El controlador generado ya tendrá métodos de apéndice para cada una de estas acciones, incluidas las notas que le informarán sobre los verbos HTTP y los URI que manejan.

## Acciones manejadas por el controlador de recursos

| Verbo | URI | Acción | Nombre de la ruta |
|---|---|---|---|
| OBTENER | `/photos` | índice | fotos.index |
| OBTENER | `/photos/create` | crear | fotos.crear |
| ENVIAR | `/photos` | almacenar | fotos.tienda |
| OBTENER | `/photos/{photo}` | espectáculo | fotos.muestra |
| OBTENER | `/photos/{photo}/edit` | editar | fotos.editar |
| PUT / PATCH | `/photos/{photo}` | actualizar | photos.update |
| BORRAR | `/photos/{photo}` | destruir | fotos.destroy |

Lea Controladores en línea: https://riptutorial.com/es/laravel/topic/10604/controladores

# Capítulo 16: Correo

## Examples

### Ejemplo basico

Puede configurar Mail simplemente agregando / cambiando estas líneas en el archivo **.ENV de** la aplicación con los detalles de inicio de sesión de su proveedor de correo electrónico, por ejemplo, para usarlo con gmail puede usar:

```
MAIL_DRIVER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=587
MAIL_USERNAME=yourEmail@gmail.com
MAIL_PASSWORD=yourPassword
MAIL_ENCRYPTION=tls
```

Luego puedes comenzar a enviar correos electrónicos usando Mail, por ejemplo:

```
$variable = 'Hello world!'; // A variable which can be use inside email blade template.
Mail::send('your.blade.file', ['variable' => $variable], function ($message) {
            $message->from('john@doe.com');
            $message->sender('john@doe.com');
            $message->to(foo@bar.com);
            $message->subject('Hello World');
        });
```

Lea Correo en línea: https://riptutorial.com/es/laravel/topic/8014/correo

# Capítulo 17: Eliminar público de la URL en laravel

## Introducción

Cómo eliminar el `public` de la URL en Laravel, hay muchas respuestas en Internet, pero la manera más fácil se describe a continuación.

## Examples

### ¿Como hacer eso?

Siga estos pasos para eliminar `public` de la url

1. Copie el archivo `.htaccess` del directorio `/public` a la carpeta raíz de `Laravel/project` .
2. Cambie el nombre de `server.php` en la carpeta raíz de `Laravel/project` a `index.php` .

Saludos, estarás bien ahora.

Tenga en cuenta: Se prueba en *Laravel 4.2* , *Laravel 5.1* , *Laravel 5.2* , *Laravel 5.3* .

Creo que esta es la forma más fácil de eliminar `public` de la url.

### Retirar el público de url

1. Renombrando el `server.php` a `index.php`
2. Copie el archivo `.htaccess` de `public` carpeta `public` a `root` carpeta `root`
3. Cambiando `.htaccess` un poco como sigue para statics:

```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^(.*)/$ /$1 [L,R=301]

RewriteCond %{REQUEST_URI} !(\.css|\.js|\.png|\.jpg|\.gif|robots\.txt)$ [NC]
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [L]

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_URI} !^/public/
RewriteRule ^(css|js|images)/(.*)$ public/$1/$2 [L,NC]
```

A veces he usado este método para eliminar url de formulario `public` .

Lea Eliminar público de la URL en laravel en línea:
https://riptutorial.com/es/laravel/topic/9786/eliminar-publico-de-la-url-en-laravel

# Capítulo 18: Elocuente

## Introducción

El Elocuente es un ORM (Objeto Modelo Relacional) incluido con el Laravel. Implementa el patrón de registro activo y se utiliza para interactuar con bases de datos relacionales.

## Observaciones

### Nombre de la tabla

La convención es utilizar "snake_case" pluralizado para nombres de tablas y "StudlyCase" singular para nombres de modelos. Por ejemplo:

- Una mesa de `cats` tendría un modelo de `Cat`
- Una mesa de `jungle_cats` tendría un modelo de `JungleCat`
- Una tabla de `users` tendría un modelo de `User`
- Una mesa de `people` tendría un modelo de `Person`

Eloquent intentará automáticamente vincular su modelo con una tabla que tenga el nombre en plural del modelo, como se indicó anteriormente.

Sin embargo, puede especificar un nombre de tabla para anular la convención predeterminada.

```
class User extends Model
{
    protected $table = 'customers';
}
```

## Examples

### Introducción

Elocuente es el ORM integrado en el marco de Laravel. Le permite interactuar con las tablas de su base de datos de una manera orientada a objetos, mediante el uso del patrón ActiveRecord .

Una clase de modelo único generalmente se asigna a una tabla de base de datos única, y también se pueden definir relaciones de diferentes tipos ( uno a uno , uno a muchos , muchos a muchos , polimórficos) entre diferentes clases de modelos.

La sección Making a Model describe la creación y definición de clases de modelos.

Antes de que pueda comenzar a usar los modelos Eloquent, asegúrese de que al menos una conexión de base de datos se haya configurado en su archivo de configuración `config/database.php` .

Para comprender el uso del generador de consultas elocuente durante el desarrollo, puede usar `php artisan ide-helper:generate` comando. Aquí está el enlace .

---

# Subtema de navegación

Relación elocuente

## Persistiendo

Además de leer datos con Eloquent, también puede usarlo para insertar o actualizar datos con el método `save()` . Si ha creado una nueva instancia de modelo, se *insertará* el registro; de lo contrario, si ha recuperado un modelo de la base de datos y ha establecido nuevos valores, se *actualizará* .

En este ejemplo creamos un nuevo registro de `User` :

```
$user = new User();
$user->first_name = 'John';
$user->last_name = 'Doe';
$user->email = 'john.doe@example.com';
$user->password = bcrypt('my_password');
$user->save();
```

También puede utilizar el método de `create` para rellenar campos utilizando una matriz de datos:

```
User::create([
    'first_name'=> 'John',
    'last_name' => 'Doe',
    'email'     => 'john.doe@example.com',
    'password'  => bcrypt('changeme'),
]);
```

Cuando utilice el método de creación, sus atributos deben declararse en la matriz `fillable` dentro de su modelo:

```
class User extends Model
{
    protected $fillable = [
         'first_name',
         'last_name',
         'email',
         'password',
    ];
}
```

Alternativamente, si desea que todos los atributos sean asignables en masa, puede definir la propiedad $ guarded como una matriz vacía:

```
class User extends Model
{
```

```
    /**
    * The attributes that aren't mass assignable.
    *
    * @var array
    */
    protected $guarded = [];
}
```

Pero también puede crear un registro sin siquiera cambiar el atributo `fillable` en su modelo usando el método `forceCreate` lugar de `create` método

```
User::forceCreate([
    'first_name'=> 'John',
    'last_name' => 'Doe',
    'email'     => 'john.doe@example.com',
    'password'  => bcrypt('changeme'),
]);
```

El siguiente es un ejemplo de la actualización de un modelo de `User` existente al cargarlo primero (mediante la `find` ), modificarlo y luego guardarlo:

```
$user = User::find(1);
$user->password = bcrypt('my_new_password');
$user->save();
```

Para lograr la misma hazaña con una sola llamada de función, puede usar el método de `update` :

```
$user->update([
    'password' => bcrypt('my_new_password'),
]);
```

Los métodos de `create` y `update` hacen que trabajar con grandes conjuntos de datos sea mucho más sencillo que tener que configurar cada par de clave / valor individualmente, como se muestra en los siguientes ejemplos:

> Tenga en cuenta el uso de `only` y `except` al recopilar datos de solicitud. Es importante que especifique las claves exactas que desea permitir / no permitir que se actualicen, de lo contrario, es posible que un atacante envíe campos adicionales con su solicitud y cause actualizaciones no deseadas.

```
// Updating a user from specific request data
$data = Request::only(['first_name', 'email']);
$user->find(1);
$user->update($data);

// Create a user from specific request data
$data = Request::except(['_token', 'profile_picture', 'profile_name']);
$user->create($data);
```

## Borrando

Puede eliminar datos después de escribirlos en la base de datos. Puede eliminar una instancia de

modelo si ha recuperado una, o especificar las condiciones para los registros que desea eliminar.

Para eliminar una instancia de modelo, recupérela y llame al método `delete()` :

```
$user = User::find(1);
$user->delete();
```

Alternativamente, puede especificar una clave primaria (o una matriz de claves primarias) de los registros que desea eliminar a través del método `destroy()` :

```
User::destroy(1);
User::destroy([1, 2, 3]);
```

También puede combinar la consulta con la eliminación:

```
User::where('age', '<', 21)->delete();
```

Esto eliminará todos los usuarios que coincidan con la condición.

> Nota: Al ejecutar una declaración de eliminación masiva a través de Eloquent, los eventos de `deleting` y `deleted` modelo no se activarán para los modelos eliminados. Esto se debe a que los modelos nunca se recuperan realmente al ejecutar la instrucción de eliminación.

## Eliminación suave

Algunas veces, no desea eliminar un registro de forma permanente, sino mantenerlo para fines de auditoría o informe. Para esto, Eloquent proporciona *una* funcionalidad de *eliminación suave* .

Para agregar la funcionalidad de borrados suaves a su modelo, necesita importar el rasgo `SoftDeletes` y agregarlo a su clase de modelo Eloquent:

```
namespace Illuminate\Database\Eloquent\Model;
namespace Illuminate\Database\Eloquent\SoftDeletes;

class User extends Model
{
     use SoftDeletes;
}
```

Al eliminar un modelo, establecerá una marca de tiempo en una columna de marca de tiempo `deleted_at` en la tabla para su modelo, así que asegúrese de crear la columna `deleted_at` en su tabla primero. O en la migración, debe llamar `softDeletes()` método `softDeletes()` en su proyecto para agregar la `deleted_at` tiempo `deleted_at` . Ejemplo:

```
Schema::table('users', function ($table) {
    $table->softDeletes();
});
```

Cualquier consulta omitirá los registros borrados. Puede forzarlos, mostrarlos si lo desea, usando el ámbito `withTrashed()` :

```
User::withTrashed()->get();
```

Si desea permitir que los usuarios *restauren* un registro después de la eliminación suave (es decir, en un área de tipo de papelera), puede usar el método `restore()` :

```
$user = User::find(1);
$user->delete();
$user->restore();
```

Para eliminar un registro a la `forceDelete()` use el método `forceDelete()` que realmente eliminará el registro de la base de datos:

```
$user = User::find(1);
$user->forceDelete();
```

## Cambiar clave principal y marcas de tiempo

De manera predeterminada, los modelos Eloquent esperan que la clave primaria se llame `'id'` . Si ese no es su caso, puede cambiar el nombre de su clave principal especificando la propiedad `$primaryKey` .

```
class Citizen extends Model
{
    protected $primaryKey = 'socialSecurityNo';

    // ...
}
```

Ahora, cualquier método Eloquent que use su clave principal (por ejemplo, `find` o `findOrFail` ) usará este nuevo nombre.

Además, Eloquent espera que la clave principal sea un entero de incremento automático. Si su clave principal no es un entero de incremento automático (por ejemplo, un GUID), debe indicar a Eloquent actualizando la propiedad `$incrementing` a `false` :

```
class Citizen extends Model
{
    protected $primaryKey = 'socialSecurityNo';

    public $incrementing = false;

    // ...
}
```

De forma predeterminada, Eloquent espera que `updated_at` columnas `created_at` y `updated_at` en sus tablas. Si no desea que Eloquent administre estas columnas automáticamente, establezca la propiedad `$timestamps` en su modelo en false:

```
class Citizen extends Model
{
    public $timestamps = false;

    // ...
}
```

Si necesita personalizar los nombres de las columnas utilizadas para almacenar las marcas de tiempo, puede configurar las constantes `CREATED_AT` y `UPDATED_AT` en su modelo:

```
class Citizen extends Model
{
    const CREATED_AT = 'date_of_creation';
    const UPDATED_AT = 'date_of_last_update';

    // ...
}
```

## Lanzar 404 si no se encuentra la entidad

Si desea lanzar automáticamente una excepción al buscar un registro que no se encuentra en un modal, puede usar

```
Vehicle::findOrFail(1);
```

o

```
Vehicle::where('make', 'ford')->firstOrFail();
```

Si no se encuentra un registro con la clave principal de `1` , se lanza una `ModelNotFoundException` . Que es esencialmente lo mismo que escribir ( ver fuente ):

```
$vehicle = Vehicle::find($id);

if (!$vehicle) {
    abort(404);
}
```

## Modelos de clonación

Puede que necesite clonar una fila, tal vez cambiar algunos atributos, pero necesita una manera eficiente de mantener las cosas en SECO. Laravel proporciona una especie de método 'oculto' para permitirte hacer esta funcionalidad. Aunque está completamente sin documentar, debe buscar a través de la API para encontrarlo.

Usando `$model->replicate()` puedes clonar fácilmente un registro

```
$robot = Robot::find(1);
$cloneRobot = $robot->replicate();
// You can add custom attributes here, for example he may want to evolve with an extra arm!
$cloneRobot->arms += 1;
```

```
$cloneRobot->save();
```

Lo anterior encontraría un robot que tiene un ID de 1 y luego lo clonará.

Lea Elocuente en línea: https://riptutorial.com/es/laravel/topic/865/elocuente

# Capítulo 19: Elocuente: Modelo

## Examples

### Haciendo un modelo

### Creación de modelos

Las clases de modelo deben extender `Illuminate\Database\Eloquent\Model` . La ubicación predeterminada para los modelos es el directorio `/app` .

El comando Artesano puede generar fácilmente una clase modelo:

```
php artisan make:model [ModelName]
```

Esto creará un nuevo archivo PHP en la `app/` por defecto, que se llamará `[ModelName].php` , y contendrá todo el texto de su nuevo modelo, que incluye la clase, el espacio de nombres y los usos necesarios para una configuración básica.

Si desea crear un archivo de migración junto con su Modelo, use el siguiente comando, donde `-m` también generará el archivo de migración:

```
php artisan make:model [ModelName] -m
```

Además de crear el modelo, esto crea una migración de base de datos que está conectada al modelo. El archivo PHP de migración de la base de datos se encuentra por defecto en la `database/migrations/` de `database/migrations/` . De manera predeterminada, esto no incluye nada más que las columnas id y `created_at` / `updated_at` , por lo que deberá editar el archivo para proporcionar columnas adicionales.

Tenga en cuenta que tendrá que ejecutar la migración (una vez que haya configurado el archivo de migración) para que el modelo comience a funcionar utilizando `php artisan migrate` desde la raíz del proyecto

Además, si desea agregar una migración más adelante, después de hacer el modelo, puede hacerlo ejecutando:

```
php artisan make:migration [migration name]
```

---

Digamos, por ejemplo, que quería crear un modelo para sus gatos, tendría dos opciones, crear con o sin migración. Elegiría crear sin migración si ya tenía una tabla de gatos o no quería crear una en este momento.

Para este ejemplo, queremos crear una migración porque todavía no tenemos una tabla, así que ejecutaríamos el siguiente comando.

```
php artisan make:model Cat -m
```

Este comando creará dos archivos:

1. En la carpeta de `app/Cat.php` : `app/Cat.php`
2. En la carpeta de la base de datos: `database/migrations/timestamp_creat_cats_table.php`

El archivo en el que estamos interesados es el último, ya que es este archivo en el que podemos decidir cómo queremos que se vea e incluya la tabla. Para cualquier migración predefinida se nos da una columna de ID de incremento automático y columnas de marcas de tiempo.

El siguiente ejemplo de un extracto del archivo de migración incluye las columnas predefinidas anteriores, así como la adición del nombre del gato, la edad y el color:

```
public function up()
    {
        Schema::create('cats', function (Blueprint $table) {

            $table->increments('id');     //Predefined ID
            $table->string('name');       //Name
            $table->integer('age');       //Age
            $table->string('colour');     //Colour
            $table->timestamps();         //Predefined Timestamps

        });
    }
```

Como puede ver, es relativamente fácil crear el modelo y la migración para una tabla. Luego, para ejecutar la migración y crearla en su base de datos, ejecutará el siguiente comando:

```
php artisan migrate
```

Lo que migrará cualquier migración pendiente a su base de datos.

---

## Ubicación del archivo de modelo

Los modelos se pueden almacenar en cualquier lugar gracias a PSR4 .

Por defecto, los modelos se crean en el directorio de la `app` con el espacio de nombres de la `App` . Para aplicaciones más complejas, generalmente se recomienda almacenar los modelos dentro de sus propias carpetas en una estructura que tenga sentido para la arquitectura de sus aplicaciones.

Por ejemplo, si tenía una aplicación que utilizaba una serie de frutas como modelos, podría crear una carpeta llamada `app/Fruits` y dentro de esta carpeta, crea `Banana.php` (manteniendo la convención de nombres de StudlyCase ), luego puede crear la clase Banana. en el espacio de nombres de `App\Fruits` :

```
namespace App\Fruits;
```

```
use Illuminate\Database\Eloquent\Model;

class Banana extends Model {
    // Implementation of "Banana" omitted
}
```

## Configuración del modelo

Elocuente sigue un enfoque de "convención sobre configuración". Al ampliar la clase de `Model` base, todos los modelos heredan las propiedades que se enumeran a continuación. A menos que se invalide, se aplican los siguientes valores predeterminados:

| Propiedad | Descripción | Defecto |
|---|---|---|
| `protected $connection` | Nombre de conexión DB | Conexión DB predeterminada |
| `protected $table` | Nombre de la tabla | De forma predeterminada, el nombre de la clase se convierte a `snake_case` y se pluraliza. Por ejemplo, `SpecialPerson` convierte en `special_people` |
| `protected $primaryKey` | Tabla PK | `id` |
| `public $incrementing` | Indica si las ID son auto-incrementales | `true` |
| `public $timestamps` | Indica si el modelo debe tener la marca de tiempo. | `true` |
| `const CREATED_AT` | Nombre de la columna de marca de tiempo de creación | `created_at` |
| `const UPDATED_AT` | Nombre de la columna de marca de tiempo de modificación | `updated_at` |
| `protected $dates` | Atributos que deben mutarse a DateTime, además de los atributos de marca de tiempo | `[]` |
| `protected $dateFormat` | Formato en el que se conservarán los atributos de fecha. | Por defecto para el dialecto SQL actual. |
| `protected $with` | Relaciones a eagerload con modelo. | `[]` |
| `protected $hidden` | Atributos omitidos en la serialización del modelo | `[]` |

| Propiedad | Descripción | Defecto |
|---|---|---|
| `protected $visible` | Atributos permitidos en la serialización del modelo. | `[]` |
| `protected $appends` | Atributos de acceso añadido a la serialización del modelo. | `[]` |
| `protected $fillable` | Atributos que son asignables en masa | `[]` |
| `protected $guarded` | Atributos que están en la lista negra de la asignación en masa | `[*]` (Todos los atributos) |
| `protected $touches` | Las relaciones que se deben tocar en guardar. | `[]` |
| `protected $perPage` | El número de modelos a devolver por paginación. | `15` |

5.0

| Propiedad | Descripción | Defecto |
|---|---|---|
| `protected $casts` | Atributos que deben ser convertidos a tipos nativos | `[]` |

## Actualizar un modelo existente

```
$user = User::find(1);
$user->name = 'abc';
$user->save();
```

También puede actualizar varios atributos a la vez usando la `update` , que no requiere usar `save` después:

```
$user = User::find(1);
$user->update(['name' => 'abc', 'location' => 'xyz']);
```

También puede actualizar un modelo (s) sin consultarlo de antemano:

```
User::where('id', '>', 2)->update(['location' => 'xyz']);
```

Si no desea activar un cambio en la marca de tiempo `updated_at` en el modelo, puede pasar la opción `touch` :

```
$user = User::find(1);
$user->update(['name' => 'abc', 'location' => 'xyz'], ['touch' => false]);
```

Lea Elocuente: Modelo en línea: https://riptutorial.com/es/laravel/topic/7984/elocuente--modelo

# Capítulo 20: Elocuente: Relación

## Examples

### Consultar sobre las relaciones

Eloquent también le permite consultar relaciones definidas, como se muestra a continuación:

```
User::whereHas('articles', function (Builder $query) {
    $query->where('published', '!=', true);
})->get();
```

Esto requiere que el nombre de su método de relación sean `articles` en este caso. El argumento que se pasa al cierre es el Generador de consultas para el modelo relacionado, por lo que puede usar cualquier consulta aquí que pueda usar en otro lugar.

**Eager Loading**

Supongamos que el modelo de usuario tiene una relación con el modelo de artículo y desea cargar con entusiasmo los artículos relacionados. Esto significa que los artículos del usuario se cargarán mientras se recupera el usuario.

`articles` es el nombre de la relación (método) en el modelo de usuario.

```
User::with('articles')->get();
```

Si tienes relaciones multiples. por ejemplo artículos y publicaciones.

```
User::with('articles','posts')->get();
```

y para seleccionar relaciones anidadas.

```
User::with('posts.comments')->get();
```

Llame a más de una relación anidada

```
User::with('posts.comments.likes')->get()
```

### Insertando Modelos Relacionados

Supongamos que tiene un modelo de `Post` con una relación `hasMany` con `Comment` . Puede insertar un objeto de `Comment` relacionado con una publicación haciendo lo siguiente:

```
$post = Post::find(1);

$commentToAdd = new Comment(['message' => 'This is a comment.']);
```

```
$post->comments()->save($commentToAdd);
```

Puedes guardar varios modelos a la vez usando la función `saveMany` :

```
$post = Post::find(1);

$post->comments()->saveMany([
    new Comment(['message' => 'This a new comment']),
    new Comment(['message' => 'Me too!']),
    new Comment(['message' => 'Eloquent is awesome!'])
]);
```

Alternativamente, también hay un método de `create` que acepta una matriz PHP simple en lugar de una instancia de modelo Eloquent.

```
$post = Post::find(1);

$post->comments()->create([
    'message' => 'This is a new comment message'
]);
```

### Introducción

Las relaciones elocuentes se definen como funciones en sus clases de modelo Eloquent. Dado que, al igual que los modelos Eloquent, las relaciones también sirven como poderosos constructores de consultas, la definición de relaciones como funciones proporciona potentes capacidades de encadenamiento y consulta de métodos. Por ejemplo, podemos encadenar restricciones adicionales en esta relación de publicaciones:

```
$user->posts()->where('active', 1)->get();
```

Navegar al tema principal

### Tipos de relacion

## Uno a muchos

Digamos que cada publicación puede tener uno o varios comentarios y cada comentario pertenece a una sola publicación.

así que la tabla de comentarios tendrá `post_id` . En este caso las relaciones serán las siguientes.

**Modelo de publicación**

```
public function comments()
{
   return $this->belongsTo(Post::class);
}
```

Si la clave externa es distinta de `post_id` , por ejemplo, la clave externa es `example_post_id` .

```
public function comments()
{
   return $this->belongsTo(Post::class, 'example_post_id');
}
```

y además, si la clave local es distinta de `id` , por ejemplo, la clave local es `other_id`

```
public function comments()
{
   return $this->belongsTo(Post::class, 'example_post_id', 'other_id');
}
```

**Modelo de comentario**

definiendo inverso de uno a muchos

```
public function post()
{
   return $this->hasMany(Comment::class);
}
```

# Doce y cincuenta y nueve de la noche

# Cómo asociar entre dos modelos (ejemplo: modelo de `User` y `Phone` )

`App\User`

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the phone record associated with the user.
     */
    public function phone()
    {
        return $this->hasOne('Phone::class', 'foreign_key', 'local_key');
    }
}
```

`App\Phone`

```
<?php

namespace App;
```

```
use Illuminate\Database\Eloquent\Model;

class Phone extends Model
{
    /**
     * Get the user that owns the phone.
     */
    public function user()
    {
        return $this->belongsTo('User::class', 'foreign_key', 'local_key');
    }
}
```

`foreign_key` : de forma predeterminada, Eloquent asumirá que este valor es `other_model_name_id` (en este caso, `user_id` y `phone_id` ), cámbielo si no es así.

`local_key` : de forma predeterminada, Eloquent asumirá que este valor es `id` (clave principal del modelo actual); cámbielo si no es así.

> Si su base de datos presentó el nombre según la norma de laravel, no necesita proporcionar una clave externa y una clave local en la declaración de relación

### Explicación

# Muchos a muchos

Digamos que hay roles y permisos. Cada rol puede pertenecer a muchos permisos y cada permiso puede pertenecer a muchos roles. así que habrá 3 mesas. Dos modelos y una mesa pivotante. Una tabla de `roles` , `users` y `permission_role` .

**Modelo a seguir**

```
public function permissions()
{
   return $this->belongsToMany(Permission::class);
}
```

**Modelo de permiso**

```
public function roles()
{
   return $this->belongsToMany(Roles::class);
}
```

***Nota 1***

*Considere lo siguiente al usar un nombre de tabla diferente para la tabla dinámica.*

Supongamos que desea usar `role_permission` lugar de `permission_role` , ya que elocuente usa orden alfabético para construir los nombres de las claves dinámicas. Tendrá que pasar el nombre

de la tabla dinámica como segundo parámetro de la siguiente manera.

**Modelo a seguir**

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission');
}
```

**Modelo de permiso**

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission');
}
```

***Nota 2***

*Considere lo siguiente al usar diferentes nombres de clave en la tabla dinámica.*

Eloquent asume que si no se pasan claves como tercer y cuarto parámetros, serán los nombres de tabla singulares con `_id` . por lo que se supone que el pivote tendrá campos `role_id` y `permission_id` . Si se van a utilizar otras claves, deben pasarse como parámetros tercero y cuarto.

Digamos si se va a `other_role_id` lugar de `role_id` y `other_permission_id` lugar de `permission_id` . Así sería como sigue.

**Modelo a seguir**

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission', 'other_role_id',
'other_permission_id');
}
```

**Modelo de permiso**

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission', 'other_permission_id',
'other_role_id');
}
```

# Polimórfico

Las relaciones polimórficas permiten que un modelo pertenezca a más de otro modelo en una sola asociación. Un buen ejemplo sería imágenes, tanto un usuario como un producto pueden tener una imagen. La estructura de la tabla puede verse como sigue:

```
user
    id - integer
```

```
    name - string
    email - string

product
    id - integer
    title - string
    SKU - string

image
    id - integer
    url - string
    imageable_id - integer
    imageable_type - string
```

Las columnas importantes a mirar están en la tabla de imágenes. La columna `imageable_id` contendrá el valor de ID del usuario o producto, mientras que la columna `imageable_type` contendrá el nombre de clase del modelo propietario. En tus modelos, configuras las relaciones de la siguiente manera:

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Image extends Model
{
    /**
     * Get all of the owning imageable models.
     */
    public function imageable()
    {
        return $this->morphTo();
    }
}

class User extends Model
{
    /**
     * Get all of the user's images.
     */
    public function images()
    {
        return $this->morphMany('Image::class', 'imageable');
    }
}

class Product extends Model
{
    /**
     * Get all of the product's images.
     */
    public function images()
    {
        return $this->morphMany('Image::class', 'imageable');
    }
}
```

También puede recuperar el propietario de una relación polimórfica del modelo polimórfico accediendo al nombre del método que realiza la llamada a `morphTo` . En nuestro caso, ese es el método `imageable` en el modelo de imagen. Entonces, accederemos a ese método como una propiedad dinámica.

```
$image = App\Image::find(1);

$imageable = $image->imageable;
```

Esta `imageable` volverá ya sea un usuario o un producto.

## Muchos a muchos

Digamos que hay roles y permisos. Cada rol puede pertenecer a muchos permisos y cada permiso puede pertenecer a muchos roles. así que habrá 3 mesas. Dos modelos y una mesa pivotante. Una tabla de `roles` , `users` y `permission_role` .

### Modelo a seguir

```
public function permissions()
{
   return $this->belongsToMany(Permission::class);
}
```

### Modelo de permiso

```
public function roles()
{
   return $this->belongsToMany(Roles::class);
}
```

### *Nota 1*

*Considere lo siguiente al usar un nombre de tabla diferente para la tabla dinámica.*

Supongamos que desea usar `role_permission` lugar de `permission_role` , ya que elocuente usa orden alfabético para construir los nombres de las claves dinámicas. Tendrá que pasar el nombre de la tabla dinámica como segundo parámetro de la siguiente manera.

### Modelo a seguir

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission');
}
```

### Modelo de permiso

```
public function roles()
{
```

```
   return $this->belongsToMany(Roles::class, 'role_permission');
}
```

***Nota 2***

*Considere lo siguiente al usar diferentes nombres de clave en la tabla dinámica.*

Eloquent asume que si no se pasan claves como tercer y cuarto parámetros, serán los nombres de tabla singulares con `_id` . por lo que se supone que el pivote tendrá campos `role_id` y `permission_id` . Si se van a utilizar otras claves, deben pasarse como parámetros tercero y cuarto.

Digamos si se va a `other_role_id` lugar de `role_id` y `other_permission_id` lugar de `permission_id` . Así sería como sigue.

**Modelo a seguir**

```
public function permissions()
{
   return $this->belongsToMany(Permission::class, 'role_permission', 'other_role_id',
'other_permission_id');
}
```

**Modelo de permiso**

```
public function roles()
{
   return $this->belongsToMany(Roles::class, 'role_permission', 'other_permission_id',
'other_role_id');
}
```

**Acceso a la tabla intermedia utilizando withPivot ()**

Supongamos que tiene una tercera columna ' **permission_assigned_date** ' en la tabla dinámica. Por defecto, solo las claves del modelo estarán presentes en el objeto pivote. Ahora para obtener esta columna en el resultado de la consulta, debe agregar el nombre en la función Pivot ().

```
   public function permissions()
        {
           return $this->belongsToMany(Permission::class, 'role_permission', 'other_role_id',
'other_permission_id')->withPivot('permission_assigned_date');
        }
```

**Fijación / Desmontaje**

Eloquent también proporciona algunos métodos de ayuda adicionales para que trabajar con modelos relacionados sea más conveniente. Por ejemplo, imaginemos que un usuario puede tener muchos roles y un rol puede tener muchos permisos. Para adjuntar un rol a un permiso insertando un registro en la tabla intermedia que une los modelos, use el método de adjuntar:

```
$role= App\Role::find(1);
$role->permissions()->attach($permissionId);
```

Al vincular una relación a un modelo, también puede pasar una matriz de datos adicionales para insertarlos en la tabla intermedia:

```
$rol->roles()->attach($permissionId, ['permission_assigned_date' => $date]);
```

Del mismo modo, para eliminar un permiso específico contra una función, use la función de desconexión

```
$role= App\Role::find(1);
//will remove permission 1,2,3 against role 1
$role->permissions()->detach([1, 2, 3]);
```

**Asociaciones de sincronización**

También puede usar el método de sincronización para construir asociaciones de muchos a muchos. El método de sincronización acepta una matriz de ID para colocar en la tabla intermedia. Cualquier ID que no esté en la matriz dada se eliminará de la tabla intermedia. Entonces, una vez que se complete esta operación, solo existirán los ID en la matriz dada en la tabla intermedia:

```
//will keep permission id's 1,2,3 against Role id 1

$role= App\Role::find(1)
$role->permissions()->sync([1, 2, 3]);
```

Lea Elocuente: Relación en línea: https://riptutorial.com/es/laravel/topic/7960/elocuente--relacion

# Capítulo 21: Eloquent: Accessors & Mutators

## Introducción

Los accesores y los mutadores le permiten dar formato a los valores de atributo Eloquent cuando los recupera o configura en instancias de modelo. Por ejemplo, es posible que desee utilizar el cifrado Laravel para cifrar un valor mientras está almacenado en la base de datos, y luego descifrar automáticamente el atributo cuando accede a él en un modelo de Eloquent. Además de los accesores y mutadores personalizados, Eloquent también puede convertir automáticamente campos de fecha en instancias de Carbon o incluso campos de texto de conversión en JSON.

## Sintaxis

- establecer {ATTRIBUTE} Atributo ($ atributo) // en caso de camello

## Examples

### Definiendo un accessors

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class User extends Model
{
    /**
     * Get the user's first name.
     *
     * @param  string  $value
     * @return string
     */
    public function getFirstNameAttribute($value)
    {
        return ucfirst($value);
    }
}
```

# Obteniendo Accessor:

Como puede ver, el valor original de la columna se pasa al descriptor de acceso, lo que le permite manipular y devolver el valor. Para acceder al valor del elemento de acceso, simplemente puede acceder al atributo `first_name` en una instancia del modelo:

```
$user = App\User::find(1);
$firstName = $user->first_name;
```

## Definiendo un mutador

```
class User extends Model
{
    public function setPasswordAttribute($password)
    {
        $this->attributes['password'] = bcrypt($password);
    }
    ...
}
```

El código anterior hace "bcrypting" cada vez que se establece la propiedad de contraseña.

```
$user = $users->first();
$user->password = 'white rabbit'; //laravel calls mutator on background
$user->save(); // password is bcrypted and one does not need to call bcrypt('white rabbit')
```

Lea Eloquent: Accessors & Mutators en línea:
https://riptutorial.com/es/laravel/topic/8305/eloquent--accessors--amp--mutators

# Capítulo 22: Empezando con laravel-5.3

## Observaciones

Esta sección proporciona una descripción general de qué es laravel-5.3 y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema importante dentro de laravel-5.3 y vincular a los temas relacionados. Dado que la Documentación para laravel-5.3 es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

## Examples

### Instalando Laravel

**Requisitos:**
Necesita `PHP >= 5.6.4` y `Composer` instalado en su máquina. Puedes verificar la versión de ambos usando el comando:
**Para PHP:**

```
php -v
```

Salida como esta:

```
PHP 7.0.9 (cli) (built: Aug 26 2016 06:17:04) ( NTS )
Copyright (c) 1997-2016 The PHP Group
Zend Engine v3.0.0, Copyright (c) 1998-2016 Zend Technologies
```

**Para compositor**
Puede ejecutar el comando en su terminal / CMD:

```
composer --version
```

Salida como esta:

```
composer version 1.2.1 2016-09-12 11:27:19
```

Laravel utiliza Composer para gestionar sus dependencias. Por lo tanto, antes de usar Laravel, asegúrese de tener Composer instalado en su máquina.

## Via Laravel Installer

Primero, descargue el instalador de Laravel usando Composer:

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

Asegúrese de colocar el directorio `$HOME/.composer/vendor/bin` (o el directorio equivalente para su sistema operativo) en su $ PATH para que el `laravel` pueda `laravel` ejecutable `laravel` .

Una vez instalado, el `laravel new` comando `laravel new` creará una nueva instalación de Laravel en el directorio que especifique. Por ejemplo, `laravel new blog` creará un directorio llamado `blog` contiene una nueva instalación de Laravel con todas las dependencias de Laravel ya instaladas:

```
laravel new blog
```

# Via Composer Create-Project

Alternativamente, también puede instalar Laravel emitiendo el comando Composer `create-project` en su terminal:

```
composer create-project --prefer-dist laravel/laravel blog
```

# Preparar

Una vez que haya completado la instalación de Laravel, deberá configurar los `permissions` para las carpetas de almacenamiento y Bootstrap.

> **Nota: la** configuración de `permissions` es uno de los procesos más importantes que se deben completar al instalar Laravel.

**Servidor de desarrollo local**

Si tiene PHP instalado localmente y le gustaría usar el servidor de desarrollo incorporado de PHP para servir su aplicación, puede usar el comando `serve` Artisan. Este comando iniciará un servidor de desarrollo en `http://localhost:8000` :

```
php artisan serve
```

Abra la url de solicitud de su navegador `http://localhost:8000`

## Requisitos del servidor

El marco de Laravel tiene algunos requisitos del sistema. Por supuesto, todos estos requisitos son cumplidos por la máquina virtual Laravel Homestead , por lo que es altamente recomendable que utilice Homestead como su entorno de desarrollo local de Laravel.

Sin embargo, si no está utilizando Homestead, deberá asegurarse de que su servidor cumpla con los siguientes requisitos:

- PHP> = 5.6.4
- Extensión PHP OpenSSL

- Extensión PHP de DOP
- Mbstring PHP Extension
- Tokenizer PHP Extension
- Extensión PHP PHP

## Servidor de desarrollo local

Si tiene PHP instalado localmente y le gustaría usar el servidor de desarrollo incorporado de PHP para servir su aplicación, puede usar el comando `serve` Artisan. Este comando iniciará un servidor de desarrollo en `http://localhost:8000` :

```
php artisan serve
```

Por supuesto, las opciones de desarrollo local más sólidas están disponibles a través de Homestead y Valet .

También es posible usar un puerto personalizado, algo así como `8080` . Puedes hacer esto con la opción `--port` .

```
php artisan serve --port=8080
```

Si tiene un dominio local en su archivo de hosts, puede configurar el nombre de host. Esto se puede hacer mediante la opción `--host` .

```
php artisan serve --host=example.dev
```

También puede ejecutar en un host y puerto personalizados, esto se puede hacer con el siguiente comando.

```
php artisan serve --host=example.dev --port=8080
```

## Ejemplo de Hello World (básico) y con el uso de una vista

**El ejemplo basico**
Abra las `routes/web.php` archivo `routes/web.php` y pegue el siguiente código en el archivo:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

aquí ' **helloworld** ' actuará como nombre de página al que desea acceder,

y si no desea crear un archivo blade y aún desea acceder a la página directamente, puede usar el enrutamiento de laravel de esta manera

ahora escriba `localhost/helloworld` en la barra de direcciones del navegador y podrá acceder a la página que muestra Hello World.

**El siguiente paso.**
¡Así que has aprendido a crear un Hello World muy simple! página devolviendo una frase de hola mundo. ¡Pero podemos hacerlo un poco más agradable!

Paso 1.
Comenzaremos de nuevo en nuestro archivo de `routes/web.php` ahora, en lugar de usar el código anterior, usaremos el siguiente código:

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

El valor de retorno esta vez no es solo un simple texto de helloworld sino una vista. Una vista en Laravel es simplemente un nuevo archivo. Este archivo "helloworld" contiene el HTML y tal vez más adelante, incluso algunos PHP del texto de Helloworld.

Paso 2.
Ahora que hemos ajustado nuestra ruta para llamar en una vista, vamos a hacer la vista. Laravel trabaja con archivos blade.php en vistas. Entonces, en este caso, nuestra ruta se llama helloworld. Entonces nuestra vista será llamada `helloworld.blade.php`

Vamos a crear el nuevo archivo en el directorio `resources/views` y lo llamaremos helloworld.blade.php

Ahora abriremos este nuevo archivo y lo editaremos creando nuestra oración Hello World. Podemos agregar varias formas diferentes de obtener nuestra oración como se muestra en el siguiente ejemplo.

```
<html>
    <body>
        <h1> Hello World! </h1>

        <?php
            echo "Hello PHP World!";
        ?>

    </body>
</html>
```

ahora vaya a su navegador y escriba su ruta nuevamente como en el ejemplo básico: ¡ `localhost/helloworld` verá su nueva vista creada con todos los contenidos!

## Ejemplo de Hello World (Básico)

Abrir archivo de rutas. Pega el siguiente código en:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

después de ir a la ruta `http://localhost/helloworld` se muestra Hello World.

El archivo de rutas se encuentra `/routes/web.php`

## Configuración del servidor web para URL bonitas

Si instaló `Laravel` través de `Composer or the Laravel installer` , debajo de la configuración necesitará.

**La configuración para Apache** Laravel incluye un archivo `public/.htaccess` que se utiliza para proporcionar URL sin el controlador frontal `index.php` en la ruta. Antes de servir a Laravel con Apache, asegúrese de habilitar el módulo `mod_rewrite` para que el servidor `.htaccess` archivo `.htaccess` .

Si el archivo `.htaccess` que viene con Laravel no funciona con su instalación de Apache, intente esta alternativa:

```
Options +FollowSymLinks
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^ index.php [L]
```

---

**Configuración para Nginx** Si está utilizando Nginx, la siguiente directiva en la configuración de su sitio dirigirá todas las solicitudes al controlador frontal `index.php` :

```
location / {
    try_files $uri $uri/ /index.php?$query_string;
}
```

Por supuesto, al usar Homestead o Valet , las URL bonitas se configurarán automáticamente.

Lea Empezando con laravel-5.3 en línea: https://riptutorial.com/es/laravel/topic/8602/empezando-con-laravel-5-3

# Capítulo 23: Encuadernación de modelos de ruta

## Examples

### Vinculación implícita

Laravel resuelve automáticamente los modelos elocuentes definidos en rutas o acciones de controlador cuyos nombres de variable coinciden con un nombre de segmento de ruta. Por ejemplo:

```
Route::get('api/users/{user}', function (App\User $user) {
    return $user->email;
});
```

En este ejemplo, dado que la variable de usuario Eloquent $ definida en la ruta coincide con el segmento {usuario} en el URI de la ruta, Laravel inyectará automáticamente la instancia del modelo que tiene una ID que coincide con el valor correspondiente del URI de solicitud. Si no se encuentra una instancia de modelo coincidente en la base de datos, se generará automáticamente una respuesta HTTP 404.

Si el nombre de la tabla del modelo se compone de varias palabras, para hacer que el enlace implícito del modelo funcione, la variable de entrada debe estar en minúsculas;
Por ejemplo, si el usuario puede realizar algún tipo de *acción* y queremos acceder a esta acción, la ruta será:

```
Route::get('api/useractions/{useraction}', function (App\UserAction $useraction) {
    return $useraction->description;
});
```

### Vinculación explícita

Para registrar un enlace explícito, use el método de modelo del enrutador para especificar la clase para un parámetro dado. Debe definir sus enlaces de modelo explícitos en el método de arranque de la clase RouteServiceProvider

```
public function boot()
{
   parent::boot();

   Route::model('user', App\User::class);
}
```

A continuación, podemos definir una ruta que contiene el parámetro {usuario}.

```
$router->get('profile/{user}', function(App\User $user) {
```

```
});
```

Como hemos vinculado todos `{user}` parámetros de `{user}` al modelo `App\User` , se inyectará una instancia de User en la ruta. Entonces, por ejemplo, una solicitud de `profile/1` inyectará la instancia de Usuario de la base de datos que tiene un **ID** de **1** .

Si no se encuentra una instancia de modelo coincidente en la base de datos, se generará automáticamente una respuesta **HTTP 404** .

Lea Encuadernación de modelos de ruta en línea:
https://riptutorial.com/es/laravel/topic/7098/encuadernacion-de-modelos-de-ruta

# Capítulo 24: Enlaces útiles

## Introducción

En este tema, puede encontrar enlaces útiles para mejorar sus habilidades de Laravel o ampliar su conocimiento.

## Examples

### Ecosistema Laravel

- Laravel Scout : Laravel Scout ofrece una solución simple y basada en controladores para agregar búsqueda de texto completo a sus modelos de Eloquent.
- Laravel Passport : autenticación API sin dolor de cabeza. Passport es un servidor OAuth2 que está listo en minutos.
- Homestead - El entorno de desarrollo oficial de Laravel. Desarrollado por Vagrant, Homestead pone a todo su equipo en la misma página con lo último en PHP, MySQL, Postgres, Redis y más.
- Laravel Cashier - Facilite la facturación de suscripciones con las integraciones integradas de Stripe y Braintree. Cupones, intercambio de suscripciones, cancelaciones e incluso facturas en PDF están listos para usar.
- Forge : aprovisione y despliegue aplicaciones PHP ilimitadas en DigitalOcean, Linode y AWS.
- Envoyer - Despliegue PHP de tiempo de inactividad cero.
- Valet - Un entorno de desarrollo Laravel para Mac minimalistas. No vagabundo, no apache, no hay problema.
- Spark - Potentes andamios de aplicación SaaS. Deja de escribir boilerplate y enfócate en tu aplicación.
- Lumen : si todo lo que necesitas es una API y una velocidad de rayo, prueba Lumen. Es Laravel super-light.
- Statamic : un verdadero CMS diseñado para que las agencias sean rentables, los desarrolladores estén contentos y los clientes lo abracen.

### Educación

- Laracasts : aprenda desarrollo web moderno y práctico a través de screencasts de expertos.
- Laravel News - Manténgase al día con Laravel con Laravel News.
- Laravel.io - Foro con código de fuente abierta.

### Podcasts

- Laravel Noticias Podcasts
- Los podcasts de Laravel

Lea Enlaces útiles en línea: https://riptutorial.com/es/laravel/topic/9957/enlaces-utiles

# Capítulo 25: Enrutamiento

## Examples

### Enrutamiento básico

El enrutamiento define un mapa entre los métodos HTTP y los URI en un lado, y las acciones en el otro. Las rutas se escriben normalmente en el archivo `app/Http/routes.php` .

En su forma más simple, una ruta se define llamando al método HTTP correspondiente en la fachada de la ruta, pasando como parámetros una cadena que coincide con el URI (relativo a la raíz de la aplicación) y una devolución de llamada.

Por ejemplo: una ruta a la URI raíz del sitio que devuelve una vista de `home` ve así:

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

Una ruta para una solicitud de publicación que simplemente hace eco de las variables de publicación:

```
Route::post('submit', function() {
   return Input::all();
});

//or

Route::post('submit', function(\Illuminate\Http\Request $request) {
   return $request->all();
});
```

# Rutas que apuntan a un método de controlador

En lugar de definir la devolución de llamada en línea, la ruta puede referirse a un método de controlador en la sintaxis de [ControllerClassName @ Method]:

```
Route::get('login', 'LoginController@index');
```

# Una ruta para múltiples verbos.

El método de `match` se puede usar para hacer coincidir una matriz de métodos HTTP para una ruta determinada:

```
Route::match(['GET', 'POST'], '/', 'LoginController@index');
```

También puede usar `all` para que coincida con cualquier método HTTP para una ruta determinada:

```
Route::all('login', 'LoginController@index');
```

# Grupos de ruta

Las rutas se pueden agrupar para evitar la repetición del código.

Digamos que todos los URI con un prefijo de `/admin` usan un cierto middleware llamado `admin` y todos viven en el espacio de nombres `App\Http\Controllers\Admin` .

Una forma limpia de representar esto usando Grupos de Ruta es la siguiente:

```
Route::group([
    'namespace' => 'Admin',
    'middleware' => 'admin',
    'prefix' => 'admin'
    ], function () {

        // something.dev/admin
        // 'App\Http\Controllers\Admin\IndexController'
        // Uses admin middleware
        Route::get('/', ['uses' => 'IndexController@index']);

        // something.dev/admin/logs
        // 'App\Http\Controllers\Admin\LogsController'
        // Uses admin middleware
        Route::get('/logs', ['uses' => 'LogsController@index']);
});
```

## Ruta nombrada

Las rutas con nombre se utilizan para generar una URL o redireccionar a una ruta específica. La ventaja de usar una ruta con nombre es que si cambiamos el URI de una ruta en el futuro, no tendríamos que cambiar la URL o los redireccionamientos que apuntan a esa ruta si estamos usando una ruta con nombre. Pero si los enlaces se generaron utilizando la url [por ejemplo. `url('/admin/login')` ], entonces tendríamos que cambiar en todas partes donde se usa.

Las rutas con nombre se crean utilizando `as` clave de matriz

```
Route::get('login', ['as' => 'loginPage', 'uses' => 'LoginController@index']);
```

o usando el `name` método

```
Route::get('login', 'LoginController@index')->name('loginPage');
```

# Generar URL usando una ruta con nombre

Para generar un url usando el nombre de la ruta.

```
$url = route('loginPage');
```

Si está utilizando el nombre de la ruta para la redirección

```
$redirect = Redirect::route('loginPage');
```

### Parámetros de ruta

Puede utilizar los parámetros de ruta para obtener la parte del segmento URI. Puede definir un / s parámetro / s de ruta opcionales o necesarios mientras crea una ruta. Los parámetros opcionales tienen una `?` adjunto al final del nombre del parámetro. Este nombre está encerrado entre llaves `{}`

# Parámetro opcional

```
Route::get('profile/{id?}', ['as' => 'viewProfile', 'uses' => 'ProfileController@view']);
```

Se puede acceder a esta ruta en `domain.com/profile/23` donde 23 es el parámetro id. En este ejemplo, la `id` se pasa como un parámetro en el método de `view` de `ProfileController` . Dado que este es un parámetro opcional, acceder a `domain.com/profile` funciona bien.

# Parámetro requerido

```
Route::get('profile/{id}', ['as' => 'viewProfile', 'uses' => 'ProfileController@view']);
```

Tenga en cuenta que el nombre del parámetro requerido no tiene un `?` al final del nombre del parámetro.

# Accediendo al parámetro en el controlador.

En su controlador, su método de vista toma un parámetro con el **mismo** nombre que el de `routes.php` y se puede usar como una variable normal. Laravel se encarga de inyectar el valor:

```
public function view($id){
    echo $id;
}
```

### Coger todas las rutas

Si desea capturar todas las rutas, podría usar una expresión regular como se muestra:

```
Route::any('{catchall}', 'CatchAllController@handle')->where('catchall', '.*');
```

**Importante:** si tiene otras rutas y no desea que interfiera el catch-all, debe ponerlo al final. Por ejemplo:

# Atrapando todas las rutas excepto las ya definidas.

```
Route::get('login',  'AuthController@login');
Route::get('logout', 'AuthController@logout');
Route::get('home',   'HomeController@home');

// The catch-all will match anything except the previous defined routes.
Route::any('{catchall}', 'CatchAllController@handle')->where('catchall', '.*');
```

## Las rutas se emparejan en el orden en que se declaran.

Este es un gotcha común con las rutas de Laravel. Las rutas se emparejan en el orden en que se declaran. La primera ruta coincidente es la que se utiliza.

Este ejemplo funcionará como se esperaba:

```
Route::get('/posts/{postId}/comments/{commentId}', 'CommentController@show');
Route::get('/posts/{postId}', 'PostController@show');
```

Una solicitud de obtención a `/posts/1/comments/1` invocará `CommentController@show` . Una solicitud de obtención a `/posts/1` invocará `PostController@show` .

Sin embargo, este ejemplo no funcionará de la misma manera:

```
Route::get('/posts/{postId}', 'PostController@show');
Route::get('/posts/{postId}/comments/{commentId}', 'CommentController@show');
```

Una solicitud de obtención a `/posts/1/comments/1` invocará `PostController@show` . Una solicitud de obtención a `/posts/1` invocará `PostController@show` .

Debido a que Laravel usa la primera ruta coincidente, la solicitud a `/posts/1/comments/1` coincide con `Route::get('/posts/{postId}', 'PostController@show');` y asigna la variable `$postId` al valor `1/comments/1` . Esto significa que `CommentController@show` nunca será invocado.

## Rutas insensibles a mayúsculas

Las rutas en Laravel distinguen entre mayúsculas y minúsculas. Significa que una ruta como

```
Route::get('login', ...);
```

coincidirá con una solicitud GET para `/login` pero no coincidirá con una solicitud GET para `/login /Login` .

Para hacer que sus rutas no distingan entre mayúsculas y minúsculas, debe crear una nueva clase de validador que haga coincidir las URL solicitadas con las rutas definidas. La única diferencia entre el nuevo validador y el existente es que agregará el modificador **i** al final de la expresión regular para que la ruta compilada habilite la coincidencia entre mayúsculas y minúsculas.

```
<?php namespace Some\Namespace;

use Illuminate\Http\Request;
use Illuminate\Routing\Route;
use Illuminate\Routing\Matching\ValidatorInterface;

class CaseInsensitiveUriValidator implements ValidatorInterface
{
  public function matches(Route $route, Request $request)
  {
    $path = $request->path() == '/' ? '/' : '/'.$request->path();
    return preg_match(preg_replace('/$/','i', $route->getCompiled()->getRegex()),
rawurldecode($path));
  }
}
```

Para que Laravel use su nuevo validador, debe actualizar la lista de emparejadores que se utilizan para hacer coincidir la URL con una ruta y reemplazar el UriValidator original con el suyo.

Para hacer eso, agregue lo siguiente en la parte superior de su archivo route.php:

```
<?php
use Illuminate\Routing\Route as IlluminateRoute;
use Your\Namespace\CaseInsensitiveUriValidator;
use Illuminate\Routing\Matching\UriValidator;

$validators = IlluminateRoute::getValidators();
$validators[] = new CaseInsensitiveUriValidator;
IlluminateRoute::$validators = array_filter($validators, function($validator) {
  return get_class($validator) != UriValidator::class;
});
```

Esto eliminará el validador original y agregará el suyo a la lista de validadores.

Lea Enrutamiento en línea: https://riptutorial.com/es/laravel/topic/1284/enrutamiento

# Capítulo 26: Estructura de directorios

## Examples

### Cambiar el directorio predeterminado de la aplicación

Hay casos de uso en los que es posible que desee cambiar el nombre del directorio de su aplicación a otra cosa. En Laravel4 solo puedes cambiar una entrada de configuración, aquí hay una forma de hacerlo en Laravel5.

En este ejemplo, cambiaremos el nombre del directorio de la `app` a `src` .

## Anular clase de aplicación

La `app` nombres de directorios está codificada en la clase de aplicación principal, por lo que debe anularse. Crear un nuevo archivo `Application.php` . Prefiero mantener el mío en el directorio `src` (con el que reemplazaremos la aplicación), pero puedes ubicarlo en otro lugar.

Así es como debería verse la clase anulada. Si quieres un nombre diferente, simplemente cambiar la cadena `src` a otra cosa.

```
namespace App;

class Application extends \Illuminate\Foundation\Application
{
    /**
     * @inheritdoc
     */
    public function path($path = '')
    {
        return $this->basePath . DIRECTORY_SEPARATOR . 'src' . ($path ? DIRECTORY_SEPARATOR .
$path : $path);
    }
}
```

Guarda el archivo. Hemos terminado con eso.

## Llamando a la nueva clase

Abre `bootstrap/app.php` y localiza

```
$app = new Illuminate\Foundation\Application(
    realpath(__DIR__.'/../')
);
```

Lo reemplazaremos con esto.

```
$app = new App\Application(
```

```
    realpath(__DIR__.'/../')
);
```

## Compositor

Abra su archivo `composer.json` y cambie la carga automática para que coincida con su nueva ubicación

```
"psr-4": {
    "App\\": "src/"
}
```

Y, finalmente, en la línea de comandos, ejecute `composer dump-autoload` y su aplicación debe ser servida desde el directorio `src` .

### Cambiar el directorio de controladores

Si queremos cambiar el directorio de `Controllers` , necesitamos:

1. Mueva y / o cambie el nombre del directorio de `Controllers` predeterminado donde lo queramos. Por ejemplo, desde `app/Http/Controllers` a `app/Controllers`
2. Actualice todos los espacios de nombres de los archivos dentro de la carpeta `Controllers` , haciendo que se adhieran a la nueva ruta, respetando el PSR-4 específico.
3. Cambie el espacio de nombres que se aplica al archivo `routes.php` , editando `app\Providers\RouteServiceProvider.php` y cambie esto:

```
protected $namespace = 'App\Http\Controllers';
```

a esto:

```
protected $namespace = 'App\Controllers';
```

Lea Estructura de directorios en línea: https://riptutorial.com/es/laravel/topic/3153/estructura-de-directorios

# Capítulo 27: Eventos y oyentes

## Examples

### Uso de eventos y escuchas para enviar correos electrónicos a un nuevo usuario registrado

Los eventos de Laravel permiten implementar el patrón Observer. Esto se puede usar para enviar un correo electrónico de bienvenida a un usuario cuando se registran en su aplicación.

Se pueden generar nuevos eventos y escuchas usando la utilidad de línea de comandos de artisan después de registrar el evento y su escucha particular en la clase `App\Providers\EventServiceProvider` .

```
protected $listen = [
    'App\Events\NewUserRegistered' => [
        'App\Listeners\SendWelcomeEmail',
    ],
];
```

Notación alternativa:

```
protected $listen = [
    \App\Events\NewUserRegistered::class => [
        \App\Listeners\SendWelcomeEmail::class,
    ],
];
```

Ahora ejecuta `php artisan generate:event` . Este comando generará todos los eventos y escuchas correspondientes mencionados anteriormente en los directorios `App\Events` y `App\Listeners` respectivamente.

Podemos tener múltiples oyentes en un solo evento como

```
protected $listen = [
    'Event' => [
        'Listner1', 'Listener2'
    ],
];
```

`NewUserRegistered` es solo una clase de envoltorio para el modelo de usuario recién registrado:

```
class NewUserRegistered extends Event
{
    use SerializesModels;

    public $user;

    /**
```

```
     * Create a new event instance.
     *
     * @return void
     */
    public function __construct(User $user)
    {
        $this->user = $user;
    }
}
```

Este `Event` será manejado por el oyente `SendWelcomeEmail` :

```
class SendWelcomeEmail
{
    /**
     * Handle the event.
     *
     * @param  NewUserRegistered  $event
     */
    public function handle(NewUserRegistered $event)
    {
        //send the welcome email to the user
        $user = $event->user;
        Mail::send('emails.welcome', ['user' => $user], function ($message) use ($user) {
                $message->from('hi@yourdomain.com', 'John Doe');
                $message->subject('Welcome aboard '.$user->name.'!');
                $message->to($user->email);
        });
    }
}
```

El último paso es llamar / disparar el evento cada vez que un nuevo usuario se registre. Esto se puede hacer en el controlador, comando o servicio, donde sea que implemente la lógica de registro de usuario:

```
event(new NewUserRegistered($user));
```

Lea Eventos y oyentes en línea: https://riptutorial.com/es/laravel/topic/4687/eventos-y-oyentes

# Capítulo 28: Formulario de solicitud (s)

## Introducción

Las solicitudes personalizadas (o solicitudes de formulario) son útiles en situaciones en las que uno quiere **autorizar** y **validar** una solicitud antes de golpear el método del controlador.

Uno puede pensar en dos usos prácticos, **crear** y **actualizar** un registro, mientras que cada acción tiene un conjunto diferente de reglas de validación (o autorización).

Usar solicitudes de formulario es trivial, uno tiene que tipear la clase de solicitud en el método.

## Sintaxis

- php artesano make: solicitar name_of_request

## Observaciones

Las solicitudes son útiles al separar su validación del Controlador. También le permite verificar si la solicitud está autorizada.

## Examples

### Creación de solicitudes

```
php artisan make:request StoreUserRequest

php artisan make:request UpdateUserRequest
```

> **Nota** : También puede considerar el uso de nombres como **StoreUser** o **UpdateUser** (sin el apéndice de **Solicitud** ) ya que sus Solicitudes de Formulario se colocan en la carpeta `app/Http/Requests/` .

### Usando solicitud de formulario

Digamos que continúe con el ejemplo del usuario (puede tener un controlador con el método de almacenamiento y el método de actualización). Para usar FormRequests, use el tipo de sugerencia específica.

```
...

public function store(App\Http\Requests\StoreRequest $request, App\User $user) {
    //by type-hinting the request class, Laravel "runs" StoreRequest
    //before actual method store is hit

    //logic that handles storing new user
```

```
    //(both email and password has to be in $fillable property of User model
    $user->create($request->only(['email', 'password']));
    return redirect()->back();
}

...

public function update(App\Http\Requests\UpdateRequest $request, App\User $users, $id) {
    //by type-hinting the request class, Laravel "runs" UpdateRequest
    //before actual method update is hit

    //logic that handles updating a user
    //(both email and password has to be in $fillable property of User model
    $user = $users->findOrFail($id);
    $user->update($request->only(['password']));
    return redirect()->back();
}
```

## Manejo Redirecciones luego de validación.

A veces es posible que desee tener algún inicio de sesión para determinar a dónde se redirige el usuario después de enviar un formulario. Las solicitudes de formulario dan una variedad de formas.

Por defecto, hay 3 variables declaradas en la solicitud `$redirect` , `$redirectRoute` y `$redirectAction` .

Encima de esas 3 variables, puede anular el controlador de redireccionamiento principal `getRedirectUrl()` .

A continuación se muestra una solicitud de muestra que explica lo que puede hacer.

```
<?php namespace App;

use Illuminate\Foundation\Http\FormRequest as Request;

class SampleRequest extends Request {

    // Redirect to the given url
    public $redirect;

    // Redirect to a given route
    public $redirectRoute;

    // Redirect to a given action
    public $redirectAction;


    /**
     * Get the URL to redirect to on a validation error.
     *
     * @return string
     */
    protected function getRedirectUrl()
    {

        // If no path is given for `url()` it will return a new instance of
```

```
`Illuminate\Routing\UrlGenerator`

        // If your form is down the page for example you can redirect to a hash
        return url()->previous() . '#contact';

        //`url()` provides several methods you can chain such as

        // Get the current URL
        return url()->current();

        // Get the full URL of the current request
        return url()->full();

        // Go back
        return url()->previous();

        // Or just redirect back
        return redirect()->back();
    }


    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [];
    }

    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }
}
```

Lea Formulario de solicitud (s) en línea: https://riptutorial.com/es/laravel/topic/6329/formulario-de-solicitud--s-

# Capítulo 29: Función de ayuda personalizada

## Introducción

Agregar ayudantes personalizados puede ayudarlo con su velocidad de desarrollo. Sin embargo, hay algunas cosas que se deben tener en cuenta al escribir estas funciones de ayuda, de ahí este tutorial.

## Observaciones

Sólo unos pocos punteros:

- Hemos puesto las definiciones de funciones dentro de una verificación ( `function_exists` ) para evitar excepciones cuando se llama dos veces al proveedor de servicios.
- Una forma alternativa es registrar el archivo de ayudantes del archivo `composer.json` . Puede copiar la lógica desde el propio marco de laravel .

## Examples

### document.php

```
<?php

if (!function_exists('document')) {
    function document($text = '') {
        return $text;
    }
}
```

Cree un archivo helpers.php, supongamos por ahora que vive en `app/Helpers/document.php` . Puedes poner muchos ayudantes en un archivo (así es como lo hace Laravel) o puedes dividirlos por nombre.

### HelpersServiceProvider.php

Ahora vamos a crear un proveedor de servicios. Vamos a ponerlo bajo la `app/Providers` :

```
<?php

namespace App\Providers;

class HelpersServiceProvider extends ServiceProvider
{
    public function register()
    {
        require_once __DIR__ . '/../Helpers/document.php';
    }
}
```

El proveedor de servicios anterior carga el archivo de ayudantes y registra su función personalizada automáticamente. Asegúrese de registrar este HelpersServiceProvider en su `config/app.php` bajo los `providers` :

```
'providers' => [
     // [..] other providers
     App\Providers\HelpersServiceProvider::class,
]
```

## Utilizar

Ahora puede usar el `document()` función `document()` en cualquier parte de su código, por ejemplo, en plantillas blade. Este ejemplo solo devuelve la misma cadena que recibe como argumento

```
<?php
Route::get('document/{text}', function($text) {
    return document($text);
});
```

Ahora vaya a `/document/foo` en su navegador (use `php artisan serve` o `valet` ), que devolverá `foo` .

Lea Función de ayuda personalizada en línea: https://riptutorial.com/es/laravel/topic/8347/funcion-de-ayuda-personalizada

# Capítulo 30: Fundamentos básicos

## Introducción

Cron es un demonio del programador de tareas que ejecuta tareas programadas en ciertos intervalos. Cron usa un archivo de configuración llamado crontab, también conocido como tabla cron, para administrar el proceso de programación.

## Examples

### Crear Cron Job

Crontab contiene trabajos cron, cada uno relacionado con una tarea específica. Los trabajos de Cron se componen de dos partes, la expresión cron y un comando de shell que se ejecutará:

```
* * * * * comando / to / run
```

Cada campo en la expresión anterior `* * * * *` es una opción para configurar la frecuencia de programación. Se compone de minuto, hora, día del mes, mes y día de la semana en orden de colocación. El símbolo de asterisco se refiere a todos los valores posibles para el campo respectivo. Como resultado, la tarea cron anterior se ejecutará cada minuto del día.

El siguiente trabajo cron se ejecuta a las **12:30** todos los días:

```
30 12 * * * comando / a / correr
```

Lea Fundamentos básicos en línea: https://riptutorial.com/es/laravel/topic/9891/fundamentos-basicos

# Capítulo 31: Guía de instalación

## Observaciones

Esta sección proporciona una descripción general de qué es laravel-5.4 y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de laravel-5.4, y vincular a los temas relacionados. Dado que la Documentación para laravel-5.4 es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

## Examples

### Instalación

Instrucciones detalladas sobre cómo configurar o instalar el laravel.

Se requiere compositor para instalar laravel fácilmente.

Existen 3 métodos para instalar laravel en su sistema:

1. **Via Laravel Installer**

   Descarga el instalador de Laravel usando el `composer`

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

   Antes de usar compositor, necesitamos agregar `~/.composer/vendor/bin` a `PATH` . Una vez finalizada la instalación, podemos usar el `laravel new` comando `laravel new` para crear un nuevo proyecto en `Laravel` .

   Ejemplo:

   ```
   laravel new {folder name}
   ```

   Este comando crea un nuevo directorio llamado `site` y se instala una nueva instalación de `Laravel` con todas las demás dependencias en el directorio.

2. **Via Composer Create-Project**

   Puede usar el comando en el `terminal` para crear una nueva `Laravel app` :

   ```
   composer create-project laravel/laravel {folder name}
   ```

3. **Via Descargar**

Descarga Laravel y descomprime.

1. `composer install`
2. Copie `.env.example` a `.env` través de `teminal` o manualmente.

```
cp .env.example .env
```

3. Abra el archivo `.env` y configure su base de datos, correo electrónico, pulsador, etc. (si es necesario)
4. `php artisan migrate` (si la base de datos está configurada)
5. `php artisan key:generate`
6. `php artisan serve`
7. Ir a localhost: 8000 para ver el sitio

Laravel docs

## Ejemplo de Hello World (Básico)

Acceder a las páginas y generar datos es bastante fácil en Laravel. Todas las rutas de la página se encuentran en `app/routes.php` . Generalmente hay algunos ejemplos para comenzar, pero vamos a crear una nueva ruta. Abra su `app/routes.php` , y pegue el siguiente código:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

Esto le dice a Laravel que cuando alguien accede a `http://localhost/helloworld` en un navegador, debe ejecutar la función y devolver la cadena proporcionada.

## Ejemplo de Hello World con vistas y controlador

Suponiendo que tenemos una aplicación de trabajo en ejecución que se ejecuta en, digamos, "mylaravel.com", queremos que nuestra aplicación muestre un mensaje "Hola mundo" cuando lleguemos a la URL `http://mylaravel.com/helloworld` . Implica la creación de dos archivos (la vista y el controlador) y la modificación de un archivo existente, el enrutador.

## La vista

En primer lugar, abrimos un nuevo archivo de vista de blade llamado `helloview.blade.php` con la cadena "Hello World". Crealo en el directorio app / resources / views

```
<h1>Hello, World</h1>
```

## El controlador

Ahora creamos un controlador que administrará la visualización de esa vista con la cadena "Hello World". Usaremos artesano en la línea de comandos.

```
$> cd your_laravel_project_root_directory
$> php artisan make:controller HelloController
```

Eso solo creará un archivo ( `app/Http/Controllers/HelloController.php` ) que contiene la clase que es nuestro nuevo controlador `HelloController` .

Edite el nuevo archivo y escriba un nuevo método `hello` que mostrará la vista que creamos anteriormente.

```
public function hello()
{
    return view('helloview');
}
```

Ese argumento 'helloview' en la función de vista es solo el nombre del archivo de vista sin el final ".blade.php". Laravel sabrá cómo encontrarlo.

Ahora, cuando llamemos al método `hello` del controlador `HelloController` , se mostrará el mensaje. Pero, ¿cómo vinculamos eso con una llamada a `http://mylaravel.com/helloworld` ? Con el paso final, el enrutamiento.

## El enrutador

Abra el archivo existente `app/routes/web.php` (en las versiones más antiguas de laravel `app/Http/routes.php` ) y agregue esta línea:

```
Route::get('/helloworld', 'HelloController@hello');
```

que es un comando muy autoexplicativo que dice a nuestra aplicación de laravel: "Cuando alguien usa el verbo `GET` para acceder a '/ helloworld' en esta aplicación de laravel, devuelva los resultados de llamar a la función `hello` en el controlador `HelloController` .

Lea Guía de instalación en línea: https://riptutorial.com/es/laravel/topic/2187/guia-de-instalacion

# Capítulo 32: HTML y Form Builder

## Examples

### Instalación

HTML y Form Builder no son un componente central desde Laravel 5, por lo que necesitamos instalarlo por separado:

```
composer require laravelcollective/html "~5.0"
```

Finalmente, en `config/app.php` necesitamos registrar el proveedor de servicios y los alias de fachadas como este:

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

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

Los documentos completos están disponibles en formularios y HTML

Lea HTML y Form Builder en línea: https://riptutorial.com/es/laravel/topic/3672/html-y-form-builder

# Capítulo 33: Implementar la aplicación Laravel 5 en alojamiento compartido en un servidor Linux

## Observaciones

Para obtener más información sobre la implementación del proyecto Laravel en un alojamiento compartido, visite este repositorio de Github.

## Examples

### Aplicación Laravel 5 en Hosting Compartido en Servidor Linux

De forma predeterminada, `public` carpeta `public` del proyecto Laravel expone el contenido de la aplicación que cualquier persona puede solicitar, el resto del código de la aplicación es invisible o inaccesible para cualquier persona sin los permisos adecuados.

Después de desarrollar la aplicación en su máquina de desarrollo, debe enviarse a un servidor de producción para que se pueda acceder a ella a través de Internet desde cualquier lugar, ¿no?

Para la mayoría de las aplicaciones / sitios web, la primera opción es utilizar el paquete de alojamiento compartido de proveedores de servicios de alojamiento como GoDaddy, HostGator, etc., principalmente debido a su bajo costo.

> **nota** : puede pedirle a su proveedor que cambie **document_root** manualmente, por lo que todo lo que tiene que hacer es cargar su aplicación Laravel en el servidor (a través de FTP), solicitar el cambio de la raíz a **{app} / public** y debería ser bueno.

Dichos paquetes de alojamiento compartido, sin embargo, tienen limitaciones en términos de acceso a terminales y permisos de archivos. De forma predeterminada, uno tiene que cargar su aplicación / código en la carpeta `public_html` en su cuenta de alojamiento compartido.

Entonces, si desea cargar un proyecto Laravel en una cuenta de hosting compartido, ¿cómo lo haría? ¿Debería cargar la aplicación completa (carpeta) en la carpeta `public_html` en su cuenta de alojamiento compartido? - **Ciertamente NO**

Debido a que todo en la carpeta `public_html` es accesible "públicamente, es decir, por cualquier persona", lo que sería un gran riesgo para la seguridad.

Pasos para cargar un proyecto a una cuenta de hosting compartido - a la manera de Laravel

**Paso 1**
Cree una carpeta llamada laravel (o cualquier cosa que desee) en el mismo nivel que la carpeta `public_html` .

```
Eg:
/
|--var
    |---www
        |----laravel       //create this folder in your shared hosting account
        |----public_html
        |----log
```

**Paso 2**
Copie todos los elementos, excepto la carpeta `public` de su proyecto de laravel (en la máquina de desarrollo) en la carpeta de `laravel` (en el servidor host - cuenta de alojamiento compartida).
Puedes usar:

- Panel C: ¿Cuál sería la opción más lenta?
- Cliente FTP: como ***FileZilla*** para conectarse a su cuenta de hosting compartido y transferir sus archivos y carpetas a través de la carga FTP
- Asignar unidad de red: también puede crear una unidad de red asignada en su máquina de desarrollo para conectarse a la carpeta raíz de su cuenta de alojamiento compartida usando " ftp: // su-nombre de dominio " como la dirección de red.

**Paso 3**
Abra la carpeta `public` de su proyecto de laravel (en la máquina de desarrollo), copie todo y pegue en la carpeta `public_html` (en el servidor host - cuenta de hosting compartido).
**Etapa 4**
Ahora abra el archivo `index.php` en la carpeta `public_html` en la cuenta de alojamiento compartido (en el editor de cpanel o en cualquier otro editor conectado) y:

***Cambio:***

```
require __DIR__.'/../bootstrap/autoload.php';
```

***A:***

```
require __DIR__.'/../laravel/bootstrap/autoload.php';
```

***Y cambio:***

```
$app = require_once __DIR__.'/../bootstrap/app.php';
```

***A:***

```
$app = require_once __DIR__.'/../laravel/bootstrap/app.php';
```

***Guardar y cerrar.***

**Paso 5**
Ahora vaya a la carpeta de `laravel` (en el `laravel` cuentas de alojamiento compartido) y abra `server.php` archivo `server.php`
***Cambio***

```
require_once __DIR__.'/public/index.php';
```

***A:***

```
require_once __DIR__.'../public_html/index.php';
```

***Guardar y cerrar.***

**Paso 6**
Establezca los permisos de archivo para la carpeta de `laravel/storage` (recursivamente) y todos los archivos, subcarpetas y archivos dentro de ellos en la cuenta de alojamiento compartido: servidor a `777` .
***Nota:*** tenga cuidado con los permisos de archivo en Linux, son como un arma de doble filo; si no se utilizan correctamente, pueden hacer que su aplicación sea vulnerable a los ataques. Para comprender los permisos de archivos de Linux, puede leer https://www.linux.com/learn/tutorials/309527-understanding-linux-file-permissions

**Paso 7**

Como el archivo `.env` del servidor local / de desarrollo es ignorado por git y debe ignorarse, ya que tiene todas las variables de entorno, incluida la APP_KEY, y no debe exponerse al público empujándolo en los repositorios '. También puede ver que el archivo `.gitignore` ha mencionado `.env` por lo que no lo cargará en los repositorios.

Después de seguir todos los pasos anteriores, `.env` un archivo `.env` en la carpeta laravel y agregue todas las variables de entorno que haya usado desde el archivo `.env` del servidor local / de desarrollo al archivo `.env` del servidor de producción.

Incluso hay archivos de configuración como `app.php` , `database.php` en la carpeta config de la aplicación laravel que define estas variables como predeterminadas en el segundo parámetro de `env()` pero no codifique los valores de estos archivos, ya que afectará la Archivos de configuración de los usuarios que extraen su repositorio. Por eso se recomienda crear el archivo `.env` manualmente!

También laravel proporciona `.env-example` archivo `.env-example` que puede usar como referencia.

Eso es.

Ahora, cuando visita la url que configuró como dominio con su servidor, su aplicación laravel debería funcionar tal como funcionaba en su máquina de desarrollo de localhost, mientras que el código de la aplicación es seguro y no está accesible para nadie sin los permisos de archivo adecuados.

Lea Implementar la aplicación Laravel 5 en alojamiento compartido en un servidor Linux en línea: https://riptutorial.com/es/laravel/topic/2410/implementar-la-aplicacion-laravel-5-en-alojamiento-compartido-en-un-servidor-linux

# Capítulo 34: Instalación

## Examples

### Instalación

Las aplicaciones de Laravel se instalan y administran con Composer , un popular administrador de dependencias de PHP. Hay dos formas de crear una nueva aplicación Laravel.

## Via Compositor

```
$ composer create-project laravel/laravel [foldername]
```

O

```
$ composer create-project --prefer-dist laravel/laravel [foldername]
```

Reemplace **[** nombre de **carpeta]** con el nombre del directorio en el que desea que se instale su nueva aplicación Laravel. No debe existir antes de la instalación. Es posible que también deba agregar el ejecutable de Composer a la ruta de su sistema.

Si desea crear un proyecto Laravel utilizando una versión específica del marco, puede proporcionar un patrón de versión, de lo contrario su proyecto utilizará la última versión disponible.

Si quisieras crear un proyecto en Laravel 5.2 por ejemplo, ejecutarías:

```
$ composer create-project --prefer-dist laravel/laravel 5.2.*
```

**Por qué --prefer-dist**

Hay dos formas de descargar un paquete: `source` y `dist` . Para versiones estables, Composer usará `dist` por defecto. La `source` es un repositorio de control de versiones. Si `--prefer-source` está habilitado, Composer se instalará desde la fuente si hay una.

`--prefer-dist` es lo opuesto a `--prefer-source` , y le dice a Composer que instale desde `dist` si es posible. Esto puede acelerar las instalaciones sustancialmente en los servidores de compilación y en otros casos de uso donde normalmente no se ejecutan las actualizaciones de los proveedores. También permite evitar problemas con Git si no tiene una configuración adecuada.

## A través del instalador de Laravel.

Laravel proporciona una útil utilidad de línea de comandos para crear rápidamente aplicaciones Laravel. Primero, instale el instalador:

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

> Debe asegurarse de que la carpeta de archivos binarios de Composer esté dentro de su variable $ PATH para ejecutar el instalador de Laravel.
>
> Primero, mira si ya está en tu variable $ PATH
>
> ```
> echo $PATH
> ```
>
> Si todo es correcto, la salida debe contener algo como esto:
>
> ```
> Users/yourusername/.composer/vendor/bin
> ```
>
> De lo contrario, edite su `.bashrc` o, si está usando ZSH, su `.zshrc` para que contenga la ruta al directorio de proveedores de Composer.

Una vez instalado, este comando creará una nueva instalación de Laravel en el directorio que especifique.

```
laravel new [foldername]
```

También puedes usar `.` (un punto) en lugar de **[nombre de carpeta]** para crear el proyecto en el directorio de trabajo actual sin crear un subdirectorio.

# Ejecutando la aplicación

Laravel viene con un servidor web basado en PHP que puede iniciarse ejecutando

```
$ php artisan serve
```

De forma predeterminada, el servidor HTTP usará el puerto 8000, pero si el puerto ya está en uso o si desea ejecutar varias aplicaciones de Laravel a la vez, puede usar el indicador `--port` para especificar un puerto diferente:

```
$ php artisan serve --port=8080
```

El servidor HTTP usará `localhost` como el dominio predeterminado para ejecutar la aplicación, pero puede usar la `--host` para especificar una dirección diferente:

```
$ php artisan serve --host=192.168.0.100 --port=8080
```

### Usando un servidor diferente

Si prefiere utilizar un software de servidor web diferente, se proporcionan algunos archivos de configuración dentro del directorio `public` de su proyecto; `.htaccess` para Apache y `web.config` para ASP.NET. Para otro software como NGINX, puede convertir las configuraciones de Apache usando varias herramientas en línea.

---

El marco necesita que el usuario del servidor web tenga permisos de escritura en los siguientes directorios:

- `/storage`
- `/bootstrap/cache`

En los sistemas operativos * nix esto se puede lograr mediante

```
chown -R www-data:www-data storage bootstrap/cache
chmod -R ug+rwx storage bootstrap/cache
```

(donde `www-data` es el nombre y el grupo del usuario del servidor web)

---

El servidor web de su elección debe configurarse para servir contenido desde el directorio `/public` de su proyecto, que generalmente se realiza estableciéndolo como la raíz del documento. El resto de su proyecto no debe ser accesible a través de su servidor web.

Si configura todo correctamente, navegar a la URL de su sitio web debería mostrar la página de destino predeterminada de Laravel.

## Requerimientos

El marco de Laravel tiene los siguientes requisitos:

5.3

- PHP> = 5.6.4
- Extensión PHP PHP
- Extensión PHP de DOP
- Extensión PHP OpenSSL
- Mbstring PHP Extension
- Tokenizer PHP Extension

5.1 (LTS) 5.2

- PHP> = 5.5.9
- Extensión PHP de DOP
- Laravel 5.1 es la primera versión de Laravel para soportar PHP 7.0.

5.0

- PHP> = 5.4, PHP <7
- Extensión PHP OpenSSL
- Tokenizer PHP Extension
- Mbstring PHP Extension
- Extensión JSON PHP (solo en PHP 5.5)

4.2

- PHP> = 5.4
- Mbstring PHP Extension
- Extensión JSON PHP (solo en PHP 5.5)

## Ejemplo de Hello World (usando el controlador y la vista)

1. Crea una aplicación Laravel:

```
$ composer create-project laravel/laravel hello-world
```

2. Vaya a la carpeta del proyecto, por ejemplo,

```
$ cd C:\xampp\htdocs\hello-world
```

3. Crear un controlador:

```
$ php artisan make:controller HelloController --resource
```

   Esto creará la **aplicación de** archivo **/ Http / Controllers / HelloController.php** . La opción `--resource` generará métodos CRUD para el controlador, por ejemplo, indexar, crear, mostrar, actualizar.

4. Registre una ruta al método de `index` de HelloController. Agregue esta línea a **app / Http / route.php** *(versión 5.0 a 5.2)* o **route / web.php** *(versión 5.3)* :

```
Route::get('hello', 'HelloController@index');
```

   Para ver las rutas recién agregadas, puede ejecutar la `$ php artisan route:list`

5. Cree una plantilla Blade en el directorio de `views` :

   **recursos / vistas / hello.blade.php:**

```
<h1>Hello world!</h1>
```

6. Ahora le decimos al método de índice que muestre la plantilla **hello.blade.php** :

   **app / Http / Controllers / HelloController.php**

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;

class HelloController extends Controller
{
    /**
```

```
         * Display a listing of the resource.
         *
         * @return \Illuminate\Http\Response
         */
        public function index()
        {
            return view('hello');
        }

        // ... other resources are listed below the index one above
```

Puede servir su aplicación usando el siguiente Comando PHP Artisan: **`php artisan serve`** ; le mostrará la dirección en la que puede acceder a su aplicación *(generalmente en http: // localhost: 8000 de manera predeterminada)* .

Alternativamente, puede dirigirse directamente a la ubicación apropiada en su navegador; *en caso de que esté utilizando un servidor como XAMPP (ya sea: http: // localhost / hello-world / public / hello si hubiera instalado su instancia de Laravel,* ***`hello-world`*** *, directamente en su directorio* ***xampp / htdocs*** *como: ejecutando el paso 1 de este Hello Word desde su interfaz de línea de comando, apuntando a su directorio* ***xampp / htdocs*** *)* .

## Ejemplo de Hello World (Básico)

Abrir archivo de rutas. Pega el siguiente código en:

```
Route::get('helloworld', function () {
    return '<h1>Hello World</h1>';
});
```

después de ir a la ruta `localhost/helloworld` muestra **Hello World** .

El archivo de rutas se encuentra:

5.3

Para web

```
routes/web.php
```

Para APIs

```
routes/api.php
```

5.2 5.1 (LTS) 5.0

```
app/Http/routes.php
```

4.2

```
app/routes.php
```

### Instalación utilizando LaraDock (Laravel Homestead for Docker)

LaraDock es un entorno de desarrollo similar a Laravel Homestead, pero para Docker en lugar de Vagrant. https://github.com/LaraDock/laradock

# Instalación

* Requiere Git y Docker.

Clona el repositorio de LaraDock:

A. Si ya tiene un proyecto Laravel, clone este repositorio en su directorio raíz Laravel:

```
git submodule add https://github.com/LaraDock/laradock.git
```

B. Si no tiene un proyecto Laravel y desea instalar Laravel desde Docker, clone este repositorio en cualquier lugar de su máquina:

```
git clone https://github.com/LaraDock/laradock.git
```

# Uso básico

1. Ejecutar contenedores: (Asegúrese de estar en la carpeta laradock antes de ejecutar los comandos de composición de la ventana acoplable).

   Ejemplo: ejecutar NGINX y MySQL: `docker-compose up -d nginx mysql`

   Hay una lista de contenedores disponibles que puede seleccionar para crear sus propias combinaciones.

   `nginx` , `hhvm` , `php-fpm` , `mysql` , `redis` , `postgres` , `mariadb` , `neo4j` , `mongo` , `apache2` , `caddy` , `memcached` , `beanstalkd` , `beanstalkd-console` , `workspace`

2. Ingrese al contenedor Workspace, para ejecutar comandos como (Artisan, Composer, PHPUnit, Gulp, ...).

   ```
   docker-compose exec workspace bash
   ```

3. Si aún no tiene un proyecto Laravel instalado, siga el paso para instalar Laravel desde un contenedor Docker.

   a. Entra en el contenedor del espacio de trabajo.

   segundo. Instala Laravel. `composer create-project laravel/laravel my-cool-app "5.3.*"`

4. Editar las configuraciones de Laravel. Abra su archivo .env de Laravel y configure DB_HOST en su mysql:

```
DB_HOST=mysql
```

5. Abra su navegador y visite su dirección localhost.

Lea Instalación en línea: https://riptutorial.com/es/laravel/topic/7961/instalacion

# Capítulo 35: Integración de Sparkpost con Laravel 5.4

## Introducción

Laravel 5.4 viene preinstalado con sparkpost api lib. Sparkpost lib requiere una clave secreta que se puede encontrar en su cuenta de sparkpost.

## Examples

### MUESTRA de datos del archivo .env

Para crear con éxito una configuración de api de correo electrónico de chispa, agregue los detalles a continuación al archivo env y su aplicación será buena para comenzar a enviar correos electrónicos.

MAIL_DRIVER = sparkpost
SPARKPOST_SECRET =

NOTA: Los detalles anteriores no le dan el código escrito en el controlador, que tiene la lógica empresarial para enviar correos electrónicos mediante la función de correo de correo electrónico de laravel.

Lea Integración de Sparkpost con Laravel 5.4 en línea:
https://riptutorial.com/es/laravel/topic/10136/integracion-de-sparkpost-con-laravel-5-4

# Capítulo 36: Introducción a laravel-5.2.

## Introducción

Laravel es un marco MVC con paquetes, migraciones y Artisan CLI. Laravel ofrece un conjunto robusto de herramientas y una arquitectura de aplicación que incorpora muchas de las mejores características de marcos como CodeIgniter, Yii, ASP.NET MVC, Ruby on Rails, Sinatra y otros. Laravel es un framework de código abierto. Tiene un conjunto muy rico de características que aumentarán la velocidad del desarrollo web. Si está familiarizado con Core PHP y Advanced PHP, Laravel facilitará su tarea. Te ahorrará mucho tiempo.

## Observaciones

Esta sección proporciona una descripción general de qué es laravel-5.1 y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema importante dentro de laravel-5.1 y vincular a los temas relacionados. Dado que la Documentación para laravel-5.1 es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

## Examples

### Instalación o configuración

Instrucciones para instalar Laravel 5.1 en una máquina Linux / Mac / Unix.

Antes de iniciar la instalación, compruebe si se cumplen los siguientes requisitos:

- PHP> = 5.5.9
- Extensión PHP OpenSSL
- Extensión PHP de DOP
- Mbstring PHP Extension
- Tokenizer PHP Extension

Comencemos la instalación:

1. Instalar compositor. Documentación del compositor
2. Ejecutar `composer create-project laravel/laravel <folder-name> "5.1.*"`
3. Asegúrese de que la carpeta de `storage` y la carpeta `bootstrap/cache` sean de escritura.
4. Abra el archivo `.env` y configure la información de configuración como las credenciales de la base de datos, el estado de depuración, el entorno de la aplicación, etc.
5. Ejecute `php artisan serve` y apunte su navegador a `http://localhost:8000` . Si todo está bien, entonces debería obtener la página

### Instale Laravel 5.1 Framework en Ubuntu 16.04, 14.04 y LinuxMint

**Paso 1 - Instalar LAMP**

Para comenzar con Laravel, primero debemos configurar un servidor LAMP en ejecución. Si ya está ejecutando la pila LAMP, omita este paso o utilice los siguientes comandos para configurar la lámpara en el sistema Ubuntu.

Instalar PHP 5.6

```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository ppa:ondrej/php
$ sudo apt-get update
$ sudo apt-get install -y php5.6 php5.6-mcrypt php5.6-gd
```

Instalar apache2

```
$ apt-get install apache2 libapache2-mod-php5
```

Instalar MySQL

```
$ apt-get install mysql-server php5.6-mysql
```

**Paso 2 - Instalar Composer**

Se requiere Composer para instalar las dependencias de Laravel. Así que use los siguientes comandos para descargar y usar como un comando en nuestro sistema.

```
$ curl -sS https://getcomposer.org/installer | php
$ sudo mv composer.phar /usr/local/bin/composer
$ sudo chmod +x /usr/local/bin/composer
```

**Paso 3 - Instalar Laravel**

Para descargar la última versión de Laravel, use el siguiente comando para clonar el repositorio maestro de laravel desde github.

```
$ cd /var/www
$ git clone https://github.com/laravel/laravel.git
```

Navegue al directorio de código Laravel y use composer para instalar todas las dependencias requeridas para el marco Laravel.

```
$ cd /var/www/laravel
$ sudo composer install
```

La instalación de dependencias llevará algún tiempo. Después de que establecer permisos adecuados en los archivos.

```
$ chown -R www-data.www-data /var/www/laravel
$ chmod -R 755 /var/www/laravel
```

```
$ chmod -R 777 /var/www/laravel/app/storage
```

**Paso 4 - Establecer clave de cifrado**

Ahora configure la clave de encriptación de números aleatorios de 32 bits, que utiliza el servicio de cifrado Illuminate.

```
$ php artisan key:generate

Application key [uOHTNu3Au1Kt7Uloyr2Py9blU0J5XQ75] set successfully.
```

Ahora edite el archivo de configuración `config/app.php` y actualice la clave de la aplicación generada de la siguiente manera. También asegúrese de que el cifrado está configurado correctamente.

```
'key' => env('APP_KEY', 'uOHTNu3Au1Kt7Uloyr2Py9blU0J5XQ75'),

'cipher' => 'AES-256-CBC',
```

**Paso 5 - Crear Apache VirtualHost**

Ahora agregue un host virtual en su archivo de configuración de Apache para acceder al marco de trabajo de Laravel desde el navegador web. Cree el archivo de configuración de Apache en el directorio `/etc/apache2/sites-available/` y agregue el contenido a continuación.

```
$ vim /etc/apache2/sites-available/laravel.example.com.conf
```

Esta es la estructura de archivos del Host Virtual.

```
<VirtualHost *:80>

        ServerName laravel.example.com
        DocumentRoot /var/www/laravel/public

        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/laravel>
                AllowOverride All
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log
        LogLevel warn
        CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Finalmente, habilitamos el sitio web y recarguemos el servicio de Apache usando el siguiente comando.

```
$ a2ensite laravel.example.com
$ sudo service apache2 reload
```

### Paso 6 - Acceso a Laravel

En este punto, ha completado con éxito el marco PHP de Laravel 5 en su sistema. Ahora haga una entrada de archivo de host para acceder a su aplicación Laravel en el navegador web. Cambie `127.0.0.1` con su servidor ip y `laravel.example.com` con su nombre de dominio configurado en Apache.

```
$ sudo echo "127.0.0.1  laravel.example.com" >> /etc/hosts
```

Y acceda a http://laravel.example.com en su navegador web favorito como se muestra a continuación.

Lea Introducción a laravel-5.2. en línea: https://riptutorial.com/es/laravel/topic/1987/introduccion-a-laravel-5-2-

# Capítulo 37: Introducción a laravel-5.3.

## Introducción

Nuevas características, mejoras y cambios de Laravel 5.2 a 5.3.

## Examples

### La variable $ loop

Se sabe por un tiempo que tratar con bucles en Blade ha sido limitado, a partir de 5.3 hay una variable llamada `$loop` disponible

```
@foreach($variables as $variable)

    // Within here the `$loop` variable becomes available

    // Current index, 0 based
    $loop->index;

    // Current iteration, 1 based
    $loop->iteration;

    // How many iterations are left for the loop to be complete
    $loop->remaining;

    // Get the amount of items in the loop
    $loop->count;

    // Check to see if it's the first iteration ...
    $loop->first;

    // ... Or last iteration
    $loop->last;

    //Depth of the loop, ie if a loop within a loop the depth would be 2, 1 based counting.
    $loop->depth;

    // Get's the parent `$loop` if the loop is nested, else null
    $loop->parent;

@endforeach
```

Lea Introducción a laravel-5.3. en línea: https://riptutorial.com/es/laravel/topic/9231/introduccion-a-laravel-5-3-

# Capítulo 38: Laravel Docker

## Introducción

Un desafío al que se enfrentan todos los desarrolladores y el equipo de desarrollo es la coherencia del entorno. Laravel es uno de los frameworks PHP más populares en la actualidad. DDocker, por otro lado, es un método de virtualización que elimina los problemas de *"funciona en mi máquina"* al cooperar en el código con otros desarrolladores. Los dos juntos crean una fusión de **utilidad** y **poder** . Aunque ambos hacen cosas muy diferentes, ambos pueden combinarse para crear productos sorprendentes.

## Examples

### Usando Laradock

**Laradock** es un proyecto que proporciona una lista de aplicaciones listas para usar diseñadas para el uso de Laravel.

Descarga o clona Laradock en la carpeta raíz de tu proyecto:

```
git clone https://github.com/Laradock/laradock.git
```

Cambie el directorio a Laradock y genere el archivo `.env` necesario para ejecutar sus configuraciones:

```
cd laradock
cp .env-example .env
```

Ahora está listo para ejecutar la ventana acoplable. La primera vez que ejecute el contenedor, descargará todos los paquetes necesarios de Internet.

```
docker-compose up -d nginx mysql redis beanstalkd
```

Ahora puede abrir su navegador y ver su proyecto en `http://localhost` .

Para la documentación y configuración completa de Laradock, haga clic aquí .

Lea Laravel Docker en línea: https://riptutorial.com/es/laravel/topic/10034/laravel-docker

# Capítulo 39: Las macros en la relación elocuente

## Introducción

Tenemos nuevas características para Eloquent Relationship en la versión 5.4.8 de Laravel. Podemos obtener una instancia única de una relación hasMany (es solo un ejemplo) definiéndola en el lugar y funcionará para todas las relaciones

## Examples

### Podemos obtener una instancia de la relación hasMany

En nuestro AppServiceProvider.php

```
public function boot()
{
  HasMany::macro('toHasOne', function() {
      return new HasOne(
          $this->query,
          $this->parent,
          $this->foreignKey,
          $this->localKey
      );
  });
}
```

Supongamos que tenemos tienda modal y obtenemos la lista de productos que hemos comprado. Supongamos que tenemos todas las relaciones adquiridas para Shop modal

```
public function allPurchased()
{
    return $this->hasMany(Purchased::class);
}

public function lastPurchased()
{
    return $this->allPurchased()->latest()->toHasOne();
}
```

Lea Las macros en la relación elocuente en línea: https://riptutorial.com/es/laravel/topic/8998/las-macros-en-la-relacion-elocuente

# Capítulo 40: Manejo de errores

## Observaciones

Recuerde configurar su aplicación para enviar por correo electrónico asegurándose de la configuración correcta de `config/mail.php`

También verifique para asegurarse de que las variables ENV están configuradas correctamente.

Este ejemplo es una guía y es mínimo. Explora, modifica y diseña la vista como desees. Ajustar el código para satisfacer sus necesidades. Por ejemplo, establece el destinatario en tu archivo *.env*

## Examples

### Enviar correo electrónico de informe de error

Las excepciones en Laravel son manejadas por *App \ Exceptions \ Handler.php*

Este archivo contiene dos funciones por defecto. Reportar y renderizar. Solo usaremos la primera

```
public function report(Exception $e)
```

> El método de informe se utiliza para registrar excepciones o enviarlas a un servicio externo como BugSnag. De forma predeterminada, el método de informe simplemente pasa la excepción a la clase base donde se registra la excepción. Sin embargo, usted es libre de registrar excepciones como desee.

Esencialmente esta función simplemente reenvía el error y no hace nada. Por lo tanto, podemos insertar lógica de negocios para realizar operaciones basadas en el error. Para este ejemplo, le enviaremos un correo electrónico con la información del error.

```
public function report(Exception $e)
{
    if ($e instanceof \Exception) {
        // Fetch the error information we would like to
        // send to the view for emailing
        $error['file']    = $e->getFile();
        $error['code']    = $e->getCode();
        $error['line']    = $e->getLine();
        $error['message'] = $e->getMessage();
        $error['trace']   = $e->getTrace();

        // Only send email reports on production server
        if(ENV('APP_ENV') == "production"){
            #1. Queue email for sending on "exceptions_emails" queue
            #2. Use the emails.exception_notif view shown below
            #3. Pass the error array to the view as variable $e
            Mail::queueOn('exception_emails', 'emails.exception_notif', ["e" => $error],
function ($m) {
```

```
                $m->subject("Laravel Error");
                $m->from(ENV("MAIL_FROM"), ENV("MAIL_NAME"));
                $m->to("webmaster@laravelapp.com", "Webmaster");
            });

        }
    }

    // Pass the error on to continue processing
    return parent::report($e);
}
```

La vista para el correo electrónico ("emails.exception_notif") está abajo.

```
<?php
$action = (\Route::getCurrentRoute()) ? \Route::getCurrentRoute()->getActionName() : "n/a";
$path = (\Route::getCurrentRoute()) ? \Route::getCurrentRoute()->getPath() : "n/a";
$user = (\Auth::check()) ? \Auth::user()->name : 'no login';
?>

There was an error in your Laravel App<br />

<hr />
<table border="1" width="100%">
    <tr><th >User:</th><td>{{ $user }}</td></tr>
    <tr><th >Message:</th><td>{{ $e['message'] }}</td></tr>
    <tr><th >Action:</th><td>{{ $action }}</td></tr>
    <tr><th >URI:</th><td>{{ $path }}</td></tr>
    <tr><th >Line:</th><td>{{ $e['line'] }}</td></tr>
    <tr><th >Code:</th><td>{{ $e['code'] }}</td></tr>
</table>
```

## Captura de toda la aplicación ModelNotFoundException

### aplicación \ Excepciones \ Handler.php

```
public function render($request, Exception $exception)
{
    if ($exception instanceof ModelNotFoundException) {
        abort(404);
    }

    return parent::render($request, $exception);
}
```

Puede capturar / manejar cualquier excepción que se lance en Laravel.

Lea Manejo de errores en línea: https://riptutorial.com/es/laravel/topic/2858/manejo-de-errores

# Capítulo 41: marco del lumen

## Examples

### Empezando con Lumen

*El siguiente ejemplo demuestra el uso de `Lumen` en entornos `WAMP / MAMP / LAMP` .*

Para trabajar con `Lumen` , primero debes configurar un par de cosas.

- Compositor
- PHPUnit
- git (no requerido pero muy recomendado)

Suponiendo que tenga todos estos tres componentes instalados (al menos necesita compositor), primero vaya a la raíz de documentos de su servidor web usando terminal. MacOSX y Linux vienen con un gran terminal. Puedes usar `git bash` (que en realidad es `mingw32` o `mingw64` ) en Windows.

```
$ cd path/to/your/document/root
```

Entonces necesitas usar compose para instalar y crear el proyecto `Lumen` . Ejecute el siguiente comando.

```
$ composer create-project laravel/lumen=~5.2.0 --prefer-dist lumen-project
$ cd lumen-project
```

`lumen-app` en el código anterior es el nombre de la carpeta. Puedes cambiarlo como quieras. Ahora necesita configurar su host virtual para que apunte a la `path/to/document/root/lumen-project/public` folder. Supongamos que ha asignado `http://lumen-project.local` a esta carpeta. Ahora, si vas a esta URL, deberías ver un mensaje como el siguiente (dependiendo de tu versión de `Lumen` instalada, en mi caso era 5.4.4) -

```
Lumen (5.4.4) (Laravel Components 5.4.*)
```

Si abre el archivo `lumen-project/routers/web.php` , debería ver lo siguiente:

```
$app->get('/', function () use($app) {
    return $app->version();
});
```

¡Felicidades! Ahora tienes una instalación de trabajo de `Lumen` . No, puede ampliar esta aplicación para escuchar sus puntos finales personalizados.

Lea marco del lumen en línea: https://riptutorial.com/es/laravel/topic/9221/marco-del-lumen

# Capítulo 42: Middleware

## Introducción

Los middleware son clases, que pueden asignarse a una o más rutas, y se utilizan para realizar acciones en las fases iniciales o finales del ciclo de solicitud. Podemos considerarlos como una serie de capas por las que debe pasar una solicitud HTTP mientras se ejecuta.

## Observaciones

Un middleware "Antes" se ejecutará antes del código de acción del controlador; mientras que un middleware "Después" se ejecuta después de que la solicitud maneje la solicitud

## Examples

### Definiendo un middleware

Para definir un nuevo middleware tenemos que crear la clase de middleware:

```
class AuthenticationMiddleware
{
    //this method will execute when the middleware will be triggered
    public function handle ( $request, Closure $next )
    {
        if ( ! Auth::user() )
        {
            return redirect('login');
        }

        return $next($request);
    }
}
```

Luego tenemos que registrar el middleware: si el middleware se debe vincular a todas las rutas de la aplicación, debemos agregarlo a la propiedad de middleware de `app/Http/Kernel.php` :

```
protected $middleware = [
        \Illuminate\Foundation\Http\Middleware\CheckForMaintenanceMode::class,
        \App\Http\Middleware\AuthenticationMiddleware::class
];
```

mientras que si solo queremos asociar el middleware a algunas de las rutas, podemos agregarlo a `$routeMiddleware`

```
//register the middleware as a 'route middleware' giving it the name of 'custom_auth'
protected $routeMiddleware = [
    'custom_auth' => \App\Http\Middleware\AuthenticationMiddleware::class
];
```

y luego enlazarlo a las rutas individuales como esta:

```
//bind the middleware to the admin_page route, so that it will be executed for that route
Route::get('admin_page', 'AdminController@index')->middleware('custom_auth');
```

## Antes vs Después de Middleware

Un ejemplo de middleware "antes" sería el siguiente:

```
<?php

namespace App\Http\Middleware;

use Closure;

class BeforeMiddleware
{
    public function handle($request, Closure $next)
    {
        // Perform action

        return $next($request);
    }
}
```

mientras que "después" del middleware se vería así:

```
<?php

namespace App\Http\Middleware;

use Closure;

class AfterMiddleware
{
    public function handle($request, Closure $next)
    {
        $response = $next($request);

        // Perform action

        return $response;
    }
}
```

La diferencia clave está en cómo se maneja el parámetro `$request` . Si las acciones se realizan antes de que `$next($request)` ocurra antes de que se ejecute el código del controlador mientras se llama `$next($request)` , las acciones se realizarán después de que se ejecute el código del controlador.

## Ruta middleware

Cualquier middleware registrado como `routeMiddleware` en `app/Http/Kernel.php` puede asignarse a una ruta.

Hay algunas formas diferentes de asignar middleware, pero todas hacen lo mismo.

```
Route::get('/admin', 'AdminController@index')->middleware('auth', 'admin');
Route::get('admin/profile', ['using' => 'AdminController@index', 'middleware' => 'auth']);
Route::get('admin/profile', ['using' => 'AdminController@index', 'middleware' => ['auth',
'admin']);
```

En todos los ejemplos anteriores, también puede pasar nombres de clase totalmente calificados como middleware, independientemente de si se ha registrado como middleware de ruta.

```
use App\Http\Middleware\CheckAdmin;
Route::get('/admin', 'AdminController@index')->middleware(CheckAdmin::class);
```

Lea Middleware en línea: https://riptutorial.com/es/laravel/topic/3419/middleware

# Capítulo 43: Migraciones de base de datos

## Examples

### Migraciones

Para controlar su base de datos en Laravel es mediante el uso de migraciones. Crear migración con artesano:

```
php artisan make:migration create_first_table --create=first_table
```

Esto generará la clase CreateFirstTable. Dentro del método up puedes crear tus columnas:

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateFirstTable extends Migration
{
    public function up()
    {
        Schema::create('first_table', function (Blueprint $table) {
            $table->increments('id');
            $table->string('first_string_column_name');
            $table->integer('secont_integer_column_name');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::drop('first_table');
    }
}
```

Al final para ejecutar todas sus clases de migraciones, puede ejecutar el comando artisan:

```
php artisan migrate
```

Esto creará sus tablas y sus columnas en su base de datos. Otros comandos de migración útiles son:

- `php artisan migrate:rollback` - Rollback la última migración de base de datos
- `php artisan migrate:reset` - Deshacer todas las migraciones de bases de datos
- `php artisan migrate:refresh` - Restablecer y volver a ejecutar todas las migraciones
- `php artisan migrate:status` - Muestra el estado de cada migración

**Modificar tablas existentes**

A veces, necesita cambiar su estructura de tabla existente como `renaming/deleting` columnas. Lo que puede lograr creando una nueva migración. Y en el método `up` de su migración.

```
//Renaming Column.

public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->renameColumn('email', 'username');
    });
}
```

El ejemplo anterior cambiará el nombre de la `email column` de `email column` de la `users table` de `users table` a `username` de `username` . Mientras que el siguiente código elimina un `username` de `username` columna de la tabla de `users` .

IMPROTANTE: para modificar columnas, debe agregar la dependencia `doctrine/dbal` al archivo `composer.json` del proyecto y ejecutar la `composer update` para reflejar los cambios.

```
//Droping Column
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn('username');
    });
}
```

## Los archivos de migración

Las migraciones en una aplicación Laravel 5 viven en el directorio `database/migrations` . Sus nombres de archivos se ajustan a un formato particular:

```
<year>_<month>_<day>_<hour><minute><second>_<name>.php
```

Un archivo de migración debe representar una actualización de esquema para resolver un problema particular. Por ejemplo:

```
2016_07_21_134310_add_last_logged_in_to_users_table.php
```

Las migraciones de la base de datos se mantienen en orden cronológico para que Laravel sepa en qué orden ejecutarlas. Laravel siempre ejecutará migraciones de la más antigua a la más nueva.

## Generando archivos de migración

Crear un nuevo archivo de migración con el nombre de archivo correcto cada vez que necesite cambiar su esquema sería una tarea. Afortunadamente, el comando `artisan` de Laravel puede generar la migración para usted:

```
php artisan make:migration add_last_logged_in_to_users_table
```

También puede usar los `--table` y `--create` con el comando anterior. Estos son opcionales y solo por conveniencia, e insertarán el código relevante en el archivo de migración.

```
php artisan make:migration add_last_logged_in_to_users_table --table=users

php artisan make:migration create_logs_table --create=logs
```

Puede especificar una ruta de salida personalizada para la migración generada usando la opción `--path` . La ruta es relativa a la ruta base de la aplicación.

```
php artisan make:migration --path=app/Modules/User/Migrations
```

## Dentro de una migración de base de datos

Cada migración debe tener un método `up()` y un método `down()` . El propósito del método `up()` es realizar las operaciones requeridas para poner el esquema de la base de datos en su nuevo estado, y el propósito del método `down()` es revertir cualquier operación realizada por el método `up()` . Asegurarse de que el método `down()` invierta correctamente sus operaciones es fundamental para poder revertir los cambios del esquema de la base de datos.

Un ejemplo de archivo de migración puede verse así:

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class AddLastLoggedInToUsersTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dateTime('last_logged_in')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn('last_logged_in');
        });
    }
}
```

Al ejecutar esta migración, Laravel generará el siguiente SQL para ejecutarse en su base de datos:

```
ALTER TABLE `users` ADD `last_logged_in` DATETIME NULL
```

## Ejecutando migraciones

Una vez que se haya escrito su migración, la ejecución aplicará las operaciones a su base de datos.

```
php artisan migrate
```

Si todo salió bien, verás una salida similar a la siguiente:

```
Migrated: 2016_07_21_134310_add_last_logged_in_to_users_table
```

Laravel es lo suficientemente inteligente como para saber cuándo está ejecutando migraciones en el entorno de producción. Si detecta que está realizando una migración destructiva (por ejemplo, una que elimina una columna de una tabla), el comando de `php artisan migrate` le pedirá confirmación. En entornos de entrega continua esto puede no ser deseado. En ese caso, use la bandera `--force` para omitir la confirmación:

```
php artisan migrate --force
```

Si acaba de ejecutar migraciones, puede confundirse al ver la presencia de una tabla de `migrations` en su base de datos. Esta tabla es lo que utiliza Laravel para realizar un seguimiento de las migraciones que ya se han ejecutado. Al emitir el comando `migrate` , Laravel determinará qué migraciones aún deben ejecutarse, las ejecutará en orden cronológico y luego actualizará la tabla de `migrations` para que se adapte.

Nunca debe editar manualmente la tabla de `migrations` menos que sepa absolutamente lo que está haciendo. Es muy fácil dejar inadvertidamente su base de datos en un estado defectuoso en el que las migraciones fallarán.

## Migraciones de retroceso

¿Qué sucede si desea revertir la última migración, es decir, la operación reciente, puede usar el comando awesome `rollback` . Pero recuerde que este comando solo retrocede la última migración, que puede incluir múltiples archivos de migración

```
php artisan migrate:rollback
```

Si está interesado en revertir todas las migraciones de su aplicación, puede usar el siguiente comando

```
php artisan migrate:reset
```

Además, si eres un vago como yo y quieres retroceder y migrar con un solo comando, puedes usar este comando

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

También puede especificar el número de pasos para revertir con la opción de `step` . Al igual que esto hará retroceder 1 paso.

```
php artisan migrate:rollback --step=1
```

Lea Migraciones de base de datos en línea:
https://riptutorial.com/es/laravel/topic/1131/migraciones-de-base-de-datos

# Capítulo 44: Mundano

## Examples

### Instalación

```
composer require laravel/socialite
```

Esta instalación asume que estás usando Composer para administrar tus dependencias con Laravel, que es una excelente manera de lidiar con eso.

### Configuración

En su `config\services.php` puede agregar el siguiente código

```
'facebook' => [
    'client_id' => 'your-facebook-app-id',
    'client_secret' => 'your-facebook-app-secret',
    'redirect' => 'http://your-callback-url',
],
```

También deberá agregar el Proveedor a su `config\app.php`

Busque la matriz `'providers' => []` y, al final de la misma, agregue lo siguiente

```
'providers' => [
      ...

      Laravel\Socialite\SocialiteServiceProvider::class,
]
```

Una fachada también se proporciona con el paquete. Si desea `config\app.php` asegúrese de que la matriz de `aliases` (también en su `config\app.php` ) tenga el siguiente código

```
'aliases' => [
    ....
    'Socialite' => Laravel\Socialite\Facades\Socialite::class,
]
```

### Uso básico - Fachada

```
return Socialite::driver('facebook')->redirect();
```

Esto redirigirá una solicitud entrante a la URL apropiada para autenticarse. Un ejemplo básico sería en un controlador.

```
<?php
```

```
namespace App\Http\Controllers\Auth;

use Socialite;

class AuthenticationController extends Controller {

    /**
     * Redirects the User to the Facebook page to get authorization.
     *
     * @return Response
     */
    public function facebook() {
        return Socialite::driver('facebook')->redirect();
    }

}
```

asegúrese de que su `app\Http\routes.php` tenga una ruta para permitir una solicitud entrante aquí.

```
Route::get('facebook', 'App\Http\Controllers\Auth\AuthenticationController@facebook');
```

## Uso básico - inyección de dependencia

```
/**
 * LoginController constructor.
 * @param Socialite $socialite
 */
public function __construct(Socialite $socialite) {
    $this->socialite = $socialite;
}
```

Dentro del constructor de su Controlador, ahora puede inyectar la clase `Socialite` que lo ayudará a manejar el inicio de sesión con las redes sociales. Esto reemplazará el uso de la fachada.

```
/**
 * Redirects the User to the Facebook page to get authorization.
 *
 * @return Response
 */
public function facebook() {
    return $this->socialite->driver('facebook')->redirect();
}
```

## Socialite for API - Stateless

```
public function facebook() {
    return $this->socialite->driver('facebook')->stateless()->redirect()->getTargetUrl();
}
```

Esto devolverá la URL que el consumidor de la API debe proporcionar al usuario final para obtener la autorización de Facebook.

Lea Mundano en línea: https://riptutorial.com/es/laravel/topic/1312/mundano

# Capítulo 45: Nombrar archivos al cargar con Laravel en Windows

## Parámetros

| Parámetro / Función | Descripción |
|---|---|
| Subir archivo | nombre del archivo `<input>` campo |
| $ sampleName | También podría ser una cadena generada dinámicamente o el nombre del archivo cargado por el usuario |
| app_path () | Laravel es el ayudante para proporcionar la ruta absoluta a la aplicación. |
| getCLientOriginalExtension () | Laravel wrapper para recuperar la extensión del archivo cargado por el usuario tal como estaba en la máquina del usuario |

## Examples

### Generación de nombres de archivo con marca de tiempo para los archivos cargados por los usuarios.

**A continuación no funcionará en una máquina con Windows**

```
$file = $request->file('file_upload');
$sampleName = 'UserUpload';
$destination = app_path() . '/myStorage/';
$fileName = $sampleName . '-' . date('Y-m-d-H:i:s') . '.' .
$file->getClientOriginalExtension();
$file->move($destination, $fileName);
```

Se lanzará un error como "No se pudo mover el archivo a / ruta ..."

**¿Por qué? - Esto funciona perfectamente en un servidor Ubuntu.**
La razón es que en el `colon ':'` Windows `colon ':'` no está permitido en un nombre de archivo como está permitido en Linux. Esto es algo tan pequeño que no lo notamos por adelantado y nos preguntamos por qué un código que funciona bien en Ubuntu (Linux) no funciona.
Nuestro primer presentimiento sería revisar los permisos de los archivos y cosas así, pero es posible que no notemos que los `colon ':'` son el culpable aquí.
Entonces, para cargar archivos en Windows, **no use los `colon':'` mientras genera nombres de archivos con marca de tiempo** , en lugar de hacer algo como a continuación:

```
$filename = $sampleName . '-' . date('Y-m-d-H_i_s') . '.' . $file-
>getClientOriginalExtension();  //ex output UserUpload-2016-02-18-11_25_43.xlsx

                                    OR

$filename = $sampleName . '-' .date('Y-m-d H i s') . '.' . $file-
>getClientOriginalExtension();  //ex output UserUpload-2016-02-18 11 25 43.xlsx

                                    OR

$filename = $sampleName . '-'.date('Y-m-d_g-i-A').'.' . $file->getClientOriginalExtension();
//ex output UserUpload-2016-02-18_11-25-AM.xlsx
```

Lea Nombrar archivos al cargar con Laravel en Windows en línea:
https://riptutorial.com/es/laravel/topic/2629/nombrar-archivos-al-cargar-con-laravel-en-windows

# Capítulo 46: Observador

## Examples

### Creando un observador

Los observadores se utilizan para escuchar devoluciones de llamada de ciclo de vida de un determinado modelo en Laravel.
Estos oyentes pueden escuchar cualquiera de las siguientes acciones:

- creando
- creado
- actualizando
- actualizado
- ahorro
- salvado
- borrando
- eliminado
- restaurando
- restaurado

Aquí hay un ejemplo de un observador.

**UserObserver**

```
<?php

namespace App\Observers;

/**
 * Observes the Users model
 */
class UserObserver
{
    /**
     * Function will be triggerd when a user is updated
     *
     * @param Users $model
     */
     public function updated($model)
     {
         // execute your own code
     }
}
```

Como se muestra en el observador del usuario, escuchamos la acción actualizada; sin embargo, antes de que esta clase escuche el modelo del usuario, primero debemos registrarlo dentro del `EventServiceProvider` .

**EventServiceProvider**

```php
<?php

namespace App\Providers;

use Illuminate\Contracts\Events\Dispatcher as DispatcherContract;
use Illuminate\Foundation\Support\Providers\EventServiceProvider as ServiceProvider;

use App\Models\Users;
use App\Observers\UserObserver;

/**
 * Event service provider class
 */
class EventServiceProvider extends ServiceProvider
{
    /**
     * Boot function
     *
     * @param DispatcherContract $events
     */
    public function boot(DispatcherContract $events)
    {
        parent::boot($events);

        // In this case we have a User model that we want to observe
        // We tell Laravel that the observer for the user model is the UserObserver
        Users::observe(new UserObserver());
    }
}
```

Ahora que hemos registrado nuestro observador, se llamará a la función actualizada cada vez que se guarde el modelo de usuario.

Lea Observador en línea: https://riptutorial.com/es/laravel/topic/7128/observador

# Capítulo 47: Paginación

## Examples

### Paginación en Laravel

En otros marcos de paginación es el dolor de cabeza. Laravel lo hace fácil, puede generar paginación al agregar algunas líneas de código en Controlador y Ver.

**Uso básico**

Hay muchas formas de paginar elementos, pero la más simple es usar el método paginate en el generador de consultas o una consulta de Eloquent . Laravel out of the box se encarga de establecer el límite y la compensación en función de la página actual que está viendo el usuario. De forma predeterminada, la página actual es detectada por el valor de? Argumento de cadena de consulta de página en la solicitud HTTP. Y seguro, Laravel detecta este valor automáticamente y lo inserta en los enlaces generados por el paginador.

Ahora digamos que queremos llamar al método `paginate` en la consulta. En nuestro ejemplo, el argumento pasado para paginar es el número de elementos que le gustaría mostrar "por página". En nuestro caso, digamos que queremos mostrar 10 artículos por página.

```
<?php

namespace App\Http\Controllers;

use DB;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
    /**
     * Show all of the users for the application.
     *
     * @return Response
     */
    public function index()
    {
        $users = DB::table('users')->paginate(10);

        return view('user.index', ['users' => $users]);
    }
}
```

> **Nota:** Actualmente, las operaciones de paginación que usan una instrucción `groupBy` no pueden ejecutarse de manera eficiente por Laravel. Si necesita usar un `groupBy` con un conjunto de resultados paginado, se recomienda que consulte la base de datos y cree un paginador manualmente.

**Paginación simple**

Digamos que solo desea mostrar los enlaces Siguiente y Anterior en su vista de paginación. Laravel te ofrece esta opción utilizando el método `simplePaginate` .

```
$users = DB::table('users')->simplePaginate(10);
```

**Visualización de resultados en una vista**

Ahora vamos a mostrar la paginación a la vista. En realidad, cuando se llama a la `paginate` o `simplePaginate` métodos de consulta elocuente, recibirá una instancia paginador. Cuando se llama al método `Illuminate\Pagination\LengthAwarePaginator` , recibe una instancia de `Illuminate\Pagination\LengthAwarePaginator` , mientras que cuando llama al método `simplePaginate` , recibe una instancia de `Illuminate\Pagination\Paginator` . Estas instancias / objetos vienen con varios métodos que explican el conjunto de resultados. Además, además de estos métodos de ayuda, las instancias del paginador son iteradores y se pueden enlazar como una matriz.

Una vez que haya recibido los resultados, puede representar fácilmente los enlaces de la página utilizando blade

```
<div class="container">
    @foreach ($users as $user)
        {{ $user->name }}
    @endforeach
</div>

{{ $users->links() }}
```

El método de `links` procesará automáticamente los enlaces a otras páginas en el conjunto de resultados. Cada uno de estos enlaces contendrá el número de página específico, es decir, la variable de cadena de consulta de la `?page` . El HTML generado por el método de enlaces es perfectamente compatible con el marco CSS de Bootstrap.

## Cambio de vistas de paginación

Mientras usa la paginación de laravel, puede usar sus propias vistas personalizadas. Por lo tanto, cuando llame al método de enlaces en una instancia de paginador, pase el nombre de la vista como primer argumento al método como:

```
{{ $paginator->links('view.name') }}
```

o

Puede personalizar las vistas de paginación exportándolas a su directorio de `resources/views/vendor` utilizando el comando vendedor: publish:

```
php artisan vendor:publish --tag=laravel-pagination
```

Este comando colocará las vistas en el directorio `resources/views/vendor/pagination` . El archivo `default.blade.php` dentro de este directorio corresponde a la vista de paginación predeterminada. Edite este archivo para modificar el HTML de la paginación.

Lea Paginación en línea: https://riptutorial.com/es/laravel/topic/2359/paginacion

# Capítulo 48: Paquetes de vacaciones en Laravel

## Examples

### laravel-ide-helper

Este paquete genera un archivo que su IDE entiende, por lo que puede proporcionar un autocompletado preciso. La generación se realiza en base a los archivos en su proyecto.

Lea más sobre esto aquí

### laravel-datatables

Este paquete se crea para manejar los trabajos del lado del servidor de DataTables jQuery Plugin a través de la opción AJAX usando ORM Eloquent, Fluent Query Builder o Colección.

Lea más sobre esto aquí o aquí

### Imagen de intervención

Intervention Image es una biblioteca de manipulación y manipulación de imágenes PHP de código abierto. Proporciona una forma más fácil y expresiva de crear, editar y componer imágenes y admite actualmente las dos bibliotecas de procesamiento de imágenes más comunes, GD Library e Imagick.

Lea más sobre esto aquí

### Generador de Laravel

Prepare sus API y el Panel de administración en minutos. Generador de nivel para generar CRUD, API, casos de prueba y documentación de Swagger.

Lea más sobre esto aquí

### Laravel Socialite

Laravel Socialite proporciona una interfaz expresiva y fluida para la autenticación OAuth con Facebook, Twitter, Google, LinkedIn, GitHub y Bitbucket. Maneja casi todo el código de autenticación social que estás temiendo escribir.

Lea más sobre esto aquí

### Paquetes Oficiales

## Cajero

Laravel Cashier proporciona una interfaz expresiva y fluida para los servicios de facturación de suscripción de Stripe y Braintree . Maneja casi todo el código de facturación de suscripción que usted está escribiendo. Además de la administración básica de suscripciones, el Cajero puede manejar cupones, intercambiar suscripciones, "cantidades" de suscripciones, períodos de gracia de cancelación e incluso generar documentos PDF de facturas.

Más información sobre este paquete se puede encontrar aquí .

## Enviado

Laravel Envoy proporciona una sintaxis limpia y mínima para definir las tareas comunes que ejecuta en sus servidores remotos. Con la sintaxis de estilo Blade, puede configurar fácilmente las tareas para la implementación, los comandos de Artisan y más. Actualmente, Envoy solo es compatible con los sistemas operativos Mac y Linux.

Este paquete se puede encontrar en Github .

## Pasaporte

Laravel ya facilita la autenticación mediante los formularios de inicio de sesión tradicionales, pero ¿qué pasa con las API? Las API suelen utilizar tokens para autenticar a los usuarios y no mantienen el estado de la sesión entre las solicitudes. Laravel hace que la autenticación API sea muy fácil con Laravel Passport, que proporciona una implementación completa del servidor OAuth2 para su aplicación Laravel en cuestión de minutos.

Más información sobre este paquete se puede encontrar aquí .

## Explorar

Laravel Scout proporciona una solución simple y basada en controladores para agregar búsqueda de texto completo a sus modelos de Eloquent. Utilizando observadores modelo, Scout mantendrá automáticamente sus índices de búsqueda sincronizados con sus registros de Eloquent.

Actualmente, Scout se envía con un conductor de Algolia; sin embargo, escribir controladores personalizados es simple y puede extender Scout con sus propias implementaciones de búsqueda.

Más información sobre este paquete se puede encontrar aquí .

## Mundano

Laravel Socialite proporciona una interfaz expresiva y fluida para la autenticación OAuth con Facebook, Twitter, Google, LinkedIn, GitHub y Bitbucket. Maneja casi todo el código de

autenticación social que estás temiendo escribir.

Este paquete se puede encontrar en Github .

Lea Paquetes de vacaciones en Laravel en línea:
https://riptutorial.com/es/laravel/topic/8001/paquetes-de-vacaciones-en-laravel

# Capítulo 49: Permisos de almacenamiento

## Introducción

Laravel requiere que algunas carpetas sean de escritura para el usuario del servidor web.

## Examples

### Ejemplo

También necesitamos establecer los permisos correctos para `storage` archivos de `storage` en el `server` . Por lo tanto, debemos otorgar un permiso de escritura en el directorio de almacenamiento de la siguiente manera:

```
$ chmod -R 777 ./storage ./bootstrap
```

o puedes usar

```
$ sudo chmod -R 777 ./storage ./bootstrap
```

Para ventanas

Asegúrate de ser un usuario administrador en esa computadora con acceso de escritura

```
xampp\htdocs\laravel\app\storage needs to be writable
```

La forma NORMAL de establecer permisos es que sus archivos sean propiedad del servidor web:

```
sudo chown -R www-data:www-data /path/to/your/root/directory
```

Lea Permisos de almacenamiento en línea: https://riptutorial.com/es/laravel/topic/9797/permisos-de-almacenamiento

# Capítulo 50: Peticiones

## Examples

### Obteniendo entrada

La principal forma de obtener información sería inyectando el `Illuminate\Http\Request` en su controlador, después de eso hay muchas formas de acceder a los datos, 4 de las cuales se encuentran en el siguiente ejemplo.

```
<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
    public function store(Request $request)
    {
        // Returns the username value
        $name = $request->input('username');

        // Returns the username value
        $name = $request->username;

        // Returns the username value
        $name = request('username');

        // Returns the username value again
        $name = request()->username;

    }

}
```

Cuando se utiliza la función de `input` , también es posible agregar un valor predeterminado para cuando la entrada de solicitud no está disponible

```
$name = $request->input('username', 'John Doe');
```

Lea Peticiones en línea: https://riptutorial.com/es/laravel/topic/3076/peticiones

# Capítulo 51: Peticiones

## Examples

### Obtener una instancia de solicitud HTTP

Para obtener una instancia de una solicitud HTTP, la clase `Illuminate\Http\Request` debe ser tipada en el constructor o en el método del controlador.

Código de ejemplo:

```
 <?php

namespace App\Http\Controllers;

/* Here how we illuminate the request class in controller */
use Illuminate\Http\Request;

use Illuminate\Routing\Controller;

class PostController extends Controller
{
 /**
  * Store a new post.
  *
  * @param  Request  $request
  * @return Response
  */
 public function store(Request $request)
 {
     $name = $request->input('post_title');

     /*
     * so typecasting Request class in our method like above avails the
     * HTTP GET/POST/PUT etc method params in the controller to use and
     * manipulate
     */
 }
}
```

### Solicitar instancia con otros parámetros de rutas en el método del controlador

A veces necesitamos aceptar parámetros de ruta, así como acceder a los parámetros de solicitud HTTP. Aún podemos escribir sugerencias de la clase Solicitudes en el controlador de laravel y lograr eso como se explica a continuación.

Por ejemplo, tenemos una ruta que actualiza una publicación determinada como esta (pasa la ID de la entrada i la ruta)

```
Route::put('post/{id}', 'PostController@update');
```

Además, desde que el usuario ha editado otros campos de formulario de edición, estará disponible en Solicitud HTTP

Aquí es cómo acceder a ambos en nuestro método.

```
    public function update(Request $request,$id){
        //This way we have $id param from route and $request as an HTTP Request object

    }
```

Lea Peticiones en línea: https://riptutorial.com/es/laravel/topic/4929/peticiones

# Capítulo 52: Plantillas Blade

## Introducción

Laravel soporta el motor de plantillas Blade fuera de la caja. El motor de plantillas Blade nos permite crear plantillas maestras y plantillas infantiles para cargar contenido de plantillas maestras, podemos tener variables, bucles y declaraciones condicionales dentro del archivo Blade.

## Examples

### Vistas: Introducción

Las vistas, en un patrón MVC, contienen la lógica sobre *cómo* presentar los datos al usuario. En una aplicación web, normalmente se utilizan para generar el resultado HTML que se envía a los usuarios con cada respuesta. De forma predeterminada, las vistas en Laravel se almacenan en el directorio `resources/views` .

Se puede llamar a una vista usando la función de ayuda de `view` :

```
view(string $path, array $data = [])
```

El primer parámetro del ayudante es la ruta a un archivo de vista, y el segundo parámetro es una matriz opcional de datos para pasar a la vista.

Por lo tanto, para llamar a `resources/views/example.php` , usaría:

```
view('example');
```

Puede ver los archivos en subcarpetas dentro del directorio `resources/views` , como `resources/views/parts/header/navigation.php` , usando la notación de puntos: `view('parts.header.navigation');`

Dentro de un archivo de vista, como `resources/views/example.php` , puede incluir tanto HTML como PHP juntos:

```
<html>
    <head>
        <title>Hello world!</title>
    </head>
    <body>
        <h1>Welcome!</h1>
        <p>Your name is: <?php echo $name; ?></p>
    </body>
</html>
```

En el ejemplo anterior (que no utiliza ninguna sintaxis específica de Blade), generamos la variable

`$name` . Para pasar este valor a nuestra vista, pasaríamos una matriz de valores al llamar al ayudante de vista:

```
view('example', ['name' => $name]);
```

o alternativamente, use el ayudante `compact()` . En este caso, la cadena pasada a compact () corresponde al nombre de la variable que queremos pasar a la vista.

```
view('example', compact('name'));
```

**CONVENIO DE NOMBRES PARA VARIABLES DE HOJA**

Mientras se envían datos a la vista. Puede usar el `underscore` para la `variable` múltiples palabras `variable` pero con `-` laravel da error.

Al igual que éste, dará un error (aviso `hyphen ( - )` dentro de la `user-address`

```
view('example',['user-address' => 'Some Address']);
```

La **forma correcta** de hacerlo será

```
view('example', ['user_address' => 'Some Address']);
```

### Estructuras de Control

Blade proporciona una sintaxis conveniente para estructuras de control PHP comunes.

Cada una de las estructuras de control comienza con `@[structure]` y termina con `@[endstructure]` . Observe que dentro de las etiquetas, solo estamos escribiendo HTML normal e incluyendo variables con la sintaxis de Blade.

# Condicionales

## 'Si' declaraciones

```
@if ($i > 10)
    <p>{{ $i }} is large.</p>
@elseif ($i == 10)
    <p>{{ $i }} is ten.</p>
@else
    <p>{{ $i }} is small.</p>
@endif
```

## 'A menos que' declaraciones

(Sintaxis corta para 'si no'.)

```
@unless ($user->hasName())
    <p>A user has no name.</p>
@endunless
```

# Bucles

## 'While' loop

```
@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

## Bucle 'Foreach'

```
@foreach ($users as $id => $name)
    <p>User {{ $name }} has ID {{ $id }}.</p>
@endforeach
```

## 'Forelse' Loop

(Igual que el bucle 'foreach', pero agrega una directiva especial `@empty` , que se ejecuta cuando la expresión de la matriz iterada sobre está vacía, como una forma de mostrar el contenido predeterminado).

```
@forelse($posts as $post)
    <p>{{ $post }} is the post content.</p>
@empty
    <p>There are no posts.</p>
@endforelse
```

---

Dentro de los bucles, estará disponible una variable especial `$loop` , que contiene información sobre el estado del bucle:

| Propiedad | Descripción |
|---|---|
| `$loop->index` | El índice de la iteración del bucle actual (comienza en 0). |
| `$loop->iteration` | La iteración del bucle actual (comienza en 1). |
| `$loop->remaining` | Las iteraciones de bucle restantes. |
| `$loop->count` | El número total de elementos en la matriz que se está iterando. |
| `$loop->first` | Si esta es la primera iteración a través del bucle. |
| `$loop->last` | Si esta es la última iteración a través del bucle. |

| Propiedad | Descripción |
|---|---|
| `$loop->depth` | El nivel de anidamiento del bucle actual. |
| `$loop->parent` | Cuando se encuentra en un bucle anidado, la variable de bucle principal. |

Ejemplo:

```
@foreach ($users as $user)
  @foreach ($user->posts as $post)
        @if ($loop->parent->first)
            This is first iteration of the parent loop.
        @endif
    @endforeach
@endforeach
```

---

Desde Laravel 5.2.22, también podemos usar las directivas `@continue` y `@break`

| Propiedad | Descripción |
|---|---|
| `@continue` | Detenga la iteración actual y comience la siguiente. |
| `@break` | Detener el bucle actual. |

Ejemplo:

```
@foreach ($users as $user)
    @continue ($user->id == 2)
        <p>{{ $user->id }} {{ $user->name }}</p>
    @break ($user->id == 4)
@endforeach
```

Luego ( *asumiendo que más de 5 usuarios están ordenados por ID y no falta ID* ), la página se mostrará

```
1 Dave
3 John
4 William
```

### Haciendo eco de las expresiones PHP

Cualquier expresión PHP entre llaves dobles `{{ $variable }}` será `echo` ed después de haber sido dirigido a través del `e` función auxiliar . (Por lo tanto, los caracteres especiales html ( `<` , `>` , `"` , `'` , `&` ) se reemplazan de manera segura para las entidades html correspondientes.)

## Haciendo eco de una variable

```
{{ $variable }}
```

### Haciendo eco de un elemento en una matriz

```
{{ $array["key"] }}
```

### Haciendo eco de una propiedad de objeto

```
{{ $object->property }}
```

### Haciendo eco del resultado de una llamada de función

```
{{ strtolower($variable) }}
```

## Comprobando la existencia

Normalmente, en PHP, para verificar si una variable está configurada e imprimirla usted haría

- Antes de PHP 7

```
<?php echo isset($variable) ? $variable : 'Default'; ?>
```

- Después de PHP 7 (usando el "operador de unión nula")

```
<?php echo $variable ?? 'Default'; ?>
```

Operador de cuchillas `or` hace esto más fácil:

```
{{ $variable or 'Default' }}
```

## Ecos crudos

Como se mencionó, la sintaxis de llaves dobles `{{ }}` , se filtra a través de la función `htmlspecialchars` de PHP, por seguridad (evitando la inyección maliciosa de HTML en la vista). Si desea evitar este comportamiento, por ejemplo, si está intentando generar un bloque de contenido HTML resultante de una expresión PHP, use la siguiente sintaxis:

```
{!! $myHtmlString !!}
```

Tenga en cuenta que se considera una práctica recomendada utilizar la sintaxis estándar `{{ }}` para escapar de sus datos, a menos que sea absolutamente necesario. Además, cuando haga eco de contenido no confiable (es decir, contenido suministrado por los usuarios de su sitio), debe evitar el uso de `{!! !!}` sintaxis.

## Incluyendo vistas parciales

Con Blade, también puede incluir vistas parciales (llamadas 'parciales') directamente en una página así:

```
@include('includes.info', ['title' => 'Information Station'])
```

El código anterior incluirá la vista en 'views / includes / info.blade.php'. También pasará en una variable `$title` con valor 'Estación de información'.

En general, una página incluida tendrá acceso a cualquier variable a la que tenga acceso la página que llama. Por ejemplo, si tenemos:

```
{{$user}} // Outputs 'abc123'
@include('includes.info')
```

Y 'includes / info.blade.php' tiene lo siguiente:

```
<p>{{$user}} is the current user.</p>
```

Entonces la página se renderizará:

```
abc123
abc123 is the current user.
```

### Incluir cada

A veces, deseará combinar una declaración de `include` con una instrucción de `foreach` , y acceder a las variables desde dentro del bucle foreach en la inclusión. En este caso, use la directiva `@each` de Blade:

```
@each('includes.job', $jobs, 'job')
```

El primer parámetro es la página a incluir. El segundo parámetro es la matriz para iterar sobre. El tercer parámetro es la variable asignada a los elementos de la matriz. La declaración anterior es equivalente a:

```
@foreach($jobs as $job)
    @include('includes.job', ['job' => $job])
@endforeach
```

También puede pasar un cuarto argumento opcional a la directiva `@each` para especificar la vista que se mostrará cuando la matriz esté vacía.

```
@each('includes.job', $jobs, 'job', 'includes.jobsEmpty')
```

## Herencia de diseño

Un diseño es un archivo de vista, que se extiende por otras vistas que inyectan bloques de código en su elemento primario. Por ejemplo:

**parent.blade.php**

```
<html>
    <head>
        <style type='text/css'>
        @yield('styling')
        </style>
    </head>
    <body>
        <div class='main'>
        @yield('main-content')
        </div>
    </body>
</html>
```

**child.blade.php**

```
@extends('parent')

@section('styling')
.main {
    color: red;
}
@stop

@section('main-content')
This is child page!
@stop
```

**otherpage.blade.php:**

```
@extends('parent')

@section('styling')
.main {
    color: blue;
}
@stop

@section('main-content')
This is another page!
@stop
```

Aquí se ven dos páginas secundarias de ejemplo, cada una de las cuales se extiende al padre. Las páginas secundarias definen una `@section` , que se inserta en el padre en la instrucción `@yield` apropiada.

Así que la vista representada por `View::make('child')` dirá "¡ **Esta es una página secundaria!** " En rojo, mientras que `View::make('otherpage')` producirá el mismo html, excepto con el texto " **This is another página!** "en azul en su lugar.

Es común separar los archivos de vista, por ejemplo, tener una carpeta de diseños

específicamente para los archivos de diseño y una carpeta separada para las distintas vistas individuales específicas.

Los diseños están diseñados para aplicar el código que debe aparecer en cada página, por ejemplo, agregando una barra lateral o encabezado, sin tener que escribir toda la plantilla de html en cada vista individual.

Las vistas se pueden ampliar repetidamente, es decir, **page3** can **@extend** `@extend('page2')` , y **page2** can **@extend** `@extend('page1')` .

El comando extender usa la misma sintaxis que se usa para `View::make` y `@include` , por lo que se accede a los `layouts/main/page.blade.php` archivo `layouts/main/page.blade.php` como `layouts.main.page` .

### Compartir datos a todas las vistas.

A veces necesitas configurar los mismos datos en muchas de tus vistas.

# Usando View :: share

```
// "View" is the View Facade
View::share('shareddata', $data);
```

Después de esto, el contenido de `$data` estará disponible en todas las vistas con el nombre `$shareddata` .

`View::share` se suele llamar en un proveedor de servicios, o tal vez en el constructor de un controlador, por lo que los datos solo se compartirán en las vistas devueltas por ese controlador.

# Usando View :: compositor

Los compositores de vistas son devoluciones de llamada o métodos de clase que se llaman cuando se procesa una vista. Si tiene datos que desea vincular a una vista cada vez que se renderiza la vista, un compositor de vistas puede ayudarlo a organizar esa lógica en una única ubicación. Puede vincular directamente la variable a una vista específica o a todas las vistas.

## Compositor basado en el cierre

```
use Illuminate\Support\Facades\View;

// ...

View::composer('*', function ($view) {
    $view->with('somedata', $data);
});
```

## Compositor basado en la clase

```
use Illuminate\Support\Facades\View;

// ...

View::composer('*', 'App\Http\ViewComposers\SomeComposer');
```

Al igual que con `View::share` , es mejor registrar a los compositores en un proveedor de servicios.

Si va con el enfoque de clase de compositor, entonces tendría `App/Http/ViewComposers/SomeComposer.php` con:

```
use Illuminate\Contracts\View\View;

class SomeComposer
{
    public function compose(View $view)
    {
        $view->with('somedata', $data);
    }
}
```

Estos ejemplos usan `'*'` en el registro del compositor. Este parámetro es una cadena que coincide con los nombres de vista para los que se registra el compositor ( `*` es un comodín). También puede seleccionar una vista única (por ejemplo, `'home'` ) de un grupo de rutas en una subcarpeta (por ejemplo, `'users.*'` ).

### Ejecutar código PHP arbitrario

Aunque podría no ser apropiado hacer tal cosa en una vista si pretende separar las preocupaciones estrictamente, la directiva `php` Blade permite una forma de ejecutar código PHP, por ejemplo, para establecer una variable:

```
@php($varName = 'Enter content ')
```

(igual que:)

```
@php
    $varName = 'Enter content ';
@endphp
```

luego:

```
{{ $varName }}
```

Resultado:

> Introducir contenido

Lea Plantillas Blade en línea: https://riptutorial.com/es/laravel/topic/1407/plantillas-blade

# Capítulo 53: Políticas

## Examples

### Creando Políticas

Dado que la definición de toda la lógica de autorización en el `AuthServiceProvider` podría volverse complicada en aplicaciones grandes, Laravel le permite dividir su lógica de autorización en clases de "Política". Las políticas son clases de PHP simples que agrupan la lógica de autorización según el recurso que autorizan.

Puede generar una política utilizando el comando `make:policy` artisan. La política generada se colocará en el directorio de `app/Policies` :

```
php artisan make:policy PostPolicy
```

Lea Políticas en línea: https://riptutorial.com/es/laravel/topic/7344/politicas

# Capítulo 54: Problemas comunes y soluciones rápidas

## Introducción

Esta sección enumera los problemas comunes y las soluciones rápidas que enfrentan los desarrolladores (especialmente los principiantes).

## Examples

### Excepción TokenMisMatch

Obtienes esta excepción principalmente con envíos de formularios. Laravel protege la aplicación de `CSRF` y valida cada solicitud y garantiza que la solicitud se originó desde dentro de la aplicación. Esta validación se realiza utilizando un `token` . Si este token no coincide, se genera esta excepción.

**Arreglo rapido**

Agregue esto dentro de su elemento de formulario. Esto envía `csrf_token` generado por laravel junto con otros datos de formulario para que laravel sepa que su solicitud es válida

```
<input type="hidden" name="_token" value="{{ csrf_token() }}">
```

Lea Problemas comunes y soluciones rápidas en línea:
https://riptutorial.com/es/laravel/topic/9971/problemas-comunes-y-soluciones-rapidas

# Capítulo 55: Programación de tareas

## Examples

### Creando una tarea

Puedes crear una tarea (comando de consola) en Laravel usando Artisan. Desde su línea de comando:

```
php artisan make:console MyTaskName
```

Esto crea el archivo en **app / Console / Commands / MyTaskName.php** . Se verá así:

```
<?php

namespace App\Console\Commands;

use Illuminate\Console\Command;

class MyTaskName extends Command
{
    /**
     * The name and signature of the console command.
     *
     * @var string
     */
    protected $signature = 'command:name';

    /**
     * The console command description.
     *
     * @var string
     */
    protected $description = 'Command description';

    /**
     * Create a new command instance.
     *
     * @return void
     */
    public function __construct()
    {
        parent::__construct();
    }

    /**
     * Execute the console command.
     *
     * @return mixed
     */
    public function handle()
    {
        //
    }
}
```

Algunas partes importantes de esta definición son:

- La propiedad `$signature` es lo que identifica su comando. Podrá ejecutar este comando más adelante a través de la línea de comandos utilizando Artisan ejecutando el `php artisan command:name` (donde `command:name` coincide con la `$signature` su comando)
- La propiedad `$description` es la ayuda / uso de Artisan al lado de su comando cuando está disponible.
- El método `handle()` es donde escribes el código para tu comando.

Eventualmente, su tarea se pondrá a disposición de la línea de comandos a través de Artisan. El `protected $signature = 'command:name';` la propiedad en esta clase es lo que usaría para ejecutarla.

## Haciendo una tarea disponible

Puede hacer que una tarea esté disponible para Artisan y para su aplicación en el archivo **app / Console / Kernel.php** .

La clase `Kernel` contiene una matriz llamada `$commands` que hace que sus comandos estén disponibles para su aplicación.

Agregue su comando a esta matriz, para que esté disponible para Artisan y su aplicación.

```
<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     *
     * @var array
     */
    protected $commands = [
        Commands\Inspire::class,
        Commands\MyTaskName::class // This line makes MyTaskName available
    ];

    /**
     * Define the application's command schedule.
     *
     * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
     * @return void
     */
    protected function schedule(Schedule $schedule)
    {

    }
}
```

Una vez hecho esto, ahora puede acceder a su comando a través de la línea de comandos,

utilizando Artisan. Suponiendo que su comando tiene la propiedad `$signature` establecida en `my:task` puede ejecutar el siguiente comando para ejecutar su tarea:

```
php artisan my:task
```

## Programando tu tarea

Cuando su comando esté disponible para su aplicación, puede usar Laravel para programarlo para que se ejecute en intervalos predefinidos, como lo haría con un CRON.

En el archivo **App / Console / Kernel.php** encontrará un método de `schedule` que puede utilizar para programar su tarea.

```
<?php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    /**
     * The Artisan commands provided by your application.
     *
     * @var array
     */
    protected $commands = [
        Commands\Inspire::class,
        Commands\MyTaskName::class
    ];

    /**
     * Define the application's command schedule.
     *
     * @param  \Illuminate\Console\Scheduling\Schedule  $schedule
     * @return void
     */
    protected function schedule(Schedule $schedule)
    {
        $schedule->command('my:task')->everyMinute();
        // $schedule->command('my:task')->everyFiveMinutes();
        // $schedule->command('my:task')->daily();
        // $schedule->command('my:task')->monthly();
        // $schedule->command('my:task')->sundays();
    }
}
```

Suponiendo que la `$signature` su tarea es `my:task` que puede programar como se muestra arriba, usando el objeto `Schedule $schedule` . Laravel proporciona muchas formas diferentes de programar su comando, como se muestra en las líneas comentadas arriba.

## Configuración del planificador para ejecutar

El planificador se puede ejecutar usando el comando:

```
php artisan schedule:run
```

El programador debe ejecutarse cada minuto para que funcione correctamente. Puede configurarlo creando un trabajo cron con la siguiente línea, que ejecuta el programador cada minuto en segundo plano.

```
* * * * * php /path/to/artisan schedule:run >> /dev/null 2>&1
```

Lea Programación de tareas en línea: https://riptutorial.com/es/laravel/topic/4026/programacion-de-tareas

# Capítulo 56: Pruebas

## Examples

### Introducción

Escribir códigos comprobables es una parte importante de la construcción de un proyecto robusto, fácil de mantener y ágil. El soporte para el marco de pruebas más utilizado de PHP, PHPUnit , está integrado directamente en Laravel. PHPUnit se configura mediante el archivo `phpunit.xml` , que reside en el directorio raíz de cada nueva aplicación Laravel.

El directorio de `tests` , también en el directorio raíz, contiene los archivos de prueba individuales que contienen la lógica para probar cada parte de su aplicación. Por supuesto, es tu responsabilidad como desarrollador escribir estas pruebas a medida que construyes tu aplicación, pero Laravel incluye un archivo de ejemplo, `ExampleTest.php` , para ponerte en marcha.

```
<?php

use Illuminate\Foundation\Testing\WithoutMiddleware;
use Illuminate\Foundation\Testing\DatabaseMigrations;
use Illuminate\Foundation\Testing\DatabaseTransactions;

class ExampleTest extends TestCase
{
    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testBasicExample()
    {
        $this->visit('/')
             ->see('Laravel 5');
    }
}
```

En el método `testBasicExample()` , visitamos la página de índice del sitio y nos aseguramos de ver el texto `Laravel 5` algún lugar de esa página. Si el texto no está presente, la prueba fallará y generará un error.

### Prueba sin middleware y con una base de datos nueva.

Para hacer que los artesanos migren una base de datos nueva antes de ejecutar pruebas, `use DatabaseMigrations` . Además, si desea evitar el middleware como Auth, `use WithoutMiddleware` .

```
<?php

use Illuminate\Foundation\Testing\WithoutMiddleware;
use Illuminate\Foundation\Testing\DatabaseMigrations;
```

```
class ExampleTest extends TestCase
{
    use DatabaseMigrations, WithoutMiddleware;

    /**
     * A basic functional test example.
     *
     * @return void
     */
    public function testExampleIndex()
    {
        $this->visit('/protected-page')
         ->see('All good');
    }
}
```

## Transacciones de base de datos para conexión de base de datos mutliple

`DatabaseTransactions` rasgo `DatabaseTransactions` permite a las bases de datos revertir todo el cambio durante las pruebas. Si desea revertir varias bases de datos, debe establecer `$connectionsToTransact` propiedades `$connectionsToTransact`

```
use Illuminate\Foundation\Testing\DatabaseMigrations;

class ExampleTest extends TestCase
{
     use DatabaseTransactions;

     $connectionsToTransact =["mysql","sqlite"] //tell Laravel which database need to rollBack

    public function testExampleIndex()
    {
        $this->visit('/action/parameter')
         ->see('items');
    }
}
```

## Configuración de prueba, utilizando en la base de datos de memoria

La siguiente configuración garantiza que el marco de prueba (PHPUnit) utiliza `:memory:` base de datos.

**config / database.php**

```
'connections' => [

    'sqlite_testing' => [
        'driver'   => 'sqlite',
        'database' => ':memory:',
        'prefix'   => '',
    ],
    .
    .
    .
```

**./phpunit.xml**

```
    .
    .
    .
    </filter>
    <php>
        <env name="APP_ENV" value="testing"/>
        <env name="APP_URL" value="http://example.dev"/>
        <env name="CACHE_DRIVER" value="array"/>
        <env name="SESSION_DRIVER" value="array"/>
        <env name="QUEUE_DRIVER" value="sync"/>
        <env name="DB_CONNECTION" value="sqlite_testing"/>
    </php>
</phpunit>
```

## Configuración

El archivo phpunit.xml es el archivo de configuración predeterminado para las pruebas y ya está configurado para las pruebas con PHPUnit.

El entorno de prueba predeterminado `APP_ENV` se define como `testing` con una `array` es el controlador de caché `CACHE_DRIVER` . Con esta configuración, no se retendrán datos (sesión / caché) durante la prueba.

Para ejecutar pruebas en un entorno específico como homestead, los valores predeterminados se pueden cambiar a:

```
<env name="DB_HOST" value="192.168.10.10"/>
<env name="DB_DATABASE" value="homestead"/>
<env name="DB_USERNAME" value="homestead"/>
<env name="DB_PASSWORD" value="secret"/>
```

O para usar una base de datos temporal *en la memoria* :

```
<env name="DB_CONNECTION" value="sqlite"/>
<env name="DB_DATABASE" value=":memory:"/>
```

Una última nota a tener en cuenta a partir de la documentación de Laravel :

> ¡Asegúrese de borrar su caché de configuración usando el comando `config:clear` Artisan antes de ejecutar sus pruebas!

Lea Pruebas en línea: https://riptutorial.com/es/laravel/topic/1249/pruebas

# Capítulo 57: Servicios

## Examples

### Introducción

Laravel permite el acceso a una variedad de clases llamadas Servicios. Algunos servicios están disponibles de forma inmediata, pero puede crearlos usted mismo.

Un servicio se puede utilizar en varios archivos de la aplicación, como controladores. Imaginemos un servicio `OurService` implementando un método `getNumber()` que devuelve un número aleatorio:

```
class SomeController extends Controller {

    public function getRandomNumber()
    {
        return app(OurService::class)->getNumber();
    }
}
```

Para crear un servicio, solo se necesita crear una nueva clase:

```
# app/Services/OurService/OurService.php

<?php
namespace App\Services\OurService;

class OurService
{
    public function getNumber()
    {
        return rand();
    }
}
```

En este momento, ya podría utilizar este servicio en un controlador, pero tendría que crear una instancia de un nuevo objeto cada vez que lo necesitaría:

```
class SomeController extends Controller {

    public function getRandomNumber()
    {
        $service = new OurService();
        return $service->getNumber();
    }

    public function getOtherRandomNumber()
    {
        $service = new OurService();
        return $service->getNumber();
    }
}
```

Es por eso que el siguiente paso es registrar su servicio en el **contenedor de servicios** . Cuando registra su Servicio en el Contenedor de Servicios, no necesita crear un nuevo objeto cada vez que lo necesite.

Para registrar un servicio en el contenedor de servicios, debe crear un **proveedor de servicios** . Este proveedor de servicios puede:

1. Registre su servicio en el contenedor de servicios con *el método de registro* )
2. Inyectar otros servicios en su servicio (dependencias) con *el método de arranque*

Un **proveedor de servicios** es una clase que extiende la clase abstracta `Illuminate\Support\ServiceProvider` . Necesita implementar el método de `register()` para **registrar** un Servicio en el **Contenedor de Servicios** :

```
# app/Services/OurService/OurServiceServiceProvider.php

<?php
namespace App\Services\OurService;

use Illuminate\Support\ServiceProvider;

class OurServiceServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->singleton('OurService', function($app) {
            return new OurService();
        });
    }
}
```

Todos los **proveedores de servicios** se guardan en una matriz en `config/app.php` . Por lo tanto, debemos registrar nuestro Proveedor de servicios en esta matriz:

```
return [

    ...

    'providers' => [

        ...

        App\Services\OurService\OurServiceServiceProvider::class,

        ...

    ],

    ...

];
```

Ahora podemos acceder a nuestro Servicio en un controlador. Tres posibilidades:

1. Inyección de dependencia:

```
<?php
namespace App\Http\Controllers;

use App\Services\OurService\OurService;

class SomeController extends Controller
{
    public function __construct(OurService $our_service)
    {
        dd($our_service->getNumber());
    }
}
```

2. Acceso a través de la `app()` helper:

```
<?php
namespace App\Http\Controllers;

use App\Services\OurService\OurService;

class SomeController extends Controller
{
    public function getRandomNumber()
    {
        return app('OurService')->getNumber();
    }
}
```

Laravel proporciona fachadas, clases imaginarias que puede utilizar en todos sus proyectos y reflejar un Servicio. Para acceder a su servicio más fácilmente, puede crear una fachada:

```
<?php
namespace App\Http\Controllers;

use Randomisator;

class SomeController extends Controller
{
    public function getRandomNumber()
    {
        return Randomisator::getNumber();
    }

}
```

Para crear una nueva fachada, debe crear una nueva clase que amplíe `Illuminate\Support\Facades\Facade` . Esta clase necesita implementar el método `getFacadeAccessor()` y devolver el nombre de un servicio registrado por un **proveedor de servicios** :

```
# app/Services/OurService/OurServiceFacade.php

<?php
namespace App\Services\OurService;

use Illuminate\Support\Facades\Facade;
```

```
class OurServiceFacade extends Facade
{
    protected static function getFacadeAccessor()
    {
        return 'OurService';
    }
}
```

También debe registrar su fachada en `config/app.php` :

```
return [

    ...

    'aliases' => [

        ....

        'Randomisator' => App\Services\OurService\OurServiceFacade::class,
    ],

];
```

La Fachada ahora es accesible en cualquier parte de tu proyecto.

Si desea acceder a su servicio desde sus vistas, puede crear una función auxiliar. Laravel se envía con algunas funciones de ayuda fuera de la caja, como la función `auth()` o la función `view()` . Para crear una función auxiliar, cree un nuevo archivo:

```
# app/Services/OurService/helpers.php

if (! function_exists('randomisator')) {
    /**
     * Get the available OurService instance.
     *
     * @return \App\ElMatella\FacebookLaravelSdk
     */
    function randomisator()
    {
        return app('OurService');
    }
}
```

También necesita registrar este archivo, pero en su archivo `composer.json` :

```
{

    ...

    "autoload": {
        "files": [
            "app/Services/OurService/helpers.php"
        ],

        ...
    }
```

```
}
```

Ahora puedes usar este ayudante en una vista:

```
<h1>Here is a random number: {{ randomisator()->getNumber() }}</h1>
```

Lea Servicios en línea: https://riptutorial.com/es/laravel/topic/1907/servicios

# Capítulo 58: Servicios

## Examples

### Enlace de una interfaz a la implementación

En un método de `register` proveedor de servicios podemos vincular una interfaz a una implementación:

```
public function register()
{
    App::bind( UserRepositoryInterface::class, EloquentUserRepository::class );
}
```

A partir de ahora, cada vez que la aplicación necesite una instancia de `UserRepositoryInterface` , Laravel inyectará automáticamente una nueva instancia de `EloquentUserRepository` :

```
//this will get back an instance of EloquentUserRepository
$repo = App::make( UserRepositoryInterface:class );
```

### Atar una instancia

Podemos usar el contenedor de servicios como un registro vinculando una instancia de un objeto en él y recuperarlo cuando lo necesitemos:

```
// Create an instance.
$john = new User('John');

// Bind it to the service container.
App::instance('the-user', $john);

// ...somewhere and/or in another class...

// Get back the instance
$john = App::make('the-user');
```

### Enlazar un Singleton al contenedor de servicio

Podemos unir una clase como Singleton:

```
public function register()
{
    App::singleton('my-database', function()
    {
        return new Database();
    });
}
```

De esta manera, la primera vez que se solicite una instancia de `'my-database'` al contenedor de

servicios, se creará una nueva instancia. Todas las solicitudes sucesivas de esta clase recuperarán la primera instancia creada:

```
//a new instance of Database is created
$db = App::make('my-database');

//the same instance created before is returned
$anotherDb = App::make('my-database');
```

## Introducción

El **contenedor de servicios** es el objeto principal de la aplicación. Se puede utilizar como un contenedor de inyección de dependencias y un registro para la aplicación mediante la definición de enlaces en los proveedores de servicios

**Los proveedores de servicios** son clases en las que definimos la forma en que nuestras clases de servicio se crearán a través de la aplicación, arrancarán su configuración y unirán las interfaces a las implementaciones.

**Los servicios** son clases que envuelven una o más tareas correlacionadas de lógica

## Uso del contenedor de servicios como un contenedor de inyección de dependencias

Podemos usar el contenedor de servicios como un contenedor de inyección de dependencias vinculando el proceso de creación de objetos con sus dependencias en un punto de la aplicación

Supongamos que la creación de un `PdfCreator` necesita dos objetos como dependencias; Cada vez que necesitamos crear una instancia de `PdfCreator` , debemos pasar estas dependencias al constructor che. Al utilizar el contenedor de servicios como DIC, definimos la creación de `PdfCreator` en la definición de enlace, tomando la dependencia requerida directamente del contenedor de servicios:

```
App:bind('pdf-creator', function($app) {

    // Get the needed dependencies from the service container.
    $pdfRender = $app->make('pdf-render');
    $templateManager = $app->make('template-manager');

    // Create the instance passing the needed dependencies.
    return new PdfCreator( $pdfRender, $templateManager );
});
```

Luego, en cada punto de nuestra aplicación, para obtener un nuevo `PdfCreator` , simplemente podemos hacer:

```
$pdfCreator = App::make('pdf-creator');
```

Y el contenedor de servicios creará una nueva instancia, junto con las dependencias necesarias para nosotros.

Lea Servicios en línea: https://riptutorial.com/es/laravel/topic/1908/servicios

# Capítulo 59: Siembra

## Observaciones

La base de datos de base de datos le permite insertar datos, datos de prueba generales en su base de datos. Por defecto, hay una clase `DatabaseSeeder` en la `database/seeds` .

Las sembradoras se pueden hacer con

```
php artisan db:seed
```

O si solo quieres procesar una sola clase.

```
php artisan db:seed --class=TestSeederClass
```

Al igual que con todos los comandos artesanales, tiene acceso a una amplia gama de métodos que se pueden encontrar en la documentación de la API.

## Examples

### Insertando datos

Hay varias formas de insertar datos:

## Usando el DB Facade

```
public function run()
{
    DB::table('users')
        ->insert([
            'name' => 'Taylor',
            'age'  => 21
        ]);
}
```

## A través de la creación de un modelo

```
public function run()
{
    $user = new User;
    $user->name = 'Taylor';
    $user->save();
}
```

## Usando el método de crear

```
public function run()
{
    User::create([
        'name' => 'Taylor',
        'age'  => 21
    ]);
}
```

# Usando la fábrica

```
public function run()
{
    factory(App\User::class, 10)->create();
}
```

### Sembrando && eliminando datos antiguos y reiniciando auto-incremento

```
public function run()
{
    DB::table('users')->delete();
    DB::unprepared('ALTER TABLE users AUTO_INCREMENT=1;');
    factory(App\User::class, 200)->create();
}
```

Consulte el ejemplo Persistente para obtener más información sobre cómo insertar / actualizar datos.

### Llamando a otros sembradores

Dentro de su clase `DatabaseSeeder` puede llamar a otros sembradores

```
$this->call(TestSeeder::class)
```

Esto le permite guardar un archivo donde puede encontrar fácilmente sus sembradoras. Tenga en cuenta que debe prestar atención al orden de sus llamadas con respecto a las restricciones de clave externa. No puede hacer referencia a una tabla que aún no existe.

### Creando una Sembradora

Para crear sembradoras, puede usar el comando `make:seeder` Artisan. Todas las sembradoras generadas se colocarán en la `database/seeds` directorio de `database/seeds` .

```
$ php artisan make:seeder MoviesTableSeeder
```

Las sembradoras generadas contendrán un método: `run` . Puede insertar datos en su base de datos en este método.

```
<?php
```

```
use Illuminate\Database\Seeder;
use Illuminate\Database\Eloquent\Model;

class MoviesTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        App\Movie::create([
            'name' => 'A New Hope',
            'year' => '1977'
        ]);

        App\Movie::create([
            'name' => 'The Empire Strikes Back',
            'year' => '1980'
        ]);
    }
}
```

Generalmente deseará llamar a todos sus sembradores dentro de la clase `DatabaseSeeder` .

Una vez que haya terminado de escribir las sembradoras, use el comando `db:seed` . Esto `run` función de ejecución de `DatabaseSeeder` .

```
$ php artisan db:seed
```

También puede especificar que se ejecute una clase de sembradora específica para ejecutar individualmente usando la opción `--class` .

```
$ php artisan db:seed --class=UserSeeder
```

Si desea revertir y volver a ejecutar todas las migraciones, y luego reinicie:

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

> El comando `migrate:refresh --seed` es un acceso directo a estos 3 comandos:
>
> ```
> $ php artisan migrate:reset     # rollback all migrations
> $ php artisan migrate           # run migrations
> $ php artisan db:seed           # run seeders
> ```

## Resiembra segura

Es posible que desee volver a sembrar su base de datos sin afectar sus semillas creadas previamente. Para este propósito, puede usar `firstOrCreate` en su sembradora:

```
EmployeeType::firstOrCreate([
```

```
    'type' => 'manager',
]);
```

Luego puedes sembrar la base de datos:

```
php artisan db:seed
```

---

Más adelante, si desea agregar otro tipo de empleado, simplemente puede agregar ese nuevo en el mismo archivo:

```
EmployeeType::firstOrCreate([
    'type' => 'manager',
]);
EmployeeType::firstOrCreate([
    'type' => 'secretary',
]);
```

Y siembra de nuevo tu base de datos sin problemas:

```
php artisan db:seed
```

Observe que en la primera llamada está recuperando el registro pero sin hacer nada con él.

Lea Siembra en línea: https://riptutorial.com/es/laravel/topic/3272/siembra

# Capítulo 60: Siembra de base de datos

## Examples

### Corriendo una sembradora

Puede agregar su nueva Sembradora a la clase DatabaseSeeder.

```
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(UserTableSeeder::class);
    }
```

Para ejecutar una sembradora de bases de datos, use el comando Artisan

```
php artisan db:seed
```

Esto ejecutará la clase DatabaseSeeder. También puede elegir usar la opción `--class=` para especificar manualmente qué sembradora ejecutar.

* Tenga en cuenta que es posible que tenga que ejecutar el dumpautoload del compositor si no se puede encontrar su clase Seeder. Esto suele suceder si crea manualmente una clase sembradora en lugar de usar el comando artisan.

### Creando una semilla

Las semillas de la base de datos se almacenan en el directorio / database / seeds. Puedes crear una semilla usando un comando Artesano.

```
php artisan make:seed UserTableSeeder
```

Alternativamente, puede crear una nueva clase que amplíe `Illuminate\Database\Seeder` . La clase debe una función pública llamada `run()` .

### Insertando datos usando una sembradora

Puede hacer referencia a modelos en una sembradora.

```
use DB;
use App\Models\User;

class UserTableSeeder extends Illuminate\Database\Seeder{
```

```
    public function run(){
        # Remove all existing entrie
        DB::table('users')->delete() ;
        User::create([
            'name' => 'Admin',
            'email' => 'admin@example.com',
            'password' => Hash::make('password')
        ]);

    }
}
```

## Insertando datos con un Model Factory

Es posible que desee utilizar Fábricas de modelos dentro de sus semillas. Esto creará 3 nuevos usuarios.

```
use App\Models\User;

class UserTableSeeder extends Illuminate\Database\Seeder{

    public function run(){
        factory(User::class)->times(3)->create();
    }
}
```

También puede definir campos específicos en su siembra como una contraseña, por ejemplo. Esto creará 3 usuarios con la misma contraseña.

```
factory(User::class)->times(3)->create(['password' => '123456']);
```

## Siembra con MySQL Dump

Siga el ejemplo anterior de crear una semilla. Este ejemplo utiliza un volcado de MySQL para generar una tabla en la base de datos del proyecto. La tabla debe ser creada antes de la siembra.

```
<?php

use Illuminate\Database\Seeder;

class UserTableSeeder extends Seeder
{

    /**
     * Run the database seeds.
     *
     * @return void
     */
     public function run()
     {
         $sql = file_get_contents(database_path() . '/seeds/users.sql');

         DB::statement($sql);
      }
```

```
}
```

Nuestro $ sql será el contenido de nuestro volcado users.sql. El volcado debe tener una instrucción INSERT INTO. Dependerá de usted dónde almacene sus vertederos. En el ejemplo anterior, se almacena en el directorio del proyecto `\database\seeds` . Utilizando la función helper de laravel `database_path()` y agregando el directorio y el nombre de archivo del volcado.

```
INSERT INTO `users` (`id`, `name`, `email`, `password`, `remember_token`, `created_at`,
`updated_at`) VALUES
(1, 'Jane', 'janeDoe@fakemail.com', 'superSecret', NULL, '2016-07-21 00:00:00', '2016-07-21
00:00:00'),
(2, 'John', 'johnny@fakemail.com', 'sup3rS3cr3t', NULL, '2016-07-21 00:00:00', '2016-07-21
00:00:00');
```

`DB::statement($sql)` ejecutará las inserciones una vez que se ejecuta la Sembradora. Como en los ejemplos anteriores, puede colocar el `UserTableSeeder` en la clase `DatabaseSeeder` proporcionada por laravel:

```
<?php

use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        $this->call(UserTableSeeder::class);
    }
}
```

y ejecute desde CLI en el directorio de proyectos `php artisan db:seed` . O puede ejecutar la Sembradora para una sola clase usando `php artisan db:seed --class=UsersTableSeeder`

## Usando faker y ModelFactories para generar semillas

### 1) VÍA SIMPLE BÁSICA

Las aplicaciones basadas en bases de datos a menudo necesitan datos pre-sembrados en el sistema para fines de prueba y demostración.

Para hacer tales datos, primero cree la clase sembradora

**ProductTableSeeder**

```
use Faker\Factory as Faker;
use App\Product;

class ProductTableSeeder extends DatabaseSeeder {
```

```
public function run()
{
    $faker = $this->getFaker();

    for ($i = 0; $i < 10; $i++)
    {
        $name =         $faker->word;
        $image =        $faker->imageUrl;

        Modelname::create([
            'name' => $name,
            'image' => $image,
        ]);
      }
    }
 }
```

Para llamar a una clase de ser capaz de ejecutar una sembradora, debe llamarla desde la clase DatabaseSeeder, simplemente al pasar el nombre de la sembradora que desea ejecutar:

utilizar Illuminate \ Database \ Seeder;

```
class DatabaseSeeder extends Seeder {

    protected $faker;

    public function getFaker() {
        if (empty($this->faker)) {
            $faker = Faker\Factory::create();
            $faker->addProvider(new Faker\Provider\Base($faker));
            $faker->addProvider(new Faker\Provider\Lorem($faker));
        }
        return $this->faker = $faker;
    }
    public function run() {
        $this->call(ProductTableSeeder::class);
    }
}
```

No se olvide de ejecutar `$ composer dump-autoload` después de crear la Sembradora, ya que el compositor no los carga automáticamente (a menos que haya creado la sembradora con el comando artesanal `$ php artisan make:seeder Name` )

Ahora está listo para sembrar ejecutando este comando `php artisan db:seed`

**2) UTILIZANDO Fábricas de modelos**

En primer lugar, debe definir un conjunto predeterminado de atributos para cada modelo en `App/database/factories/ModelFactory.php`

Tomando como ejemplo un modelo de usuario, este es el aspecto de ModelFactory

```
$factory->define(App\User::class, function (Faker\Generator $faker) {
    return [
        'name' => $faker->name,
```

```
        'email' => $faker->email,
        'password' => bcrypt(str_random(10)),
        'remember_token' => str_random(10),
    ];
});
```

Ahora `php artisan make:seeder UsersTableSeeder` una tabla sembradora `php artisan make:seeder UsersTableSeeder`

Y agrega esto

```
public function run()
{
    factory(App\User::class, 100)->create()
}
```

A continuación, agregue esto al `DatabaseSeeder`

```
public function run()
{
    $this->call(UsersTableSeeder::class);
}
```

Esto sembrará la tabla con 100 registros.

Lea Siembra de base de datos en línea: https://riptutorial.com/es/laravel/topic/1118/siembra-de-base-de-datos

# Capítulo 61: Sistema de archivos / almacenamiento en la nube

## Examples

### Configuración

El archivo de configuración del sistema de archivos se encuentra en `config/filesystems.php` . Dentro de este archivo puede configurar todos sus "discos". Cada disco representa un controlador de almacenamiento y una ubicación de almacenamiento particulares. Las configuraciones de ejemplo para cada controlador compatible se incluyen en el archivo de configuración. Entonces, simplemente modifique la configuración para reflejar sus preferencias de almacenamiento y credenciales.

Antes de usar los controladores S3 o Rackspace, deberá instalar el paquete apropiado a través de Composer:

- Amazon S3: `league/flysystem-aws-s3-v2 ~1.0`
- Rackspace: `league/flysystem-rackspace ~1.0`

Por supuesto, puede configurar tantos discos como desee, e incluso puede tener varios discos que utilicen el mismo controlador.

Cuando use el controlador local, tenga en cuenta que todas las operaciones de archivos son relativas al directorio raíz definido en su archivo de configuración. De forma predeterminada, este valor se establece en el `storage/app directory` . Por lo tanto, el siguiente método almacenaría un archivo en `storage/app/file.txt` :

```
Storage::disk('local')->put('file.txt', 'Contents');
```

### Uso básico

La fachada de `Storage` se puede utilizar para interactuar con cualquiera de sus discos configurados. Alternativamente, puede escribir una sugerencia de tipo del `Illuminate\Contracts\Filesystem\Factory` en cualquier clase que se resuelva a través del contenedor de servicio Laravel.

**Recuperando un disco particular**

```
$disk = Storage::disk('s3');

$disk = Storage::disk('local');
```

**Determinar si existe un archivo**

```
$exists = Storage::disk('s3')->exists('file.jpg');
```

### Métodos de llamada en el disco predeterminado

```
if (Storage::exists('file.jpg'))
{
    //
}
```

### Recuperar el contenido de un archivo

```
$contents = Storage::get('file.jpg');
```

### Configuración de los contenidos de un archivo

```
Storage::put('file.jpg', $contents);
```

### Antepuesto a un archivo

```
Storage::prepend('file.log', 'Prepended Text');
```

### Anexar a un archivo

```
Storage::append('file.log', 'Appended Text');
```

### Borrar un archivo

```
Storage::delete('file.jpg');

Storage::delete(['file1.jpg', 'file2.jpg']);
```

### Copiar un archivo a una nueva ubicación

```
Storage::copy('old/file1.jpg', 'new/file1.jpg');
```

### Mover un archivo a una nueva ubicación

```
Storage::move('old/file1.jpg', 'new/file1.jpg');
```

### Obtener el tamaño del archivo

```
$size = Storage::size('file1.jpg');
```

### Obtener el último tiempo de modificación (UNIX)

```
$time = Storage::lastModified('file1.jpg');
```

### Obtener todos los archivos dentro de un directorio

```
$files = Storage::files($directory);

// Recursive...
$files = Storage::allFiles($directory);
```

### Obtener todos los directorios dentro de un directorio

```
$directories = Storage::directories($directory);

// Recursive...
$directories = Storage::allDirectories($directory);
```

### Crear un directorio

```
Storage::makeDirectory($directory);
```

### Eliminar un directorio

```
Storage::deleteDirectory($directory);
```

## Sistemas de archivos personalizados

La integración Flysystem de Laravel proporciona controladores para varios "controladores" listos para usar; Sin embargo, Flysystem no se limita a estos y tiene adaptadores para muchos otros sistemas de almacenamiento. Puede crear un controlador personalizado si desea usar uno de estos adaptadores adicionales en su aplicación Laravel. No te preocupes, no es demasiado difícil!

Para configurar el sistema de archivos personalizado, deberá crear un proveedor de servicios como `DropboxFilesystemServiceProvider` . En el método de `boot` del proveedor, puede inyectar una instancia del `Illuminate\Contracts\Filesystem\Factory` y llamar al método `extend` de la instancia inyectada. Alternativamente, puede utilizar el método de `extend` la fachada del `Disk` .

El primer argumento del método `extend` es el nombre del controlador y el segundo es un cierre que recibe las variables `$app` y `$config` . El cierre de resolución debe devolver una instancia de `League\Flysystem\Filesystem` .

> Nota: la variable $ config ya contendrá los valores definidos en `config/filesystems.php` para el disco especificado. Ejemplo de Dropbox

```
<?php namespace App\Providers;

use Storage;
use League\Flysystem\Filesystem;
use Dropbox\Client as DropboxClient;
use League\Flysystem\Dropbox\DropboxAdapter;
use Illuminate\Support\ServiceProvider;

class DropboxFilesystemServiceProvider extends ServiceProvider {
```

```
    public function boot()
    {
        Storage::extend('dropbox', function($app, $config)
        {
            $client = new DropboxClient($config['accessToken'], $config['clientIdentifier']);

            return new Filesystem(new DropboxAdapter($client));
        });
    }

    public function register()
    {
        //
    }

}
```

## Creando un enlace simbólico en un servidor web usando SSH

En la documentación de Laravel, se debe crear un enlace simbólico (enlace simbólico o enlace flexible) de public / storage a storage / app / public para que los archivos sean accesibles desde la web.

*(ESTE PROCEDIMIENTO CREARÁ UN ENLACE SIMBÓLICO DENTRO DEL DIRECTORIO DE PROYECTOS LARAVEL)*

Estos son los pasos sobre cómo puede crear un enlace simbólico en su servidor web Linux utilizando el cliente SSH:

1. Conéctese e inicie sesión en su servidor web utilizando el cliente SSH (por ejemplo, PUTTY).

2. Enlace de **almacenamiento / aplicación / público** a **público / almacenamiento** usando la sintaxis

   ```
   ln -s target_path link_path
   ```

   Ejemplo (en el directorio de archivos de CPanel)

   ```
   ln -s /home/cpanel_username/project_name/storage/app/public
   /home/cpanel_sername/project_name/public/storage
   ```

*(Se creará una carpeta denominada **almacenamiento** para vincular la ruta con un indicador **>>>** en el icono de la carpeta).*

Lea Sistema de archivos / almacenamiento en la nube en línea:
https://riptutorial.com/es/laravel/topic/3040/sistema-de-archivos---almacenamiento-en-la-nube

# Capítulo 62: Solicitud de dominio cruzado

## Examples

### Introducción

A veces necesitamos la solicitud de dominios cruzados para nuestras API en laravel. Necesitamos agregar encabezados apropiados para completar con éxito la solicitud de dominio cruzado. Por lo tanto, debemos asegurarnos de que los encabezados que estemos agregando sean precisos, de lo contrario nuestra API se volverá vulnerable. Para agregar encabezados necesitamos agregar middleware en laravel que agregará los encabezados apropiados y reenviará las solicitudes.

### CorsHeaders

```
<?php

namespace laravel\Http\Middleware;

class CorsHeaders
{
  /**
   * This must be executed _before_ the controller action since _after_ middleware isn't
executed when exceptions are thrown and caught by global handlers.
   *
   * @param $request
   * @param \Closure $next
   * @param string [$checkWhitelist] true or false Is a string b/c of the way the arguments
are supplied.
   * @return mixed
   */
  public function handle($request, \Closure $next, $checkWhitelist = 'true')
  {
    if ($checkWhitelist == 'true') {
      // Make sure the request origin domain matches one of ours before sending CORS response
headers.
      $origin = $request->header('Origin');
      $matches = [];
      preg_match('/^(https?:\/\/)?([a-zA-Z\d]+\.)*(?<domain>[a-zA-Z\d-\.]+\.[a-z]{2,10})$/',
$origin, $matches);

      if (isset($matches['domain']) && in_array($matches['domain'], ['yoursite.com']) {
        header('Access-Control-Allow-Origin: ' . $origin);
        header('Access-Control-Expose-Headers: Location');
        header('Access-Control-Allow-Credentials: true');

        // If a preflight request comes then add appropriate headers
        if ($request->method() === 'OPTIONS') {
          header('Access-Control-Allow-Methods: GET, POST, PUT, OPTIONS, DELETE, PATCH');
          header('Access-Control-Allow-Headers: ' . $request->header('Access-Control-Request-
Headers'));
            // 20 days
          header('Access-Control-Max-Age: 1728000');
        }
```

```
        }
    } else {
        header('Access-Control-Allow-Origin: *');
    }

    return $next($request);
  }
}
```

Lea Solicitud de dominio cruzado en línea: https://riptutorial.com/es/laravel/topic/7425/solicitud-de-dominio-cruzado

# Capítulo 63: Token Mismatch Error en AJAX

## Introducción

He analizado que la proporción de error de TokenMismatch es muy alta. Y este error se produce debido a algunos errores tontos. Hay muchas razones por las que los desarrolladores están cometiendo errores. Estos son algunos de los ejemplos, es decir, No _token en los encabezados, No _token pasó los datos cuando se usa Ajax, el problema de permisos en la ruta de almacenamiento, una ruta de almacenamiento de sesión no válida.

## Examples

### Configurar token en el encabezado

Establece el token en `<head>` de tu `default.blade.php` .

```
<meta name="csrf-token" content="{{csrf_token()}}">
```

Agregue `ajaxSetup` en la parte superior de su script, que será accesible a todas partes. Esto establecerá los encabezados en cada llamada `ajax`

```
$.ajaxSetup({
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
});
```

### Establecer token en etiqueta

Agregue la siguiente función a su etiqueta `<form>` . Esta función generará un campo oculto llamado `_token` y valor rellenado con el token.

```
{{csrf_field()}}
```

Agregue la función `csrf_token ()` a su `_token` oculto en el atributo de valor. Esto generará solo cadena encriptada.

`<input type="hidden" name="_token" value="{{csrf_token()}}"/>` .

### Compruebe la ruta de almacenamiento de la sesión y el permiso

Aquí asumo que la url de la aplicación del proyecto es `APP_URL=http://project.dev/ts/toys-store`

1. Establezca el permiso de escritura en `storage_path('framework/sessions')` la carpeta.

2. Verifique la ruta de su proyecto de laravel `'path' => '/ts/toys-store',` la raíz de su proyecto de laravel.
3. Cambie el nombre de su cookie `'cookie' => 'toys-store',`

```
return [
    'driver' => env('SESSION_DRIVER', 'file'),
    'lifetime' => 120,
    'expire_on_close' => false,
    'encrypt' => false,
    'files' => storage_path('framework/sessions'),
    'connection' => null,
    'table' => 'sessions',
    'lottery' => [2, 100],
    'cookie' => 'toys-store',
    'path' => '/ts/toys-store',
    'domain' => null,
    'secure' => false,
    'http_only' => true,
];
```

## Utilice el campo _token en Ajax

Hay muchas formas de enviar `_token` en una llamada AJAX

1. Obtenga todo el valor del campo de entrada dentro de la etiqueta `<form>` usando `var formData = new FormData($("#cart-add")[0]);`
2. Utilice `$("form").serialize();` o `$("form").serializeArray();`
3. Añadir `_token` manualmente en los `data` de Ajax. utilizando `$('meta[name="csrf-token"]').attr('content')` o `$('input[name="_token"]').val()` .
4. Podemos establecer como encabezado en una llamada Ajax particular como el código de abajo.

```
$.ajax({
    url: $("#category-add").attr("action"),
    type: "POST",
    data: formData,
    processData: false,
    contentType: false,
    dataType: "json",
    headers: {
        'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
    }
});
```

Lea Token Mismatch Error en AJAX en línea: https://riptutorial.com/es/laravel/topic/10656/token-mismatch-error-en-ajax

# Capítulo 64: usar campos alias en Eloquent

Lea usar campos alias en Eloquent en línea: https://riptutorial.com/es/laravel/topic/7927/usar-campos-alias-en-eloquent

# Capítulo 65: Validación

## Parámetros

| Parámetro | Detalles |
|---|---|
| necesario | El campo es obligatorio |
| algunas veces | Ejecute las verificaciones de validación en un campo solo si ese campo está presente en la matriz de entrada |
| correo electrónico | La entrada es un correo electrónico válido. |
| valor máximo | El valor de entrada debe estar por debajo del valor máximo |
| único: db_table_name | El valor de entrada debe ser único en el nombre de la tabla de la base de datos proporcionada |
| aceptado | Sí / Encendido / 1 verdadero, útil para verificar TOS |
| active_url | Debe ser una URL válida de acuerdo con checkdnsrr |
| después *: fecha* | El campo bajo validación debe proporcionar un valor después de la fecha dada |
| alfa | El campo bajo validación debe ser enteramente caracteres alfabéticos. |
| alpha_dash | El campo bajo validación puede tener caracteres alfanuméricos, así como guiones y guiones bajos. |
| alfa_num | El campo bajo validación debe ser enteramente caracteres alfanuméricos. |
| formación | Debe ser una matriz de PHP |
| antes *: fecha* | El campo debe ser un valor bajo la fecha dada. |
| entre: min, max | El valor de entrada debe estar entre el valor mínimo (mínimo) y máximo (máximo) |
| booleano | El campo bajo validación debe poder ser lanzado como un booleano. Las entradas aceptadas son `true` , `false` , `1` , `0` , `"1"` y `"0"` . |
| confirmado | El campo bajo validación debe tener un campo coincidente de `foo_confirmation` . Por ejemplo, si el campo bajo validación es `password` , un campo coincidente `password_confirmation` debe estar presente en la entrada. |

| Parámetro | Detalles |
| --- | --- |
| fecha | El campo bajo validación debe ser una fecha válida de acuerdo con la función PHP de strtotime . |
| entero | El campo bajo validación debe ser un entero. |
| cuerda | El campo bajo validación debe ser un tipo de cadena . |

# Examples

## Ejemplo básico

Puede validar los datos de solicitud utilizando el método de `validate` (disponible en el Controlador base, proporcionado por el rasgo `ValidatesRequests` ).

Si las reglas pasan, su código seguirá ejecutándose normalmente; sin embargo, si la validación falla, una respuesta de error que contiene los errores de validación se devolverá automáticamente:

- para solicitudes de formulario HTML típicas, el usuario será redirigido a la página anterior, con el formulario manteniendo los valores enviados
- para solicitudes que esperan una respuesta JSON, se generará una respuesta HTTP con el código 422

Por ejemplo, en su `UserController` , puede estar guardando un nuevo usuario en el método de `store` , que necesitaría validación antes de guardar.

```
/**
 * @param  Request  $request
 * @return Response
 */
public function store(Request $request) {
    $this->validate($request, [
        'name' => 'required',
        'email' => 'email|unique:users|max:255'
    ],
    // second array of validation messages can be passed here
    [
        'name.required' => 'Please provide a valid name!',
        'email.required' => 'Please provide a valid email!',
    ]);

    // The validation passed
}
```

En el ejemplo anterior, validamos que el campo de `name` existe con un valor no vacío. En segundo lugar, verificamos que el campo de `email` tenga un formato de correo electrónico válido, sea único en la tabla de la base de datos "usuarios" y tenga una longitud máxima de 255 caracteres.

> El `|` El carácter (pipe) combina diferentes reglas de validación para un campo.

En ocasiones, es posible que desee dejar de ejecutar reglas de validación en un atributo después del primer error de validación. Para hacerlo, asigne la regla de `bail` al atributo:

```
$this->validate($request, [
    'name' => 'bail|required',
    'email' => 'email|unique:users|max:255'
]);
```

La lista completa de reglas de validación disponibles se puede encontrar en la sección de parámetros a continuación .

## Validación de Array

La validación de los campos de entrada de la matriz es muy simple.

Supongamos que tiene que validar cada nombre, correo electrónico y nombre de padre en una matriz dada. Podrías hacer lo siguiente:

```
$validator = \Validator::make($request->all(), [
    'name.*'       => 'required',
    'email.*'      => 'email|unique:users',
    'fatherName.*' => 'required'
]);

if ($validator->fails()) {
    return back()->withInput()->withErrors($validator->errors());
}
```

Laravel muestra los mensajes por defecto para la validación. Sin embargo, si desea mensajes personalizados para campos basados en matrices, puede agregar el siguiente código:

```
[
    'name.*' => [
        'required' => 'Name field is required',
    ],
    'email.*' => [
        'unique'   => 'Unique Email is required',
    ],
    'fatherName.*' => [
        'required' => 'Father Name required',
    ]
]
```

Tu código final se verá así:

```
$validator = \Validator::make($request->all(), [
    'name.*'       => 'required',
    'email.*'      => 'email|unique:users',
    'fatherName.*' => 'required',
], [
    'name.*'       => 'Name Required',
    'email.*'      => 'Unique Email is required',
    'fatherName.*' => 'Father Name required',
]);
```

```
if ($validator->fails()) {
    return back()->withInput()->withErrors($validator->errors());
}
```

## Otros enfoques de validación

### 1) Formulario de Validación de Solicitud

Puede crear una "solicitud de formulario" que puede contener la lógica de autorización, las reglas de validación y los mensajes de error para una solicitud particular en su aplicación.

El comando `make:request` Artisan CLI genera la clase y la coloca en el directorio `app/Http/Requests` :

```
php artisan make:request StoreBlogPostRequest
```

El método de `authorize` se puede anular con la lógica de autorización para esta solicitud:

```
public function authorize()
{
    return $this->user()->can('post');
}
```

El método de las `rules` se puede anular con las reglas específicas para esta solicitud:

```
public function rules()
{
    return [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ];
}
```

El método de `messages` se puede anular con los mensajes específicos para esta solicitud:

```
public function messages()
{
    return [
        'title.required' => 'A title is required',
        'title.unique' => 'There is another post with the same title',
        'title.max' => 'The title may not exceed :max characters',
        'body.required' => 'A message is required',
    ];
}
```

Para validar la solicitud, escriba la clase de solicitud específica en el método del controlador correspondiente. Si la validación falla, se enviará una respuesta de error.

```
public function store(StoreBlogPostRequest $request)
{
    // validation passed
}
```

### 2) Creación manual de validadores

Para una mayor flexibilidad, puede crear un validador manualmente y manejar la validación fallida directamente:

```
<?php
namespace App\Http\Controllers;

use Validator;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class PostController extends Controller
{
    public function store(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'title' => 'required|unique:posts|max:255',
            'body' => 'required',
        ]);

        if ($validator->fails()) {
            return redirect('post/create')
                    ->withErrors($validator)
                    ->withInput();
        }

        // Store the blog post...
    }
}
```

### 2) Crear reglas con fluidez.

Ocasionalmente, puede que necesite crear reglas únicas sobre la marcha, ya que trabajar con el método `boot()` dentro de un Proveedor de Servicios puede ser superior, ya que en Laravel 5.4 puede crear nuevas reglas con fluidez usando la clase de `Rule` .

Como ejemplo, vamos a trabajar con `UserRequest` para cuando desee insertar o actualizar un usuario. Por ahora queremos que se requiera un nombre y la dirección de correo electrónico debe ser única. El problema con el uso de la regla `unique` es que si está editando un usuario, es posible que mantengan el mismo correo electrónico, por lo que debe excluir al usuario actual de la regla. El siguiente ejemplo muestra cómo puede hacer esto fácilmente utilizando la nueva clase de `Rule` .

```
<?php
namespace App\Http\Requests;
use Illuminate\Foundation\Http\FormRequest;
use Illuminate\Http\Request;
use Illuminate\Validation\Rule;

class UserRequest 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(Request $request)
    {
        $id = $request->route()->getParameter('user');

        return [
            'name'           =>  'required',

            // Notice the value is an array and not a string like usual
            'email'         =>  [
                'required',
                Rule::unique('users')->ignore($id)
            ]
        ];
    }
}
```

## Clase de solicitud de formulario único para POST, PUT, PATCH

Siguiendo el ejemplo de 'Validación de solicitud de formulario' , la misma clase de solicitud puede usarse para `POST` , `PUT` , `PATCH` para que no tenga que crear otra clase utilizando las validaciones iguales / similares. Esto es útil si tiene atributos en su tabla que son únicos.

```
/**
 * Get the validation rules that apply to the request.
 *
 * @return array
 */
public function rules() {
    switch($this->method()) {
        case 'GET':
        case 'DELETE':
            return [];
        case 'POST':
            return [
                'name'     => 'required|max:75|unique',
                'category' => 'required',
                'price'    => 'required|between:0,1000',
            ];
        case 'PUT':
        case 'PATCH':
            return [
                'name'     => 'required|max:75|unique:product,name,' . $this->product,
                'category' => 'required',
                'price'    => 'required|between:0,1000',
            ];
        default:break;
    }
```

```
}
```

A partir de la parte superior, nuestra instrucción de conmutación examinará el tipo de método de la solicitud ( `GET` , `DELETE` , `POST` , `PUT` , `PATCH` ).

Dependiendo del método se devolverá la matriz de reglas definida. Si tiene un campo que es único, como el campo de `name` en el ejemplo, debe especificar un ID particular para que la validación se ignore.

```
'field_name' => 'unique:table_name,column_name,' . $idToIgnore`
```

Si tiene una clave primaria etiquetada de otra manera que no sea `id` , especificará la columna de clave primaria como el cuarto parámetro.

```
'field_name' => 'unique:table_name,column_name,' . $idToIgnore . ',primary_key_column'
```

En este ejemplo, usamos `PUT` y pasamos a la ruta ( `admin/products/{product}` ) el valor de la identificación del producto. Así que `$this->product` será igual al `id` para ignorar.

Ahora sus reglas de validación `PUT|PATCH` y `POST` no necesitan ser las mismas. Define tu lógica que se ajuste a tus necesidades. Esta técnica le permite reutilizar los mensajes personalizados que puede haber definido dentro de la Clase de solicitud de formulario personalizada.

### Error de mensajes

## Personalizando mensajes de error

Los archivos `/resources/lang/[lang]/validation.php` contienen los mensajes de error que utilizará el validador. Puede editarlos según sea necesario.

La mayoría de ellos tienen marcadores de posición que se reemplazarán automáticamente al generar el mensaje de error.

Por ejemplo, en `'required' => 'The :attribute field is required.'` , el marcador de posición del `:attribute` será reemplazado por el nombre del campo (alternativamente, también puede personalizar el valor de visualización de cada campo en la matriz de `attributes` en el mismo archivo).

*Ejemplo*

configuración del mensaje:

```
'required' => 'Please inform your :attribute.',
//...
'attributes => [
    'email' => 'E-Mail address'
]
```

reglas:

```
`email' => `required`
```

mensaje de error resultante:

> "Por favor, informe a su dirección de correo electrónico".

---

## Personalizando mensajes de error dentro de una clase de Solicitud

La clase de Solicitud tiene acceso a un método de `messages()` que debería devolver una matriz, esto se puede usar para anular mensajes sin tener que ir a los archivos lang. Por ejemplo, si tenemos una validación personalizada de `file_exists` , puede enviar los siguientes mensajes.

```
class SampleRequest extends Request {

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'image' =>  'required|file_exists'
        ];
    }

    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    public function messages()
    {
        return [
            'image.file_exists' =>  'That file no longer exists or is invalid'
        ];
    }

}
```

---

## Mostrando mensajes de error

Los errores de validación se transmiten a la sesión y también están disponibles en la variable `$errors`

, que se comparte automáticamente en todas las vistas.

Ejemplo de visualización de los errores en una vista Blade:

```
@if (count($errors) > 0)
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

## Reglas de validación personalizadas

Si desea crear una regla de validación personalizada, puede hacerlo, por ejemplo, en el método de `boot` de un proveedor de servicios, a través de la fachada Validator.

```
<?php
namespace App\Providers;

use Illuminate\Support\ServiceProvider;
use Validator;

class AppServiceProvider extends ServiceProvider
{
    public function boot()
    {
        Validator::extend('starts_with', function($attribute, $value, $parameters, $validator)
{
            return \Illuminate\Support\Str::startsWith($value, $parameters[0]);
        });

        Validator::replacer('starts_with', function($message, $attribute, $rule, $parameters)
{
            return str_replace(':needle', $parameters[0], $message);
        });
    }
}
```

El método de `extend` toma una cadena que será el nombre de la regla y una función que, a su vez, pasará el nombre del atributo, el valor que se valida, una matriz de los parámetros de la regla y la instancia del validador, y debe devolver si La validación pasa. En este ejemplo, estamos comprobando si la cadena de valor comienza con una subcadena dada.

El mensaje de error para esta regla personalizada se puede configurar como es habitual en el archivo `/resources/lang/[lang]/validation.php` , y puede contener marcadores de posición, por ejemplo, para valores de parámetros:

```
'starts_with' => 'The :attribute must start with :needle.'
```

El método de `replacer` toma una cadena que es el nombre de la regla y una función que, a su vez, pasará el mensaje original (antes de reemplazarlo), el nombre del atributo, el nombre de la regla y

una matriz de los parámetros de la regla. y debe devolver el mensaje después de reemplazar los marcadores de posición según sea necesario.

Usa esta regla como cualquier otra:

```
$this->validate($request, [
    'phone_number' => 'required|starts_with:+'
]);
```

Lea Validación en línea: https://riptutorial.com/es/laravel/topic/1310/validacion

# Creditos

| S. No | Capítulos | Contributors |
|---|---|---|
| 1 | Empezando con Laravel | alepeino, Alphonsus, boroboris, Colin Herzog, Community, Ed Rands, Evgeniy Maynagashev, Gaurav, Imam Assidiqqi, James, Ketan Akbari, Kovah, Lance Pioch, Marek Skiba, Martin Bean, Misa Lazovic, nyedidikeke, Oliver Adria, Prakash, rap-2-h, Ru Chern Chong, SeinopSys, Tatranskymedved, Tim |
| 2 | Artesano | Alessandro Bassi, Gaurav, Harshal Limaye, Himanshu Raval, Imam Assidiqqi, Kaspars, Laurel, Rubens Mariuzzo, Safoor Safdar, Sagar Naliyapara, SeinopSys |
| 3 | Autenticación | Aykut CAN, Imam Assidiqqi |
| 4 | Autorización | Daniel Verem |
| 5 | Ayudante de cámara | David Lartey, Dov Benyomin Sohacheski, Imam Assidiqqi, Misa Lazovic, Ru Chern Chong, Shog9 |
| 6 | Ayudantes | aimme |
| 7 | Base de datos | A. Raza, adam, caoglish, Ian, Iftikhar uddin, Imam Assidiqqi, liamja, Panagiotis Koursaris, RamenChef, Rubens Mariuzzo, Sanzeeb Aryal, Vucko |
| 8 | Cajero | littleswany, RamenChef |
| 9 | Cambiar el comportamiento de enrutamiento predeterminado en Laravel 5.2.31 + | Frank Provost |
| 10 | Clase CustomException en Laravel | ashish bansal |
| 11 | Colas | Alessandro Bassi, Kyslik |
| 12 | Colecciones | A. Raza, Alessandro Bassi, Alex Harris, bhill77, caoglish, Dummy Code, Gras Double, Ian, Imam Assidiqqi, Josh Rumbut, Karim Geiger, matiaslauriti, Nicklas Kevin Frank, Ozzy, rap-2-h, simonhamp, Vucko |

| | | |
|---|---|---|
| 13 | Conexiones DB múltiples en Laravel | 4444, A. Raza, Rana Ghosh |
| 14 | Constantes | Mubashar Iqbal, Oscar David, Zakaria Acharki |
| 15 | Controladores | Ru Chern Chong |
| 16 | Correo | Yohanan Baruchel |
| 17 | Eliminar público de la URL en laravel | A. Raza, Rana Ghosh, ultrasamad |
| 18 | Elocuente | aimme, alepeino, Alessandro Bassi, Alex Harris, Alfa, Alphonsus, andretzermias, andrewtweber, Andrey Lutskevich, aynber, Buckwheat, Casper Spruit, Dancia, Dipesh Poudel, Ian, Imam Assidiqqi, James, James, jedrzej.kurylo, John Slegers, Josh Rumbut, Kaspars, Ketan Akbari, KuKeC, littleswany, Lykegenes, Maantje, Mahmood, Marco Aurélio Deleu, marcus.ramsden, Marek Skiba, Martin Bean, matiaslauriti, MM2, Nicklas Kevin Frank, Niklas Modess, Nyan Lynn Htut, patricus, Pete Houston, Phroggyy, Prisoner Raju, RamenChef, rap-2-h, Rubens Mariuzzo, Sagar Naliyapara, Samsquanch, Sergio Guillen Mantilla, Tim, tkausl, whoan, Yasin Patel |
| 19 | Elocuente: Modelo | Aeolingamenfel, alepeino, Alex Harris, Imam Assidiqqi, John Slegers, Kaspars, littleswany, Marco Aurélio Deleu, marcus.ramsden, Marek Skiba, matiaslauriti, Nicklas Kevin Frank, Samsquanch, Tim |
| 20 | Elocuente: Relación | Advaith, aimme, Alex Harris, Alphonsus, bhill77, Imam Assidiqqi , Ketan Akbari, Phroggyy, rap-2-h, Ru Chern Chong, Zulfiqar Tariq |
| 21 | Eloquent: Accessors & Mutators | Diego Souza, Kyslik |
| 22 | Empezando con laravel-5.3 | A. Raza, Advaith, Community, davejal, Deathstorm, Manish, Matthew Beckman, Robin Dirksen, Shital Jachak |
| 23 | Encuadernación de modelos de ruta | A. Raza, GiuServ, Vikash |
| 24 | Enlaces útiles | Jakub Kratina |
| 25 | Enrutamiento | A. Raza, alepeino, Alessandro Bassi, Alex Juchem, beznez, Dwight, Ilker Mutlu, Imam Assidiqqi, jedrzej.kurylo, Kyslik, Milan Maharjan, Rubens Mariuzzo, SeinopSys, Vucko |
| 26 | Estructura de directorios | Kaspars, Moppo, RamenChef |

| | | |
|---|---|---|
| 27 | Eventos y oyentes | Bharat Geleda, matiaslauriti, Nauman Zafar |
| 28 | Formulario de solicitud (s) | Bookeater, Ian, John Roca, Kyslik, RamenChef |
| 29 | Función de ayuda personalizada | Ian, Luceos, rap-2-h, Raunak Gupta |
| 30 | Fundamentos básicos | A. Raza |
| 31 | Guía de instalación | Advaith, Amarnasan, aynber, Community, davejal, Dov Benyomin Sohacheski, Imam Assidiqqi, PaladiN, rap-2-h, Ru Chern Chong |
| 32 | HTML y Form Builder | alepeino, Casper Spruit, Himanshu Raval, Prakash |
| 33 | Implementar la aplicación Laravel 5 en alojamiento compartido en un servidor Linux | Donkarnash, Gayan, Imam Assidiqqi, Kyslik, PassionInfinite, Pete Houston, rap-2-h, Ru Chern Chong, Stojan Kukrika, ultrasamad |
| 34 | Instalación | A. Raza, alepeino, Alphonsus, Black, boroboris, Gaurav, Imam Assidiqqi, James, Ketan Akbari, Lance Pioch, Marek Skiba, Martin Bean, nyedidikeke, PaladiN, Prakash, rap-2-h, Ru Chern Chong, Sagar Naliyapara, SeinopSys, Tim |
| 35 | Integración de Sparkpost con Laravel 5.4 | Alvin Chettiar |
| 36 | Introducción a laravel-5.2. | A. Raza, ashish bansal, Community, Edward Palen, Ivanka Todorova, Shubhamoy |
| 37 | Introducción a laravel-5.3. | Ian |
| 38 | Laravel Docker | Dov Benyomin Sohacheski |
| 39 | Las macros en la relación elocuente | Alex Casajuana, Vikash |
| 40 | Manejo de errores | Isma, Kyslik, RamenChef, Rubens Mariuzzo |
| 41 | marco del lumen | maksbd19 |
| 42 | Middleware | Alex Harris, Kaspars, Kyslik, Moppo, Pistachio |

| | | |
|---|---|---|
| 43 | Migraciones de base de datos | Chris, Chris White, Hovsep, hschin, Iftikhar uddin, Imam Assidiqqi, Kaspars, liamja, littleswany, mnoronha, Nauman Zafar, Panagiotis Koursaris, Paulo Freitas, Vucko |
| 44 | Mundano | Jonathon, Marco Aurélio Deleu |
| 45 | Nombrar archivos al cargar con Laravel en Windows | Donkarnash, RamenChef |
| 46 | Observador | matiaslauriti, Szenis |
| 47 | Paginación | Himanshu Raval, Iftikhar uddin |
| 48 | Paquetes de vacaciones en Laravel | Casper Spruit, Imam Assidiqqi, Ketan Akbari, rap-2-h, Ru Chern Chong, Tosho Trajanov |
| 49 | Permisos de almacenamiento | A. Raza |
| 50 | Peticiones | Ian, Jerodev, RamenChef, Rubens Mariuzzo |
| 51 | Plantillas Blade | A. Raza, agleis, Akshay Khale, alepeino, Alessandro Bassi, Benubird, cbaconnier, Christophvh, Imam Assidiqqi, matiaslauriti, Nauman Zafar, rap-2-h, Safoor Safdar, Tosho Trajanov, yogesh |
| 52 | Políticas | Tosho Trajanov |
| 53 | Problemas comunes y soluciones rápidas | Nauman Zafar |
| 54 | Programación de tareas | Jonathon |
| 55 | Pruebas | Alessandro Bassi, Brayniverse, caoglish, Julian Minde, Kyslik, rap-2-h, Sven |
| 56 | Servicios | A. Raza, El_Matella |
| 57 | Siembra | A. Raza, Alphonsus, Ian, Imam Assidiqqi, Kyslik, SupFrost, whoan |
| 58 | Siembra de base de datos | Achraf Khouadja, Andrew Nolan, Dan Johnson, Isma, Kyslik, Marco Aurélio Deleu |
| 59 | Sistema de archivos / almacenamiento en la nube | Imam Assidiqqi, Nitish Kumar, Paulo Laxamana |

| | | |
|---|---|---|
| 60 | Solicitud de dominio cruzado | Imam Assidiqqi, Suraj |
| 61 | Token Mismatch Error en AJAX | Pankaj Makwana |
| 62 | usar campos alias en Eloquent | MM2 |
| 63 | Validación | A. Raza, alepeino, Alessandro Bassi, Alex Harris, Andrew Nolan , happyhardik, Himanshu Raval, Ian, Iftikhar uddin, John Slegers, Marco Aurélio Deleu, matiaslauriti, rap-2-h, Rubens Mariuzzo, Safoor Safdar, Sagar Naliyapara, Stephen Leppik, sun, Vucko |