



UA.MASTER MOVILES

MÁSTER UNIVERSITARIO EN DESARROLLO DE SOFTWARE
PARA DISPOSITIVOS MÓVILES

PROGRAMACIÓN HIPERMEDIA PARA DISPOSITIVOS MÓVILES

Laravel 5 – Paquetes, Rest y Curl

- Instalación de paquetes
- Controladores RESTful
- Pruebas con cURL
- Transformar datos a JSON o array
 - Respuestas especiales
- Autenticación HTTP básica
 - Pruebas con cURL

- Laravel permite instalar de forma sencilla “paquetes” para complementar su funcionalidad.
- Para instalar un paquete usamos el comando de composer:

```
$ composer require <nombre-del-paquete-a-instalar>
```

- También podemos editar el fichero “composer.json” de nuestro proyecto, añadir el paquete en su sección “require” y por último ejecutar el comando:

```
$ composer update
```

- Esta última opción nos permitirá una mayor configuración.

- La lista de todos los paquetes disponibles la podemos encontrar en <https://packagist.org>
- Algunos de los más utilizados son:

Nombre	Descripción	URL
Debugbar	Barra de depuración de Laravel	https://github.com/barryvdh/laravel-debugbar
IDE Helper	Helper para IDEs	https://github.com/barryvdh/laravel-ide-helper
Permission	Gestión de permisos y roles	https://github.com/spatie/laravel-permission
MongoDB	Extensión para soportar MongoDB	https://github.com/jenssegers/laravel-mongodb
Alert	Notificaciones	https://github.com/bpocallaghan/alert
Former	Automatización de formularios	https://github.com/formers/former
Image	Manipulación de imágenes	https://github.com/Intervention/image
Stapler	Manipulación de ficheros	https://github.com/CodeSleeve/laravel-stapler
User Agent	Obtener info. del <i>user agent</i>	https://github.com/jenssegers/agent
Sitemap	Generación de <i>Sitemaps</i>	https://github.com/spatie/laravel-sitemap
Excel	Trabajar con Excel y CSV	https://github.com/Maatwebsite/Laravel-Excel
DOM PDF	Trabajar con PDF	https://github.com/barryvdh/laravel-dompdf

- Después de instalar un paquete tendremos que modificar el fichero de configuración “`config/app.php`” para añadir las rutas.
- Por ejemplo, instalar un paquete de notificaciones:
 1. Editar el fichero “`composer.json`” y en su sección “`require`” añadir la línea:

```
"bpocallaghan/alert": "1.*"
```
 2. Ejecutar el comando: `$ composer update`
 3. Actualizar la configuración “`config/app.php`”.
 - En la sección “`providers`” añadir:

```
Bpocallaghan\Alert\AlertServiceProvider::class,
```
 - En la sección “`aliases`” añadir:

```
'Alert' => Bpocallaghan\Alert\Facades\Alert::class,
```
 4. Actualizar el *middleware* si fuera necesario.

- Después de instalar un nuevo paquete y añadir la configuración ya podremos usarlo desde cualquier parte de la aplicación.
 - Solo tendremos que importar su uso con: “`use Alert;`”
- Por ejemplo, con el paquete de notificaciones del ejemplo podremos:
 - Añadir notificaciones de los siguientes tipos desde un controlador:

```
Alert::info('Título', 'Info message');  
Alert::success('Título', 'Success message');  
Alert::warning('Título', 'Warning message');  
Alert::error('Título', 'Error message');
```

- Mostrar las notificaciones en la vista:

```
@include('alert::alert')
```

- Laravel incorpora un tipo especial de controlador, llamado controlador de recuso (“*resource controller*”), que facilita la construcción de controladores tipo “*RESTful*”.
- Para utilizarlo simplemente tenemos que:
 - Usar el comando de Artisan ``make:controller`` con la opción “`--resource`”.
 - Crear las rutas en ``routes/web.php`` con “`Route::resource`”.
- Por ejemplo, para crear un controlador RESTful para fotos:
 1. Creamos el controlador con:

```
php artisan make:controller PhotoController --resource
```
 2. Añadimos las rutas al fichero de rutas con:

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

Al añadir una ruta con:

```
“Route::resource('photo', 'PhotoController');”
```

se crean automáticamente todas las rutas RESTful asociadas:

Verbo	Ruta	Acción	Controlador / método
GET	/photo	index	PhotoController@index
GET	/photo/create	create	PhotoController@create
POST	/photo	store	PhotoController@store
GET	/photo/{resource}	show	PhotoController@show
GET	/photo/{resource}/edit	edit	PhotoController@edit
PUT/PATCH	/photo/{resource}	update	PhotoController@update
DELETE	/photo/{resource}	destroy	PhotoController@destroy

- Si no queremos declarar todas las rutas RESTful podemos usar ``only`` o ``except``, por ejemplo:

```
Route::resource('photo', 'PhotoController',  
                ['only' => ['index', 'show']]);  
  
Route::resource('photo', 'PhotoController',  
                ['except' => ['destroy']]);
```

- Los métodos que no usemos los podremos borrar del controlador.

Si queremos definir rutas adicionales para un controlador de recursos RESTful las tenemos que añadir al fichero de rutas `routes/web.php` **antes de la declaración del recurso**, por ejemplo:

```
// Ruta adicional
Route::get('photos/popular', 'PhotoController@getPopular');

// Controlador de recursos
Route::resource('photos', 'PhotoController');
```

- Para definir una API tenemos que utilizar el fichero de rutas `“routes/api.php”` en lugar de `“routes/web.php”`.
- Las rutas se definen en ambos de la misma forma y además aparecerán juntas en el listado de rutas, pero tienen varias diferencias:
 - Middleware que utilizan: el de API carga menos filtros (ver `“app/Http/Kernel.php”`).
 - Prefijo de las rutas: a las rutas almacenadas en `“routes/api.php”` se les añadirá el prefijo `“api”`.
- La respuesta de una API también es muy distinta: no tendrá que devolver una vista en HTML sino el valor de respuesta directamente en texto plano o en formato JSON o XML.

- Para probar una API lo podemos hacer fácilmente utilizando el comando ``curl`` desde consola.
- Por ejemplo, para realizar una petición tipo GET a una URL simplemente tenemos que hacer:

```
$ curl -i http://localhost/recurso
```

```
HTTP/1.1 200 OK
```

```
Transfer-Encoding: chunked
```

```
Date: Fri, 27 Jul 2012 05:11:00 GMT
```

```
Content-Type: text/plain
```

```
¡Hola Mundo!
```

- La opción ``-i`` indica que se **muestren las cabeceras** de respuesta.

- Opcionalmente, al hacer la petición podemos **indicar las cabeceras** con el parámetro `-H`.
- Por ejemplo, para solicitar datos en formato JSON tenemos que hacer:

```
$ curl -i -H "Accept: application/json" http://localhost/recurso
```

```
HTTP/1.1 200 OK
Date: Fri, 27 Jul 2012 05:12:32 GMT
Cache-Control: max-age=42
Content-Type: application/json
Content-Length: 27
```

```
{
  "text": "¡Hola Mundo!"
}
```

- Para realizar peticiones tipo POST, PUT o DELETE lo tenemos que indicar con la opción “-X <tipo>”.
- Para añadir parámetros usamos la opción “-d <params>”.
- Por ejemplo:

```
$ curl -i -H "Accept: application/json" -X POST -d  
"name=javi&phone=800999800" http://localhost/users
```

```
$ curl -i -H "Accept: application/json" -X PUT -d  
"name=pedro" http://localhost/users/1
```

```
$ curl -i -H "Accept: application/json" -X DELETE http://  
localhost/users/1
```

- Para añadir más de una cabecera tenemos que indicar varias veces la opción `-H`:

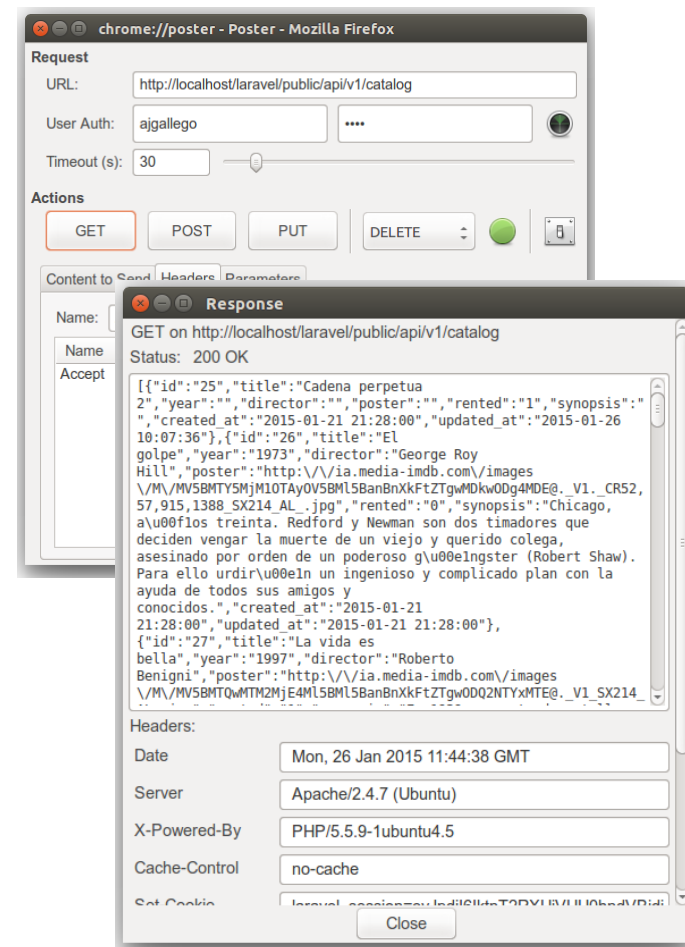
```
$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" http://localhost/resource
```

```
$ curl -i -H "Accept: application/xml" -H "Content-Type: application/xml" http://localhost/resource
```

- Ejemplo: petición tipo POST con datos en formato JSON y que espera respuesta en formato JSON:

```
$ curl -i -H "Accept: application/json" -H "Content-Type: application/json" -X POST -d '{"title":"x","year":"x"}' http://localhost/resource
```

- Existen muchos programas para realizar pruebas de este tipo.
- En los navegadores Firefox y Chrome podemos añadir extensiones o *plugins* para ello.
- En Firefox podemos utilizar:
 - RESTClient:
<https://addons.mozilla.org/en-US/firefox/addon/restclient/>
 - Rested:
<https://addons.mozilla.org/en-US/firefox/addon/rested>
- En Chrome podéis encontrar en su market varias extensiones buscando por “API REST”:
<https://chrome.google.com/webstore/category/apps>



- Laravel incluye métodos para transformar fácilmente el resultado obtenido de una consulta a formato JSON o a formato array.
- Por ejemplo:

```
$user = User::first();  
$arrayUsuario = $user->toArray() ;  
$jsonUsuario = $user->toJson() ;  
  
// O todo un conjunto de datos:  
$arrayUsuarios = User::all()->toArray() ;  
$jsonUsuarios = User::all()->toJson() ;
```

- Al realizar la transformación se incluirán los datos de las relaciones que se hayan cargado al hacer la consulta.

- En ocasiones nos interesará ocultar determinados atributos en la conversión.
- Para hacer esto tenemos que definir el campo protegido `hidden` de nuestro modelo con el array de atributos a ocultar:

```
class User extends Model
{
    protected $hidden = ['id', 'password'];

    // O también podemos indicar solamente aquellos
    // que queremos mostrar con:
    // protected $visible = ['name', 'address'];
}
```

- Para devolver como **respuesta de un controlador** datos en formato JSON tenemos que utilizar el método `response() -> json`.
- Este método además de realizar la conversión asigna las **cabeceras de la respuesta**.
- Por ejemplo:

```
$usuarios = User::all();  
return response() -> json( $usuarios );
```

- También podemos utilizarlo para devolver variables, arrays, etc.:

```
return response() -> json( [ 'name' => 'Steve',  
                             'state' => 'CA' ] );
```

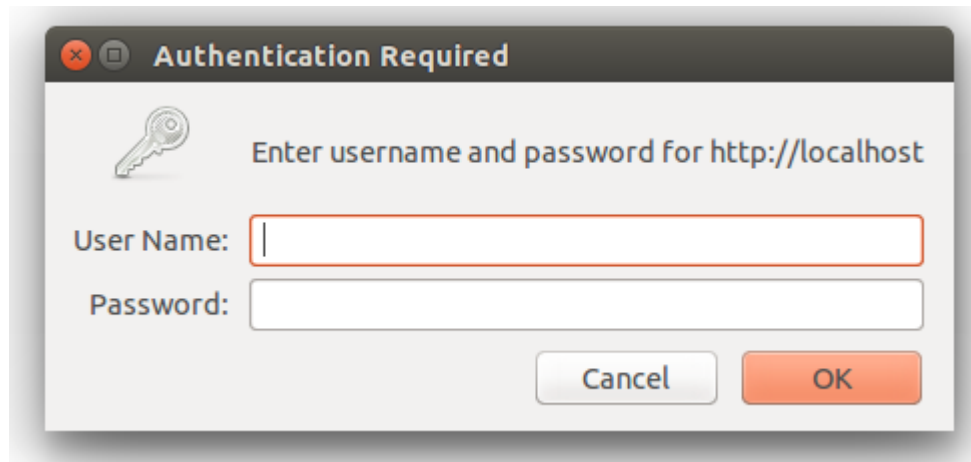
- También podemos especificar el **código de la respuesta** como segundo parámetro del método “`response() -> json`”.
- Por ejemplo, si queremos indicar que ha sucedido algún error, además del JSON podemos indicar el código de estado 500:

```
return response() -> json(  
    ['error'=>true,  
    'msg'=>'Error al procesar la petición'],  
    500 );
```

- La lista completa de los códigos que podemos utilizar la podéis encontrar en:

http://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP

- Este sistema proporciona una forma rápida de identificar a los usuarios sin crear una página de login.
- Cuando se accede a través de la web se mostrará una ventana emergente para solicitar los datos de acceso:



- Este sistema se suele utilizar para proteger las rutas de una API.
- Las credenciales se tendrán que enviar en la cabecera de la petición.
- Para proteger una ruta usando este sistema simplemente tenemos que añadir el *middleware* o filtro llamado ``auth.basic``:

```
Route::get('profile', function() {  
    // Zona de acceso restringido  
})->middleware('auth.basic');
```

- Una vez superada la autenticación básica se crea la sesión del usuario y en cliente se almacenaría una *cookie* con el identificador de la sesión.
- Por defecto este filtro utiliza la columna ``email`` de la tabla de usuarios.

- Si **no** queremos que la sesión se mantenga y que se soliciten las credenciales en cada petición podemos usar un filtro **sin estado**.
- **Importante:** este filtro no viene implementado por defecto. Si lo queremos utilizar tendremos que crear un nuevo *middleware*:

```
$ php artisan make:middleware AuthenticateOnceWithBasicAuth
```

- A continuación registramos el nuevo *middleware* en el fichero “app/Http/Kernel.php” dentro del array “routeMiddleware” con:

```
'auth.basic.once' => \App\Http\Middleware\AuthenticateOnceWithBasicAuth::class,
```

- Completamos el fichero “AuthenticateOnceWithBasicAuth.php” que se habrá creado en la carpeta “app/Http/Middleware”:

```
<?php
namespace App\Http\Middleware;
use Illuminate\Support\Facades\Auth;
use Closure;

class AuthenticateOnceWithBasicAuth {
    public function handle($request, Closure $next) {
        return Auth::onceBasic() ?: $next($request);
    }
}
```

- Ahora ya podemos utilizar el filtro para proteger nuestra API:

```
Route::get('api/user', function() {
    // Zona de acceso restringido
})->middleware('auth.basic.once');
```


- Si intentamos acceder a una ruta protegida mediante autenticación básica utilizando los comando de *cURL* que hemos visto obtendremos el siguiente error:

```
HTTP/1.1 401 Unauthorized
```

- *cURL* permite indicar el usuario y contraseña añadiendo el parámetro `-u` o también `--user` (equivalente):

```
$ curl -u username:password http://localhost/recurso
```

- Si solamente indicamos el usuario (y no el password) se nos solicitará inmediatamente y además al introducirlo no se verá escrito en la pantalla.

¿PREGUNTAS?