



# UA.MASTER MOVILES

MÁSTER UNIVERSITARIO EN DESARROLLO DE SOFTWARE  
PARA DISPOSITIVOS MÓVILES

## PROGRAMACIÓN HIPERMEDIA PARA DISPOSITIVOS MÓVILES

Laravel 2 - Controladores, filtros y formularios

1. Controladores
2. *Middleware* o Filtros
3. Rutas avanzadas
4. Redirecciones
5. Formularios

- Permiten separar mucho mejor el código y crear clases (controladores) que agrupen toda la funcionalidad de un determinado recurso.
- Son el punto de entrada de las peticiones de los usuarios y contendrán toda la lógica asociada para:
  - Acceder a la base de datos si fuese necesario.
  - Preparar los datos.
  - Llamar a la vista con los datos asociados.
- Se almacenan en la carpeta `“app/Http/Controllers”` como ficheros PHP.
- Se les añade el sufijo `“Controller”`, por ejemplo `“UserController.php”` o `“MoviesController.php”`.

- Ejemplo de controlador básico almacenado en el fichero “app/Http/Controllers/UserController.php”:

```
<?php
namespace App\Http\Controllers;

use App\Models\User;

class UserController extends Controller
{
    public function showProfile($id)
    {
        $user = User::findOrFail($id);
        return view('user.profile', ['user' => $user]);
    }
}
```

Es importante el espacio de nombre y añadir las clases a usar.

Controlador base situado en app/Http/Controllers

- Para **crear un nuevo controlador** usaremos el siguiente comando de Artisan:

```
php artisan make:controller MoviesController
```

- Este comando creará el controlador `MoviesController.php` dentro de la ruta “`app/Http/Controllers`”.
- Lo completará con el código básico para un controlador que hemos visto antes.
- Por defecto el controlador no contendrá ningún método. Más adelante veremos cómo generar los métodos de un controlador tipo RESTful.

- Para enlazar el controlador con una ruta tenemos que modificar el fichero “routes/web.php”:

```
use App\Http\Controllers\UserController;  
Route::get('user/{id}', [UserController::class, 'show']);
```

- Para generar la URL que apunte a una acción de un controlador tenemos dos opciones:

```
action([UserController::class, 'show'], [1])  
url('user/', [1])
```

- Ejemplo de uso en una plantilla:

```
<a href="{ { action([App\Http\Controllers\UserController::class, 'index']) } }">  
    ¡Aprieta aquí!  
</a>
```

- Permiten realizar **validaciones antes o después** de que se ejecute el código asociado a una ruta.
- Los filtros se definen como una clase PHP almacenada dentro de la carpeta “app/Http/Middleware”.
- Cada *middleware* aplicará un filtro concreto sobre una petición y podrá permitir su ejecución, dar un error o redireccionar.
- Laravel incluye por defecto algunos filtros “auth”, “guest”, “auth.basic”, ...
- Para crear nuestros propios filtros podemos usar el comando de Artisan:

```
php artisan make:middleware MyMiddleware
```

- Ejemplo de código de un *middleware* propio:

```
<?php
namespace App\Http\Middleware;
use Closure;

class MyMiddleware {
    public function handle($request, Closure $next) {
        return $next($request);
    }
}
```

Parámetros de entrada

Método que tiene que procesar la petición

- Podemos devolver:
  - Para que continúe la petición: `return $next($request);`
  - Redirección a otra ruta: `return redirect('home');`
  - Lanzar excepción o abortar: `abort(403, 'Unauthorized');`



- Los *middleware* se pueden utilizar a tres niveles:
  - Global: se ejecutará en TODAS las peticiones.
  - A nivel de tipo o grupo de rutas (web o api).
  - A nivel de ruta.
- Para añadir un *middleware* **global**, que se ejecute en todas las peticiones, tenemos que modificar el fichero “bootstrap/app.php” y añadir:

```
use App\Http\Middleware\MyMiddleware;  
  
...  
->withMiddleware(function (Middleware $middleware) {  
    $middleware->append(MyMiddleware::class);  
})
```

- Como ya hemos visto, según el fichero de rutas que usemos (“routes/api.php” o “routes/web.php”) estaremos creando rutas API o rutas Web.
- Para cada uno de estos ficheros se aplican *middlewares* distintos durante el proceso de carga.
- Si queremos añadir un *middleware* a estos grupos tenemos que modificar el fichero “bootstrap/app.php” y añadir:

```
use App\Http\Middleware\MyMiddlewareWeb;  
use App\Http\Middleware\MyMiddlewareApi;  
  
->withMiddleware(function (Middleware $middleware) {  
    $middleware->web(append: [ MyMiddlewareWeb::class, ] );  
    $middleware->api(prepend: [ MyMiddlewareApi::class, ] );  
})
```

- El tipo de *middleware* que **más usaremos** será el que podemos asignar a **nivel de ruta**.
- Para asociar un *middleware* a una ruta tenemos que modificar el fichero de rutas “`routes/web.php`” o “`routes/api.php`”:

```
use App\Http\Middleware\MyMiddleware;

Route::get('user', function() {
    return 'Has superado el filtro!';
})->middleware(MyMiddleware::class);

// Podemos indicar varios filtros...
Route::get('user', [UserController::class, 'show'])
    ->middleware(First::class, Second::class);
```

- También podemos **agrupar** varias rutas y aplicar un filtro a todas ellas a la vez:

```
Route::middleware('auth')->group(function()  
{  
    Route::get('/', function()  
    {  
        //  
    });  
  
    Route::get('user/profile', function()  
    {  
        //  
    });  
});
```

- También podemos usar los grupos de rutas para aplicar un **prefijo** a todas ellas, por ejemplo:

```
Route::prefix('admin')->group(function()
{
    // Esta ruta tendrá será: admin/dashboard
    Route::get('dashboard', [AdminController::class, 'index']);

    Route::prefix('user')->group(function()
    {
        // Estas rutas tendrán el prefijo admin/user/*
        Route::get('index', [UserAdminController::class, 'index']);
        Route::get('show/{id}', [UserAdminController::class, 'show']);
    });
});
```

Podemos redirigir de una ruta a otra si por ejemplo hay un error de validación o de permisos, volver a la ruta anterior o incluso devolver los valores de la petición con la redirección:

```
return redirect('user/login');  
return back();           // Volver a la ruta anterior  
  
// O redirigir a un método de un controlador:  
return redirect()->action([HomeController::class, 'index']);  
  
// Para añadir parámetros:  
return redirect()->action([UserController::class, 'profile'], [1]);  
  
// Para devolver los valores de entrada del usuario:  
return redirect('form')->withInput();  
return back()->withInput();  
  
// O para reenviar los datos de entrada excepto algunos:  
return redirect('form')->withInput($request->except('password'));
```

# FORMULARIOS

- Laravel NO incluye métodos para la generación de formularios.
- Solo incluye algunas funciones de ayuda.
- En una vista, para abrir y cerrar formularios, haríamos:

```
<form action="{{ url('foo/bar') }}" method="POST">
    ...
</form>
```

- Para cambiar el **método** de envío (por defecto HTML solo acepta GET y POST):

```
<form action="{{ url('foo/bar') }}" method="POST">
    @method('PUT')
    ...
</form>
```



- Protección contra CSRF (*Cross-site request forgery* o falsificación de petición en sitios cruzados):
  - Tipo de *exploit* malicioso de un sitio web en el que comandos no autorizados son transmitidos por un usuario en el cual el sitio web confía.
- Laravel proporciona una forma fácil de protegernos:

```
<form action="{ {action([App\Http\Controllers\UserController::class, 'method'])} }"
        method="POST">

    @csrf

    ...

</form>
```

- Campos de texto:

```
<input type="text" name="nombre" id="nombre">
{{--También podemos especificar un valor por defecto--}}
<input type="text" name="edad" id="edad" value="18">
```

- *Textarea, password, hidden:*

```
<input type="password" name="password" id="password">
<input type="hidden" name="oculto" value="valor">
<textarea name="texto" id="texto" rows="4" cols="50">
    Texto por defecto
</textarea>
```

- También podemos crear otro tipo de *inputs* (email, number, tel, etc.).

- Añadir las etiquetas de un formulario:

```
<label for="correo">Correo electrónico:</label>  
<input type="email" name="correo" id="correo">
```

- Es importante que el valor del atributo “for” coincida con el identificador (id) del campo asociado.
- Podemos añadir tres tipos de **botones** a un formulario:

```
<button type="submit">Enviar</button>  
<button type="reset">Borrar</button>  
<button type="button">Volver</button>
```

- *Checkbox:*

```
<label for="terms">Aceptar términos</label>  
<input type="checkbox" name="terms" id="terms" value="1">
```

- *Radio buttons:*

```
<label for="color">Elige un color:</label>  
  
<input type="radio" name="color" id="color" value="red">Rojo  
<input type="radio" name="color" id="color" value="blue">Azul  
<input type="radio" name="color" id="color" value="green">Verde
```

- Es importante que todos tengan el mismo nombre en la propiedad “name”.
- Además podemos marcar alguno por defecto añadiendo “checked”.

- Listas desplegables tipo “*select*”:

```
<select name="marca">  
  <option value="volvo">Volvo</option>  
  <option value="saab">Saab</option>  
  <option value="mercedes">Mercedes</option>  
  <option value="audi" selected>Audi</option>  
</select>
```

- Podemos marcar una opción por defecto añadiendo el atributo “selected”.

**¿PREGUNTAS?**