



# LARAVEL: CONTROLADORES Y FORMULARIOS

DISEÑO DE SISTEMAS SOFTWARE

# Contenido

1. Controladores
2. Redirecciones
3. Formularios
4. Datos de entrada

Laravel: Controladores y formularios

# CONTROLADORES

# Controladores

- 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`

# Controladores

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

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

use Illuminate\Http\Request;
use App\User;

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

# Controladores

- Para crear un nuevo controlador podemos usar 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

# Controladores

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

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

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

```
$url = action('UserController@showProfile', [$id]);  
$url = url("user/", [$id]);
```

- Ejemplo de uso en una plantilla:

```
<a href="{{ action('FooController@method') }}">  
    ¡Aprieta aquí!  
</a>
```

Laravel: Controladores y formularios

## REDIRECCIONES



# Redirecciones

- 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@index');  
  
// Para añadir parámetros:  
return redirect()->action('UserController@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'));
```

Laravel: Controladores y formularios

# FORMULARIOS

# Formularios

- Laravel 5 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_field('PUT') }}  
    ...  
</form>
```

# Formularios: CSRF

- 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('Controller@method')}}" method="POST">
    {{ csrf_field() }}
    {{ method_field('PUT') }}
    ...
</form>
```

# Formularios: campos de texto

- 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.).

# Formularios: etiquetas y botones

- 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>
```

# Formularios: *checkbox* y *radio* buttons

- **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`

# Formularios: listas desplegables

- 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`



Laravel: Controladores y formularios

## **DATOS DE ENTRADA**

# Datos de entrada

- Laravel facilita el acceso a los datos de entrada
- No importa el método de la petición (POST, GET, PUT, DELETE), los datos se obtendrán de la misma forma
- Para obtener los datos de una petición utilizaremos la clase `Request`
- Esta clase la cargaremos en los métodos del controlador mediante **inyección de dependencias**
- Para obtener los datos siempre lo hacemos de la misma forma:

```
$nombre = $request->input('nombre');  
// O simplemente: $nombre = $request->nombre;  
// También podemos especificar un valor por defecto  
$nombre = $request->input('nombre', 'Pedro');
```

# Ejemplo:

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

use Illuminate\Http\Request;

class UserController extends Controller {
    // Inyección de la clase Request
    public function store(Request $request) {
        $nombre = $request->input('nombre');
        //...
    }

    // Método con más parámetros
    public function edit(Request $request, $id) {
        $validated = $request->input('validated', false);
        //...
    }
}
```

# Datos de entrada

- Podemos comprobar si un determinado valor existe con `$request->has()`:

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

use Illuminate\Http\Request;

class UserController extends Controller {
    public function edit(Request $request, $id) {
        $user = User::findOrFail( $id );
        if( $request->has('name') ) {
            $user->name = $request->input('name');
            $user->save();
        }
    }
}
```

# Datos de entrada

- También podemos obtener los datos de entrada agrupados:

```
// Obtener todos:  
$input = $request->all();  
  
// Obtener solo los campos indicados:  
$input = $request->only('username', 'password');  
  
// Obtener todos excepto los indicados:  
$input = $request->except('credit_card');
```

- Si el campo es tipo array podemos utilizar la notación:

```
$input = $request->input('products.0.name');
```

- Además si la entrada está en formato JSON también podremos acceder a los datos de forma normal con `$request->input`

# Ficheros de entrada

- Laravel incluye clases para trabajar con los ficheros de entrada
- Para obtener un fichero enviado en el campo `photo` hacemos:

```
$file = $request->file('photo');
```

- Si queremos comprobar si la variable contiene un fichero podemos hacer:

```
if( $request->hasFile('photo') ) { /* ... */ }
```

- El objeto devuelto es una instancia de la clase `Symfony\Component\HttpFoundation\File\UploadedFile`, la cual extiende ``SplFileInfo`` (<http://php.net/manual/es/class.splfileinfo.php>), por lo tanto disponemos de muchos métodos para obtener datos del fichero o para gestionarlo

# Ficheros de entrada

- Podemos comprobar si un fichero es válido:

```
if( $request->file('photo')->isValid() ) { /* ... */ }
```

- Mover el fichero a una ruta:

```
$request->file('photo')->move($destinationPath);  
// Mover el fichero a la ruta con un nuevo nombre:  
$request->file('photo')->move($destinationPath, $fileName);
```

- Recuperar información del fichero:

```
$path = $request->file('photo')->getRealPath();  
$name = $request->file('photo')->getClientOriginalName();  
$ext  = $request->file('photo')->getClientOriginalExtension();  
$size = $request->file('photo')->getSize();  
$mime = $request->file('photo')->getMimeType();
```