



UA.MASTER MOVILES

MÁSTER UNIVERSITARIO EN DESARROLLO DE SOFTWARE
PARA DISPOSITIVOS MÓVILES

PROGRAMACIÓN HIPERMEDIA PARA DISPOSITIVOS MÓVILES

Laravel 4 – Datos de entrada y Control de usuarios

1. Datos de entrada
2. Ficheros de entrada
3. Control de usuarios:
 1. Configuración
 2. Añadir nuevos usuarios
 3. Autenticar usuarios
 4. Acceso a los datos de un usuario
 5. Cerrar sesión
 6. Proteger rutas

- 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');
```

DATOS DE ENTRADA. EJEMPLO:

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

use Illuminate\Http\Request;
use Illuminate\Routing\Controller;

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

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

Inyección de la clase
Request

Resto de parámetros

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

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

use Illuminate\Http\Request;
use Illuminate\Routing\Controller;

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

- 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`”.

- 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 `Illuminate\Http\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.

- 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();
```


CONTROL DE USUARIOS

- Laravel incluye métodos y clases que hacen que la implementación y uso del control de usuarios sea muy sencilla.
- La **configuración** del sistema de autenticación se puede encontrar en el fichero ``config/auth.php``, en el cual podremos:
 - Cambiar el sistema de autenticación (“*Eloquent*” por defecto).
 - Cambiar el modelo de datos (“*User*” por defecto).
 - Cambiar la tabla de usuarios (“*users*” por defecto).
- Al crear un nuevo proyecto de Laravel **ya se incluye el modelo** “*User*” en la carpeta “*app*” configurado para utilizar la autenticación.

- También se incluye la **migración** de la tabla “users” con el siguiente esquema (función up):

```
Schema::create('users', function($table) {  
    $table->bigIncrements('id');  
    $table->string('name');  
    $table->string('email')->unique();  
    $table->timestamp('email_verified_at')->nullable();  
    $table->string('password');  
    $table->rememberToken();  
    $table->timestamps();  
});
```

- **Importante:**
 - Incluye un “id” único autoincremental para identificar a los usuarios.
 - El campo “email” es “*unique*”.
 - El campo *password* estará cifrado mediante el método `bcrypt()`.
 - Podemos **añadir todos los campos que queramos** a esta tabla, por ejemplo apellidos, dirección, teléfono, etc.

- Laravel también **incorpora** por defecto los controladores para la gestión de usuarios:
 - `LoginController` y `RegisterController`:
Incluyen métodos para ayudarnos en el proceso de autenticación (o *login*), registro y cierre de sesión.
 - `ResetPasswordController` y `ForgotPasswordController`:
Contienen la lógica para ayudarnos en el proceso de restaurar una contraseña.
- Los podemos encontrar en la carpeta (y espacio de nombres):
“`App\Http\Controllers\Auth`”

- Lo único que falta son **las rutas y las vistas**, estas no vienen por defecto pero las podemos **generar** mediante el comando:

```
$ composer require laravel/ui --dev  
$ php artisan ui vue --auth
```

- Al ejecutar este comando se añadirán todas las vistas necesarias a la carpeta “resources/views/auth” y las rutas al fichero “routes/web.php”.
- Si editamos el fichero “routes/web.php” podremos ver que únicamente nos ha añadido las siguientes dos líneas:

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

Si ejecutamos `php artisan route:list` podremos ver las nuevas rutas:

Método	Ruta	Acción	Vista	Filtros
GET	<i>login</i>	<i>LoginController@showLoginForm</i>	<i>login.blade</i>	<i>web,guest</i>
POST	<i>login</i>	<i>LoginController@login</i>		<i>web,guest</i>
POST	<i>logout</i>	<i>LoginController@logout</i>		<i>web</i>
GET	<i>register</i>	<i>RegisterController@showRegistrationForm</i>	<i>register.blade</i>	<i>web,guest</i>
POST	<i>register</i>	<i>RegisterController@register</i>		<i>web,guest</i>
GET	<i>password/reset</i>	<i>ForgotPasswordController@showLinkRequestForm</i>	<i>email.blade</i>	<i>web,guest</i>
POST	<i>password/email</i>	<i>ForgotPasswordController@sendResetLinkEmail</i>		<i>web,guest</i>
GET	<i>password/reset/{token}</i>	<i>ResetPasswordController@showResetForm</i>	<i>reset.blade</i>	<i>web,guest</i>
POST	<i>password/reset</i>	<i>ResetPasswordController@reset</i>		<i>web,guest</i>
GET	<i>home</i>	<i>HomeController@index</i>		<i>web,auth</i>

- Al ejecutar `php artisan ui vue --auth` se **generan** también las vistas necesarias para: login, registro y recuperar la contraseña.
- Estas vistas las podemos encontrar en `resources/views/auth`.
- Es importante que **no cambiemos** ni el nombre ni la ruta de las vistas pues los controladores ya están preparados para acceder con esos datos.
- Sin embargo **sí** que podemos **modificar** el contenido y apariencia de las vistas, con la única precaución de respetar la URL a la que se envía el formulario y los nombres de los *inputs*.
- Las vistas heredan del **layout** `layouts/app.blade.php`, el cual lo podemos modificar o cambiar por otro.

- Si accedemos a la ruta “login” nos aparecerá el formulario de login para iniciar sesión mediante nuestro email y contraseña.
- En caso de que el *login* se realice **correctamente**:
 - Por defecto se redirigirá a la ruta “/home”.
 - Para cambiar la ruta tenemos que modificar el controlador “LoginController” y establecer la propiedad:

```
protected $redirectTo = '/dashboard';
```
- Además podemos definir esta propiedad en RegisterController y ResetPasswordController para cambiar la URL de redirección después del registro y después de restablecer la contraseña, respectivamente.

- Si accedemos a la ruta “register” nos aparecerá el formulario de registro para crear nuevos usuarios.
- Si además de los campos nombre, email y contraseña queremos almacenar otros, tenemos que modificar:
 - La migración de la tabla de usuarios con los nuevos campos.
 - Las siguientes funciones de “RegisterController”:
 - `validator`: realiza la validación de los datos.
 - `create`: se encarga de crear el nuevo registro.

```
protected function create(array $data) {  
    return User::create([  
        'name' => $data['name'],  
        'email' => $data['email'],  
        'phone' => $data['phone'],          // Campo añadido  
        'password' => Hash::make($data['password']),  
    ]);  
}
```

AÑADIR UN NUEVO USUARIO MANUALMENTE

- Lo podemos crear usando *Eloquent* de forma normal.
- La única precaución que tenemos que llevar es cifrar el *password* manualmente:

```
use Illuminate\Support\Facades\Hash;
```

```
$password_cifrado = Hash::make( 'mi-super-password' );
```

- Por ejemplo, para recoger los datos de un formulario y crear un nuevo registro:

```
public function store(Request $request)
{
    $user = new User;
    $user->name = $request->input('name');
    $user->email = $request->input('email');
    $user->password = Hash::make($request->input('password'));
    $user->save();
}
```

- Una vez que el usuario está autenticado podemos acceder a los datos del mismo a través del método ``Auth::user()``, por ejemplo:

```
$email = Auth::user()->email;
```

- El método “`Auth::user()`” devolverá “`null`” si el usuario no está autenticado.
- Importante: para utilizar la clase `Auth` tenemos que añadir el espacio de nombres “`use Illuminate\Support\Facades\Auth;`”

- Para comprobar si el usuario actual se ha autenticado en la aplicación podemos utilizar el método `Auth::check()` de la forma:

```
if( Auth::check() ) {  
    // El usuario está correctamente autenticado  
}
```

- Por ejemplo, en una vista con Blade podríamos hacer:

```
@if( Auth::check() )  
    Usuario autenticado: {{ Auth::user()->name }}  
@endif  
  
@auth  
    // Alternativa para validar usuario autenticado...  
@endauth
```

- También podéis usar el método `Auth::guest()` o `@guest ... @endguest` para comprobar si es un usuario invitado.

- Si accedemos a la ruta “logout” (por POST) se cerrará la sesión y se redirigirá a la ruta “/”.
- Para cerrar la sesión del usuario **manualmente** podemos utilizar el método:

```
Auth::logout();
```

- Posteriormente podemos hacer una redirección a una página principal para usuarios no autenticados.

- Laravel incorpora el filtro o *middleware* “**auth**” que podemos usar para comprobar que el usuario que accede a una determinada ruta o grupo de rutas esté autenticado:

```
Route::get('admin', function() {  
    // Solo se permite el acceso a usuarios autenticados  
})->middleware('auth');  
  
// Para proteger una acción de un controlador:  
Route::get('profile', 'ProfileController@show')->middleware('auth');  
  
// O por ejemplo para proteger un grupo de rutas:  
Route::group(['middleware' => 'auth'], function() {  
    Route::get('catalog', 'CatalogController@getIndex');  
    Route::get('catalog/create', 'CatalogController@getCreate');  
});
```

¿PREGUNTAS?