LARAVEL

CLASE 06

REQUESTS

Uso de Request en Laravel

Laravel nos facilita todos los datos de la solicitud actual (los viejos y conocidos $_POST y $_GET) a través del Illuminate\Http\Request, un objeto sobre el que podremos consultar información sobre la solicitud y los datos que pueda estar enviando.

Para poder trabajar con el objeto Request en un controlador lo hacemos a través de inyección de dependencias.

```
public function nombreMetodo(Request $request)
{
    // Código
}
```

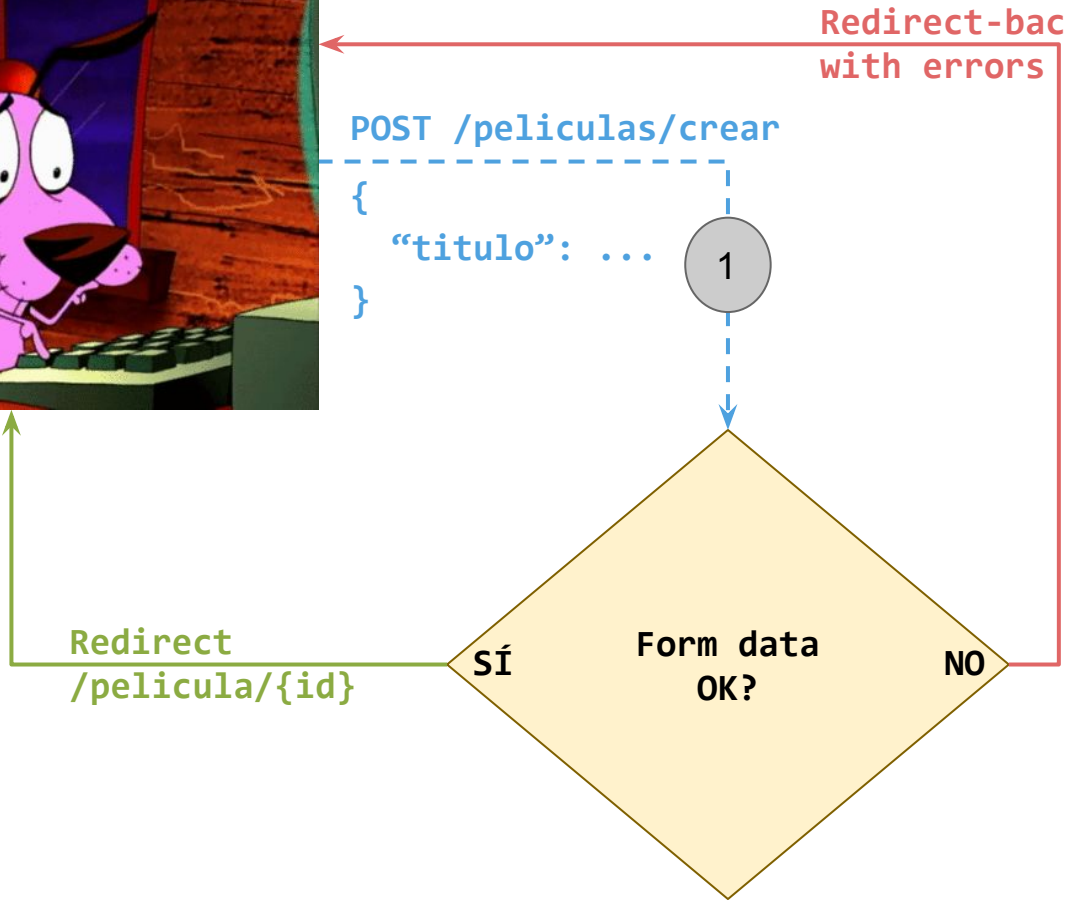
```php
// Método All - Nos devuelve un array con todos los inputs y sus valores
$inputs = $request->all();

// Método Input - Nos devuelve el valor de un único input
$input = $request->input('nombre');

// Método Only - Nos devuelve un array sólo con los inputs solicitados
$inputs = $request->only('nombre', 'edad');
```

VALIDACIONES

¿Cómo validar en Laravel?


Redirect-back
with errors
POST /peliculas/crear
{
"titulo": ...
}
1
Form data
OK?
SÍ
NO
Redirect
/pelicula/{id}

Laravel nos proporciona algunos métodos muy sencillos para validar los datos que recibimos en un pedido HTTP.

Analicemos la manera más rápida de validar, manteniendo la lógica en el controlador.

```
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required|unique:movies|max:255',
        'body'  => 'required',
    ]);
}
```

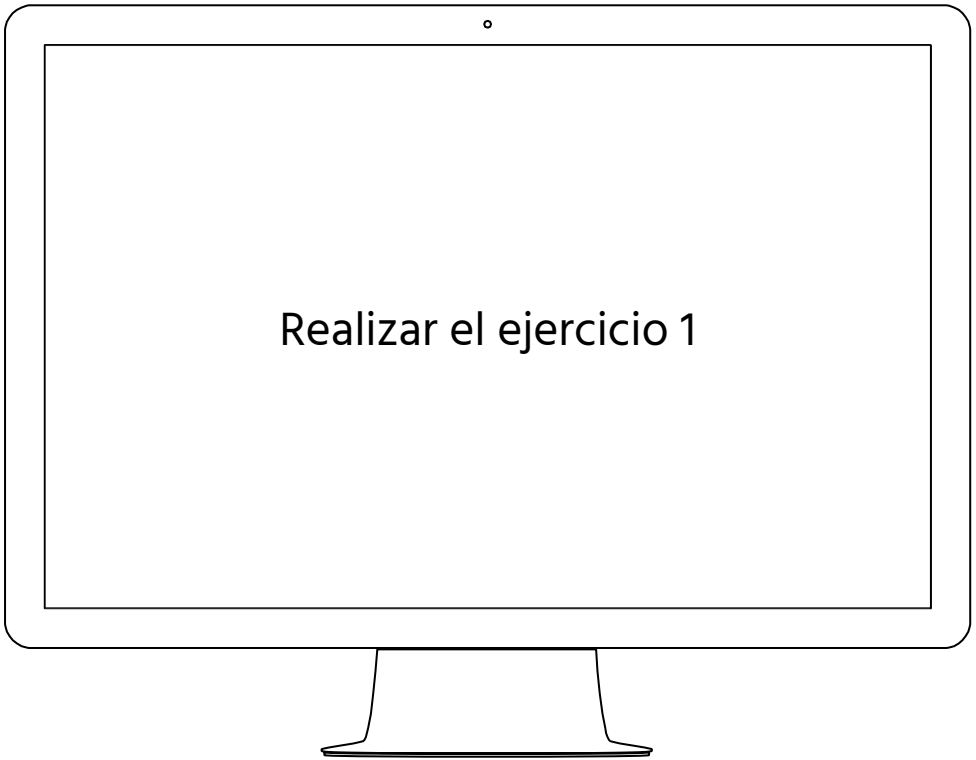
Reglas de Validación

- required
- sometimes
- min - max
- in - not_in
- between
- confirmed
- before - after
- numeric
- integer
- boolean
- array
- string
- date
- image
- unique
- exists
- mimes - mimetypes
- email
- url
- alpha - alpha_dash - alpha_num

```
// Mostrar errores de Validación en la vista

@if (count($errors) > 0)
        <div class="alert alert-danger">
            <ul>
             @foreach ($errors->all() as $error)
                 <li>{{ $error }}</li>
             @endforeach
            </ul>
    </div>
@endif
```

ES MOMENTO DE
PRACTICAR !



MÉTODOS ELOQUENT

Crear, Actualizar, Eliminar

Create ( );

Recibe un array asociativo (clave => valor), **inserta** un nuevo registro en la base de datos. Devuelve un objeto de la clase con los atributos generados.

```
$usuario = User::create(['nombre' => 'Francisco']);
```

```
// Para poder utilizar este método debemos recordar colocar el atributo que en este caso es “nombre” como $fillable dentro del modelo.
```

Save ( );

Nos permite **insertar** un nuevo registro ó **actualizar** los datos de un registro existente. Para que esto funcione, es importante tener una instancia del modelo creada.

```
//instanciamos un objeto Usuario vacío:
    $usuario = new Usuario( );

//ó podemos enviarle un array asociativo similar al método create:

    $usuario = new Usuario([
      'name' => $request->input('name'),

      'email' => $request->input('email'),

    ]);
```

Save ( );

Así lo usamos para crear un nuevo objeto en la BD.

```
public function store(Request $request)
{
    //instanciamos un objeto Usuario y lo guardamos
        $usuario = new Usuario([
            'name' => $request->input('name'),
        ]);

    $usuario->save();
}
```

Save ( );

Así editamos un objeto en la BD.

```
public function store($id, Request $request)
{
    //Buscamos el Objeto, editamos los atributos y los guardamos en
la BD.
    $usuario =Usuario::find($id);
    //cambiamos las propiedades
    $usuario->name = $request->input('name');

    $usuario->save();
}
```

Delete ( );

Nos permite **eliminar** datos de la base de datos, siempre y cuando tengamos instanciado al objeto.

```
$flight = App\Flight::find(1);

$flight->delete();
```

¡Luego no será posible recuperar esa info!

Resources Routes

Doc

Usamos la consola

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

```
// luego colocamos:
```

> routes/web.php

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

```
//ademas creara en el controlador: la documentacion
```

ES MOMENTO DE
PRACTICAR !

¡ HASTA LA PROXIMA !