

Universidad Nacional de San Agustín

Facultad de Producción y Servicio

Escuela Profesional de Ciencia de la Computación



CURSO ING. DE SOFTWARE

Principios SOLID

Alumno:

- Diego Alonso Zanabria Sacsi

Arequipa – Perú

2023

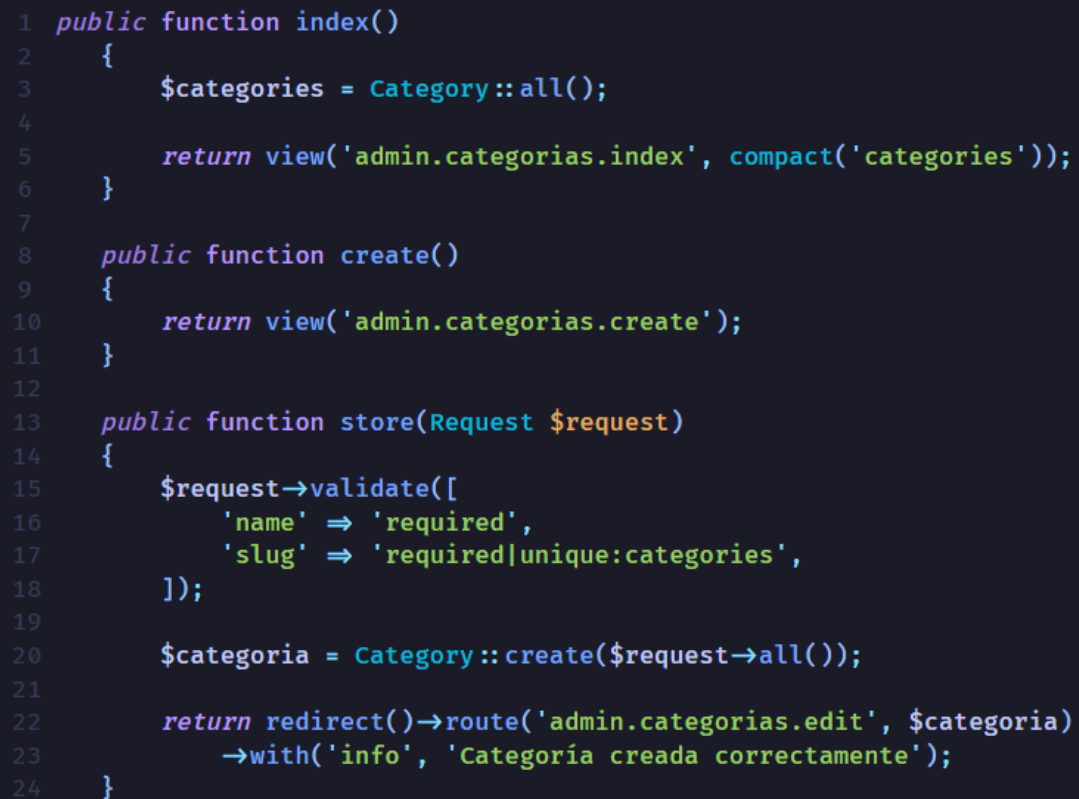
1. Single-responsability o Principio de Responsabilidad Unica:

Si una clase tiene muchas responsabilidades, aumenta la posibilidad de errores porque hacer cambios en una de sus responsabilidades podría afectar a las otras sin que usted lo sepa.

"Una clase debe tener solo una razón para cambiar" Se eligió esta clase porque cumple con las características de este, es decir, la Clase User se encarga únicamente de recopilar la información de una persona como nombre, email, password.

```
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Category extends Model
9  {
10     use HasFactory;
11
12     protected $fillable = [
13         'name',
14         'slug',
15     ];
16
17     public function getRouteKeyName()
18     {
19         return 'slug';
20     }
21
22     public function posts()
23     {
24         return $this->hasMany(Post::class);
25     }
26 }
27
```

El principio de responsabilidad única busca que el código quede encapsulado y exista independencia entre las clases, sus funcionalidades. Al utilizar clases hemos procurado cumplir con este criterio, ya que encapsulamos la funcionalidad de cada una para que realicen una única función.



```

1  public function index()
2  {
3      $categories = Category::all();
4
5      return view('admin.categorias.index', compact('categories'));
6  }
7
8  public function create()
9  {
10     return view('admin.categorias.create');
11 }
12
13 public function store(Request $request)
14 {
15     $request->validate([
16         'name' => 'required',
17         'slug' => 'required|unique:categories',
18     ]);
19
20     $categoria = Category::create($request->all());
21
22     return redirect()->route('admin.categorias.edit', $categoria)
23         ->with('info', 'Categoría creada correctamente');
24 }

```

Por ejemplo, en la imagen se ve que se han independizado las funciones, entre ellas editar la sesión, y otras. Esto también ayuda a la reutilización del código en caso de cambios o mantenimiento.

2. Segregación de la Interfaz:

Según este principio es mejor tener una clase pequeña y especializada que una muy grande, para poder hacer un mejor objetivo hacia las necesidades del sistema. En nuestro trabajo, hemos procurado seguir esta norma al no sobrecargar las funcionalidades de las clases sin más de lo que se necesite. Por ejemplo en la

captura se ve que cada función está dirigida a un único fin, aunque pertenezcan al mismo sistema, están especializados.

```
1 public function index()
2 {
3     $categories = Category::all();
4
5     return view('admin.categorias.index', compact('categories'));
6 }
7
8 public function create()
9 {
10     return view('admin.categorias.create');
11 }
12
13 public function store(Request $request)
14 {
15     $request->validate([
16         'name' => 'required',
17         'slug' => 'required|unique:categories',
18     ]);
19
20     $categoria = Category::create($request->all());
21
22     return redirect()->route('admin.categorias.edit', $categoria)
23         ->with('info', 'Categoría creada correctamente');
24 }
```

3. Single-responsability o Principio de Responsabilidad Unica:

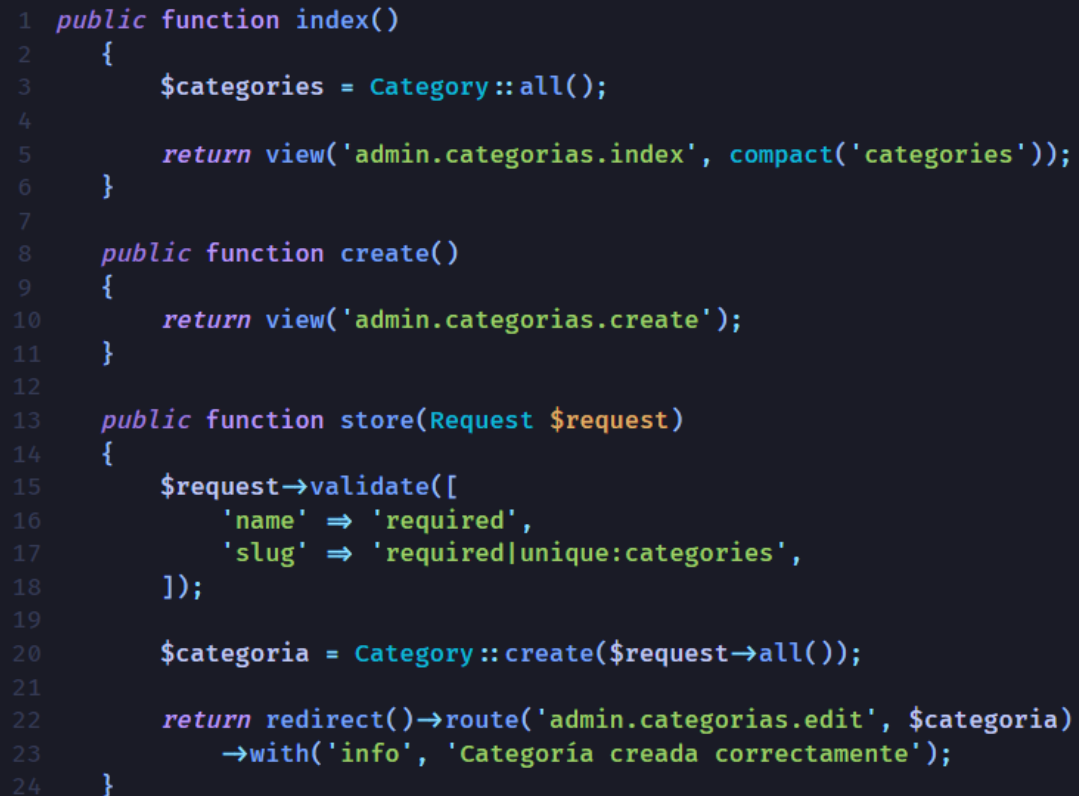
Si una clase tiene muchas responsabilidades, aumenta la posibilidad de errores porque hacer cambios en una de sus responsabilidades podría afectar a las otras sin que usted lo sepa.

"Una clase debe tener solo una razón para cambiar" Se eligió esta clase porque cumple con las características de este, es decir, la Clase User se encarga

únicamente de recopilar la información de una persona como nombre, email, password.

```
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Factories\HasFactory;
6  use Illuminate\Database\Eloquent\Model;
7
8  class Category extends Model
9  {
10     use HasFactory;
11
12     protected $fillable = [
13         'name',
14         'slug',
15     ];
16
17     public function getRouteKeyName()
18     {
19         return 'slug';
20     }
21
22     public function posts()
23     {
24         return $this->hasMany(Post::class);
25     }
26 }
27
```

El principio de responsabilidad única busca que el código quede encapsulado y exista independencia entre las clases, sus funcionalidades. Al utilizar clases hemos procurado cumplir con este criterio, ya que encapsulamos la funcionalidad de cada una para que realicen una única función.



```
1 public function index()
2 {
3     $categories = Category::all();
4
5     return view('admin.categorias.index', compact('categories'));
6 }
7
8 public function create()
9 {
10    return view('admin.categorias.create');
11 }
12
13 public function store(Request $request)
14 {
15     $request->validate([
16         'name' => 'required',
17         'slug' => 'required|unique:categories',
18     ]);
19
20     $categoria = Category::create($request->all());
21
22     return redirect()->route('admin.categorias.edit', $categoria)
23         ->with('info', 'Categoría creada correctamente');
24 }
```

Por ejemplo, en la imagen se ve que se han independizado las funciones, entre ellas editar la sesión, y otras. Esto también ayuda a la reutilización del código en caso de cambios o mantenimiento.

4. Segregación de la Interfaz:

Según este principio es mejor tener una clase pequeña y especializada que una muy grande, para poder hacer un mejor objetivo hacia las necesidades del sistema. En nuestro trabajo, hemos procurado seguir esta norma al no sobrecargar las funcionalidades de las clases sin más de lo que se necesite. Por ejemplo en la

captura se ve que cada función está dirigida a un único fin, aunque pertenezcan al mismo sistema, están especializados.

```
1 public function index()
2 {
3     $categories = Category::all();
4
5     return view('admin.categorias.index', compact('categories'));
6 }
7
8 public function create()
9 {
10     return view('admin.categorias.create');
11 }
12
13 public function store(Request $request)
14 {
15     $request->validate([
16         'name' => 'required',
17         'slug' => 'required|unique:categories',
18     ]);
19
20     $categoria = Category::create($request->all());
21
22     return redirect()->route('admin.categorias.edit', $categoria)
23         ->with('info', 'Categoría creada correctamente');
24 }
```