
UD10.3 – Laravel

Controladores

2º CFGS
Desarrollo de Aplicaciones Web
2022-23

1.- Controlador

En el patrón MVC (Modelo Vista Controlador) los controladores son los encargados de coordinar todas las acciones.

Una petición, a través de una ruta llegará a un **controlador**, este realizará toda la lógica relacionada con la ruta.

Si es **necesario accederá a la base de datos** y por último se encargará de **devolver la vista asociada a la ruta** que ha generado la petición.

1.- Controlador

Los controladores van muy ligados al modelo de datos.

Es habitual que se tenga **un controlador por cada tabla de la base de datos.**

Además, puede haber **controladores extra para otras funcionalidades.**

En Laravel los controladores son **clases.**

2.- Crear controladores Laravel

Para crear un controlador se utiliza el siguiente comando artisan:

```
# php artisan make:controller NombreController
```

Los **nombres** de los controladores deben:

- Estar en **singular** y acabar con la palabra **Controller**.
- Usar la técnica **UpperCamelCase** de manera que todas las palabras deben empezar por mayúscula y no puede haber espacios.
- Evitar el uso del carácter **_** siempre.

2.- Crear controladores Laravel

Cuando se ejecuta el comando anterior se crea el archivo correspondiente al controlador en el directorio **app/Http/Controllers**.

Ese archivo se crea con el código mínimo necesario para la **clase PHP**.

Dentro de la clase se tendrán que **definir los métodos** que realizarán las diferentes acciones que el controlador realizará.

2.- Crear controladores Laravel

```
# php artisan make:controller PrimerController
```

app\Http\Controllers\PrimerController.php

```
app > Http > Controllers > 🐘 PrimerController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class PrimerController extends Controller
8  {
9      //
10 }
```

2.- Crear controladores Laravel

app\Http\Controllers\PrimerController.php

```
app > Http > Controllers > PrimerController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class PrimerController extends Controller
8  {
9      public function primeraAccion()
10     {
11         return 'Este texto se genera desde el controlador PrimerController';
12     }
13 }
```

2.- Crear controladores Laravel

app\Http\Controllers\PrimerController.php

Para cumplir con el patrón MVC se debería devolver una vista:

```
app > Http > Controllers > 🐘 PrimerController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class PrimerController extends Controller
8  {
9      public function primeraAccion()
10     {
11         return view('primeraAccion');
12     }
13 }
```

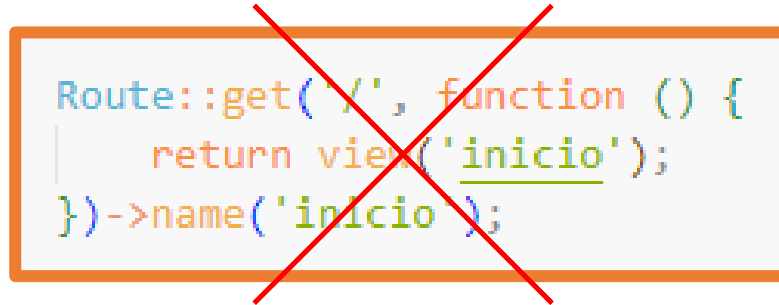

2.- Crear controladores Laravel

La vista **primeraaccion.blade.php** podría ser:

```
resources > views > primeraaccion.blade.php > ...
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width={device-width}, initial-scale=1.0">
6      <title>Primera vista desde un controlador</title>
7  </head>
8  <body>
9      Vista cargada desde el método primeraAcción del controlador PrimerControlador
10 </body>
11 </html>
```

3.- Redirigir a un controlador desde una ruta

Hasta ahora se habían definido las rutas para que ellas se encargaran de devolver la vista adecuada a la ruta, pero esta práctica no hace uso del patrón MVC:



```
Route::get('/', function () {  
    return view('inicio');  
})->name('inicio');
```

Así, una vez creados los controladores y definidos sus métodos, **se debe indicar a cada ruta qué método de qué controlador será el encargado de realizar las acciones correspondientes a dicha ruta.**

3.- Redirigir a un controlador desde una ruta

```
routes > web.php
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  use App\Http\Controllers\PrimerController; ←
6
7  /*
8  |-----
9  | Web Routes
10 |-----
11 |
12 | Here is where you can register web routes for your application. These
13 | routes are loaded by the RouteServiceProvider within a group which
14 | contains the "web" middleware group. Now create something great!
15 |
16 */
17
18 Route::get('/', function () {
19     return view('inicio');
20 }->name('inicio');
21
22 Route::get('primeraaccion', [PrimerController::class, 'primeraAccion']);
```

3.- Redirigir a un controlador desde una ruta

Analizando la ruta anterior se tiene:

```
Route::get('primeraaccion', [PrimerController::class, 'primeraAccion']);
```

- La URL de la ruta es: **/primeraaccion**
- El controlador que gestiona la ruta es: **PrimerController**
- El método de Primer Controller que se ejecuta es: **primeraAccion**

4.- Controladores de una sola acción

En ocasiones se necesitan controladores que **no están relacionados con la base de datos** y que **solo realizan una acción**.

Para crearlos se debe usar el parámetro **-i** en el comando:

```
# php artisan make:controller SaludoController -i
```

De esta manera dentro de la clase SaludoController se creará un método llamado **__invoke** donde se tendrá que crear la funcionalidad de dicho controlador.

(el método va precedido de **__** que son dos caracteres **_** seguidos).

4.- Controladores de una sola acción

SaludoController.php

```
app > Http > Controllers > SaludoController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class SaludoController extends Controller
8  {
9      /**
10       * Handle the incoming request.
11       *
12       * @param  \Illuminate\Http\Request  $request
13       * @return \Illuminate\Http\Response
14       */
15     public function __invoke(Request $request)
16     {
17         //
18     }
19 }
```

4.- Controladores de una sola acción

SaludoController.php

En el método **__invoke** se puede observar que recibe una variable llamada **\$request**.

Esta variable contiene mucha información sobre la petición realizada. Es muy útil cuando un método recibe datos de un formulario.

Si no se reciben datos desde un formulario se puede eliminar dicha variable de la función para optimizar el código.

Si se quiere, esta variable se puede utilizar en cualquier método de cualquier controlador, pero se debe añadir en la parte superior del archivo la línea que importa la clase para su uso.

```
app > Http > Controllers > SaludoController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class SaludoController extends Controller
8  {
9      /**
10       * Handle the incoming request.
11       *
12       * @param  \Illuminate\Http\Request  $request
13       * @return \Illuminate\Http\Response
14       */
15     public function __invoke(Request $request)
16     {
17         //
18     }
19 }
```



4.- Controladores de una sola acción

SaludoController.php

```
app > Http > Controllers > SaludoController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class SaludoController extends Controller
8  {
9      /**
10       * Handle the incoming request.
11       *
12       * @param \Illuminate\Http\Request $request
13       * @return \Illuminate\Http\Response
14       */
15     public function __invoke(Request $request)
16     {
17         $nombre = 'Invitado';
18         return view('saludo', compact('nombre'));
19     }
20 }
```



4.- Controladores de una sola acción

La vista **saludo.blade.php** podría ser:

```
resources > views > saludo.blade.php > ...
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width={device-width}, initial-scale=1.0">
6      <title>Saludo</title>
7  </head>
8  <body>
9      Hola {{ $nombre }}
10 </body>
11 </html>
```

4.- Controladores de una sola acción

A continuación, se muestra una **ruta** para esa acción.

En la ruta **no es necesario indicar el método** que gestiona la ruta porque el controlador es para una única acción (-i) y solo contiene el método `__invoke`.

```
routes > web.php
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  use App\Http\Controllers\PrimerController;
6  use App\Http\Controllers\SaludoController;
7
8  /*
9   |-----
10  | Web Routes
11  |-----
12  |
13  | Here is where you can register web routes for your application. These
14  | routes are loaded by the RouteServiceProvider within a group which
15  | contains the "web" middleware group. Now create something great!
16  |
17  */
18
19  Route::get('/', function () {
20      return view('inicio');
21  }->name('inicio');
22
23  Route::get('primeraaccion', [PrimerController::class, 'primeraAccion']);
24
25  Route::get('saludo', SaludoController::class->name('saludo');
```

4.- Controladores de una sola acción

Como se puede observar, como en cualquier ruta, en las rutas gestionadas por un controlador también se puede indicar un nombre.

```
Route::get('saludo', SaludoController::class)->name('saludo');
```



5.- Controladores de varias acciones

En otras ocasiones se requiere crear un controlador que **realiza varias acciones** pero que no están relacionadas con ninguna tabla de la base de datos.

En este caso el nombre del controlador se puede indicar en plural pero es una excepción debido a la funcionalidad del controlador:

```
# php artisan make:controller OperacionesController
```

Al crearlos contendrán el código mínimo de la clase y se deberá añadir tantos métodos como acciones realice el controlador.

5.- Controladores de varias acciones

OperacionesController.php

```
app > Http > Controllers > 🐘 OperacionesController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class OperacionesController extends Controller
8  {
9      //
10 }
```

5.- Controladores de varias acciones

OperacionesController.php

```
app > Http > Controllers > OperacionesController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class OperacionesController extends Controller
8  {
9      public function listar10Primos() ←
10     {
11         $primos = [];
12         $numero = 2;
13         while (count($primos) < 10) {
14             $esPrimo = true;
15             for ($i = 2; $i < $numero; $i++) {
16                 if ($numero % $i == 0) {
17                     $esPrimo = false;
18                     break;
19                 }
20             }
21             if ($esPrimo) {
22                 $primos[] = $numero;
23             }
24             $numero++;
25         }
26
27         return view('primos', compact('primos'));
28     }
29
30     public function factorial($numero) ←
31     {
32         $factorial = 1;
33         for ($i = 1; $i <= $numero; $i++) {
34             $factorial *= $i;
35         }
36         return view('factorial', compact('numero', 'factorial'));
37     }
38 }
```

Vistas: primos.blade.php

```
resources > views > primos.blade.php > ...
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width={device-width}, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      @foreach ($primos as $primo)
10         {{ $primo }}
11         @if(! $loop->last)
12             -
13         @endif
14     @endforeach
15 </body>
16 </html>
```

factorial.blade.php

```
resources > views > factorial.blade.php > ...
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width={device-width}, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9      Cálculod el factorial de {{ $numero }} <br>
10     {{ $numero }}! = {{ $factorial }}
11 </body>
12 </html>
```

5.- Controladores de varias acciones

Rutas:

```
routes > web.php
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  use App\Http\Controllers\PrimerController;
6  use App\Http\Controllers\SaludoController;
7  use App\Http\Controllers\OperacionesController;
8
9  /*
10 |-----
11 | Web Routes
12 |-----
13 |
14 | Here is where you can register web routes for your application. These
15 | routes are loaded by the RouteServiceProvider within a group which
16 | contains the "web" middleware group. Now create something great!
17 |
18 */
19
20 Route::get('/', function () {
21     return view('inicio');
22 }->name('inicio');
23
24 Route::get('primeraaccion', [PrimerController::class, 'primeraAccion']);
25
26 Route::get('saludo', SaludoController::class->name('saludo'));
27
28 Route::get('listar10primos', [OperacionesController::class, 'listar10Primos']);
29 Route::get('factorial/{numero}', [OperacionesController::class, 'factorial']);
30
```

5.- Controladores de varias acciones

Se puede observar que igual que en todas las rutas, en las gestionadas por un controlador también se pueden usar parámetros y Laravel es capaz de enlazarlo directamente en el controlador simplemente con usar su nombre:

```
Route::get('factorial/{numero}', [OperacionesController::class, 'factorial']);
```



```
public function factorial($numero)
{
    $factorial = 1;
    for ($i = 1; $i <= $numero; $i++) {
        $factorial *= $i;
    }
    return view('factorial', compact('numero', 'factorial'));
}
```


5.- Controladores de varias acciones

Los parámetros también puede ser opcionales, en ese caso en el método se debe indicar un valor por defecto:

```
Route::get('saludo/{nombre?}', SaludoController::class)->name('saludo');
```



```
public function __invoke(Request $request, $nombre = 'Invitado')
{
    return view('saludo', compact('nombre'));
}
```

Se podría acceder al parámetro opcional mediante la variable \$request, pero no es práctico.

```
public function __invoke(Request $request)
{
    if(array_key_exists('nombre', $request->route()->parameters()))
        $nombre = $request->route()->parameters()['nombre'];
    else
        $nombre = 'Invitado';
    return view('saludo', compact('nombre'));
}
```

Práctica

En tu proyecto `blogTuNombre` crea los controladores, vistas y rutas de los ejemplos anteriores y prueba que todo funciona en el navegador.

Crea un método nuevo en `OperacionesController` que reciba un número `N` y muestre los primeros `N` números primos.

6.- CRUD, controladores tipo recurso

CRUD → Controlador Laravel asociado a una tabla de la BBDD

Las siglas CRUD corresponden a las palabras en inglés:

- **Create**
- **Read**
- **Update**
- **Delete**

Estas son las acciones típicas que se realizan sobre una tabla de una base de datos.

6.- CRUD, controladores tipo recurso

Para visualizarlo mejor, si se está realizando una aplicación para el alquiler de vehículos, es posible que se tenga una tabla llamada **coches**.

Así las acciones del CRUD serían las siguientes:

- Create: añadir un nuevo coche a la base de datos.
- Read: leer datos de la tabla coches.
- Update: actualizar datos de un coche.
- Delete: eliminar un coche de la base de datos.

6.- CRUD, controladores tipo recurso

En Laravel, y en general en todos los proyectos que usan el patrón MVC se suele usar **un controlador** para gestionar todo el CRUD que se realiza sobre una tabla de la base de datos.

Se conocen como **controladores tipo recurso** → resource.

Para crear un controlador tipo recurso se utiliza el comando:

```
# php artisan make:controller NombreController -r
```

Este comando crea un archivo como los vistos anteriormente pero que por defecto contiene unos métodos ya creados en los que habrá que completar el código, estos métodos se relacionan con las diferentes acciones del CRUD.

6.- CRUD, controladores tipo recurso

Nota importante sobre el nombre de los componentes de Laravel

Laravel está desarrollado en inglés, por ello dispone de diferentes herramientas que facilitan el desarrollo del proyecto siguiendo el patrón MVC siempre que se cumplan las siguientes pautas:

- Los nombres de las tablas de la base de datos deben estar en plural.
- Los nombres de los modelos deben estar en singular.
- Los nombres de controladores deben estar en singular.
- Los nombres de las rutas a los controladores deben estar en plural.
- Todos los nombres anteriores deben estar en inglés.

No es obligado seguir estas pautas pero es una buena práctica como se verá a partir de este momento.

6.- CRUD, controladores tipo recurso

php artisan make:controller CarController -r

```
app > Http > Controllers > CarController.php > ...
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6
7  class CarController extends Controller
8  {
9      /**
10       * Display a listing of the resource.
11       *
12       * @return \Illuminate\Http\Response
13       */
14     public function index() ←
15     {
16         //
17     }
18
19     /**
20     * Show the form for creating a new resource.
21     *
22     * @return \Illuminate\Http\Response
23     */
24     public function create() ←
25     {
26         //
27     }
28 }
```

```
29     /**
30      * Store a newly created resource in storage.
31      *
32      * @param \Illuminate\Http\Request $request
33      * @return \Illuminate\Http\Response
34      */
35     public function store(Request $request) ←
36     {
37         //
38     }
39
40     /**
41     * Display the specified resource.
42     *
43     * @param int $id
44     * @return \Illuminate\Http\Response
45     */
46     public function show($id) ←
47     {
48         //
49     }
50
51     /**
52     * Show the form for editing the specified resource.
53     *
54     * @param int $id
55     * @return \Illuminate\Http\Response
56     */
57     public function edit($id) ←
58     {
59         //
60     }
```

```
61
62     /**
63      * Update the specified resource in storage.
64      *
65      * @param \Illuminate\Http\Request $request
66      * @param int $id
67      * @return \Illuminate\Http\Response
68      */
69     public function update(Request $request, $id) ←
70     {
71         //
72     }
73
74     /**
75     * Remove the specified resource from storage.
76     *
77     * @param int $id
78     * @return \Illuminate\Http\Response
79     */
80     public function destroy($id) ←
81     {
82         //
83     }
84 }
```

6.- CRUD, controladores tipo recurso

Los métodos creados son:

- **index:** mostrar todos los elementos de la tabla.
- **create:** mostrar el formulario para insertar un elemento en la base de datos.
- **store:** recibir los datos del formulario anterior y almacenar los datos en la base de datos.
- **show:** mostrar los datos de un elemento determinado de la tabla.
- **edit:** mostrar el formulario para modificar un elemento de la base de datos.
- **update:** recibir los datos del formulario anterior y almacenar los cambios en la base de datos.
- **destroy:** borrar un elemento determinado de la tabla.

6.- CRUD, controladores tipo recurso

Una vez creado un controlador tipo recurso, se pueden añadir todas las rutas de una vez de la siguiente manera en el archivo de rutas:

```
use App\Http\Controllers\CarController;
```

```
Route::resource('cars', CarController::class);
```

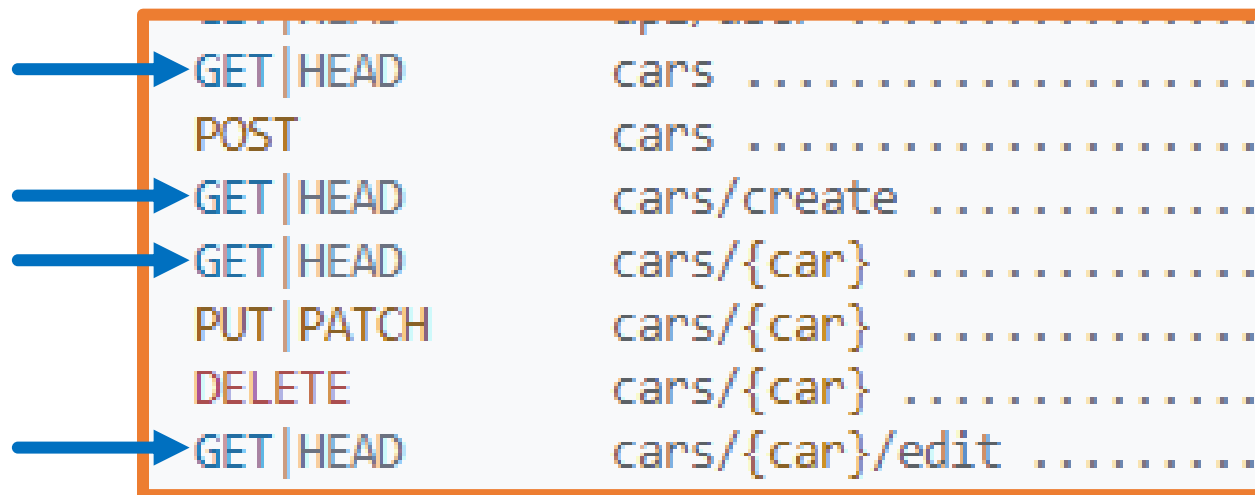
Esto crea automáticamente las rutas y les asigna un nombre: php artisan route:list

```
GET|HEAD      cars ..... cars.index > CarController@index
POST          cars ..... cars.store > CarController@store
GET|HEAD      cars/create ..... cars.create > CarController@create
GET|HEAD      cars/{car} ..... cars.show > CarController@show
PUT|PATCH    cars/{car} ..... cars.update > CarController@update
DELETE        cars/{car} ..... cars.destroy > CarController@destroy
GET|HEAD      cars/{car}/edit ..... cars.edit > CarController@edit
```

6.- CRUD, controladores tipo recurso

En el listado de rutas se puede ver el método de acceso a cada una de ellas.

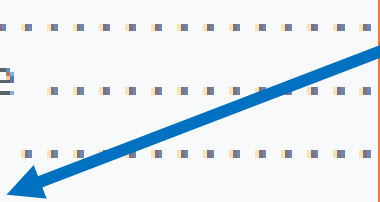
De momento solo se usarán las que usan el método GET ya que las otras necesitan del uso de formularios web.



→ GET HEAD	cars
POST	cars
→ GET HEAD	cars/create
→ GET HEAD	cars/{car}
PUT PATCH	cars/{car}
DELETE	cars/{car}
→ GET HEAD	cars/{car}/edit

6.- CRUD, controladores tipo recurso

Al utilizar en la ruta el nombre en inglés como se aconsejó, Laravel automáticamente indica el nombre del parámetro de las rutas (consultando un diccionario interno para saber el singular).



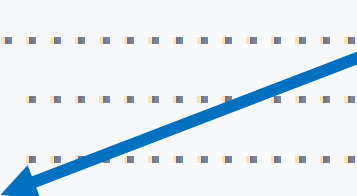
GET HEAD	cars
POST	cars
GET HEAD	cars/create
GET HEAD	cars/{car}
PUT PATCH	cars/{car}
DELETE	cars/{car}
GET HEAD	cars/{car}/edit

6.- CRUD, controladores tipo recurso

En el caso de querer tener las rutas en español (u otro idioma diferente al inglés) el comportamiento puede no ser el que se quiere:

```
Route::resource('coches', CarController::class);
```

GET HEAD	coches
POST	coches
GET HEAD	coches/create
GET HEAD	coches/{coch}
PUT PATCH	coches/{coch}
DELETE	coches/{coch}
GET HEAD	coches/{coch}/edit

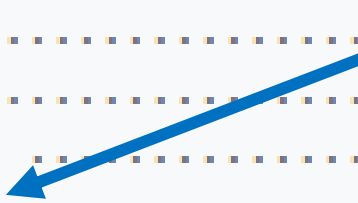


6.- CRUD, controladores tipo recurso

El problema anterior se soluciona indicando el nombre del parámetro en la ruta en el idioma deseado:

```
Route::resource('coches', CarController::class)->parameters(['coches' => 'coche']);
```

GET HEAD	coches
POST	coches
GET HEAD	coches/create
GET HEAD	coches/{coche}
PUT PATCH	coches/{coche}
DELETE	coches/{coche}
GET HEAD	coches/{coche}/edit



6.- CRUD, controladores tipo recurso

De todas formas esto no es ningún problema porque Laravel cambia internamente el nombre al parámetro en los métodos del CRUD asignándole el nombre **\$id**.

GET HEAD	coches
POST	coches
GET HEAD	coches/create
GET HEAD	coches/{coche}
PUT PATCH	coches/{coche}
DELETE	coches/{coche}
GET HEAD	coches/{coche}/edit

```
    * @return \Illuminate\Http\Response
    */
    public function show($id)
    {
        //
    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param int $id
     * @return \Illuminate\Http\Response
     */
    public function edit($id)
    {
        //
    }
```

Incluso se podría cambiar ese nombre si se quiere.

Al estudiar los Modelos se verá que se utilizará el parámetro de forma diferente.

6.- CRUD, controladores tipo recurso

Si se quiere traducir también el resto de la ruta (edit y create) se debe añadir el siguiente código en el archivo **app\Providers\RouteServiceProvider.php**

```
* @return void
*/
public function boot()
{
    Route::resourceverbs([
        'create' => 'crear',
        'edit' => 'editar',
    ]);

    $this->configureRateLimiting();


    $this->routes(function () {
```

GET HEAD	coches
POST	coches
GET HEAD	coches/crear
GET HEAD	coches/{coche}
PUT PATCH	coches/{coche}
DELETE	coches/{coche}
GET HEAD	coches/{coche}/editar

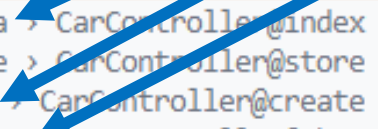
6.- CRUD, controladores tipo recurso

También se puede cambiar el nombre de las rutas (de todas o solo de algunas) al definir la ruta:

```
Route::resource('coches', CarController::class)->parameters(['coches' => 'coche'])
->names([
    'index' => 'coches.lista',
    'create' => 'coches.crear',
    'show' => 'coches.mostrar',
]);
```



GET HEAD	coches	coches.lista > CarController@index
POST	coches	coches.store > CarController@store
GET HEAD	coches/crear	coches.crear > CarController@create
GET HEAD	coches/{coche}	coches.mostrar > CarController@show
PUT PATCH	coches/{coche}	coches.update > CarController@update
DELETE	coches/{coche}	coches.destroy > CarController@destroy



6.- CRUD, controladores tipo recurso

Hay ocasiones en las que no interesa incluir todas las rutas de un controlador de tipo recurso, en ese caso se usan los métodos **only** y **except** de la siguiente manera:

```
Route::resource('coches', CarController::class)->only([  
    'index', 'show'  
]);
```

```
Route::resource('coches', CarController::class)->except([  
    'create', 'store', 'edit', 'update', 'destroy'  
]);
```

6.- CRUD, controladores tipo recurso

Se pueden usar combinadas todas las técnicas vistas anteriormente:

```
Route::resource('coches', CarController::class)
    ->parameters(['coches'=>'coche'])
    ->only([
        'index', 'show'
    ])
    ->names([
        'index' => 'coches.lista',
        'show' => 'coches.mostrar'
    ]);
```

```
GET|HEAD api/user .....
GET|HEAD coches ..... coches.lista > CarController@index
GET|HEAD coches/{coche} ..... coches.mostrar > CarController@show
GET|HEAD factorial/{numero} ..... OperacionesController@factorial
```

6.- CRUD, controladores tipo recurso

Cuando se tienen varios controladores tipo recurso y se quieren añadir todas sus rutas, se pueden añadir con una única instrucción:

```
Route::resources([  
    'cars' => CarController::class,  
    'customers' => CustomerController::class,  
]);
```

La instrucción anterior es similar a las dos siguientes:

```
Route::resource('cars', CarController::class);  
Route::resource('customers', CustomerController::class);
```

6.- CRUD, controladores tipo recurso

Los controladores tipo recurso al final son controladores como otros cualquiera, por ello **se pueden añadir otros métodos si se necesitan**.

Se pueden añadir métodos:

- Para responder a rutas: por ejemplo un método que muestre los coches en oferta.
- Que se usen internamente en el proyecto.

En el caso de añadir métodos que **respondan a rutas, habrá que declarar esas rutas antes de las rutas del tipo recurso**.

Para el ejemplo del método que mostrará los coches en oferta:

```
Route::get('coches/oferta', [CarController::class, 'oferta']);  
Route::resource('coches', CarController::class);
```

7.- Vistas asociadas a controladores tipo recurso

Para mantener la organización de los archivos en el proyecto, se deben **agrupar todas las vistas de un controlador dentro de una carpeta.**

Así, todas las vistas que se usen en el controlador **CarController** se almacenarán en la carpeta **resources/views/cars.**

En ese caso es importante indicar bien la ruta a la vista, por ejemplo para la vista `resources/views/cars/show`:

```
return view('cars.show');
```

Práctica

Actividad 5: Primer controlador.