

Clase Storage

UD 2



¿Qué veremos hoy?

- Rutas en Laravel
- Controlador (procesar datos)
- Clase Storage (cargar y persistir datos)



¿Qué son las rutas en Laravel?

Las rutas son la manera en la que Laravel mapea las URLs de tu aplicación hacia controladores.

En Laravel, las rutas son manejadas a través de archivos de rutas en la carpeta `routes`.



Tipos de rutas en Laravel:

- `web.php` : Rutas para la interfaz web, generalmente usan sesiones y cookies.
- `api.php` : Rutas específicas para APIs, usualmente no usan sesiones y están pensadas para ser "stateless" (sin estado).
 - **Usaremos este fichero de rutas en la asignatura.**
 - Para habilitarlo `php artisan install:api` .
 - Todas las peticiones a una API deben tener el header:
accept: application/json



Archivo de rutas para APIs (`api.php`):

Todas las rutas dentro de este archivo automáticamente tienen el prefijo `/api` en la URL.

Cada ruta en una API generalmente corresponde a una operación CRUD:

- **GET:** Obtener datos, como una lista de usuarios o un usuario específico.
- **POST:** Crear un nuevo recurso, como agregar un usuario nuevo.
- **PUT:** Actualizar un recurso existente.
- **DELETE:** Eliminar un recurso.



Definiendo Rutas API en Laravel

Para crear rutas en el archivo `routes/api.php`, puedes usar el siguiente formato:

```
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\UsuarioController;

// Obtener lista de usuarios
Route::get('/usuarios', [UsuarioController::class, 'index']);

// Crear un nuevo usuario
Route::post('/usuarios', [UsuarioController::class, 'store']);

// Obtener un usuario específico por ID
Route::get('/usuarios/{id}', [UsuarioController::class, 'show']);

// Actualizar un usuario por ID
Route::put('/usuarios/{id}', [UsuarioController::class, 'update']);

// Eliminar un usuario por ID
Route::delete('/usuarios/{id}', [UsuarioController::class, 'destroy']);
```




Definiendo Rutas API en Laravel

Puedes definir todas las rutas en una única línea con:

```
use App\Http\Controllers\PhotoController;  
  
Route::apiResources([  
    'photos' => PhotoController::class,  
]);
```

+info: <https://laravel.com/docs/11.x/controllers#api-resource-routes>



Controlador

Es el componente que usaremos para procesar los datos del almacenamiento primario.

Para crear un controlador de API:

```
php artisan make:controller UsuarioController --api
```

La opción `--api` crea métodos de controlador comunes para una API, como `index`, `store`, `show`, `update`, y `destroy`.



Definir los métodos en el controlador

En el controlador, define la lógica para cada método. Por ejemplo, los métodos *index* y *store* del UsuarioController:

```
namespace App\Http\Controllers;  
use Illuminate\Http\Request;
```

```
class UsuarioController extends Controller {
```

```
    function index() {
```

```
        //Logica de negocio para cargar, procesar y mostrar  
        usuarios
```

```
    }
```

```
    function store(Request $request) {
```

```
        //Logica de negocio para cargar, procesar y almacenar  
        usuarios en secundaria
```

```
    }
```

```
}
```

Debes saber ...





Acceso a parámetros de Request

Ejemplo: Crear un fichero

- Petición HTTP:
POST http://localhost:8000/api/file?
filename=jose.txt&content=helloJose
- Ruta en api.php:
Route::post('file',)
- Metodo del controlador:

```
function store  
(Request $request) {  
    $content = $request->input('content');  
    $filename = $request->input('filename');  
}
```



Acceso a parámetros de Request

Ejemplo: Actualizar un fichero

- Petición HTTP:
PUT <http://localhost:8000/api/file/jose.txt?content=helloJose>
- Ruta en api.php:
Route::put('file/{filename}',)
- Metodo update del controlador:

```
function update  
(Request $request, string $filename) {  
    $content = $request->input('content');  
    Storage::disk('local')->put($filename, $content);  
}
```



Validación de parámetros

Es importante asegurarse que los parámetros de entrada son los correctos. Este proceso se llama validación de parámetros. En caso de no superar esta validación, se devolverá un código HTTP 422.

Ejemplo: actualizar el contenido de un fichero con validación

```
function store  
(Request $request) {  
    $request->validate([  
        'content' => 'required',  
        'filename' => 'required',  
    ]);  
}
```

Haciendo uso de `$request->validate`, Laravel responderá automáticamente un error 422 si no se validan los parámetros de entrada correctamente.

+info: <https://laravel.com/docs/11.x/requests#input>



Clase Storage

En Laravel, la clase `Storage` permite manejar archivos en el sistema de archivos de manera sencilla.

Laravel usa el sistema de archivos configurado en `config/filesystems.php`.

Por defecto, el sistema está configurado para usar el disco `local`, que guarda archivos en la carpeta `storage/app`.



Métodos de la clase Storage

- `Storage::disk()`

Permite especificar el disco que deseas usar

```
Storage::disk('local')
```



Métodos de la clase Storage

- `put()`

Almacena un archivo en el disco especificado

```
Storage::put('ruta/archivo.txt', 'Contenido del  
archivo');
```




Métodos de la clase Storage

- `get()`

Obtiene el contenido de un archivo.

```
$contenido = Storage::get('ruta/archivo.txt');
```



Métodos de la clase Storage

- exists()

Verifica si un archivo existe en el disco

```
if (Storage::exists('ruta/archivo.txt')) {  
    // El archivo existe  
}
```



Métodos de la clase Storage

- delete()

Elimina uno o varios archivos.

```
Storage::delete('ruta/archivo.txt');  
Storage::delete(['ruta/archivo1.txt',  
                'ruta/archivo2.txt']);
```




Métodos de la clase Storage

- `copy()`

Copia un archivo a otra ubicación.

```
Storage::copy('ruta/archivo.txt',  
'ruta/nuevo_archivo.txt');
```



Métodos de la clase Storage

- `move()`

Mueve o renombra un archivo.

```
Storage::move('ruta/archivo.txt',  
'ruta/nuevo_archivo.txt');
```



Métodos de la clase Storage

- `size()`

Obtiene el tamaño de un archivo en bytes.

```
$size = Storage::size('ruta/archivo.txt');
```



Métodos de la clase Storage

- `download()`

Descarga el archivo como respuesta HTTP.

```
return Storage::download('ruta/archivo.txt');
```




Métodos de la clase Storage

- files()

Obtiene una lista de archivos en un directorio.

```
$files = Storage::files('ruta/directorio');
```



Métodos de la clase Storage

- allFiles()

Obtiene todos los archivos de un directorio de manera recursiva.

```
$allFiles = Storage::allFiles('ruta/directorio');
```



Métodos de la clase Storage

- `directories()`

Obtiene una lista de los subdirectorios en un directorio.

```
$directories = Storage::directories('ruta');
```



Métodos de la clase Storage

- `allDirectories()`

Obtiene todos los subdirectorios de un directorio de manera recursiva.

```
$allDirectories = Storage::allDirectories('ruta');
```




Métodos de la clase Storage

- `makeDirectory()`

Crea un nuevo directorio.

```
Storage::makeDirectory('ruta/nuevo_directorio');
```



Métodos de la clase Storage

- `deleteDirectory()`

Elimina un directorio y todos sus archivos.

```
Storage::deleteDirectory('ruta/directorio');
```



Importante

Asegúrate de incluir la clase Storage en tu controlador.

```
use Illuminate\Support\Facades\Storage;
```



```
namespace App\Http\Controllers;

use Illuminate\Support\Facades\Storage;

class FileController extends Controller
{
    public function crearArchivo()
    {
        // Contenido del archivo
        $nombre = 'Tu Nombre Aquí';
        $contenido = "Hola Mundo\nNombre: " . $nombre;

        // Crear el archivo en storage/app/hola_mundo.txt
        Storage::put('hola_mundo.txt', $contenido);

        return "Archivo hola_mundo.txt creado con éxito!";
    }
}
```



¿Cómo compruebo que funciona?

1. Creo una ruta en api.php apuntando a ese controlador.
2. Realizo una llamada:
 - Usar postman
 - Crear un test
3. Compruebo la salida



Postman

1. Descarga y configura el plugin de postman en VSCode.
2. Levanta el servidor de desarrollo con:
php artisan serve
3. Realiza la petición

POSTMAN

My Workspace

New HTTP Request

Filter

adipr1

auth

employee

ud2-storage

post hello

get files

get one file

update file

elimina fichero

user

get files

ud2-storage / get files

Save

View Documentation

No Environment

GET

http://localhost:8000/api/hello

Send

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Code

Cookies

Query Params

	Key	Value
	Key	Value

Body

Cookies

Headers (7)

Test Results

Status: 200 OK

Time: 141 ms

Size: 317 B

Pretty

Raw

Preview

JSON

```
1  {
2    "mensaje": "Listado de ficheros",
3    "contenido": [
4      ".gitignore",
5      "example.txt",
6      "jose.txt",
7      "jose2.txt"
8    ]
9  }
```





Test

```
php artisan make:test CrearArchivoApiTest --  
feature
```

Archivo: tests/Feature/CrearArchivoApiTest.php

```
public function test_crear_archivo_api_con_nombre_jose()
{
    // Utilizar el disco 'local' de forma simulada
    Storage::fake('local');

    // Nombre que queremos probar
    $nombre = 'jose';

    // Realizar la solicitud GET a /api/hola-mundo/jose
    $response = $this->get("/api/hola-mundo/{$nombre}");

    // Verificar que la respuesta tiene el código 201 (Creado)
    $response->assertStatus(201)
        ->assertJson([
            'mensaje' => 'Archivo hola_mundo.txt creado con éxito!',
            'contenido' => 'Hola jose',
        ]);

    // Verificar que el archivo hola_mundo.txt fue creado
    Storage::disk('local')->assertExists('hola_mundo.txt');

    // Verificar el contenido del archivo
    $contenidoArchivo = Storage::disk('local')->get('hola_mundo.txt');
    $this->assertEquals('Hola jose', $contenidoArchivo);
}
```



Test

php artisan test



Ejercicios

Los ejercicios para esta unidad consistirán en relizar un ***fork*** de un proyecto Laravel 11.x e implementar todo lo necesario para superar todos los test con exito.

Cada test corresponderá a un ejercicio y superar ese test implicará superar ese ejercicio y, por lo tanto, obtener esa puntuación.

La entrega consistirá en abrir una **Pull Request** y entregar dicha URL. Tener en cuenta que la corrección será automática, es decir, por cada PR abierta se disparará un Job de GitHub Actions que pasará todos los test. Por lo tanto, es conveniente revisar la salida de de ese Job y comprobar que efectivamente, se han superado los tests.