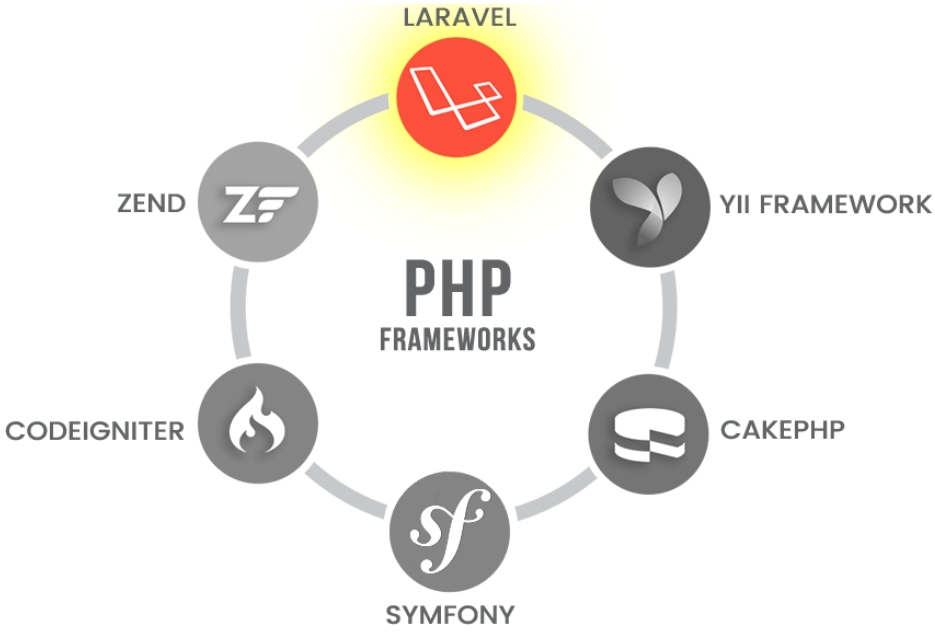
unidad didáctica 7

actividades



A tener en cuenta:

1. Dado que crearemos distintos elementos del framework Laravel desde Artisan con usuario *root* deberemos de tener en cuenta que, una vez creado dicho elemento, cambiar el *usuario*:*grupo* de ese fichero. Para acelerar esta tarea, desde la carpeta de Laravel:

```
sudo chown tuUsuario:tuGrupo -R tuProyecto
```

2. Si, al intentar ejecutar la *migración* o un *seeder* el navegador muestra un error de permisos de acceso a la base de datos, deberemos de realizar tres tareas:

   a) Acceder al fichero `.env` de tu proyecto laravel y configurar el nombre de base de datos, usuario y contraseña:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=dwes
DB_USERNAME=dwes
DB_PASSWORD=dwes
```

   b) Acceder a la base de datos (por ejemplo, mediante phpMyAdmin) y poner permisos al usuario en la base de datos del punto *a*.

   c) Si todavía persisten los errores de permisos, ejecutar desde Artisan:

```
sudo docker-compose exec myapp php artisan cache:clear
sudo docker-compose exec myapp php artisan config:clear
sudo docker-compose exec myapp php artisan config:cache
```

1. ejercicio 1. mostrar tabla

Sobre el proyecto de la sesión anterior, vamos a crear una página (vista) en la que se muestre el contenido de una tabla.a añadir un formulario (parecido) al que hemos estado creando (registro) pero que muestre (por ejemplo) un producto (como por ejemplo libros):

1.1. crear controlador y modelo

Crear un **controlador** para la tabla `libros`. Para esto, recuerda:

1. crear un controlador:

```
sudo docker-compose exec myapp php artisan make:controller
LibroController --model=Libro
```

   Este comando generará, a la vez, un controlador y su modelo relacionado. `database/migrations`.

   Esto creará el controlador en la carpeta `App\Http\Controllers` y el modelo en la carpeta `App\Models`.

2. Si queremos mostrar todas las filas de la tabla `libros` creamos el siguiente método en el **controlador** anteriormente generado:

```
    public function mostrarTodos()
    {
        // Obtener todas las filas de la tabla "libro"
        $libros = Libro::all();

        // Pasar los datos a la vista
        return view('libros.mostrar', ['libros' => $libros]);
    }
```

   En dicho método se aprecia que guarda en la variable `$libros` todos los libros de la tabla libros. A continuación, lanza una vista de nombre `mostrar.blade.php` almacenada en la carpeta `resources/views/libros` en la que pasa por parámetro la variable $libros anteriormente creada.

3. Por otro lado, en el **modelo** debemos insertar en la clase `Libro` :

```
protected $table = 'libros';
```

1.2. crear tabla

Crea una **tabla** `libros` en tu base de datos. Para esto, recuerda:

1. crear una migración para crear la tabla libros:

```
sudo docker-compose exec myapp php artisan make:migration
create_libros_table
```

Este comando generará un nuevo archivo de migración en el directorio `database/migrations`.

2. abre el archivo de migración recién creado. Puedes encontrarlo en el directorio `database/migrations` y tendrá un nombre similar a `2024_01_01_000000_create_libros_table.php` (la fecha y la hora pueden variar).
3. dentro del archivo de migración, encontrarás dos métodos: `up` y `down`. En el método `up`, define la estructura de la tabla "libro". Puedes hacerlo utilizando la sintaxis del constructor de esquemas de Laravel. Aquí hay un ejemplo básico:

```php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateLibroTable extends Migration
{
    public function up()
    {
        Schema::create('libros', function (Blueprint $table) {
            $table->id();
            $table->string('titulo');
            $table->text('descripcion')->nullable();
            // Agrega más columnas según sea necesario para tu aplicación
            $table->decimal('precio', 8, 2);
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('libros');
    }
}
```

En este ejemplo, la tabla `libros` tiene una columna de ID, un título, una descripción, un precio (que hemos añadido nosotros pues podemos personalizar la estructura de la tabla según nuestras necesidades) y las marcas de tiempo (`created_at` y `updated_at`).

4. Después de definir la estructura de la tabla, ejecuta las migraciones con el siguiente comando:

```
sudo docker-compose exec myapp php artisan migrate
```

Este comando aplicará la migración y creará la tabla "libro" en tu base de datos.

1.3. crear seeder

Crea un **seeder** para introducir información:

```
sudo docker-compose exec myapp php artisan make:seeder LibroSeeder
```

Este comando generará un nuevo archivo de seeder en el directorio `database/seeders`.

1. abre el archivo de seeder recién creado. Puedes encontrarlo en el directorio `database/seeders` y tendrá un nombre similar a `LibroSeeder.php`.

   Dentro del archivo de seeder, dentro del método `run`, puedes utilizar el modelo `Libro` para insertar datos en la tabla. Asegúrate de tener el modelo `Libro` creado en tu aplicación. Aquí tienes un ejemplo básico:

```php
use Illuminate\Database\Seeder;
use App\Models\Libro;

class LibroSeeder extends Seeder
{
    public function run()
    {
        // Ejemplo de inserción de datos en la tabla "libro"
        Libro::create([
            'titulo' => 'Libro 1',
            'descripcion' => 'Descripción del Libro 1',
            'precio' => 15.99,
        ]);

        Libro::create([
            'titulo' => 'Libro 2',
            'descripcion' => 'Descripción del Libro 2',
            'precio' => 21.50,
        ]);

        // ***********************************
        // AGREGA MÁS LIBROS (POR LO MENOS 5)
        // ***********************************
    }
}
```

2. ejecutar el seeder y insertar los datos en la tabla, utiliza el siguiente comando:

```
sudo docker-compose exec myapp php artisan db:seed --class=LibroSeeder
```

   Esto ejecutará el seeder que acabas de crear y agregaría los datos a la tabla "libro".

> **Recuerda** que, además de ejecutar un seeder específico, también puedes utilizar el comando `php artisan db:seed` sin especificar un seeder en particular para ejecutar todos los seeders definidos en tu aplicación.

1.4. crear ruta

Crear una ruta para acceder a dicho formulario de inserción de libros (productos).

Para ello, recuerda introducir en el fichero `web.php` de la carpeta `routes` la ruta para lanzar el controlador-mostrarTodos (y, si no se introduce de forma automática, poner también el `use` que apunte a dicho controlador).

```
Route::get('/libros', [LibroController::class, 'mostrarTodos']) ->
name('libros');
```

1.5. crear vista

Crear la vista `resources/views/libros/mostrar.blade.php` e introducir el código para recorrer la variable $libros que se pasa desde el método `mostrarTodos` del controlador `LibroController`. Para esto, recuerda, utilizar la directiva `@foreach` ... `@foreach` y acceder a cada campo mediante `$libro->id` (por ejemplo).

Crea en tu página principal (del ejemplo seguido durante esta unidad) un enlace a dicha ruta para probar el ejercicio (también lo podrás probar añadiendo a la URL de la aplicación `/mostrar-libros`).

2. ejercicio 2. insertar registro

Crear una página (vista) en la que se inserte un registro de la tabla libros. Un formulario (parecido) al que hemos estado creando (registro) pero que muestre un libro:

1. Recuerda poner en el método `store` en el controlador:

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

        $this->validate($request, [
            'titulo' => ['required', 'unique:libros'],
            'precio' => ['required', 'numeric',
                        'min: 0','max:10000',
                        'regex:/^\d+(\.\d{1,2})?$/']
        ]);
    }
```

   - `'max:10000'` para establecer un valor máximo permitido.
   - `'regex:/^\d+(\.\d{1,2})?$/'` para permitir decimales con hasta dos lugares decimales.

2. Por otro lado, en el **modelo** debemos insertar en la clase `Libro` :

```php
    protected $fillable = [
        'titulo',
        'descripcion',
        'precio',
    ];
```

En Laravel, el atributo `protected $fillable` se utiliza para especificar qué campos de una tabla de base de datos pueden ser asignados masivamente. Cuando se realiza una operación de asignación masiva, como al crear un nuevo modelo utilizando el método `create` o al actualizar un modelo mediante el método `update`, Laravel utiliza el conjunto de campos especificados en `$fillable` para determinar qué datos se deben incluir en la operación.

En este caso, al crear o actualizar un usuario, solo los campos `titulo`, `descripcion` y `precio` serán aceptados en una asignación masiva. Si intentas asignar otros campos no especificados en `$fillable`, Laravel lanzará una excepción `MassAssignmentException` para proteger contra la asignación masiva no autorizada.

Este enfoque ayuda a mejorar la seguridad al limitar qué campos pueden ser modificados de manera masiva, evitando posibles ataques de asignación masiva no autorizada. Es una práctica recomendada utilizar `$fillable` para definir explícitamente los campos que pueden ser asignados masivamente en lugar de permitir la asignación masiva de todos los campos con `$guarded`.

2.1. crear formulario

[...]

2.2. crear página

[...]

¿Qué entregar?

Como entrega de esta sesión deberás comprimir tu proyecto **laravel** con los cambios incorporados, y eliminando las carpetas `vendor` y `node_modules`. Renombra el archivo comprimido a `proyecto_07.zip`.