Seeders y factories

En las pruebas que hemos hecho hasta ahora, para tener datos con que probar la aplicación, nos hemos limitado a añadirlos a mano desde *phpMyAdmin*, o bien desde el código con algunos datos simples como "Título de prueba 1" o cosas similares.

Dado que los datos de inicio son necesarios para probar algunas funcionalidades básicas de la aplicación, como son las búsquedas y filtrados, y dado que los formularios para dar de alta y gestionar estos datos normalmente no se tienen listos hasta etapas más tardías, puede resultar conveniente disponer de algún mecanismo que genere estos datos de prueba al inicio, sin preocuparnos de tocar la base de datos a mano o alterar el código de la aplicación para ello. En este aspecto, los *seeders* y *factories* juegan un papel importante.

1. Los *seeders*

Los *seeders* son clases especiales que permiten sembrar (*seed*) de contenido una aplicación. Para crearlos, utilizamos el comando `php artisan` como sigue:

```
php artisan make:seeder NombreSeeder
```

Esto creará una clase llamada `NombreSeeder` en la carpeta `database/seeds` (hasta Laravel 7) o `database/seeders` (desde Laravel 8). En el método `run` de dicha clase podemos crear los elementos que necesitemos añadir a la base de datos.

Por ejemplo, vamos a crear en nuestro proyecto *biblioteca* un seeder llamado `LibrosSeeder`:

```
php artisan make:seeder LibrosSeeder
```

Editamos el método `run` del *seeder* que hemos creado, y definimos este código para crear un autor con un libro asociado (deberemos incorporar con `use` los modelos de `Autor` y `Libro` previamente):

```
public function run()
{
    $autor = new Autor();
    $autor->nombre = "Juan Seeder";
    $autor->nacimiento = 1960;
    $autor->save();
    $libro = new Libro();
    $libro->titulo = "El libro del Seeder";
    $libro->editorial = "Seeder S.A.";
    $libro->precio = 10;
    $libro->autor()->associate($autor);
    $libro->save();
}
```

1.1. Añadiendo los *seeders* a la aplicación

Por defecto, los *seeders* que creamos no forman parte de la aplicación aún, en el sentido de que aún no los podemos ejecutar. Para ello, debemos darlos de alta en el *seeder* general, llamado `DatabaseSeeder`, ubicado en la misma carpeta que los *seeders* que definimos:

```
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        ...
        $this->call(LibrosSeeder::class);
    }
}
```

1.2. Lanzar los *seeders*

Si queremos ejecutar los *seeders* para que añadan los datos, emplearemos este comando:

```
php artisan db:seed
```

Esto lanzará todos los *seeders* que tengamos declarados en la clase `DatabaseSeeder`. Si sólo queremos lanzar uno en concreto, podemos hacer lo siguiente:

```
php artisan db:seed --class=LibrosSeeder
```

También puede ser necesario (y a veces conveniente) limpiar la base de datos y llenarla desde cero con los datos de los seeds para empezar a probar la aplicación. En este caso, el comando es el siguiente:

```
php artisan migrate:fresh --seed
```

2. Los *factories*

Los *seeders* son una herramienta útil para poblar nuestra aplicación con datos al inicio. Podemos, por ejemplo, dar de alta una serie de usuarios iniciales con acceso a la aplicación, para que con ellos se puedan rellenar el resto de datos. También podemos dar de alta una serie de datos predefinidos en ciertas tablas, o datos de prueba que luego poder borrar.

Sin embargo, los *seeders* por sí solos se quedan algo "cojos". ¿Qué tendríamos que hacer para dar de alta 10 o 20 libros en nuestra base de datos de *biblioteca*? Tendríamos que definir algún tipo de bucle en el *seeder*, y definir datos diferentes (por ejemplo, con identificadores o contadores aleatorios) para cada libro. Para facilitar esta tarea, podemos echar mano de los *factories*.

Los *factories* son clases que permiten generar datos por lotes. Se crean con el siguiente comando, almacenándose la clase en la carpeta `database/factories`:

```
php artisan make:factory NombreFactory
```

Por ejemplo, vamos a crear un *factory* para generar autores y otro para libros:

```
php artisan make:factory AutorFactory
php artisan make:factory LibroFactory
```

2.1. Usando *factories* hasta Laravel 8

Hasta la aparición de Laravel 8, los *factories* eran básicamente un archivo PHP en la carpeta anteriormente citada `database/factories`, con un método `define` que debíamos completar con los datos que se van a emplear para generar objetos de esa factoría. Por ejemplo, así generamos autores con nombres al azar ("Autor X") y nacimientos al azar entre 1950 y 1990:

```
use App\Models\Autor;
use Faker\Generator as Faker;

$factory->define(Autor::class, function(Faker $faker) {
    return [
        'nombre' => "Autor " . rand(1, 100),
        'nacimiento' => rand(1950, 1990)
    ];
})
```

Notar también que utilizamos el modelo `Autor` (`Autor::class`) en lugar del modelo que viene por defecto al crear el *factory* (`Model:class`), deberemos cambiarlo en el código.

Ahora, para crear, por ejemplo, 5 autores aleatorios usando este *factory*, creamos el *seeder* correspondiente...

```
php artisan make:seeder AutoresSeeder
```

... llamamos al *factory* en su método `run` para crear 5 autores...

```
class AutoresSeeder extends Seeder
{
    public function run()
    {
        factory(Autor::class, 5)->create();
    }
}
```

... y damos de alta el nuevo *seeder* en `DatabaseSeeder`:

```
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        ...
        $this->call(AutoresSeeder:class);
        $this->call(LibrosSeeder::class);
    }
}
```

Usando los *fakers*

Estaremos de acuerdo en que generar datos del tipo "Autor 1", "Autor 2", etc, no queda demasiado "real" en una aplicación, por mucho que sean datos de prueba. Por ello, Laravel nos proporciona los *fakers* para generar datos al azar con una apariencia determinada. Así, podemos generar nombres reales aleatorios, o direcciones de correo electrónico, o frases, o textos largos. Si nos fijamos, cuando definimos un *factory* existe un parámetro de tipo `Faker` en la función `define`, que podemos emplear:

```
$factory->define(Autor::class, function(Faker $faker) {
    ...
```

Este objeto `Faker` dispone de una serie de propiedades que generan datos de un cierto tipo. Algunos de los más habituales son:

- `name`: genera un nombre de persona. Admite como parámetro opcional "male" o "female" para generar nombres masculinos o femeninos, respectivamente.
- `sentence`: genera una frase corta. Admite como parámetro opcional un número, indicando cuántas palabras generar.
- `word`: genera una palabra aleatoria.
- `text`: genera un texto largo.
- `phoneNumber`: genera un número de teléfono.
- `email`: genera un e-mail aleatorio.
- `randomNumber`: genera un número aleatorio. Como alternativa, también se tiene `numberBetween`, que genera un número aleatorio entre un mínimo y un máximo pasados como parámetro.
- ... etc (aquí podéis consultar más posibilidades al respecto).

Además, también tenemos disponible el método `unique()` para asegurarnos de que alguno de los campos que generemos no se repita entre registros.

Volviendo a nuestro ejemplo, vamos a modificar el *factory* de autores para que genere nombres reales, y años de nacimiento entre 1950 y 1990 usando el `Faker`:

```
use App\Models\Autor;
use Faker\Generator as Faker;

$factory->define(Autor::class, function(Faker $faker) {
    return [
        'nombre' => $faker->name,
        'nacimiento' => $faker->numberBetween(1950, 1990)
    ];
})
```

Si ahora actualizamos la base de datos, veremos los nuevos nombres generados:

```
php artisan migrate:fresh --seed
```

Generando datos relacionados

Para terminar con esta sección, hagamos las cosas bien. Hemos generado autores, pero esos autores escriben libros. ¿Cómo podemos generar N autores, cada uno con M libros asignados?

En primer lugar, vamos a modificar el *factory* de los libros para que genere un título, editorial y precio al azar (el precio entre 5 y 20 euros, por ejemplo, con 2 decimales):

```
use App\Models\Libro;
use Faker\Generator as Faker;

define(Libro::class, function(Faker $faker) {
    return [
        'titulo' => $faker->sentence,
        'editorial' => $faker->sentence(2),
        'precio' => $faker->randomFloat(2, 5, 20)
    ];
})
```

Así generamos 2 libros asignados a cada uno de los 5 autores creados con el *seeder* de autores:

```
class LibrosSeeder extends Seeder
{
    public function run()
    {
        $autores = Autor::all();
        $autores->each(function($autor) {
            factory(Libro::class, 2)->create([
                'autor_id' => $autor->id
            ]);
        });
    }
}
```

Como vemos, lo que hacemos es recorrer los autores previamente creados (por lo que el *seeder* de autores debe lanzarse ANTES que el de libros), y para cada uno, crear 2 libros asociados a ese *id* de autor.`

2.2. Usando *factories* desde Laravel 8

Uno de los cambios importantes que ha traído la versión 8 de Laravel es que ahora los *factories* están orientados a objetos, por lo que se engloban en clases. Además, por defecto se asocian a los modelos que

creamos, de forma que podemos generar una factoría de objetos a partir de una clase, como veremos a continuación. Por este motivo, cuando creamos un modelo se añade una cláusula `use` indicando que emplea el *trait* `HasFactory`.

```
class Libro extends Model
{
    use HasFactory;

    ...
}
```

Un *trait* básicamente es un conjunto de métodos que se puede emplear por cualquier clase que quiera utilizarlos. De este modo, se amortigua en parte la limitación de sólo poder heredar de una clase, y mediante estos *traits* podemos incorporar la funcionalidad de otras.

Cuando creamos una factoría en Laravel 8 empleando el comando `php artisan make:factory` comentado anteriormente, obtendremos una clase con el nombre que hayamos indicado, en la carpeta `database/factories`. Habrá un atributo `model` que deberá corresponder con el modelo (clase) asociada a este *factory*. Por ejemplo:

```
namespace Database\Factories;

use App\Models\Autor;
use Illuminate\Database\Eloquent\Factories\Factory;

class AutorFactory extends Factory
{
    /**
     * The name of the factory's corresponding model.
     *
     * @var string
     */
    protected $model = Autor::class;

    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            //
        ];
    }
}
```

Ahora deberemos rellenar el método `definition` con los datos que queramos generar para cada objeto que se cree. Por ejemplo, así emplearíamos el `faker` (ahora automáticamente incorporado en el propio objeto `$this`), para generar datos al azar para los autores:

```
public function definition()
{
    return [
        'nombre' => $this->faker->name,
        'nacimiento' => $this->faker->numberBetween(1950, 1990)
    ];
}
```

Del mismo modo, completamos la información del método `definition` para el *factory* de libros (`LibroFactory`):

```
public function definition()
{
    return [
        'titulo' => $this->faker->sentence,
        'editorial' => $this->faker->sentence(2),
        'precio' => $this->faker->randomFloat(2, 5, 20)
    ];
}
```

Finalmente, en los *seeder* correspondientes, podemos utilizar estos *factory* para generar N objetos del modelo asociado. Por ejemplo, para el caso de los autores:

```
class AutoresSeeder extends Seeder
{
    public function run()
    {
        Autor::factory()->count(5)->create();
    }
}
```

En el caso de los libros, procedemos igual que en las versiones anteriores de Laravel, pero teniendo en cuenta que para llamar a la factoría se debe utilizar el método estático del modelo asociado. Por ejemplo:

```
class LibrosSeeder extends Seeder
{
    public function run()
    {
        $autores = Autor::all();
        $autores->each(function($autor) {
            Libro::factory()->count(2)->create([
                'autor_id' => $autor->id
            ]);
        });
    }
}
```