Seeders en Laravel 5

Conoce los seeders, para alimentar una base de datos y crear datos de prueba o configurar el estado inicial de las tablas para un proyecto.

En esta ocasión vamos a hablar de los "seeders". Seed es "semilla" en inglés y "seeders" serían algo así como "sembradores", aunque prefiero traducir como "alimentador", ya que sería como yo llamaría a estos sistemas corrientemente.

Los seeders no son más que componentes del frameworkLaravel que sirven para inicializar las tablas con datos. Así como las migraciones nos permiten especificar el esquema de la base de datos, los seeders nos permiten también por medio de código alimentar las tablas con datos.

Su uso, como decimos puede ser para:

1. Crear datos de prueba con los que trabajar durante el desarrollo de la aplicación
2. Configurar el estado de las tablas que necesita nuestra aplicación para comenzar a trabajar

Para aclarar este segundo caso piensa en el ejemplo de los artículos. Cuando subes un artículo puede tener varios estados: "borrador", "publicado", "borrado", etc. Puede que esos estados estén definidos en una tabla, así luego puedes crear otros estados si lo necesitas, como "obsoleto". Pero quizás no vas a hacer en el backoffice para administrar los cambios sobre esa tabla y prefieres dejar una configuración inicial de los estados que prevees vas a necesitar para que la aplicación funcione según los requisitos pedidos. Entonces usarás el seeder para generar los estados de los artículos iniciales que te han pedido.

Crear un seeder

El proceso para crear un seeder comienza, como tantos otros, con el asistente artisan. Podemos pedirle que nos genere el esqueleto de un alimentador mediante el comando make:seeder indicando luego el nombre del seeder que queremos crear:

```
phpartisanmake:seederReviewsSeeder
```

Los seeder pueden tener cualquier nombre que necesites. La recomendación es indicar un nombre que te sirva para saber exactamente para qué se creó ese seeder. Por ejemplo podrás usar el nombre de la tabla que se va a alimentar y el sufijo "Seeder". Además como los seeders en código son clases, colocaremos la primera letra en mayúscula por convención. Algo como "BookSeeder" podría ser buen nombre o "BookTableSeeder".

```
phpartisanmake:seederBookSeeder
```

Los seeders una vez creados se colocan en la carpeta "database/seeds". Ejecuta el comando anterior y allí encontrarás el seeder creado por artisan.

Escribir el código de los seeder

Los seeder no son más que clases, de programación orientada a objetos, en las que tendrás el código de los datos que quieras insertar para alimentar tus tablas inicialmente. Estas clases tendrán un método llamado run() que contiene el código de los inserts que desees realizar dentro de tus tablas.

Existen varias maneras de "atacar" a la base de datos y producir los inserts que necesitas. Puedes usar sentencias construidas con "QueryBuilder" o directamente "Eloquent" a través de las operaciones disponibles en un modelo, eso es indiferente.

Nota: Esta parte la podemos describir de manera muy esquemática, porque todavía no hemos profundizado en el conocimiento que necesitamos para trabajar con la base de datos. Por ello nos vamos a limitar a insertar los datos valiéndonos de las operaciones del modelo. Para entender lo que viene a continuación tendrás que haber leído el capítulo de introducción a los modelos en Laravel 5. Si hace tiempo que lo leíste y no has practicado lo suficiente te recomendamos hacer una segunda lectura antes de proseguir.

Utilizando las operaciones de un modelo, y el ORM Eloquent, podemos insertar datos a través del método create() del modelo. Por ejemplo, si tenemos el modelo "Review", que afecta a la tabla "reviews". Entonces podrás hacer Review::create() para acceder a la operación de inserción de datos.

Este método create() recibe un array asociativo, que contiene cada uno de los campos que se quieren aplicar con los valores a definir. Son pares claves / valor, la clave es el nombre del campo y el valor es el valor que pretendemos asignar.

```
Review::create([
'campo'                 =>      'Valor campo',
'campo_num'             =>      0
]);
```

Nota: Recuerda que los modelos son clases, que se llaman con el mismo nombre que la tabla (la diferencia es que la primera letra de la tabla es en minúsculas y la primera legra de la clase de un modelo es en mayúsculas, así como la tabla está en plural y el modelo en singular). Esas clases en principio pueden estar vacías de código, puesto que la herencia de la clase principal Model ya nos ofrece la mayoría de las operaciones listas para usar. Para crear un modelo simplemente lanzas el correspondiente comando de artisanmake:model. Será algo que necesitarás realizar antes de poder invocar el método create() del modelo. De nuevo, lee el artículo de los modelos si no te suena lo que estamos diciendo.

Además, en el caso que te apoyes en el modelo para crear los alimentadores, tendrás que asegurarte que haces el correspondiente "use" para tener disponible el espacio de nombres (namespace) de la clase del modelo que vas a usar.

Por ejemplo, este sería el código para nuestro seeder de Reviews. El tema del namespace, para conocer el modelo Review está en la línea "use App\Review;".

```
<?php

useIlluminate\Database\Seeder;
useApp\Review;

classReviewsSeederextendsSeeder
{
publicfunctionrun()
{
Review::create([
        'name'                          =>      'Vacaslocas',
        'votes'                         =>      33,
        'fulldescription'                       =>      'lalala'
]);
}
}
```

Ejecutar los seeders

En un proyecto podemos haber creado varios seeders, para alimentar de manera individual varias tablas. Estos alimentadores se pueden ejecutar de manera global (todos a la vez) configurando el código de una clase que está en la carpeta "database/seeds" llamada "DatabaseSeeder".

Nota: DatabaseSeeder la encontrarás de manera predeterminada en toda instalación de Laravel 5 y con un código básico para comenzar. Esta clase podríamos decir que es el seeder "global". Podríamos situar todo el código de las alimentaciones en este mismo archivo, en el método run(), pero no es lo habitual, puesto que resulta de mayor utilidad separar los seeders en diversos archivos como hemos aprendido.

Dentro de DatabaseSeeder, en el método run() podremos hacer tantas llamadas a seeders particulares a través del método call(), indicando qué seeders queremos ejecutar a través del nombre de la clase.

```
publicfunctionrun()
{
Model::unguard();

$this->call(BookSeeder::class);
$this->call(ReviewsSeeder::class);

Model::reguard();
```

```
}
```

Las distintas clases seeder que hayas especificado se ejecutarán en el orden en el que estén escritas en este código.

Nota: Los métodos de Modelunguard() y reguard() sirven para anular temporalmente y luego reactivar ciertas protecciones de seguridad en el modelo, que evitan que te inyecten datos indeseados en las tablas. Hablaremos más adelante de ellas.

Una vez configurado el DatabaseSeeder, podemos lanzar toda la secuencia de seeders a través del comando de artisan:

```
phpartisandb:seed
```

Nota: Para realizar el "migrate" y a la vez procesar los seeders, en un solo paso, tienes como alternativa de lanzar el siguiente comando.

```
phpartisanmigrate:refresh--seed
```

Este comando lo podemos ejecutar un número indeterminado de veces. Si se repite su ejecución simplemente volverá a insertar los datos de nuevo en las tablas.

Ejecutar seeders de manera individual

En ocasiones podemos necesitar realizar la ejecución de un seeder en concreto y no todos los generados para un proyecto. Para conseguir esto podemos usar el comando de artisan "bd:seed" seguido del parámetro --class, al que le asignamos como valor el nombre del seeder a ejecutar de manera individual.

Quedaría algo como puedes ver en siguiente código:

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

Quizás nos ocurra entonces que recibamos un error:
[Illuminate\Database\Eloquent\MassAssignmentException]

Eso es por una protección que tiene Laravel para evitar que introduzcan datos que no deseamos en tablas. Lo veremos con detalle más adelante pero se soluciona colocando una pequeña declaración en la propiedad $fillable del modelo. No te preocupes de momento por ello, nosotros vamos a saltarnos las protecciones para este paso de la siguiente manera. Vamos a avisar al modelo que vamos a realizar la alimentación saltándonos la protección con el método unguard() de la clase Model.

```
Model::unguard();
```

Nota: La llamada a unguard() es una operación que encontrarás en el código del "seeder global" DatabaseSeeder. Verás ahí el código para levantar las protecciones de nuevo.

Claro, si vas a usar la clase Model debes indicarlo con el correspondiente "use" para conocer su namespace.

```
useIlluminate\Database\Eloquent\Model;
```

Con estas dos líneas, el código de un seeder para que puedas ejecutarlo de manera individual te quedaría parecido a esto:

```
<?php

useIlluminate\Database\Seeder;
useIlluminate\Database\Eloquent\Model;
useApp\Book;

classBookSeederextendsSeeder
{
publicfunctionrun()
{
Model::unguard();

Book::create([
        'name'          =>      'Viaje al centro de la tierra',
        'author'        =>      'Julio Verne',
        'isbn'          =>      '14445884'
]);
}
}
```

Conclusión

De los seeders no hay mucho más que hablar. Pero lógicamente nuestro código se puede complicar mucho en función de diversas situaciones que queramos resolver, como cargar juegos de datos (varios reg en vez de uno), borrar datos que pudiera haber que no se desean en el estado inicial, etc.

Antes de terminar te damos una sugerencia de código rápido para poder alimentar una tabla con una gran cantidad de elementos, simplemente a través de un bucle for.

```
for($i=0;$i<100;$i++){
        Book::create([
        'name'          =>      'Libro '.$i,
        'author'        =>      'Autor '.$i,
        'isbn'          =>      '14445884'.$i
]);
}
```

Para mejorar estos resultados podrías aprender a manejar Faker, una librería pensada para generar datos de prueba de modo que parezcan más reales, pero no es algo que de momento no vamos a tocar, pues es una librería externa a Laravel.

Además nos quedaría explorar toda la parte de acceso a datos con los métodos de los modelos, pero eso es algo que ya no depende del sistema de seeder y que veremos en los siguientes artículos.