Introducción a modelos en Laravel

Introducción a los modelos, parte del patrón MVC, en el framework PHP Laravel 5.

Siguiendo con una introducción básica a los componentes principales de Laravel, queremos hacer una primera aproximación a los modelos, de los que no habíamos hablado todavía. Pero antes que nada, conviene hacer una aclaración conceptual sobre qué es un modelo:

Los modelos son uno de los componentes principales de las aplicaciones desarrolladas bajo el patrón MVC, que tienen la responsabilidad de acceder a los datos, modificarlos, etc. En el patrón además los modelos mantienen lo que se llama la lógica de negocio, que son las reglas que deben cumplirse para trabajar con los datos.

Por tanto, el tipo de acciones que le vamos a solicitar a un modelo es por ejemplo, obtener datos, insertarlos, modificarlos, etc. En las operaciones que modifiquen los datos además se tendrá que realizar cierta validación de esos datos, para asegurarnos que tienen la forma que es necesaria antes de guardarlos.

Cuando pensamos en modelos muchos hacemos una conexión directa con la base de datos: "un modelo guarda el código de acceso a la base de datos". Pero no es exactamente así, ya que un modelo trabaja con datos que pueden venir de varias fuentes. Generalmente será la base de datos, pero podría ser un API, Servicio web, sistema de archivos, etc.

Modelos en Laravel 5

En Laravel 5 los modelos se gestionan en la carpeta "app", colocando los archivos de nuestro modelo sueltos ahí. En la instalación limpia de Laravel 5.1 tenemos un primer modelo que podemos abrir para echar un primer vistazo rápido sobre ellos. Es el archivo que está en la ruta "app/User.php".

Nota: Una de las modificaciones principales que aparecieron en la versión 5 de Laravel es que han quitado la carpeta de los modelos. Esto es porque te animan a que uses la estructura de carpetas que prefieras para los modelos. En principio pueden ir todos colgando de la carpeta "app", pero también podría crearse una carpeta "models" para situarlos allí, o crear cualquier otra estructura si lo ves conveniente.

Laravel además separa código que en el patrón MVC se ubica en la responsabilidad del modelo en diversas clases dispersas por diversos directorios. Por ejemplo, las validaciones y filtrado de los datos que se reciben se pueden colocar en el middleware o en los archivos de Requests para validación que no hemos visto todavía. Así que, los que conocemos otras arquitecturas de aplicaciones basadas en MVC debemos de abrir un poco la mente cuando entramos en Laravel.

Como puedes ver, los modelos tienen la primera letra en mayúscula, por implementarse mediante clases. Los archivos donde guardamos el código de los modelos también deben tener esa primera letra en mayúscula.

En Laravel los modelos se controlan por un ORM llamado Eloquent, al menos los modelos que están implementados como datos en una base de datos, pero no es un requisito, de modo que podríamos trabajar con otros ORM o incluso bajar a un nivel más bajo y trabajar con PDO directamente, o con las extensiones de nuestra base de datos en particular.

En el caso de ser un modelo Eloquent, los modelos están directamente asociados a una entidad y a su vez a una tabla de la base de datos, por lo que un modelo que se llama User está directamente relacionado con una tabla llamada con el mismo nombre en la base de datos, pero en minúscula y acabado en plural, ej "users".

Como siempre, te recomendamos comenzar por abrir el mencionado archivo con el modelo User.php para ver cómo se implementan en Laravel. Ese modelo está bien pero tiene un par de modificaciones un poco avanzadas que nos pueden despistar, así que preferimos explicarte los modelos con respecto a un ejemplo más vacío.

Crear un modelo vacío en Laravel

Como en otras ocasiones, podemos ayudarnos de artisan para crear un modelo de partida. Con el comando make:model, seguido del nombre del nuevo modelo, creamos un modelo vacío.

```
phpartisanmake:modelArticle
```

Eso nos crea en el directorio "app" el correspondiente modelo de Eloquent, que contiene un código como el que puedes ver a continuación.

```
<?php

namespaceApp;

useIlluminate\Database\Eloquent\Model;

classArticleextendsModel
{
 //
}
```

Eso es todo lo que necesita un modelo básico en Laravel. Como te puedes imaginar, la explicación de su sencillez es que esta clase extiende la clase Model de Laravel.

Estos modelos ya vienen con funcionalidades para solicitar datos que estén en la base de datos. Los modelos de Eloquent estarán asociados directamente con una tabla llamada "articles", sin que tengamos que configurar nada en nuestro código, aunque más adelante

aprenderemos a cambiar el nombre de la tabla asociada a un modelo, por si nos resultase necesario.

Este modelo usará el motor del ORM de Eloquent y lo puedes ver porque está haciendo uso de la clase Model que está en el namespaceIlluminate\Database\Eloquent. Esa clase se le asigna un alias llamado "Model" (el mismo nombre de la clase que luego hacemos el extends) gracias a la sentencia:

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

Nota: Si nuestro modelo trabajase con otro ORM, o con otra base de datos que no soporte Eloquent como MondoDB, no usaríamos esa clase Model para extenderlo, sino otra, y por tanto el trabajo sería diferente al que realizamos con Eloquent.

Fíjate también que el modelo se crea dentro del namespace "App", definido por la primera línea de código:

```
namespaceApp;
```

Acceder a datos del modelo

Desde los controladores querremos acceder a datos que mantienen los modelos: consultas, modificaciones, etc. Esas operaciones se hacen a través de la clase del modelo que acabamos de implementar.

De momento veamos cómo implementar una selección de todos los datos que tenemos en el modelo, invocando el método all() sobre el modelo que acabamos de crear.

```
\App\Article::all()
```

Este código estaría en un controlador, o en otra clase desde la que queramos acceder a los datos del modelo. Como puedes ver, para referirnos al modelo debemos indicar el espacio de nombres donde lo podemos encontrar, que en nuestro caso era "App".

Esa línea de código, como decíamos, nos devuelve una colección con los datos encontrados. Aunque de momento todavía no nos va a funcionar, porque la tabla "articles" no está creada en nuestro sistema gestor. En cambio obtendremos un error como este: "[...] Base tableorviewnotfound: 1146 Table 'proyecto.articles' [...]".

En futuros artículos veremos cómo crear nuestras tablas, con el sistema de migraciones y podremos comenzar a usar más a fondo los modelos. Pero como seguro estamos impacientes por comprobar si esa instrucción verdaderamente funciona, vamos a adelantar alguna cosa.

Crear una tabla manualmente de MySQL

Podemos crear manualmente la tabla que estamos necesitando en la base de datos. Como decimos, no sería el modo correcto de proceder pero de momento con lo que sabemos vamos a conformarnos. Usaremos nuestro cliente MySQL de preferencia, como MySQLWorkbench, Sequel Pro o incluso podríamos instalar PhpMyAdmin.

> Ahora creas la tabla y los datos de prueba.

```
1. CREATETABLE `articles` (
2.   `id` int(11) unsigned NOTNULLAUTO_INCREMENT,
3.   `name` varchar(200)COLLATE utf8_unicode_ci DEFAULTNULL,
4. PRIMARYKEY(`id`)
5. )ENGINE=InnoDBDEFAULTCHARSET=utf8 COLLATE=utf8_unicode_ci;
6.
7. INSERTINTO `articles` (`id`, `name`)
8. VALUES
9.   (1,'Probando'),
10.         (2,'Algo'),
11.         (3,'Lindo');
```

Teóricamente ahora ya podrás acceder a la página de antes, donde habías puesto en el controlador la instrucción para mostrar todos los artículos. Solo recuerda que para mostrar la salida por la página y así poder leerla debes hacer un print_r() o var_dump() porque es una colección. También puedes usar la función dd() que te ofrece Laravel.

```
dd(\App\Article::all());
```

Nota: Realmente no necesitas ni usar un controlador, podrías hacerlo directamente con una closure dentro del sistema de rutas.

```
Route::get('articulos',function(){
        dd(\App\Article::all());
});
```

Conclusión

Insistimos en que más adelante vamos a conocer mecanismos por los que se crean las tablas o se insertan datos de prueba directamente desde Laravel, cuando hablemos de "migrations y seeders". Aunque para trabajar en Laravel podríamos tener el schema de la base de datos hecho a mano directamente con SQL en el gestor de base de datos que estemos usando, no es la manera más habitual de proceder.

Además, hay otros métodos de acceder al sistema gestor de base de datos, como ya hemos advertido, con programas profesionales como MySQLWorkbench que dan muchas mejores prestaciones y aumentan la productividad, en comparación con trabajar directamente por el terminal.

De momento creemos que es suficiente para cumplir con lo que sabemos nuestro objetivo de poner en marcha esa llamada al modelo y recuperar información que hay en MySQL.