Las migraciones

Las migraciones son un mecanismo de definición de datos ofrecido por Laravel para, a través de ciertas clases y opciones de configuración, generar la estructura completa de una base de datos. A su vez, suponen una especie de control de versiones para una base de datos, y permiten crear y modificar el esquema de la misma fácilmente.

1. Estructura de las migraciones

Por defecto, Laravel trae unas migraciones predefinidas, que se hallan en la carpeta `database/migrations`. Cada una tiene un nombre de archivo que comienza por la fecha en que se hizo, seguida de una breve descripción de lo que contiene (creación de la tabla de usuarios, reseteo de contraseñas...). Puede que algunas de estas migraciones no nos vayan a ser necesarias, con lo que podemos borrarlas directamente, y puede que otras (en especial la creación de la tabla de usuarios) sí nos sirva, pero con otros campos, con lo que deberemos editarla, como veremos a continuación.

Si examinamos el contenido de una migración, todas deben tener dos métodos:

- `up` : permite agregar tablas, columnas o índices a la base de datos
- `down` : revierte lo hecho por el método anterior

Si observamos el contenido de un método `up` de los que vienen predefinidos para crear una tabla, vemos que se utilizan distintos métodos para definir los tipos de datos de cada campo de la tabla, como por ejemplo `id()` para campos que puedan contener enteros autoincrementales, o `string()` para campos de tipo texto. Además, existen otros métodos modificadores para agregar propiedades adicionales, como por ejemplo `unique()` para indicar valores únicos (claves alternativas), o `nullable()` para indicar que un campo admite nulos. Aquí tenemos un ejemplo de método `up` :

```
public function up()
{
    Schema::create('usuarios', function(BluePrint $tabla) {
        $tabla->id();
        $tabla->string('nombre');
        $tabla->string('email')->unique();
        ...
        $tabla->timestamps();
    });
}
```

Por defecto, como vemos en los ejemplos que se proporcionan, los esquemas se crean con un *id* autonumérico, y unos *timestamps* para indicar la fecha de creación y de modificación de cada registro, y que Laravel gestiona de forma automática cuando insertamos o actualizamos contenidos, lo cual resulta muy útil.

Sobre esta base, podemos añadir o quitar los campos que queramos. Para ver los tipos disponibles para las columnas de la tabla, podemos visitar la documentación de Laravel sobre migraciones, en concreto buscaremos el subapartado *Available Column Types*. Conviene tener presente, por ejemplo, que el tipo `string` que hemos utilizado en el ejemplo anterior tiene una limitación de 255 caracteres. Para textos más grandes, se puede emplear el tipo `text` (20.000 caracteres aproximadamente) o `longText`.

Podemos especificar una clave primaria con el método `primary`, al que le podemos pasar o bien el nombre del campo clave, o un array de campos clave, en el caso de que ésta sea compuesta. Por defecto, los campos de tipo `id` se auto-establecen como claves primarias.

```
$table->primary(['campo1', 'campo2']);
```

2. Creación de migraciones

Creamos migraciones con el comando:

```
php artisan make:migration nombre_migracion
```

Por ejemplo:

```
php artisan make:migration crear_tabla_prueba
```

Notar que Laravel ya asigna automáticamente la fecha de la migración, sólo debemos especificar el nombre descriptivo de la misma. Además, si Laravel detecta la palabra *create* en el nombre de la migración, finalizada

en *table*, intuye que es para crear una tabla nueva. En cambio, si detecta la palabra *to* (entre otras), y al final la palabra *table*, intuye que se va a alterar o modificar una tabla existente. Esto es gracias a la clase `TableGuesser` incorporada en Laravel, que detecta ciertos patrones en los nombres de migraciones. La diferencia entre la creación y la modificación es que en el método `up` de la migración se utilizará `Schema::create` o `Schema::table` sobre la tabla en cuestión, respectivamente.

En cualquier caso, también podemos especificar un parámetro adicional en el comando de migración para indicar si queremos crear o modificar una tabla, y de este modo podemos definir el nombre de la migración en el idioma que queramos, y sin restricciones de patrones. Estas dos migraciones crean una tabla (*pedidos*) y modifican otra (*usuarios*), respectivamente:

```
php artisan make:migration crear_tabla_pedidos --create=pedidos
php artisan make:migration nuevo_campo_usuario --table=usuarios
```

En el caso de la segunda migración, si, por ejemplo, queremos añadir una columna con el número de teléfono de los usuarios, puede quedar así (tanto el método `up` como el `down`):

```
public function up()
{
    Schema::table('usuarios', function(Blueprint $tabla) {
        $tabla->string('telefono')->nullable();
    });
}

public function down()
{
    Schema::table('usuarios', function(Blueprint $tabla) {
        $tabla->dropColumn('telefono');
    });
}
```

Si queremos que el campo en cuestión esté en un orden concreto, podemos usar el método `after` para indicar detrás de qué campo queremos ponerlo (en el método `up`):

```
$tabla->string('telefono')->after('email')->nullable();
```

2.1. Ejecución y borrado de migraciones

Para ejecutar las migraciones (el método `up` de cada una), lanzamos el siguiente comando desde la carpeta de nuestro proyecto (habiendo creado la base de datos ya previamente, y modificado las credenciales de acceso en el archivo `.env`):

```
php artisan migrate
```

Adicionalmente a las tablas afectadas, se tendrá otra tabla `migrations` en la base de datos con un histórico de las migraciones realizadas. Para cada una, se almacena su *id* (autonumérico), el nombre de la migración, y el número de proceso por lotes en que se hizo (aquellas que compartan el mismo número se hicieron a la vez en el mismo lote). De este modo, aquellas que ya se hayan hecho no se volverán a realizar.

Para deshacer las migraciones realizadas (ejecutar el método `down` de las mismas), ejecutamos:

```
php artisan migrate:rollback
```

Esto eliminará TODAS las migraciones del último lote existente en la tabla `migrations`. Si no queremos deshacerlo todo, sino retroceder un número determinado de migraciones dentro de ese lote, ejecutamos el comando anterior con un parámetro `--step`, indicando el número de pasos o migraciones a deshacer (en orden cronológico de más reciente a más antigua):

```
php artisan migrate:rollback --step=2
```

Si volvemos a hacer la migración, se restablecerán las migraciones deshechas de ese lote.

Otro comando también muy utilizado es `migrate:fresh`. Lo que hace es eliminar todas las migraciones realizadas y volverlas a lanzar. Es útil cuando, estando en desarrollo, añadimos campos nuevos a alguna tabla y queremos rehacer las tablas completamente.

```
php artisan migrate:fresh
```

> **NOTA**: el comando `migrate:fresh` es DESTRUCTIVO, elimina los contenidos de las tablas, y sólo debe utilizarse en entornos de desarrollo, no de producción.

2.2. Aplicando las migraciones a nuestro ejemplo

Vamos a poner en práctica todo lo visto en este apartado sobre nuestro proyecto `biblioteca`. Anteriormente ya hemos comentado cómo modificar el archivo `.env` del proyecto para darle los parámetros de conexión correctos a la base de datos, y cómo crear la base de datos "biblioteca" desde *phpMyAdmin*. Revisa ese apartado para hacer estos pasos, si no los has hecho ya.

A continuación, vamos a eliminar las migraciones que no nos van a ser necesarias de la carpeta `database/migrations`. En concreto, borramos todas salvo la de creación de la tabla de usuarios `create_users_table`.

Después, editamos la migración para la tabla de usuarios (`create_users_table`), ya que la utilizaremos en sesiones posteriores. Podemos renombrar el archivo a `crear_tabla_usuarios` . La clase interna en versiones recientes de Laravel no tiene nombre, se crea simplemente un subtipo de *Migration*:

```
...

return new class extends Migration
{
    ...
}
```

> **NOTA:** en versiones anteriores donde sí tenga nombre, podemos reemplazar el nombre viejo (*CreateUsersTable*) por *CrearTablaUsuarios*, por ejemplo.

La tabla a la que se alude en los métodos `up` y `down` también la renombramos a `usuarios` , para dejarlo en nuestro idioma (respetando la fecha de creación en el nombre del archivo), y después editamos el método `up` para dejarlo así:

```
public function up()
{
    Schema::create('usuarios', function(Blueprint $table) {
        $table->id();
        $table->string('login')->unique();
        $table->string('password');
        $table->timestamps();
    });
}
```

Ahora vamos a crear una nueva migración para definir la estructura de los libros:

```
php artisan make:migration crear_tabla_libros --create=libros
```

Editamos después el contenido de esta migración, en concreto el método `up` para definir estos campos en los libros:

```
public function up()
{
    Schema::create('libros', function(Blueprint $table) {
        $table->id();
        $table->string('titulo');
        $table->string('editorial')->nullable();
        $table->float('precio');
        $table->timestamps();
    });
}
```

Cargamos las migraciones con el comando:

```
php artisan migrate
```

Tras esto, ya deberíamos ver en nuestra base de datos "*biblioteca*" las dos tablas creadas (*usuarios* y *libros*), junto con la tabla *migrations* que crea Laravel para gestionar las migraciones realizadas.

> **NOTA:** si, además de las tablas anteriores, aparece una tabla de *personal_access_tokens*, ésta la crea automáticamente el paquete *Sanctum*, incorporado por defecto en Laravel. No nos molesta para lo que vamos a hacer durante el curso, pero si queréis quitarla, hay que editar el archivo `app\Providers\AppServiceProvider.php` y añadir esta línea en el método `register` (junto con el correspondiente `use` para la clase `Sanctum`):

```
...
use Laravel\Sanctum\Sanctum;

class AppServiceProvider ...
{
    ...
    public function register()
    {
        Sanctum::ignoreMigrations();
    }
    ...
}
```

> Después, lanzamos `php artisan migrate:fresh` para ejecutar desde cero las migraciones, y esa tabla habrá desaparecido.