Query builder y uso de fechas

En este documento veremos un par de cuestiones secundarias que nos han quedado en el tintero. La primera es una forma alternativa de "atacar" la base de datos, sin emplear el ORM Eloquent: el *query builder*. La segunda es cómo trabajar cómodamente con fechas en aplicaciones Laravel.

1. El *query builder*

A la hora de obtener datos de la base de datos, en lugar de usar modelos de Eloquent, podemos emplear también el *query builder*, una herramienta incorporada con Laravel que permite realizar operaciones sobre la base de datos sin utilizar un modelo de objetos por detrás, y con una sintaxis diferente a SQL.

1.1. Consultas

Para utilizar estas consultas, utilizamos el elemento `DB`. Dicho elemento corresponde al espacio de nombres `Illuminate\Support\Facades\DB` que deberemos incluir. Internamente, tiene un método `table` para especificar la tabla sobre la que se quiere consultar. Una vez referenciada, con el método `get` obtenemos todos los registros:

```
use Illuminate\Support\Facades\DB;

...

$personas = DB::table('personas')->get();
```

A pesar de no estar trabajando con clases, lo que obtenemos aquí es un array de objetos, no un array asociativo.

En el caso de buscar un registro concreto (por su *id*, por ejemplo), utilizamos el método `where`, pasándole como parámetros el nombre del campo a comparar, y el valor que debe tener. Después, enlazamos con el método `first` para obtener sólo el primer registro de la búsqueda (de lo contrario, obtendríamos un array con un resultado, si buscamos por *id*):

```
$persona = DB::table('personas')->where('id', $id)->first();
```

1.2. Actualizaciones

Si lo que queremos hacer es una **inserción**, empleamos el método `insert` de la tabla. En este caso, le pasamos un array asociativo con los nombres de cada campo del nuevo registro, y sus valores:

```
DB::table('personas')->insert([
    'nombre' => 'Juan',
    'edad' => 56
]);
```

En el caso de **modificaciones**, utilizamos el método `where` para filtrar el registro o registros a modificar, y empleamos el método `update` con el array de campos a modificar:

```
DB::table('personas')->where('id', $id)->update([
    'nombre' => 'Juan',
    'edad' => 56
]);
```

Para **borrados**, usamos una estructura similar a la anterior, reemplazando la llamada a `update` por `delete`, que no necesita parámetros:

```
DB::table('personas')->where('id', $id)->delete();
```

2. Uso de fechas

En algunas tablas que hemos visto o creado, se ha usado un tipo *timestamp*, que básicamente genera un tipo fecha en la tabla correspondiente. Estos campos de tipo tabla son instancias de una librería PHP llamada *Carbon*, muy útil para trabajar con fechas. Así que, si tenemos un registro de tipo `Persona` con un campo `created_at` de tipo fecha, podemos trabajar con él como una fecha *Carbon*, y, por ejemplo, mostrarla en una vista con un formato específico:

```
<p>
Fecha creación:
{{ Carbon\Carbon::parse($persona->created_at)->format('d/m/Y') }}
</p>
```

Además, para trabajar sobre los campos `created_at` y `updated_at` que por defecto se crean en una tabla desde una migración Laravel, podemos emplear esta librería *Carbon* para darles valor, aunque de esto ya se encarga Eloquent automáticamente, pero por si lo queremos hacer manualmente, aquí va un ejemplo:

```
DB::table('personas')->insert([
    'nombre' => 'Juan',
    'edad' => 56,
    'created_at' => Carbon::now(),
    'updated_at' => Carbon::now()
]);
```

Para poder emplear la clase `Carbon`, debemos importarla (`use Carbon\Carbon`), o bien anteponerle siempre el prefijo del *namespace* `Carbon\Carbon`, como en el ejemplo de `format` anterior.