UD10.4 – Laravel Migraciones

2º CFGS
Desarrollo de Aplicaciones Web

2022-23

1.- Modelo de datos

En el patrón MVC (Modelo Vista Controlador) **el modelo** es el encargado de representar la información con la que trabaja la aplicación.

El modelo gestiona todos los accesos a la información.

Desde los controladores se hará uso de los modelos para almacenar, obtener, actualizar o borrar datos de la base de datos.

2.- Base de datos

Para la **persistencia** de los datos se usa generalmente una **base de datos**.

Laravel ofrece las herramientas necesarias para crear las tablas, campos, claves, relaciones… pero necesita tener acceso a una base de datos ya creada.

Por ello es necesario **crear previamente la base de datos**.

Como se está utilizando XAMPP, **para crear la base de datos se usará phpMyAdmin.**

2.- Base de datos: parámetros de conexión

Como ya se ha visto, la configuración más importante de un proyecto Laravel se centraliza en el archivo .env.

Para configurar el acceso a la base de datos se deben configurar los siguientes parámetros:

- DB_CONNECTION
- DB_HOST
- DB_PORT
- DB_DATABASE
- DB_USERNAME
- DB_PASSWORD

2.- Base de datos: parámetros de conexión

Por defecto se configura para el uso de mysql (MariaDB es compatible) y por ello se configura también el puerto por defecto de mysql: 3306.

Si se usa un sistema gestor de base de datos diferente habrá que cambiar la configuración:

SGBD	DB_CONNECTION	DB_PORT
MySQL/MariaDB	mysql	3306
Oracle	oracle	1521
PostgreSQL	pgsql	5432
SQL Server	sqlsrv	1433
…		

2.- Base de datos: parámetros de conexión

Tras crear la base de datos en phpMyAdmin se debe configurar en el archivo .env (es habitual que la base de datos se llame como el proyecto).

Para trabajar en local se puede usar tanto el usuario root como su contraseña (si no la tiene configurada se deja en blanco). Pero cuando se pasa a la aplicación a producción (se despliega para su uso por los clientes) no es conveniente esta configuración y se deberá usar un usuario y su contraseña de la base de datos del servidor público.

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=blog
DB_USERNAME=root
DB_PASSWORD=
```

2.- Base de datos: parámetros de conexión

Como bien se sabe las base de datos relacionales usan índices para las restricciones de integridad: campos únicos, claves primarias, claves ajenas…

Para que Laravel funcione correctamente con la longitud de los índices, si se está usando una versión de MySQL o MariaDB antigua se deberá añadir la siguiente configuración al método **boot** del archivo **app\Providers\AppServiceProvider.php** (además se deberá indicar el uso de la clase **Schema** en la parte superior del archivo

Solo para versiones:

MySQL menor que 5.7.7

MariaDB menor que 10.2.2

```php
use Illuminate\Support\Facades\Schema;
```

```php
public function boot()
{
    Schema::defaultStringLength(191);
}
```

3.- Archivos de la base de datos en Laravel

En el directorio **database** se encuentran los archivos que permiten gestionar la estructura de la base de datos.

- **migrations**: archivos de creación y modificación de la estructura de la base de datos.
- **factories**: fabricas de registros de pruebas para la base de datos.
- **seeders**: archivos que permiten inicializar con datos las tablas de la base de datos. Los datos pueden estar indicados de manera literal o bien usar archivos del directorio factories.

4.- Migraciones

Las **migraciones** permiten **crear** y **modificar tablas** así como sus **columnas**.

Mediante el uso de migraciones no será necesario el uso de phpMyAdmin para crear/modificar las tablas y sus columnas.

Además, las migraciones son como un **control de versiones** para la estructura de la base de datos y como control de versiones permiten revertir las modificaciones si fuera necesario.

Automáticamente Laravel añade una **marca de tiempo** en el nombre de archivo de la migración como herramienta para poder volver a versiones anteriores.

4.- Migraciones

Cada migración **solo actúa sobre una tabla** de la base de datos.

Se necesita una migración para cada acción de creación/modificación que se realice sobre una tabla.

Como se indicó anteriormente el idioma nativo de Laravel es inglés.

Si se usan nombres en inglés para controladores, vistas, modelos, migraciones… Laravel realizará muchas acciones de manera automática.

4.- Migraciones

Se puede seguir la siguiente estructura en los nombres de las migraciones:

Acción	Nombre de la migración
Crear una tabla	create_tableName_table
Añadir columna a una tabla	add_columna_to_tableName_table
Actualizar columna de una tabla	update_columnName_in_tableName_table
Eliminar columna de una tabla	remove_columnName_in_tableName_table
Eliminar una tabla	drop_tableName_table

Ejemplos:

create_products_table

update_price_to_products_table

4.- Migraciones

Laravel reconoce los siguientes patrones en los nombres de las migraciones.

```
const CREATE_PATTERNS = [
    '/^create_(\w+)_table$/',
    '/^create_(\w+)$/',
];
```

```
const CHANGE_PATTERNS = [
    '/_(to|from|in)_(\w+)_table$/',
    '/_(to|from|in)_(\w+)$/',
];
```

Esto significa que para Laravel los siguientes nombres significan lo mismo:

create_products_table

créate_products

La diferencia en los nombres es lo que utiliza Laravel para añadir el código mínimo a la migración para que esta funcione. Por conveniencia se usarán los patrones indicados en la página anterior.

4.- Migraciones

Como se ha indicado anteriormente, las migraciones se almacenan en la carpeta **database/migrations.**

Se puede observar que Laravel incluye varias migraciones por defecto:

- Para el sistema de autenticación para loguearse en la aplicación
- Para el reseteo de contraseñas.
- Para la tabla de tokens personales
- Para almacenar información sobre instrucciones fallidas sobre la base de datos.

Según la versión de Laravel la cantidad de migraciones por defecto y su contenido puede variar.

4.- Migraciones

Las migraciones son clases PHP que contienen dos métodos:

- **up**: instrucciones para agregar tablas, columnas y opciones sobre estas.
- **down**: instrucciones para revertir los cambios realizados en up.

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

...
return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('password_resets', function (Blueprint $table) {
            $table->string('email')->index();
            $table->string('token');
            $table->timestamp('created_at')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('password_resets');
    }
};
```

4.- Migraciones

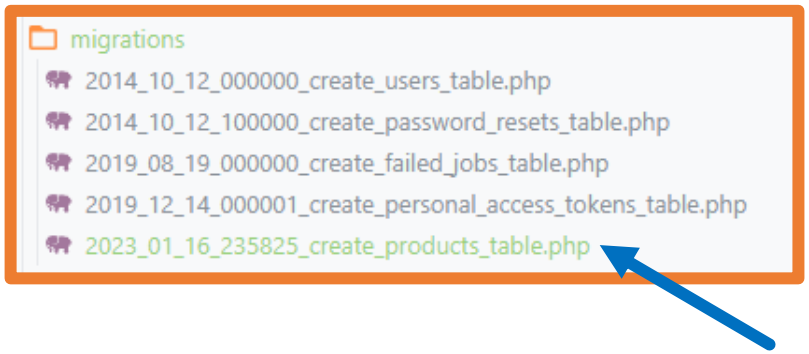
Para crear migraciones se usa el siguiente comando:

```
# php artisan make:migration nombreDeLaMigración
```

5.- Migraciones: creación de tablas

Migración que **crea** una tabla llamada **products**:

```
# php artisan make:migration create_products_table
```




```php
return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('products');
    }
};
```

5.- Migraciones: creación de tablas

Por defecto Laravel añade dos líneas de código en la creación de la tabla.

```php
public function up()
{
    Schema::create('products', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}
```

Esas líneas permiten añadir unos campos predefinidos a la tabla:

- **$table->id()**: creará un campo numérico con autoincremento que además será la clave primaria de la tabla.

- **$table->timestamps()**: creara los campos created_at y updated_at que Laravel gestionará automáticamente.

5.- Migraciones: creación de tablas

Al crear las tablas Laravel acepta un conjunto de métodos para crear los atributos y sus opciones:

```php
Schema::create('products', function (Blueprint $table) {
    $table->id();
    $table->string('nombre', 50);
    $table->text('descripcion');
    $table->double('precio', 8, 2);
    $table->boolean('disponible');
    $table->timestamp('verificado');
    $table->timestamps();
});
```

Se pueden consultar aquí todos los tipos de datos disponibles.

5.- Migraciones: creación de tablas

Existen modificadores que se pueden aplicar a los atributos:

```php
Schema::create('products', function (Blueprint $table) {
    $table->id();
    $table->string('nombre', 50);
    $table->text('descripcion')->nullable();
    $table->double('precio', 8, 2);
    $table->boolean('disponible');
    $table->timestamp('verificado')->nullable();
    $table->timestamps();
});
```

5.- Migraciones: creación de tablas

Algunos modificadores:

```
->primary()
->unique()
->nullable()
->autoincrement()
->first()
->after('nombreColumna')
->default('valor')
->unsigned()
```

Los modificadores se pueden anidar:

```
$table->string('clase')->nullable()->default('consola');
```

5.- Migraciones: creación de tablas

Algunas modificaciones sobre los atributos **se deben/pueden realizar después de crear las columnas** en una migración que modifique la tabla:

```
$table->primary('nombre');

$table->primary(['cliente', 'vehiculo']);

$table->unique('email');

$table->index('estado');

$table->foreign('cliente_id')->references('id')->on('clientes');
```

5.- Migraciones: creación de tablas

Si se usan las convenciones de Laravel a la hora de nombrar elementos, las claves ajenas se pueden definir así:

```
$table->foreign(customer_id')->references('id')->on('customers');
```

```
$table->foreignId('customer_id')->constrained();
```

5.- Migraciones: creación de tablas

También se puede indicar cómo se actuará al modificar/borrar un campo clave ajena:

```
$table->foreignId('user_id')
      ->constrained()
      ->onUpdate('cascade')
      ->onDelete('cascade');

$table->cascadeOnUpdate();
$table->restrictOnUpdate();
$table->cascadeOnDelete();
$table->restrictOnDelete();
$table->nullOnDelete();
```

6.- Tablas de relaciones N:N

Cuando se tiene una relación N:N (muchos a muchos) se tiene que crear una tabla para poder almacenar dicha información.

El nombre de la migración para crear la tabla seguirá el siguiente patrón:

create_Tabla1Singular_Tabla2Singular_table

En esa tabla habrá dos claves ajenas y a su vez esas dos columnas juntas serán la clave primaria.

Pero Laravel **no soporta claves primarias compuestas**.

6.- Tablas de relaciones N:N

Así, para crear la migración de una tabla de una relación N:N se deberán añadir las dos claves ajenas y a continuación indicar que esos dos campos forman un **campo compuesto único**.

Si se tienen dos tablas, clients y vehicles, donde un cliente puede tener muchos coches y un coche puede tener muchos clientes.

Siguiendo las convenciones de nombres de Laravel, el contenido de la migración podría ser como el siguiente código:

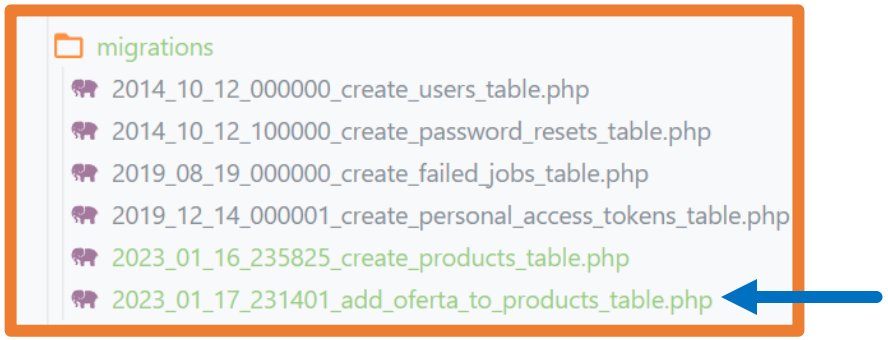
```php
public function up()
{
    Schema::create('client_vehicle', function (Blueprint $table) {
        $table->foreignId('client_id')->constrained();
        $table->foreignId('vehicle_id')->constrained();
        $table->unique(['client_id', 'vehicle_id'], 'claves_ajenas');
    });
}
```

Además, si se necesita se podrían añadir campos adicionales como por ejemplo los timestamps.

7.- Migraciones: modificación de tablas

Migración que **modifica** una tabla llamada **products**:

```
# php artisan make:migration add_oferta_to_products_table
```




```php
return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('products', function (Blueprint $table) {
            $table->string('name');
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('products', function (Blueprint $table) {
            $table->dropColumn('name');
        });
    }
};
```

7.- Migraciones: modificación de tablas

Para modificar columnas de una tabla Laravel necesita unos prerrequisitos.

En la documentación se puede encontrar todo lo necesario.

Si la base de datos aún no tiene datos puede ser más sencillo crear una migración para eliminar la columna y a continuación crear otra migración creando la nueva columna modificada.

8.- Todo sobre las migraciones

La documentación de Laravel es muy completa, por ello si se necesita más información lo aconsejable es acudir a ella:

https://laravel.com/docs/9.x/migrations

9.- Lanzar las migraciones

Para ejecutar las migraciones se utiliza el comando:

```
# php artisan migrate
```

Mediante este comando se ejecutarán en **lote** todas las migraciones pendientes (las creadas desde la última ejecución).

Con la ejecución del comando anterior se modificará la base de datos según lo definido en las migraciones.

9.- Lanzar las migraciones

Si se desea revertir el último lote de migraciones se utiliza el comando:

```
# php artisan migrate:rollback
```

Para revertir más lotes a la vez se utiliza la opción **step**:

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

9.- Lanzar las migraciones

Durante el **desarrollo** hay ocasiones en las que interesa partir de cero y lanzar todas las migraciones de golpe.

Para realizar esta acción se utiliza el siguiente comando:

```
# php artisan migrate:fresh
```

Este comando elimina también los datos por lo que **no debería usarse** cuando la aplicación está **en producción** (desplegada).

Práctica

Actividad 6:

Migraciones.