Práctica UD-6 Sesión-1

Objetivos	• Aprender a trabajar en Laravel con base de datos y crear un CRUD con **ORM Eloquent** y conectando a MySQL.
Instrucciones de entrega	• No hay que entregar nada.

- Abre una línea de comandos desde **VS Code** y ve a la ruta donde está el directorio ***htdocs*** de Xampp (**c:\xampp\htdocs**) después crea un nuevo proyecto con nombre ***PHPPracticaUD6Sesion1***:

```
C:\xampp\htdocs>laravel new PHPPracticaUD6Sesion1
```

- O como alternativa usando *composer*:

```
C:\xampp\htdocs>composer create-project --prefer-dist laravel/laravel
PHPPracticaUD6Sesion1
```

- Una vez creado carga el proyecto desde ***VS Code*** simplemente abriendo la carpeta.

Laravel y base de datos

- Documentación oficial:
  - https://laravel.com/docs/8.x/database

- Para trabajar con base de datos en Laravel tenemos tres opciones que pueden combinarse:
  - Sentencias planas SQL
  - Generador de consultas fluidas o ***fluent query builder*** en inglés
  - El framework de mapeo Eloquent ORM

- Además Laravel soporta por defecto las siguientes base de datos:
  - MySQL 5.6+
  - PostgreSQL 9.4+
  - SQLite 3.8.8+
  - SQL Server 2017+

Configuración

- El fichero de configuración para base de datos en Laravel está situado en ***config/database.php.*** Aquí se puede especificar qué proveedor de base de datos utiliza nuestra aplicación y por defecto es ***MySQL***.

- Los valores para la conexión están separados en un fichero de entorno (***.env***) y es este el que debemos editar normalmente en nuestros proyectos.

Ejecutar sentencias SQL

- Para ejecutar sentencias SQL desde nuestro código en Laravel hacemos uso de una clase especial de tipo ***facade*** con nombre ***DB*** donde encontramos los métodos *select*, *update*, *insert*, *delete*, and *statement*, por ejemplo:

```
use Illuminate\Support\Facades\DB;
...
$users = DB::select('select * from users');
foreach ($users as $user) {
    echo $user->name;
}
```

Query Builder

- Documentación oficial:
    - https://laravel.com/docs/8.x/queries

- Si queremos dar un paso más en la abstracción y no utilizar directamente sentencias SQL la siguiente opción es utilizar el generador de consultas o *query builder* que nos facilita Laravel.

- Para usarlo también se utiliza la clase fachada ***DB*** pero a través del método ***table***, veamos el siguiente ejemplo:

```
use Illuminate\Support\Facades\DB;
...
$users = DB::table('users')->get();
foreach ($users as $user) {
    echo $user->name;
}
```

- Si queremos hacer un *select* de algunas columnas solamente sería así:

```
use Illuminate\Support\Facades\DB;
...
$users = DB::table('users')
                ->select('name', 'email as user_email')
                ->get();
```

- Por último un *where* sería así:

```
use Illuminate\Support\Facades\DB;
...
$users = DB::table('users')
                  ->where('votes', '=', 100)
                  ->where('age', '>', 35)
                  ->get();
```

Migraciones

- Documentación oficial:
  - https://laravel.com/docs/8.x/migrations

- Las migraciones son como un control de versiones para la base de datos, lo que le permite definir y compartir la definición del esquema de la base de datos de la aplicación.

- Laravel facilita otra clase fachada (facade) llamada ***Schema*** que da soporte independiente de la base de datos para crear y manipular tablas.

- Los ficheros de migración están localizados en el directorio ***database/migrations*** y para crearlos podemos hacer uso del siguiente comando de *artisan*, siendo *<nombre tabla>* el nombre de la tabla en plural:
  - *php artisan make:migration create_<nombre tabla>_table*
  - *php artisan make:migration create_flights_table*

- Este comando nos creará un fichero con dos métodos principales:
  - *up()*: donde pondremos el código para crear o actualizar tablas.
  - *down()*: donde pondremos el código para borrar tablas.

- Para ejecutar las migraciones se hace también a través de un comando *artisan*:
  - *php artisan migrate*

Inicialización o Seeding

- Documentación oficial:
  - https://laravel.com/docs/8.x/seeding

- Laravel incluye la capacidad de inicializar (sembrar) o *seeding* en inglés la base de datos con datos de prueba.

- Los ficheros de migración están localizados en el directorio ***Database/Seeders*** y para crearlos podemos hacer uso del siguiente comando de *artisan*:
  - *php artisan make:seeder <NombreModelo>Seeder*
  - *php artisan make:seeder UserSeeder*

- Una vez creado el nuevo fichero se pueden añadir sentencias al método *run()*, por ejemplo:

```
public function run()
 {
     DB::table('users')->insert([
         'name' => Str::random(10),
         'email' => Str::random(10).'@gmail.com',
         'password' => Hash::make('password'),
     ]);
}
```

- Para ejecutar los ficheros de inicialización se hace a través del comando artisan:
  - *php artisan db:seed*
  - *php artisan db:seed --class=UserSeeder*

Ejercicio

- Esta práctica vamos a hacerla con *MySQL* por lo que arranca ***Xampp*** e inicia los servicios de *Apache* y *MySQL*, está basada en el artículo:
  - https://vegibit.com/how-to-use-the-laravel-query-builder/

- Crea una nueva base de datos con nombre ***phppracticaud6sesion1***

- Edita el fichero .evn y verifica que el valor de la variable DB_DATABASE es:
  - DB_DATABASE=phppracticaud6sesion1

- Crea una nueva migración para la tabla *games*:
  - *php artisan make:migration create_games_table*

- Añade el siguiente código al método up() de la clase *CreateGamesTable*:

```
public function up()
{
    Schema::create('games', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->string('publisher');
        $table->integer('releasedate');
        $table->timestamps();
    });
}
```

- Ejecuta las migraciones con el comando:
  - *php artisan migrate*

- Ahora crea una nueva clase de inicialización:
  - *php artisan make:seeder GameSeeder*

- Actualiza el contenido y pon el siguiente código:

```
public function run()
{
    //
    DB::table('games')->insert([
        'title' => 'Mega Man 2',
        'publisher' => 'Capcom',
        'releasedate' => '1989',
    ]);

    DB::table('games')->insert([
        'title' => 'Metroid',
        'publisher' => 'Nintendo',
        'releasedate' => '1986',
    ]);

    DB::table('games')->insert([
        'title' => 'Tecmo Bowl',
        'publisher' => 'Koei Tecno',
        'releasedate' => '1989',
    ]);
}
```

- Ejecuta la clase de inicialización:
    - *php artisan db:seed --class=GameSeeder*

- Abre el *PHPMyAdmin* y comprueba que se han creado las tablas y se ha insertado el contenido.

- Por último edita el fichero *web.php* y añade lo siguiente:

```
Route::get('games/dd', function () {
    $games = DB::table('games')->get();
    dd($games);
});
```

- Hemos utilizado la función auxiliar ***dd*** (die and dump) para mostrar el contenido de la variable con nombre *$games* que tiene los registros recuperados de la base de datos. Prueba el resultado en la URL:
    - http://127.0.0.1:8000/games/dd

- Como alternativa podemos simplemente devolver el contenido de los registros en formato JSON, para ello añade una nueva ruta:

```
Route::get('games/json', function () {
    $games = DB::table('games')->get();
    return $games;
});
```

- Prueba el resultado en la URL:
    - http://127.0.0.1:8000/games/json

- Por último podemos devolver una vista y ahí mostrar los registros, para ellos añade una nueva ruta:

```
Route::get('games/view', function () {
    $games = DB::table('games')->get();
    return view('games', ['games' => $games]);
});
```

- Crea la vista en el fichero ***resources/views/games.blade.php*** con el siguiente contenido:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Games</title>
</head>
<body>
<table border="1">
    <tr>        <th>Game</th>
        <th>Publisher</th>
        <th>Release Date</th>
    </tr>
    @for($i = 0; $i < count($games); $i++)
        <tr>
            <td>{{ $games[$i]->title }}</td>
            <td>{{ $games[$i]->publisher }}</td>
            <td>{{ $games[$i]->releasedate }}</td>
        </tr>
    @endfor
</table>
</body>
</html>
```

- Prueba el resultado en la URL:
    - http://127.0.0.1:8000/games/view

- A continuación añadamos una nueva ruta donde hagamos uso del *query build* con el método where:

```
Route::get('games/where', function () {
    $games = DB::table('games')->where('title','Metroid')->get();

    return view('games', ['games' => $games]);
});
```

- Prueba el resultado en la URL:
    - http://127.0.0.1:8000/games/where

- Por último añadamos dos nuevas rutas para poder mostrar una página de detalles de cada juego:

```php
Route::get('games', function () {
    $games = DB::table('games')->latest()->get();
    return view('games.index', ['games' => $games]);
});
Route::get('games/{id}', function ($id) {
    $game = DB::table('games')->find($id);
    return view('games.show', ['game' => $game]);
});
```

- Crea un nuevo directorio ***resources/views/games*** y añade un nuevo fichero ***index.blade.php***:

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Games</title>
</head>
<body>
<ul>
    @foreach($games as $game)
        <li><a href="/games/{{ $game->id }}">{{ $game->title }}</a></li>
    @endforeach
</ul>
</body>
</html>
```

- Añade otro nuevo fichero ***show.blade.php***:

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Game</title>
</head>
<body>
{{ $game->title }} is published by {{ $game->publisher }}
</body>
</html>
```

- Prueba el resultado en la URL:
    - http://127.0.0.1:8000/games

Laravel Eloquent ORM

- Documentación oficial:
  - https://laravel.com/docs/8.x/eloquent

- Laravel está empaquetado con ***Eloquent Object Relational Mapper*** (ORM), que proporciona una forma extremadamente fácil de comunicarse con una base de datos, esto ayuda a acelerar el desarrollo y proporciona una solución adecuada a la mayoría de los problemas que nos encontramos en las aplicaciones.

- Eloquent puede trabajar con múltiples bases de datos de manera eficiente utilizando una implementación de ***ActiveMethod***. Es un patrón arquitectónico donde el modelo creado en la estructura Modelo-Vista-Controlador (MVC) corresponde a una tabla en la base de datos. Las ventajas son:
  - Los modelos pueden realizar operaciones de bases de datos comunes sin codificar largas consultas SQL.
  - Los modelos permiten la consulta de datos en sus tablas, así como la inserción de nuevos registros en tablas.
  - Se simplifica el proceso de sincronización de múltiples bases de datos que se ejecutan en diferentes sistemas.
  - No es necesario escribir consultas SQL en absoluto. Todo lo que se tiene que hacer es definir las tablas de la base de datos y las relaciones entre ellas y Eloquent hará el resto del trabajo.

- Realiza el siguiente tutorial:
  - https://www.itsolutionstuff.com/post/laravel-8-crud-application-tutorial-for-beginnersexample.html

Añadir Bootstrap (opcional)

- Para añadir **bootstrap** a nuestro proyecto ejecute desde línea de comandos los siguientes comandos:
  - *composer require laravel/ui*
  - *php artisan ui bootstrap*
  - *npm install*
  - *npm run dev* (si es necesario otra vez npm run dev)

- Modifica el archivo de layout y añade:

```
<head>
    <title>Laravel 8 CRUD Application</title>
    <!-- Scripts -->
    <script src="{{ asset('js/app.js') }}" defer></script>
    <!-- Styles -->
    <link href="{{ asset('css/app.css') }}" rel="stylesheet">
</head>
```