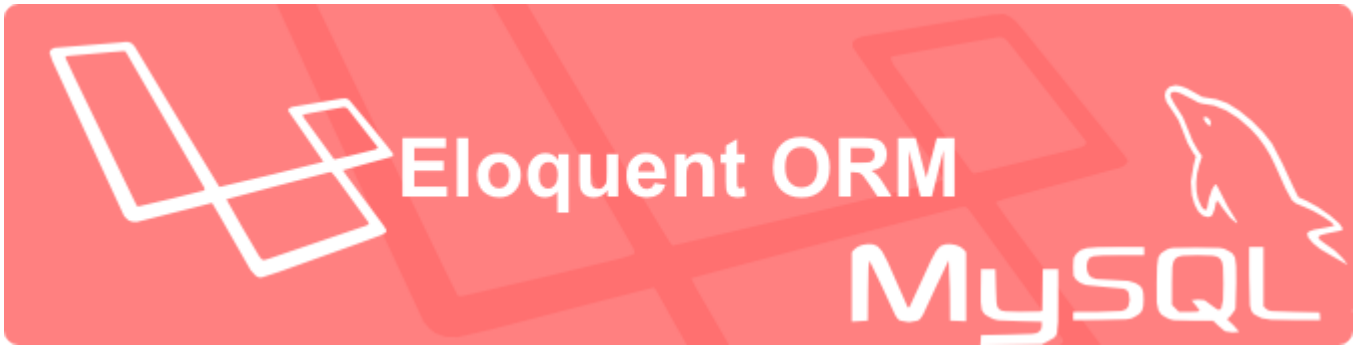


Eloquent es el ORM que incluye Laravel para manejar de una forma fácil y sencilla los procesos correspondientes al manejo de bases de datos en nuestro proyecto, gracias a las funciones que provee podremos realizar complejas consultas y peticiones de base de datos sin escribir una sola línea de código SQL.

Configurar Base de datos

El primer paso a iniciar es crear una base de datos, en caso de que aun no lo hayas hecho puedes seguir las siguientes líneas desde la consola de comando

```
$ mysql -u root -p
$ password:
mysql> CREATE DATABASE movies;
mysql> exit
```

Una vez creada se deben configurar los datos de conexión en el archivo config/database.php, yo prefiero configurar esta información “delicada” en el archivo .env de la aplicación

```
DB_HOST=localhost
DB_DATABASE=movies
DB_USERNAME=root
DB_PASSWORD=
```

Crear un nuevo modelo de Eloquent

Gracias a los comandos de artisan podemos crear los modelos desde la consola

```
$ php artisan make:model Movie -m
```

Por lo general el nombre del modelo debe escribirse con la primera letra mayúscula y en singular, laravel hará el resto por nosotros. Con el comando anterior se habrán creado dos archivos nuevos, uno llamado “Movie.php” dentro de la carpeta /app , el cual es el modelo y otro llamado *create_movies_table_000000.php* dentro de *database/migration*s que es la migración para dicha tabla.

Para mas información sobre migraciones visita el post Creando Migraciones en Laravel 5

Creando la migración

Vamos a definir los campos “name” y “description” en el archivo de migración creado anteriormente

(*create_movies_table.php*).

```
public function up()
    {
        Schema::create('movies', function(Blueprint $table)
        {
            $table->increments('id');
            $table->string('name');
            $table->string('description');
            $table->timestamps();
         });
     }
```

Atributos protected, fillable y guarded

Laravel permite asignar valores a cada uno de los atributos de la tabla de una manera muy sencilla con un array asociativo de elementos (que pueden ser los campos de un formulario), de esta forma se facilita el proceso de almacenaje desde un form. Sin embargo esto puede originar un problema de seguridad, ya que en ocasiones hay campos que no deberían enviarse a los formularios y tienen que permanecer protegidos a la vista de los usuarios finales.

Por defecto, todos los campos se tratan como protegidos y se debe indicar cuales pueden ser llenados de esta forma.

```
$fillable = [];
```

Dentro de este array se pueden especificar cuáles de los campos de la tabla pueden ser llenados con asignación masiva (que es el caso cuando enviamos un formulario creando un array asociativo para ser guardado).

```
$guarded = [];
```

Esta propiedad indica que los campos definidos allí no pueden ser llenados usando *Mass Assignment*, por lo cual nunca debería enviarse un Input::get() o cualquier otro tipo de datos o arreglo proveniente de un

controlador manipulable por un usuario.

Configuración del modelo

```php
<?php namespace App;

use Illuminate\Database\Eloquent\Model;

class Movie extends Model {
  protected $table = 'movies';
  protected $fillable = ['name', 'description'];
  protected $guarded = ['id'];
}
```

Configurando rutas de acceso

```php
//rutas para el recurso movie
Route::resource('movie','MovieController');
//una nueva ruta para eliminar registros con el metodo get
Route::get('movie/destroy/{id}', ['as' => 'movie/destroy',
'uses'=>'MovieController@destroy']);
//ruta para realizar busqueda de registros.
Route::post('movie/search', ['as' => 'movie/search',
'uses'=>'MovieController@search']);
```

Guardar un nuevo modelo con Eloquent

Cuando creamos un registro, instanciamos el modelo, asociamos los valores correspondientes a cada campo y procedemos a guardar, con Eloquent podemos hacer esto de dos formas diferentes.

Primero iniciemos creando un controlador para el recurso movie, llamado MovieController

```
$ php artisan make:controller MovieController
```

En la cabecera de este nuevo archivo importamos el modelo Movie con

```
use App\Movie as Movie;
```

La función store() será quien reciba por post la petición de guardado del nuevo registro, en este caso nos apoyaremos en la clase Illuminate\Http\Request para manejar las peticiones.

```
public function store(Request $request)
{
    $movie = new Movie;
    $movie->name = $request->name;
    $movie->description = $request->description;
    $movie->save();
    return redirect('movie');

}
```

De esta forma, instanciamos el modelo Movie, y asignamos uno a uno los valores a cada campo de la tabla, y luego guardamos usando el método save(). Hasta ahora todo bien, pero podemos hacer esto usando menos código de la siguiente manera.

```
public function store(Request $request)
{
    $movie = new Movie;
    $movie->create($request->all());
    return redirect('movie');

}
```

En este caso hacemos uso del método Create() y ya que en el formulario se han definido los campos usando los mismos nombres de la tabla movies, no habrá ningún problema al enviar el request como array asociativo y guardar el nuevo registro.

Para ver esto en funcionamiento debemos crear primero el formulario, creamos un archivo llamado

new.blade.php

Te recomiendo el tutorial sobre Integrando los componentes Html y Form a Laravel

```
@extends('app')
@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">
            {!! Form::open(['route' => 'movie.store', 'method' => 'post',
'novalidate']) !!}
                <div class="form-group">
                    {!! Form::label('full_name', 'Nombre') !!}
                    {!! Form::text('name', null, ['class' => 'form-control' ,
'required' => 'required']) !!}
                </div>
                <div class="form-group">
                    {!! Form::label('email', 'Descripci&oacute;n') !!}
                    {!! Form::text('description', null, ['class' => 'form-
control' , 'required' => 'required']) !!}
                </div>
                <div class="form-group">
                    {!! Form::submit('Enviar', ['class' => 'btn btn-success ' ]
 ) !!}
                </div>
            {!! Form::close() !!}
        </div>
    </div>
</div>
@endsection
```

Editamos la funcion create dentro de MovieController para acceder a la vista de este formulario

```
public function create()
```

```
{
    return \View::make('new');
}
```

Y por ultimo la lista a la que nos redirige el controlador luego de guardar el registro llamada

list.blade.php

```
@extends('app')
@section('content')
<div class="container">
    <div class="row">
        {!! Form::open(['route' => 'movie/search', 'method' => 'post',
'novalidate', 'class' => 'form-inline']) !!}
          <div class="form-group">
            <label for="exampleInputName2">Name</label>
            <input type="text" class="form-control" name = "name" >
          </div>
          <button type="submit" class="btn btn-default">Search</button>
        <a href="{{ route('movie.index') }}" class="btn btn-primary">All</a>
         <a href="{{ route('movie.create') }}" class="btn btn-primary">Create</a>
        {!! Form::close() !!}
        <br>
        <table class="table table-condensed table-striped table-bordered">
            <thead>
                <tr>
                  <th>Name</th>
                  <th>Description</th>
                  <th>Action</th>

                </tr>
            </thead>
            <tbody>
                @foreach($movies as $movie)
                <tr>
                    <td>{{ $movie->name }}</td>
```

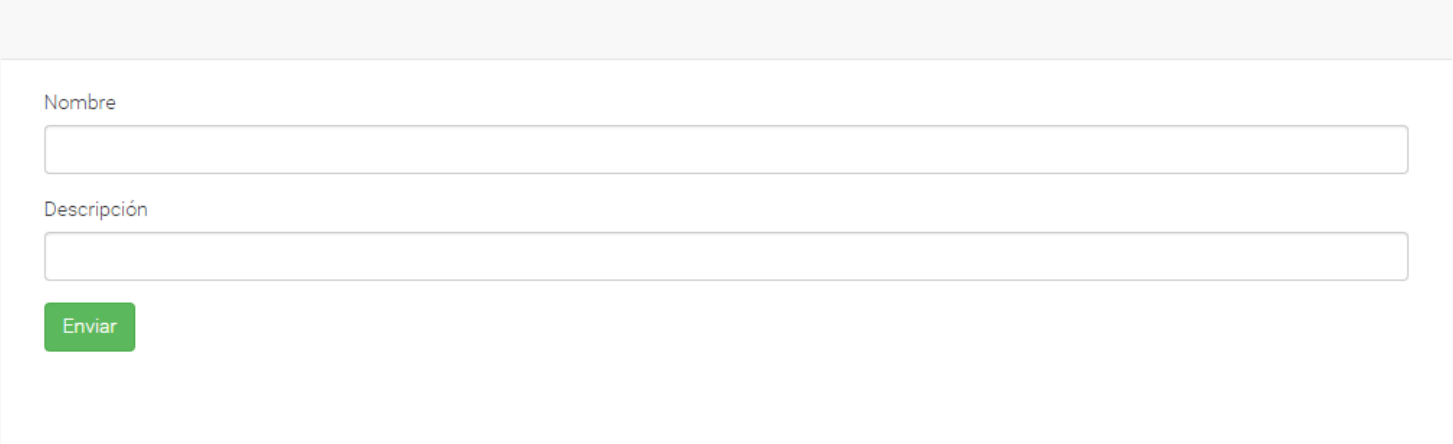
```
                    <td>{{ $movie->description }}</td>
                    <td>
                        <a class="btn btn-primary btn-xs" href="{{
route('movie.edit',['id' => $movie->id] )}}" >Edit</a>
                        <a class="btn btn-danger btn-xs" href="{{
route('movie/destroy',['id' => $movie->id] )}}" >Delete</a>
                    </td>

                </tr>
                @endforeach
            </tbody>
        </table>
    </div>
</div>
@endsection
```

Esta vista se muestra en el index del controlador

```
public function index(){
    $movies = Movie::all();
    return \View::make('list',compact('movies'));
}
```

Si ingresamos a *http:://rutadetuproyecto/movie* veremos algo similar a esto

para crear un nuevo registro hacemos click en Create y vamos al formulario



Actualizando registros con Eloquent

Creamos un nuevo formulario de actualización con algunos cambios llamado update.blade.php

```
@extends('app')

@section('content')
<div class="container">
    <div class="row">
        <div class="col-md-10 col-md-offset-1">
            {!! Form::model($movie,['route' => 'movie.update', 'method' => 'put',
 'novalidate']) !!}

                {!! Form::hidden('id', $movie->id) !!}

                <div class="form-group">
                      {!! Form::label('full_name', 'Nombre') !!}
                      {!! Form::text('name', null, ['class' => 'form-control' ,
'required' => 'required']) !!}
                  </div>

                  <div class="form-group">
                      {!! Form::label('email', 'Descripci&oacute;n') !!}
```

```
                    {!! Form::text('description', null, ['class' => 'form-
control' , 'required' => 'required']) !!}
                </div>

                <div class="form-group">
                    {!! Form::submit('Enviar', ['class' => 'btn btn-success ' ]
 ) !!}
                </div>
            {!! Form::close() !!}
        </div>
    </div>
</div>
@endsection
```

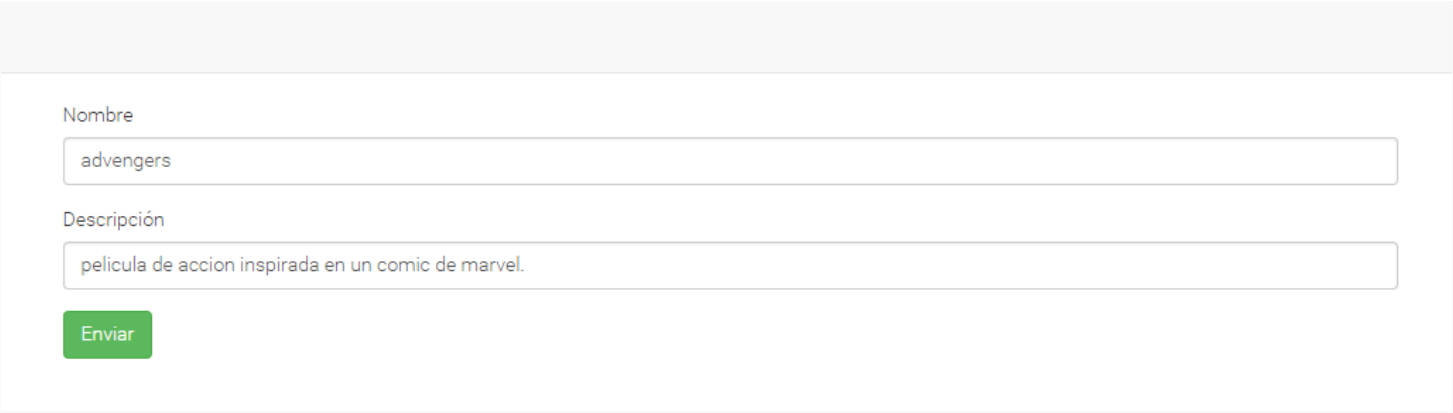
Para este formulario hemos cambiado el método a put y la ruta apunta a movie.update.

La función edit debe quedar de la siguiente manera

```
    public function edit($id)
    {
        $movie = Movie::find($id);
                return \View::make('update',compact('movie'));
    }
```

Esta función recibe como parámetro el id del registro a modificar, usando Eloquent buscamos el registro y lo pasamos al formulario para cargar los datos en pantalla.

Ahora podemos hacer click sobre el boton Edit de cualquier registro de la lista y veremos el siguiente formulario.



Para almacenar estos cambios debemos editar la función update de nuestro controlador.

```
public function update(Request $request)
    {
                $movie = Movie::find($request->id);
                $movie->name = $request->name;
                $movie->description = $request->description;
        $movie->save();
                return redirect('movie');
    }
```

Se busca el registro usando el método find() pasando como parámetro el id definido en el campo hidden del formulario, luego asignamos cada uno de los valores al modelo y por ultimo usamos nuevamente el método Save() para almacenar los cambios.

Querys básicos con Eloquent

En la lista de registro creamos un formulario de búsqueda, con el vamos a buscar las películas (movies) por el nombre usando el operador Like de Sql. Gracias a Eloquent no es necesario escribir toda una linea de Sql para crear la búsqueda, veamos como se usa.

Creamos una nueva función dentro de MovieController.php llamada search()

```
    public function search(Request $request){
```

```
        $movies = Movie::where('name','like','%'.$request->name.'%')->get();
        return \View::make('list', compact('movies'));

    }
```

el método where(), recibe como parámetros, el nombre de la columna en la base de datos a comparar, el operador y por ultimo el valor que deseamos buscar, por lo cual podemos hacer otros tipos de búsquedas como

```
//Busca el nombre exactamente igual al la variable
$movies = Movie::where('name','=',$request->name)->get();

//creados despues de una fecha dada
$movies = Movie::where('created_at','>',$date)->get();

//creados antes de una fecha dada
$movies = Movie::where('created_at','<',$date)->get();

//creados despues de una fecha dada y el nombre similar (like) a una variable
$movies = Movie::where('created_at','>',$date)
                ->where('name','=',$name)->get();

//similares a uno u otro nombre
$movies = Movie::where('name','=',$name_1)
                ->orWhere('name','=',$name_2)
                ->get();
```

Cuando usamos varios ->where() Eloquent los agrupa usando el operador AND de sql, si queremos trabajar con OR debemos usar ->orWhere()

Eliminando Registros con Eloquent

Para eliminar registros se hace uso del método delete() sobre un registro, editamos la función destroy() en el archivo MovieController.php

```php
public function destroy($id)
{
    $movie = Movie::find($id);
        $movie->delete();
        return redirect()->back();
    }
```

Nuevamente encontramos el registro usando find(), y lo eliminamos con delete()

Esto ha sido todo sobre el uso básico de Eloquent, veamos un resumen de lo aprendido.

- Crear modelos con php artisan make:model.
- Uso de los atributos fillable, public, guarded.
- Repaso sobre creación de migraciones.
- Crear nuevos registros en la base de datos.
- Buscar registros con Eloquent y el método find().
- Modificar Registros existentes.
- Realizar búsquedas (Query) con Eloquent.
- Uso de operadores con Eloquent.
- Eliminar registros con Eloquent.