

# FRAMEWORKS PHP

Manuel Alejandro Romero Miguel



# Manuel A. Romero Miguel

## Formación profesional

Ingeniero en informática.

## Experiencia profesional

En *Information Service Vision*: 3 años desarrollando un proyecto para el ministerio de Industria, Comercio y Turismo.

Colaboración en proyectos en la empresa Iternova y en el grupo GIGA de la universidad de Zaragoza (4 años)

Cursos de Formación en IFES, MisterChip, Cámara de Comercio en Burgos. (8000 horas de formación y coordinación de cursos)

Desde 2010, Coordinación, Impartición y gestión de prácticas del certificado de profesionalidad Desarrollo de Aplicaciones con tecnologías Web en el CPIFP Los Enlaces.

Tutor de Máster de educación en la universidad de Zaragoza

Desde 1998 - Actualidad Profesor de secundaria en la especialidad de Informática.



900 11 77 77



[nettdigitalschool.com](mailto:nettdigitalschool.com)



Búscame en las redes: @



# PROFESOR: una parte del proceso de aprendizaje

## Habilidades

Entusiasmo en mi trabajo

Formador de vocación, disfruto compartiendo mis conocimientos y me gusta buscar nuevos retos.

Lógica y abstracción de ideas. Optimización de algoritmos

## Contacto

[manuelromeromiguel@gmail.com](mailto:manuelromeromiguel@gmail.com)



# ÍNDICE DE CONTENIDOS

Controladores



# LARAVEL: BASES DE DATOS y ELOQUENT



Gestionar y Administrar Bases de datos  
*El framework se hace fuerte y muestra su  
potencial para facilitar estas típicas acciones y  
actuaciones con las bases de datos*

# 01 Los controladores

Qué son y para qué sirven

Ante una solicitud (request) enviamos una respuesta (response).  
Estas acciones las especificamos en el fichero de rutas



**En este caso sólo queremos retornar una página web, una vista.**

# 01 Los controladores

## Acciones entre el **request** y el **response**

Supongamos que queremos retornar un número aleatorio en la vista. Esto implica dos cosas:

- Tenemos que generar un número en el back
- Tenemos que visualizarlo en la vista

OPCIONES:

```
$numero = rand(1,100);  
Route::view("saludo", "saludo", ['numero'=>$numero]);
```

*Poco elegante declarar las variables que solo uso en una ruta*

```
Route::get("saludo", fn()=>{  
    $numero = rand(1,100);  
    return view("saludo", ['numero'=>$numero]);  
});
```

*Mejor crear una función anónima que ante la solicitud haga la acción*

***La mejor opción es crear un controlador que tenga métodos (uno o varios) que realice las acciones necesarias y retorne la vista.***



# 01 Los controladores

## Creando un controlador

Para crear un controlador usaremos mejor el comando en artisan (puedes revisar todos los comandos con **php artisan help**).

```
php artisan make:controller --help
```

Este es el comando para crear un controlador. Podemos ver las opciones con el método help.

```
→ layout git:(main) x php artisan make:controller -h
Description:
  Create a new controller class

Usage:
  make:controller [options] [--] <name>

Arguments:
  name                  The name of the controller

Options:
```



# 01 Los controladores

## Creando un controlador: opciones

El comando tiene una serie de opciones que iremos viendo

```
Options:
  --api                Exclude the create and edit methods from the controller
  --type=TYPE          Manually specify the controller stub file to use
  --force              Create the class even if the controller already exists
  -i, --invokable      Generate a single method, invokable controller class
  -m, --model[=MODEL] Generate a resource controller for the given model
  -p, --parent[=PARENT] Generate a nested resource controller class
  -r, --resource        Generate a resource controller class
  -R, --requests        Generate FormRequest classes for store and update
  -s, --singleton       Generate a singleton resource controller class
  --creatable          Indicate that a singleton resource should be creatable
  --test               Generate an accompanying PHPUnit test for the Controller
  --pest               Generate an accompanying Pest test for the Controller
  -h, --help            Display help for the given command. When no command is give
n display help for the list command
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
  --ansi|--no-ansi     Force (or disable --no-ansi) ANSI output
  -n, --no-interaction Do not ask any interactive question
  --env[=ENV]          The environment the command should run under
  -v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2
for more verbose output and 3 for debug
```

# 01 Los controladores

Creando un controlador: método `__invoke`

Si un controlador solo tiene una acción, la podemos implementar en el método mágico `__invoke`.

Este método es un método mágico en php que se ejecuta cuando intentamos llamar al objeto como si fuera una función

(<https://www.php.net/manual/en/language.oop5.magic.php#object.invoke>)

Podemos especificar en la construcción del controlador que cree el método

# 01 Los controladores

Creando un controlador: método `__invoke`

```
php artisan make:controller SaludoController -i
```

*Creamos un controlador*

```
Route::get("saludo",  
SaludoController::class);
```

*Añadimos la url en el fichero de rutas*

```
class SaludoController extends Controller  
{  
    //  
    public function invoke() {  
        return "<h1>Hola desde el controlador  
SaludoController</h1> ";  
    }  
}
```

*Implementamos el método `__invoke` en el controlador*

*Llamamos a la ruta en el navegador*

← → ↻ ⓘ localhost:8000/saludo

## Hola desde el controlador SaludoController

# 01 Los controladores

En las rutas recuerda incluir la clase del controlador

En las rutas, hay que incluir el namespace de la clase controlador que se va a usar o hacer referencia a ella de forma absoluta al especificarla

```
use App\Http\Controllers\SaludoController;  
//...  
Route::get("saludo", SaludoController::class);
```

```
Route::get("saludo", \App\Http\Controllers\SaludoController::class);
```

---

# 01 Los controladores

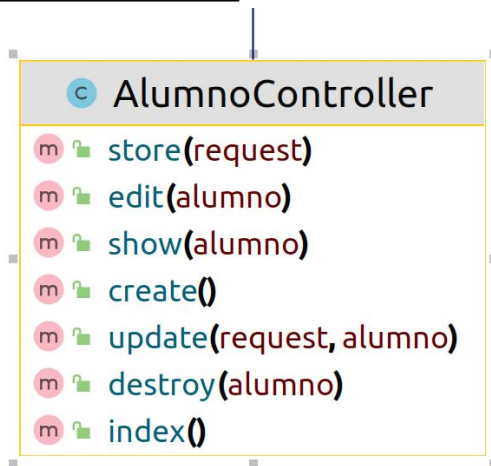
## Opciones resources

```
php artisan make:controller AlumnoController --resource
```

```
php artisan make:controller -r
```

Creo en el controlador los 7 métodos **REST** :

- **alumno** es un identificador de un alumno
- **request** es la solicitud donde vienen los datos de un formulario (datos de un nuevo alumno)



# 01 Los controladores

Las rutas a los métodos    Cada método es para una acción concreta

<b>index ()</b>	listar todos los alumnos
<b>delete(\$alumno)</b>	Eliminar o borrar un alumno concreto de la tabla correspondiente
<b>show (\$alumno)</b>	Mostrar un alumno
<b>create()</b>	Un formulario para rellenar los datos de un alumno
<b>store(\$request)</b>	Almacenar el alumno del formulario de create
<b>edit (\$alumno)</b>	Un formulario con los datos de una alumno que queremos modificar
<b>update(\$request, \$alumno)</b>	Guardar los datos de un alumno modificado en el formulario de edit

# 01 Los controladores

## Las rutas a los métodos

Para crear las entradas en el fichero de rutas, podríamos crear de una en una:

```
Route::get("alumnos", [AlumnosController::class, "index"]);
Route::get("alumnos/create", [AlumnosController::class, "create"]);
Route::post("alumnos", [AlumnosController::class, "store"]);
Route::get("alumnos/{alumno}", [AlumnosController::class, "show"]);
Route::get("alumnos/{alumno}/edit", [AlumnosController::class, "edit"]);
Route::put("alumnos/{alumno}", [AlumnosController::class, "update"]);
Route::delete("alumnos/{alumno}", [AlumnosController::class, "destroy"]);
```

En lugar de ello, también podemos usar el método **resource** de la facade Route

```
Route::resource("alumnos", AlumnosController::class);
```

---



# 01 Los controladores

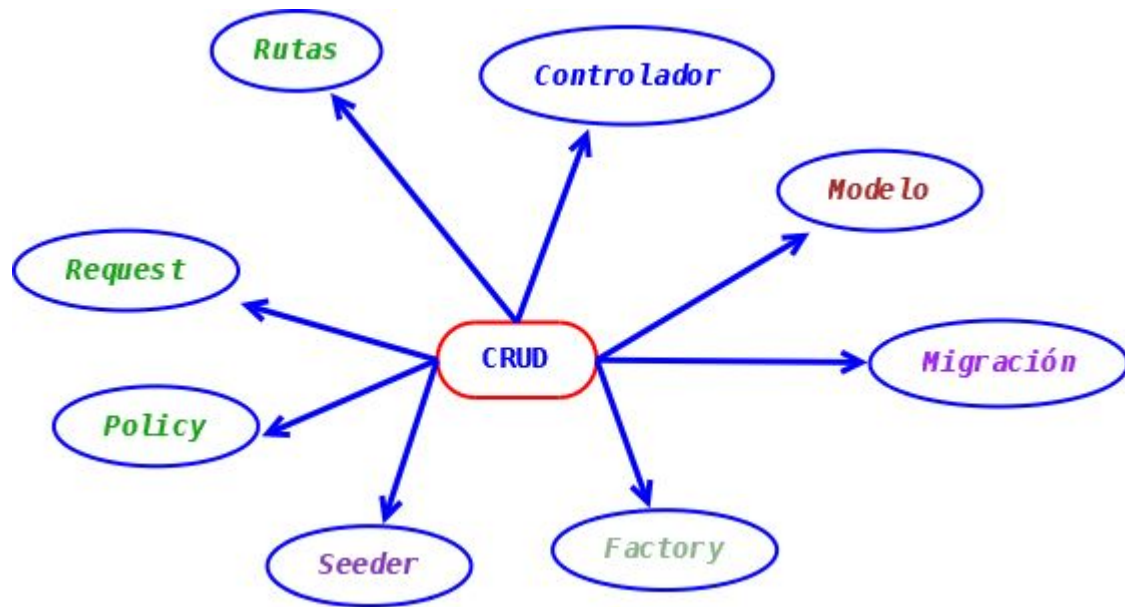
## Las rutas a los métodos

Ambos dos crean las rutas a los métodos del controlador de tipo resource

```
GET|HEAD      alumnos ..... alumnos.index > AlumnosController@index
POST          alumnos ..... alumnos.store > AlumnosController@store
GET|HEAD      alumnos/create ..... alumnos.create > AlumnosController@create
GET|HEAD      alumnos/{alumno} ..... alumnos.show > AlumnosController@show
PUT|PATCH    alumnos/{alumno} ..... alumnos.update > AlumnosController@update
DELETE        alumnos/{alumno} ..... alumnos.destroy > AlumnosController@destroy
GET|HEAD      alumnos/{alumno}/edit ..... alumnos.edit > AlumnosController@edit
```

# Creando un crud

Necesitamos crear un CRUD Create Read Update Delete



## 02 Creando un crud

Creamos un modelo con todo el ecosistema para gestionarlo

```
php artisan make:model Proyecto --all
```

```
C:\Users\Profesor\laravel\al\layout\estudios>php artisan make:model Proyecto --all
```

```
INFO Model [C:\Users\Profesor\laravel\al\layout\estudios\app\Models/Proyecto.php] created successfully.
```

```
INFO Factory [C:\Users\Profesor\laravel\al\layout\estudios\database/factories/ProyectoFactory.php] created successfully.
```

```
INFO Migration [C:\Users\Profesor\laravel\al\layout\estudios\database/migrations/2023_06_27_182920_create_proyectos_table.php] created successfully.
```

```
INFO Seeder [C:\Users\Profesor\laravel\al\layout\estudios\database/seeder/ProyectoSeeder.php] created successfully.
```

```
INFO Request [C:\Users\Profesor\laravel\al\layout\estudios\app\Http/Requests/StoreProyectoRequest.php] created successfully.
```

```
INFO Request [C:\Users\Profesor\laravel\al\layout\estudios\app\Http/Requests/UpdateProyectoRequest.php] created successfully.
```

```
INFO Controller [C:\Users\Profesor\laravel\al\layout\estudios\app\Http/Controllers/ProyectoController.php] created successfully.
```

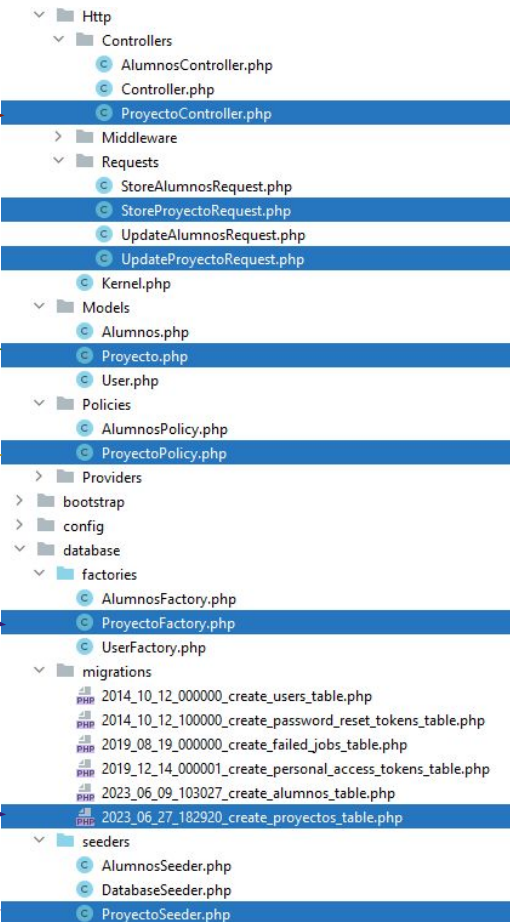
```
INFO Policy [C:\Users\Profesor\laravel\al\layout\estudios\app/Policies/ProyectoPolicy.php] created successfully.
```

## 02 Creando un crud

### Clases creadas:

- **Controlador**: las acciones de php
- **Request**: Validación y autorizaciones
- **Modelo**: interactuar con la base de datos
- **Policies**: Verificar accesos
- **Factories**: Crear valores para las filas
- **Migration**: Crear la tabla
- **Seeder**: insertar los datos creados en las tablas

Vamos a crear un modelo de **Alumno**  
`php artisan make:model Alumno --all`



## 03 Migraciones: Qué son

Son clases que se crean en nuestro proyecto y nos van a permitir:



**Control de versión de una  
base de datos**



**Crear y modificar tablas**



**Reconstruir el esquema  
de la base de datos**

---

Lo primero que tenemos que tener es **configurar la conexión** con la base de datos  
El nombre de la base de datos tiene que existir

*base de datos*

*.env*



# Migraciones crear una migración

Tenemos el comando explícito para ello (ver la ayuda para distintas opciones)

```
php artisan make:migration nombre_migracion --create=alumnos
```

Ya hemos visto que hay comandos que llevan implícita la construcción de una migración:

```
php artisan make:model -all
```

El nombre del fichero de la migración lleva **la fecha de construcción** y el nombre relacionado con la tabla que va a crear

```
→ auth git:(main) × php artisan make:migration alumnos --create alumnos
```

```
INFO Migration [database/migrations/2023_06_28_051813_alumnos.php] created successfully.
```

```
PHP 2023_06_28_051813_create_alumnos_table.php
```

*Es importante el orden de las migraciones (PK-FK)*



# Migraciones: método up - down

Por defecto crea la clase de la migración con dos métodos.

Si hemos especificado la opción **--create** los métodos ya tendrán código

```
public function up(): void
{
    Schema::create('alumnos', function (Blueprint $table) {
        $table->id();

        $table->timestamps();
    });

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

## Clases creadas:

- **up**: cuando ejecuta la migración
- **Schema** facade relacionada con **DDL**
- **id()**: Método que crea el campo **id** como clave y autoincrement (integer)
- **\$table->...()** Puedo especificar los campos de la tabla
- **timestamps()** Método que crea los campos **update\_up** y **create\_up** como marcas de tiempo. Permiten auditorías y control
- **down**: cuando ejecute que se deshaga la migración

# Migraciones: rollback-reset-refresh-fresh

Ejecutar la migración

```
php artisan migrate
```

Deshace la última migración (ejecutando métodos down de cada una de ellas)

```
php artisan migrate:roollback
```

Deshace las 3 últimas migracion (por ejemplo)

```
php artisan migrate:roollback --step=3
```

Deshace todo (ejecuta los métodos down de cada migración )

```
php artisan migrate:reset
```

Deshace todo y vuelve a ejecutar las migraciones (métodos down y up de cada migración)

```
php artisan migrate:refresh
```

Deshace todo y vuelve a ejecutar las migraciones y ejecuta la población de datos

```
php artisan migrate:refresh --seed
```

Borra todas las tablas de la base de datos y ejecuta las migraciones (solo métodos up)

```
php artisan migrate:fresh
```

---

# Migraciones: modificar una tabla sin perder los datos

Necesito crear una migración de modificación de la tabla  
Por ejemplo vamos a agregar el campo **teléfono** en la tabla **alumnos**

```
php artisan make:migration add_telefono --table:alumnos
```

```
public function up(): void
{
    Schema::table('alumnos', function (Blueprint $table) {
        $table->string("telefono")->default('11111');
    });
}

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::table('alumnos', function (Blueprint $table) {
        $table->dropColumn("telefono");
    });
}
```

Añadimos estas líneas

# Migraciones: modificar una tabla sin perder los datos

Otra opción es incluir la palabra **to\_nombre\_tabla** en el nombre de la migración

```
php artisan make:migration add_telefono_to_alumnos
```

*Laravel usando **ese patrón** del nombre de la migración interpreta que se quiere **modificar la tabla alumnos***

---

## Creando las factorías de datos

Los factory, son clases que nos van a permitir crear valores para los modelos. Para usar un factory, podemos llamar al método static `factory` del modelo. Esto ejecutará el método `definition` de la case factory asociada al modelo.

En este método retornaremos un valor para cada campo del registro que queramos aportar.

Podemos usar o bien el helper **fake()** o bien el atributo del factory **faker**. Ellos tendrán disponibles métodos para generar valores aleatorios

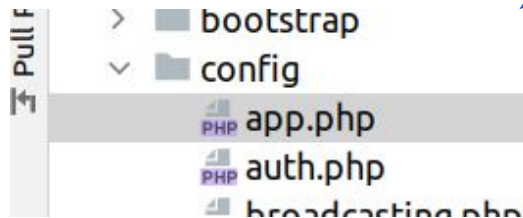
```
return [  
  'nombre'=>fake()->name(),  
  'apellido'=>$this->faker->firstName(),  
  'email'=>fake()->email(),  
  
];
```

---

## Creando las factorías de datos

Podemos establecer el idioma **en un parámetro del helper**, o bien establecer el idioma, debemos hacerlo en **el fichero** correspondiente de la carpeta **config**

```
'nombre' => fake('fr_FR')->name(),
```



Pull F

- > bootstrap
- ▼ config
  - PHP app.php
  - PHP auth.php
  - broadcasting.php

111

112

113

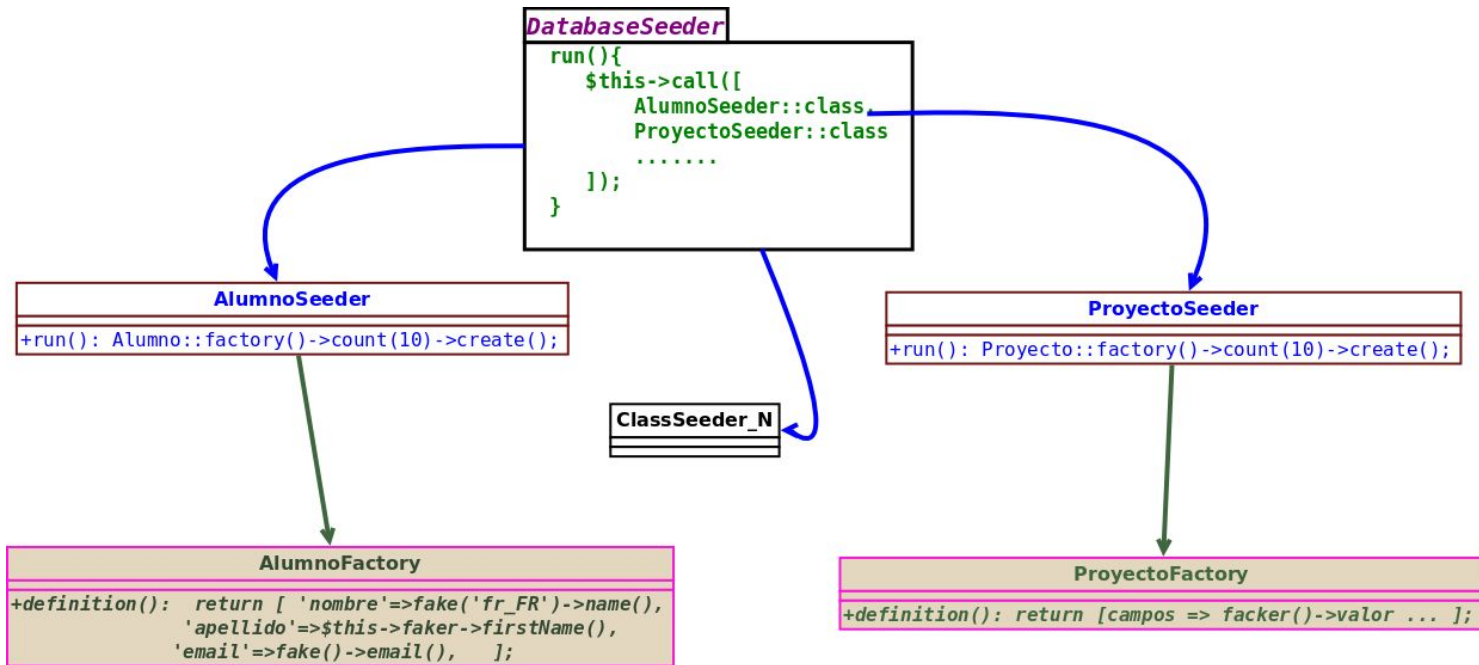


```
'faker_locale' => 'en_US',
```

# Creando los seeder y ejecutarlo

La clase seeder **DatabaseSeeder** es una clase central que se va a encargar de ejecutar los seeder de cada clase.

Un **seeder** es una clase que va a obtener los datos que creamos en **una factory** para por ejemplo insertarlos en la base de datos





## Creando los seeder y ejecutarlo

Una vez creado el seeder, se puede invocar con el siguiente comando

```
php artisan DB:seed
```

```
caravel git:(main) x cd layout
→ layout git:(main) x php artisan DB:seed

INFO Seeding database.

Database\Seeders\AlumnoSeeder ..... RUNNING
Database\Seeders\AlumnoSeeder ..... 101.94 ms DONE
```

## Creando los seeder y ejecutarlo

También lo podríamos ejecutar a la vez que creamos las tablas

```
php artisan migrate --seed
```

Es frecuente (**en desarrollo**) eliminar la base de datos y volver a crear todas las tablas realizando su población

```
php artisan migrate:fresh --seed
```

---

## Los modelos y ORM

ORM es un enfoque o una técnica que va a consistir en establecer una relación directa (**Mapeo**) entre la base de datos (**Relacion o tabla**) y la aplicación (**Objeto**).

The diagram consists of a large blue hand-drawn oval that encloses the text above. A line extends from the top of this oval to the title 'Los modelos y ORM'. Inside the oval, there are two blue arrows: one pointing from the word 'Relacion' to 'Objeto', and another pointing from 'Objeto' back to 'Relacion', indicating a bidirectional relationship.

Un modelo va a interactuar con una tabla de forma bastante transparente, si se respetan ciertos valores establecidos por defecto

---

# Los modelos: convenciones

Cuando creamos un modelo se crea una clase con el siguiente código

Por convención, Se asumen los siguientes valores

(si son otros, habría que especificarlos)

```
class Alumnos extends Model{  
    use HasFactory;  
}
```

```
protected $table="alumnos"; //Nombre del modelo en plural  
protected $primaryKey = 'id'; //La primary key es id  
protected $keyType = 'integer'; //El tipo de la clave es entero  
public $timestamps = true; //Existen los campos created_at y updated_at  
const CREATED_AT = 'created_at'; //Nombre del campo created_at  
const UPDATED_AT = 'updated_at'; //Nombre del campo updated_at  
  
public $incrementing = false; //La clave es autoincrement  
  
public $hidden=["created_at", 'updated_at'];  
  
protected $fillable= ['nombre', 'apellidos', 'dni', 'email'];  
protected $connection = "mysql"; //Si queremos una conexión diferente
```

# Los métodos en el controlador y recoger datos en la vista

<b>index ()</b>	listar todos los alumnos
<b>edit (\$alumno)</b>	Editar un alumno concreto
<b>show (\$alumno)</b>	Mostrar un alumno
<b>create()</b>	Un formulario para rellenar los datos de un alumno
<b>store(\$request)</b>	Almacenar el alumno del formulario de create
<b>edit (\$alumno)</b>	Un formulario con los datos de una alumno que queremos modificar
<b>update(\$request, \$alumno)</b>	Guardar los datos de un alumno modificado en el formulario de edit

# Los métodos index: listar todos los alumnos

```
$alumnos = Alumno::all(10); Controlador: AlumnosController@index  
return view("alumnos.listado", ["alumnos" => $alumnos]);
```

**Vista: alumnos/listado.blade.php**

```
<div class="overflow-x-auto" >  
<table class="table table-xs table-pin-rows table-pin-cols" >  
  <thead>  
    <tr>  
      <th>Nombre</th>  
      <th>Email</th>  
      <th>Edad</th>  
    </tr>  
  </thead>  
  <tbody>  
    @foreach($alumnos as $alumno)  
      <tr>  
        <td>{{ $alumno->nombre }}</td>  
        <td>{{ $alumno->edad }}</td>  
        <td>{{ $alumno->email }}</td>  
      </tr>  
    @endforeach  
  </tbody>  
</table>
```

# Los métodos Crear un nuevo alumno

Primero visualizamos el formulario para un nuevo alumno

```
<div class = "flex flex-col space-y-4" >
<a href="{{route("alumnos.create") }}" class="btn btn-primary">Nuevo alumno</a>
<hr>
<h1 class="text-green-700 text-4xl " >Listado de alumnos</h1>

<div class="overflow-x-auto" >

<table class="table table-xs table-pin-rows table-pin-cols" >
.....
```

*Vista: alumnos/listado.blade.php*

```
public function create() Controlador: AlumnosController@create
{
return view("alumnos.create");
//
}
```



*Vista: alumnos/create.blade.php*

```
<x-layout.app>
<div class=" flex flex-col justify-center items-center" >
  <h1 class=" text-4xl text-green-700" >Datos nuevo alumno</ h1>
  <form action="{{route('alumnos.store')}}" method="post" class="space-y-4">
    @csrf
    <div>
      <label for="">Nombre</label>
      <input name="nombre" type="text" placeholder="nombre"
        class="input input-bordered input-primary w-full max-w-xs" />
    </div>
    <div>
      <label for="">Edad</label>
      <input name="edad" type="text" placeholder="edad"
        class="input input-bordered input-primary w-full max-w-xs" />
    </div>
    <div>
      <label for="">Email</label>
      <input name="email" type="text" placeholder="email"
        class="input input-bordered input-primary w-full max-w-xs" />
    </div>
    <button type="submit" class=" btn btn-primary" >Guardar</button>
  </form>
</div>
</x-layout.app>
```

## Los métodos Crear un nuevo alumno

Hay que establecer en el modelo el conjunto de campos que **son rellenables en bloque o masa (masivamente)**, por ejemplo aportando un array.

Ésta es una **medida de seguridad** para evitar que se puedan aportar ciertos campos que no quiere que se puedan asignar para realizar actualizaciones o inserciones, por ejemplo el **id** o los campos de **timestamps (create\_at o update\_at)**.

```
class Alumno extends Model
{
  use HasFactory;
  protected $fillable=['nombre','edad','email'];
}
```

El método **authorize** de la clase **StoreAlumnoRequest**, debe de dar permiso para realizar la acción de store, e igualmente **UpdateAlumnoRequest**

```
public function authorize(): bool
{
    return true;
}
```

*Aquí debería de haber código que verifique que el usuario puede realizar esta acción, o true, como en este caso, para permitir a todos realizar esta acción. Lo analizaremos al ver los roles*

## Los métodos Crear un nuevo alumno: método store

En el método store del controlador recibimos en el **request** toda la solicitud, incluyendo el contenido de los inputs del formulario

En laravel puedo leer todos los inputs con el método **\$request->input()**

```
class Alumno extends Model
{
    use HasFactory;
    protected $fillable=['nombre','edad','email'];
}
```

El método **authorize** de la clase **StoreAlumnoRequest**, debe de dar permiso para realizar la acción de store, e igualmente **UpdateAlumnoRequest**

```
public function authorize(): bool
{
    return true;
}
```

*Aquí debería de haber código que verifique que el usuario puede realizar esta acción, o true, como en este caso, para permitir a todos realizar esta acción. Lo analizaremos al ver los roles*

---

# Los métodos modificar

Necesitamos generar la solicitud en la plantilla.

Dentro del **listado.blade.php**, para cada fila, añadimos *un botón de **editar** y de **borrar***.

Accedemos a **heroicons** (<https://heroicons.dev/>) para localizar el svg más idóneo, copiamos el svg correspondiente y lo pegamos en la fila.

La ruta para editar un registro es **alumnos/{id}/edit** o con nombre **alumno.edit** pasando el **id**

La solicitud es por **get**, por lo que lo podemos poner un un ancla

Edad	Teléfono	Opciones
44	998965914	 

```

<td>
<a href="{{route('alumnos.edit', $alumno->id)}}">
  <svg class="h-8 w-8 text-green-600 hover:text-green-900" . . . >
    . . .
  </svg>
</a>

```

# Los métodos modificar: el back

Modificar es un acto de dos acciones:

Por un lado nos entregarán un formulario con los datos a modificar

```
public function edit(Alumno $alumno){  
    //  
    return view ("alumnos.edit", compact ("alumno"));  
}
```

## Datos del alumno 1

Nombre

Julia Cruz

Edad

44

Email

bueno.miguel@nunez.es

GUARDAR

```
public function update(UpdateAlumnoRequest $request, Alumno $alumno)  
{  
    $datos_nuevos = $request->input();  
    $alumno->update($datos_nuevos);  
    return redirect (route("alumnos.index"));  
}
```

***El alumno que  
quiero modificar***

***Los nuevos datos modificados que vienen del formulario***

## 07 Los métodos Borrar

La ruta para borrar un registro es **alumnos/{id}** o con nombre **alumno.destroy** pasando el **id**  
La solicitud es por **delete**, por lo que lo podemos poner un formulario con un botón submit

```
<td>
  <form action="{{route('alumnos.destroy', $alumno->id)}}" method="POST">
    @csrf
    @method('DELETE')
    <button type="submit">
      <svg class="h-8 w-8 text-red-600 hover:text-red-900" . . . >.
    </button>
  </form>
</td>
```

Edad	Teléfono	Opciones
44	998965914	 

# Los métodos delete o destroy: el back

Modificar es un acto de dos acciones:

Por un lado nos entregarán un formulario con los datos a modificar

```
public function destroy(Alumno $alumno)
{
    $alumno->delete();
    return redirect (route("alumnos.index"));
}
```



***El alumno que  
quiero eliminar***

## 07 Relación 1:N

Agregamos una nueva relación : **Los idiomas que habla un alumno**



La tabla alumnos contendrá las tuplas del idioma que habla un alumno. Es una relación 1:N ya que no vamos a tener una tabla propia de los posibles idiomas. Generaremos una lista de idiomas, y a partir de ahí rellenaremos la tabla



## Relación 1:N

La tabla Alumno no se ve modificada

La tabla Idioma debe de tener una foreign key que identifique el alumno con el que se relaciona esa tupla.

Para hacerlo tenemos dos formas de especificar la **foreign key**

Alternativamente puedo especificar otro nombre para el campo de la clave foránea.

En este caso he de indicar el campo y tabla a la que hace referencia (**references (campo)**

**on(tabla)**

```
$table->foreign("alumno")
->references("id")
->on("alumnos")
->cascadeOnDelete();
```

```
$table->foreignId("alumno_id")
->constrained()
->cascadeOnDelete();
```

- He de poner el nombre de esta forma: ***nombreTabla\_id***
- La opción **cascadeOnDelete** es opcional y especifica que hago con esta tupla si elimino el alumno con que que está relacionada.

Opciones:

- `$table->cascadeOnUpdate();` (actualiza en cascada)
- `$table->restrictOnUpdate();` Updates should be restricted.
- `$table->noActionOnUpdate();` No actualiza
- `$table->cascadeOnDelete();` Borra en cascada
- `$table->restrictOnDelete();` Deletes should be restricted.
- `$table->nullOnDelete();` Deletes should set the foreign key value to null.

## 07 Tabla Idioma

```
public function up(): void
{
    Schema::create('idiomas', function (Blueprint $table) {
        $table->id();
        $table->foreignId("alumno_id")
            ->constrained()
            ->cascadeOnDelete();
        $table->string("idioma");
        $table->timestamps();
    });
}
```

---

## Tabla Idioma: poblarla

Poblar una tabla relacionada es muy sencillo y tendremos diferentes formas de hacerlo.

Una opción frecuente es la que vamos a utilizar aquí: **que cada vez que inserte un alumno, voy a insertar entre 0 y 4 idiomas para ese alumno.**

Para ello tenemos el método `each()` (<https://laravel.com/docs/11.x/collections#method-each>) de laravel

```
$listaAlumnos->each(function (Alumno $alumno) {  
    //Acciones a hacer  
});
```

La función `each` de las colecciones en Laravel permite iterar sobre cada elemento de una lista, en este caso de alumnos.

Durante la iteración, se aplica una función especificada como argumento a cada uno de estos elementos.

Esta función recibe el elemento actual de la colección sobre el que se está iterando

---

# Tabla Idioma: poblarla

Ahora el código. Tenemos un array de idiomas:

```
$idiomas = ["Francés", "Inglés", "Alemán", "Portugués", "Rumano",  
            "Italiano", "Chino mandarín", "Japonés", "Árabe", "Coreano" ];
```

Ahora para cada alumno que creamos, establecemos entre 0 y 4 idiomas que habla

```
Alumno::factory(50)->create()->each(function (Alumno $alumno) use ($idiomas) {  
    $n = rand(0, 4); //Cada alumno hablará entre 0 y 4 idiomas  
    $lista_idiomas_hablados = match ($n) { //obtenemos en un array los idiomas  
        0 => [],  
        1 => [$idiomas[array_rand($idiomas)]],  
        default => array_map(function ($index) use ($idiomas) {  
                                return $idiomas[$index];  
                            }, array_rand($idiomas, $n))  
    };  
  
    //Ahora creamos una fila en la tabla idiomas por cada idioma que hable esa alumna  
    foreach ($lista_idiomas_hablados as $idioma_hablado) {  
        $idioma = new Idioma();  
        $idioma->alumno_id = $alumno->id;  
        $idioma->idioma = $idioma_hablado;  
        $idioma->save();  
    }  
});
```

## Adaptando el modelo

Laravel es un entorno que facilita mucho el trabajo.

Ahora Modificaremos **el Modelo** para disponer de **idiomas** como si fuera **un atributo** multivaluado dentro de cada alumno. *Puede parecer como una colección de Mongo.*

Agregamos un método en el modelo **Alumno**

```
public function idiomas() {  
    return $this->hasMany(Idioma::class);  
}
```

Igualmente deberemos a partir de un idioma recuperar **el alumno al que pertenece esa tupla**. Para ello en el model **Idioma**

```
public function alumno() {  
    return $this->belongsTo(Alumno::class);  
}
```

---

# Usando los métodos del modelo

Con esto, cada vez que recuperemos un **alumno** tenemos acceso a los atributos:

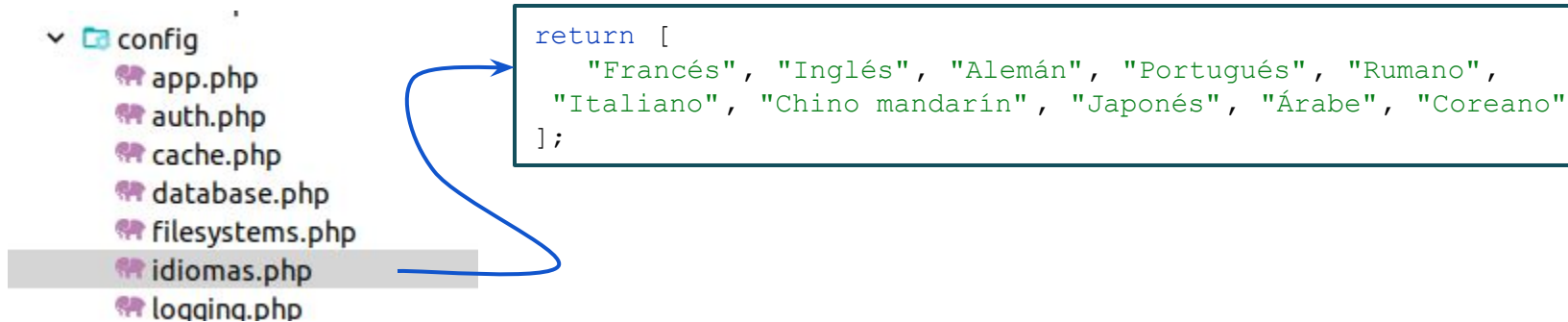
```
@foreach($alumnos as $alumno)
  <tr>
    <td>{{ $alumno->nombre }}</td>
    <td>{{ $alumno->edad }}</td>
    <td>{{ $alumno->email }}</td>
    <td>
      <select name="" id="">
        @foreach($alumno->idiomas as $idioma)
          <option >{{ $idioma->idioma }}</option>
        @endforeach
      </select></td>
    </tr>
  @endforeach
```

Diagrama: Una flecha azul curva conecta el código de la plantilla (específicamente la línea `@foreach($alumno->idiomas as $idioma)`) con la interfaz de usuario que muestra la lista de idiomas.

Nombre	Email	Edad	Idiomas
Mrs. Jakayla Hogan Sr.	54	douglas01@willms.com	<div>Idiomas</div> <div>           Francés            Inglés            Rumano            Chino mandarín            Coreano            ...         </div> <div>Rumano</div> <div>Inglés</div>
Ms. Christiana Fritsch DVM	63	eliane51@yahoo.com	
Ben Renner	21	xrohan@nitsche.com	
Emile Ortiz II	21	sandrinc.williamson@harvey.org	
Raymundo Little	47	bechtelar.yadira@gmail.com	
Hertha Zemlak MD	90	mcglynn.brenden@hessel.com	
Alberta Altenwerth III	54	jhamill@hemann.com	

# Cómo insertar alumnos con sus idiomas

Primero preparamos la plantilla blade y damos opción de visualizar todos los idiomas  
Para ello vamos a declarar el listado los idiomas en un fichero de configuración



***El controlador que retorna la vista para crear un nuevo registro***

```
public function create()  
{  
    $idiomas = config("idiomas");  
    return view ("alumnos.create", compact("idiomas"));  
    //  
}
```

# Cómo modificar los idiomas de un alumno

```
<form action="{{route('alumnos.store')}}" class="p-3 m-3" method="POST">
  @csrf
  Nombre <input type="text" name="nombre" id=""><br />
  Email <input type="text" name="email" id=""><br />
  Edad <input type="text" name="edad" id=""><br />
  @foreach($idiomas as $idioma)
    {{ $idioma }}
    <input type="checkbox" name="idiomas[]"
      value="{{ $idioma }}" />
    <br />
  @endforeach
  <input class="btn btn-secondary" type="submit" value="Guardar">
</form>
```

Nombre

Email

Edad

Francés	<input checked="" type="checkbox"/>
Inglés	<input checked="" type="checkbox"/>
Alemán	<input checked="" type="checkbox"/>
Portugués	<input type="checkbox"/>
Rumano	<input type="checkbox"/>
Italiano	<input checked="" type="checkbox"/>
Chino	<input type="checkbox"/>
mandarín	<input type="checkbox"/>
Japonés	<input checked="" type="checkbox"/>
Árabe	<input type="checkbox"/>
Coreano	<input type="checkbox"/>

Guardar



# Cómo modificar El método store guardando datos

```
public function store(StoreAlumnoRequest $request)
```

```
{  
    $datos = request()->input();  
    $alumno = new Alumno($datos);  
    $alumno->save();  
    dump ($alumno);  
    if (isset($datos['idiomas'])){  
        dump ($datos['idiomas']);  
        foreach ($datos['idiomas'] as $idioma_hablado){  
            $idioma = new Idioma();  
            $idioma->alumno_id= $alumno->id;  
            $idioma->idioma = $idioma_hablado;  
            $idioma->save();  
        }  
    }  
    //  
}
```

```
class Alumno extends Model  
{  
    use HasFactory;  
    no usages  
    protected $fillable=["nombre", "edad", "email"];
```

El controlador **edit**

```
public function edit(Alumno $alumno)
{
    $idiomas = config("idiomas");
    return view('alumnos.edit', compact('alumno', 'idiomas'));
    //
}
```

---

## 07 Cómo actualizar

### La vista

```
<form action="{{route('alumnos.update', $alumno->id)}}" class="p-3 m-3" method="POST"
    @method("PUT")
    @csrf
    Nombre <input type="text" value="{{ $alumno->nombre }}" name="nombre" id=""><br/>
    Email <input type="text" value="{{ $alumno->email }}" name="email" id=""><br/>
    Edad <input type="text" value="{{ $alumno->edad }}" name="edad" id=""><br/>
    <div class="rounded-2xl border-2 p-4 m-3 space-x-6 w-1/2">
        @foreach($idiomas as $idioma)
            <div class="flex justify-between">
                {{ $idioma }}
                <input type="checkbox" name="idiomas[]" value="{{ $idioma }}"
                    {{ $alumno->idiomas->contains('idioma', $idioma)? "checked":"" }}
                />
            </div>
        @endforeach
    </div>
    <input class="btn btn-secondary" type="submit" value="Guardar">
</form>
```

---

```
public function update(UpdateAlumnoRequest $request, Alumno $alumno)
{
    $datos = request()->input();
    $alumno->update($datos);
    $alumno->idiomas()->delete(); //borro todos los idiomas
    if (isset($datos['idiomas'])) { //ahora si tengo idiomas los agrego
        foreach ($datos['idiomas'] as $idioma_hablado) {
            $idioma = new Idioma();
            $idioma->alumno_id= $alumno->id;
            $idioma->idioma = $idioma_hablado;
            $idioma->save();
        }
    }
    return redirect (route('alumnos.index'));

    //
}
```

---

# Usando botón de confirmación al modificar

Vamos a usar una librería de js para ventanas emergentes pidiendo confirmación de acciones

Usamos la librería sweet alert de js

<https://sweetalert.js.org/guides/>

<https://sweetalert2.github.io/>

Podemos instalar el paquete desde npm o realizar una importación del cdn

Con el cdn:

Con npm :

**npm install sweetalert --save**

Ahora incluimos la librería en **app.js**

```
import swal from 'sweetalert';
```

Y tendremos que hacer que **vite** incluya el js, por lo que en nuestro layout incluiremos

```
@vite(["resources/css/app.css" , "resources/js/app.js" ])
```

---

# Usando botón de confirmación al modificar

Ahora insertamos la acción dentro de la vista que nos interese

Para borrar

*Aquí insertaríamos un svg para dar mejor experiencia de usuario*

```
<td>
  <form onsubmit=event.preventDefault() id="formulario{{ $alumno->id }}"
    action="{{ route('alumnos.destroy', $alumno->id) }}"
    method="POST">
    @csrf
    @method('DELETE')
    <button onclick="confirmarDelete('{{ $alumno->id }}')">Borrar</button>
  </form>
</td>
```

## 07 El script para borrar

```
<script>
  function confirmarDelete(id) {
    console.log("en borrar")
    swal({
      title: "~¿Confirmas el borrado?",
      text: "Esta acción no se podrá deshacer",
      icon: "warning",
      buttons: true
    }).then((ok)=> {
      if (ok) {
        console.log("Borrando .....");
        document.getElementById("formulario" + id).submit()
      }
    });
  }
</script>
```

