


## Eloquent Dossier

- [Práctica 01](#)
  - [Práctica 02](#)
  - [Práctica 03](#)
  - [Práctica 04](#)
  - [Práctica 05](#)
  - [Práctica 06](#)
  - [Práctica 07](#)
  - [Práctica 08](#)
  - [Práctica 09](#)
  - [Práctica 10](#)
  - [Práctica 11](#)
  - [Práctica 12](#)
  - [Práctica 13](#)
  - [Práctica 14](#)
  - [Práctica 15](#)
  - [Práctica 16](#)
  - [Práctica 17](#)
  - [Práctica 18](#)
  - [Práctica 19](#)
  - [Práctica 20](#)
  - [Práctica 21](#)
  - [Práctica 21.1](#)
  - [Práctica 22](#)
  - [Práctica 23](#)
- 

### Práctica 01

 Crear un elemento del modelo mediante php artisan para luego guardarlo en sqlite. En concreto crearemos: Alumno ( para esta prueba lo único que guardaremos será el nombre apellidos y edad )

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Alumno extends Model{
    use HasFactory;
    /**
     * @var string
     */
    private $name;
    /**
     * @var string
```

```


    */
    private $surname;
    /**
     * @var int
     */
    private $age;

    //constructor
    public function __construct() {
        $this->name = "";
        $this->surname = "";
        $this->age = 1;
    }

    //getters and setters
    public function getName(): string {
        return $this->name;
    }
    public function setName(string $name): void {
        $this->name = $name;
    }
    public function getSurname(): string {
        return $this->surname;
    }
    public function setSurname(string $surname): void {
        $this->surname = $surname;
    }
    public function getAge(): int {
        return $this->age;
    }
    public function setAge(int $age): void {
        $this->age = $age;
    }
}

```

## Práctica 02

 Realizar la migración para Alumno ( para esta prueba lo único que guardaremos será el nombre apellidos y edad ) Comprobar mediante el addon sqlite que se ha creado en la database

```

<?php

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

return new class extends Migration
{
    /**

```

```

    * Run the migrations.
    */
    public function up(): void
    {
        Schema::create('alumnos', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
            $table->string('name',30)->nullable(false);
            $table->string('surename');
            $table->integer('age');
        });
    }


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

```

- Captura:



## Práctica 03

 Haciendo uso de la documentación oficial y ejecutando las migraciones Crear una tabla Productos con string nombre y también un campo llamado precio que soporte precios con decimales. Así como un campo cantidad que represente la cantidad de producto de la que disponemos

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Productos extends Model
{
    public $timestamps = false;

    use HasFactory;
    /**
     * @var string
     */
    private $name;
    /**

```

```
* @var float
*/
private $price;
/**
 * @var integer
 */
private $quantity;

/**
 * Default constructor
 */
public function __construct() {
    $this->name = "";
    $this->price = 0.0;
    $this->quantity = 0;
}

/**
 * Get the product's name.
 *
 * @return string
 */
public function getName(): string
{
    return $this->name;
}

/**
 * Set the product's name.
 *
 * @param string $name
 */
public function setName(string $name): void
{
    $this->name = $name;
}

/**
 * Get the product's price.
 *
 * @return float
 */
public function getPrice(): float
{
    return $this->price;
}

/**
 * Set the product's price.
 *
 * @param float $price
 */
public function setPrice(float $price): void
{

```

```

        $this->price = $price;
    }

    /**
     * Get the product's quantity.
     *
     * @return integer
     */
    public function getQuantity(): int
    {
        return $this->quantity;
    }

    /**
     * Set the product's quantity.
     *
     * @param integer $quantity
     */
    public function setQuantity(int $quantity): void
    {
        $this->quantity = $quantity;
    }
}

```

- Migracion:

```

<?php

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

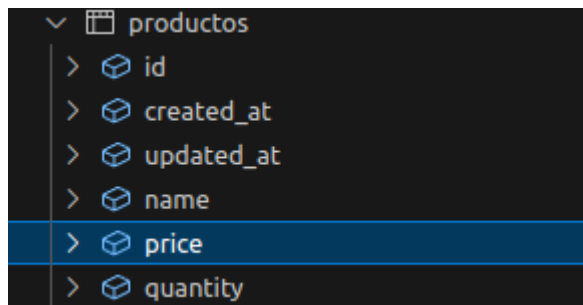
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('productos', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
            $table->string('name', 50)->nullable(false);
            $table->float('price', 10, 2)->nullable(false);
            $table->integer('quantity')->nullable(false);
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void

```

```
{
    Schema::dropIfExists('productos');
}
```

- Captura:



## Práctica 04

📁 Esta es una práctica de autoformación. Buscar como hacer uso de los: "seeder" y rellenar datos aleatorios en la tabla productos de la base de datos con ese sistema

- Version 1:

```
<?php

namespace Database\Seeders;

use Faker\Factory as Faker;
use Illuminate\Support\Facades\DB;

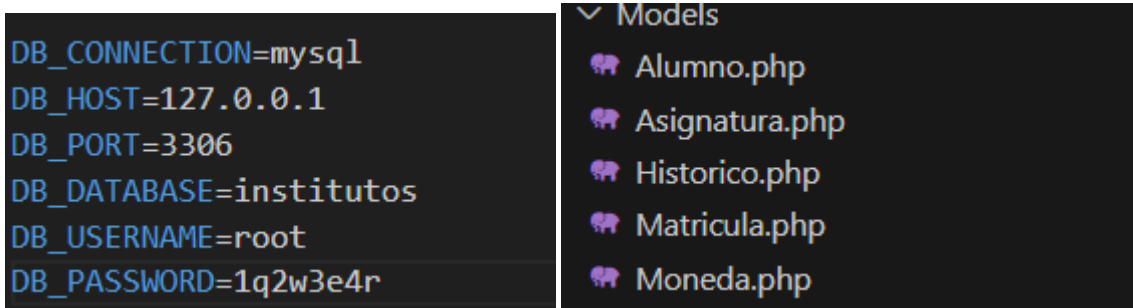
class ProductosSeeder extends DatabaseSeeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        $faker = Faker::create();

        DB::table('productos')->insert([
            'name' => $faker->word,
            'price' => $faker->randomFloat(2, 1, 20),
            'quantity' => $faker->numberBetween(1, 100),
        ]);
    }
}
```

## Práctica 05

📁 Hacer los cambios pertinentes en: .env para que la aplicación use la base de datos mysql con las tablas pertinentes para instituto matrículas. Crear con generate:model las clases correspondientes. Nota: observar que posiblemente haya que borrar las clase Alumno de prácticas anteriores

- Captura:



## Práctica 06

📁 Construir la ruta para las peticiones GET a: veralumno que nos lleve a un controlador Este controlador hará la búsqueda por id del alumno y lo enviará a una vista que mostrará la información

- web.php:

```
Route::get('/practice06/{dni}',[Practice06Controller::class, 'searchByDNI']);
```

- Controller:

```
public function searchByDNI($dni){
    DB::connection()->enableQueryLog();
    $student = Alumno::find($dni)

    return view('Practice06', compact('student'));
}
```

- Captura:

```
Info: {"dni":"12312312K","nombre":"María
Luisa","apellidos":"Gutiérrez","fechanacimiento":821234400000}
```

## Práctica 07

📁 Modificar la actividad anterior de tal forma que se sepa que está ejecutando en la base de datos como sentencia. Toma captura de pantalla del dd() obtenido

```
public function searchByDNI($dni){
    DB::connection()->enableQueryLog();
    $student = Alumno::find($dni);
    $lastQuery = DB::getQueryLog();
    dd($lastQuery);

    return view('Practice06', compact('student'));
}
```

- Captura:

```
array:1 [▼ // app\Http\Controllers\Practice06Controller.php:14
  0 => array:3 [▼
    "query" => "select * from `alumnos` where `alumnos`.`dni` = ? limit 1
"
    "bindings" => array:1 [▼
      0 => "12312312K"
    ]
    "time" => 1.65
  ]
]
```

## Práctica 08

📁 Listar todos los alumnos de nuestra base de datos (debe mostrarse en una vista blade)

- Routes:

```
Route::get('/practice08/findAll',[Practice08Controller::class, 'findAll']);
```

- Controller:

```
public function findAll(){
    $students = Alumno::all();
    return view('Practice08', compact('students'));
}
```

- Captura:



- {"dni":"12312312K","nombre":"María Luisa","apellidos":"Gutiérrez","fechanacimiento":821234400000}
- {"dni":"12345678Z","nombre":"Ana","apellidos":"Martín","fechanacimiento":968972400000}
- {"dni":"87654321X","nombre":"Marcos","apellidos":"Afonso Jiménez","fechanacimiento":874278000000}

## Práctica 09

📁 Buscar mediante where y mostrar en una vista las matrículas anteriores a 2021 obtener la consulta sql que ejecuta Eloquent en la base de datos

- Routes:

```
Route::get('/practice09',[Practice09Controller::class, 'findByDate']);
```

- Controller:

```
public function findByDate(){
    $data = Matricula::where('year', 2021)->get();
    return view('Practice09', compact('data'));
}
```

- Captura:

No hay datos disponibles.

## Práctica 10

📁 Continuando con la anterior, ordenaremos las matrículas por fecha y ejecutaremos un take(1) (seguir el ejemplo de encima: :where(..)->orderBy(...)->take(1) ) y finalmente terminaremos con un get() Mediante var\_dump() o dd() veremos que estructura nos devuelve. Hacer lo mismo de nuevo pero ahora en lugar de terminar con un get() terminamos con un first() ¿ da estructuras diferentes ? Nota: observar que la idea de first es tomar un objeto y get era el conjunto de objetos

- Routes:

```
Route::get('/practice10',[Practice09Controller::class, 'find']);
```

- Controller

```
public function find(){
    $dataGet = Matricula::where('year', 2021)
        ->orderBy('year','desc')
        ->take(1)
        ->get();

    $dataFirst = Matricula::where('year', 2021)
        ->orderBy('year','desc')
        ->take(1)
        ->first();

    dd($dataGet, $dataFirst);
}
```

- Captura:

```
Illuminate\Database\Eloquent\Collection {#1008 ▼ // app\Http\Controllers\Practice09Controller.php:26
  #items: []
  #escapeWhenCastingToString: false
}
```

```
null // app\Http\Controllers\Practice09Controller.php:26
```

## Práctica 11

📁 Obtener la cantidad total de asignaturas de 1ºDAM y mostrarlo

- Routes:

```
Route::get('/practice11',[Practice11Controller::class, 'find1DAM']);
```

- Controller

```
public function find1DAM(){
    $data = Asignatura::where('curso', '1º DAM')->get();
    return view('Practice11', compact('data'));
}
```

- Captura:

- {"id":1,"nombre":"BAE","curso":"1º DAM"}
- {"id":3,"nombre":"LND","curso":"1º DAM"}
- {"id":7,"nombre":"PRO","curso":"1º DAM"}

## Práctica 12

📁 Crear 2 nuevas asignaturas Siguiendo lo que se detallado: Una de 1ºDAM mediante new Asignatura() y save() y otra de 2ºDAM mediante: Asignatura::create() Se debe haber especificado mediante: fillable los campos que se aceptan para rellenar en la DB.

- Routes:

```
Route::get('/practice12',[Practice12Controller::class, 'create']);
```

- Controller:































```
public function create(){
    $asignatura = new Asignatura();
    $asignatura->nombre='ETS';
    $asignatura->curso='1º DAM';
    $asignatura->save();

    Asignatura::create([
        'nombre' => 'DAD',
        'curso' => '2º DAM',
    ]);

    $data1ro = Asignatura::where('nombre', 'ETS')->get();
    $data2do = Asignatura::where('nombre', 'DAD')->get();

    return view('practice12', compact('data1ro', 'data2do'));
}
```


- Captura:

				id	nombre	curso
<input type="checkbox"/>	 Edit	 Copy	 Delete	4	AED	2º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	BAE	1º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete	10	DAD	2º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete	6	DPL	2º DAW
<input type="checkbox"/>	 Edit	 Copy	 Delete	5	DSW	2º DAW
<input type="checkbox"/>	 Edit	 Copy	 Delete	9	ETS	1º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	LND	1º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete	8	PGL	2º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete	2	PGV	2º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete	7	PRO	1º DAM

Asignatura 1º: [{"id":9,"nombre":"ETS","curso":"1º DAM"}]

Asignatura 2º: [{"id":10,"nombre":"DAD","curso":"2º DAM"}]

Práctica 13

 Modificar las dos asignaturas anteriores de tal forma que la que era de primero pase a segundo y viceversa. Finalmente borrar de la base de datos la que sea de segundo

- Routes:

```
Route::get('/practice13',[Practice12Controller::class, 'modify']);
```

- Controller:

```
public function modify(){
    $asignatura2do = null;
    $asignatura2do = Asignatura::find(9);
    $asignatura2do->nombre='ETS modify';
    $asignatura2do->curso='2º DAM';

    $asignatura2do->save();

    $asignatura1ro = Asignatura::find(10);
    $asignatura1ro->nombre='DAD modify';
    $asignatura1ro->curso='1º DAM';

































    $asignatura1ro->save();

    $this->delete($asignatura2do);

    return view('practice13', compact('asignatura2do', 'asignatura1ro'));
}






























public function delete($asignatura2do){
    $asignatura2do->delete();
}
```

- Captura:

					id	nombre	curso
<input type="checkbox"/>	 Edit	 Copy	 Delete		4	AED	2º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		1	BAE	1º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		10	DAD modify	1º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		6	DPL	2º DAW
<input type="checkbox"/>	 Edit	 Copy	 Delete		5	DSW	2º DAW
<input type="checkbox"/>	 Edit	 Copy	 Delete		9	ETS modify	2º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		3	LND	1º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		8	PGL	2º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		2	PGV	2º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		7	PRO	1º DAM

Asignatura 1º: {"id":10,"nombre":"DAD modify","curso":"1º DAM"}


Asignatura 2º: {"id":9,"nombre":"ETS modify","curso":"2º DAM"}

					id	nombre	curso
<input type="checkbox"/>	 Edit	 Copy	 Delete		4	AED	2º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		1	BAE	1º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		10	DAD modify	1º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		6	DPL	2º DAW
<input type="checkbox"/>	 Edit	 Copy	 Delete		5	DSW	2º DAW
<input type="checkbox"/>	 Edit	 Copy	 Delete		3	LND	1º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		8	PGL	2º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		2	PGV	2º DAM
<input type="checkbox"/>	 Edit	 Copy	 Delete		7	PRO	1º DAM

Asignatura 1º: {"id":10,"nombre":"DAD modify","curso":"1º DAM"}

Asignatura 2º: null

## Práctica 14

 Basándose en el ejemplo anterior ( usando Historico::create ) crear un histórico para la moneda dólar que sea para la fecha actual al tipo de cambio actual con el euro

- Route:

```
Route::get('/practice14',[Practice14Controller::class, 'createHistoric' ]);
```

- Controller:

```
public function createHistoric(){  
    $moneda = new Moneda();
```

```

$moneda->pais = "Estados Unidos";
$moneda->nombre = "Dolar";
$moneda->save();

$newHistoric = new Historico();
$newHistoric->moneda_id = 1;
$newHistoric->fecha = '2024-11-06';
$newHistoric->equivalenteeuro = 0.92;
$newHistoric->save();

$historic = Historico::where('moneda_id', 1)->get();
return view('Practice14', compact('historic'));
}

```

- Captura:

	id	moneda_id	equivalenteeuro	fecha
	1	1	0.92	2024-11-06

Historico: [{"id":1,"moneda\_id":1,"equivalenteeuro":0.92,"fecha":"2024-11-06"}]

## Práctica 15

📁 Crear un histórico para la moneda dólar que sea para la fecha de mañana con tipo de cambio con el euro actual menos un céntimo, usando save() y associate()

- Routes:

```
Route::get('/practice15',[Practice15Controller::class, 'createHistoric15']);
```

- Controller:

```

public function createHistoric15(){
    $moneda = Moneda::find(1);

    $newHistoric = new Historico();
    $newHistoric->fecha = '2024-11-07';
    $newHistoric->equivalenteeuro = 0.92;
    $newHistoric->moneda()->associate($moneda);

    $newHistoric->save();

    $historic = Historico::where('id', 5)->get();
}

```



```
        return view('Practice15', compact('historic'));
    }
```

- Captura:

Historico: [{"id":5,"moneda\_id":1,"equivalenteeuro":0.92,"fecha":"2024-11-07"}]

## Práctica 16

📁 Crear un histórico para la moneda dólar que sea para la fecha de pasado mañana con tipo de cambio con el euro actual menos dos céntimos usando save() desde la entidad Moneda ( siguiendo el ejemplo que acabamos de ver )

- Routes:

```
Route::get('/practice16',[Practice16Controller::class, 'createHistoric']);
```

- Controller:

```
public function createHistoric(){
    $dolar = Moneda::find(1);
    $newDate = date('Y-m-d', strtotime('+2 days'));

    $historicDolar1 = Historico::where('id', 1)->take(1)
    ->first();

    $newValue = $historicDolar1->equivalenteeuro;
    $newValue -= 0.02;

    $newHistoric = new Historico();
    $newHistoric->fecha = $newDate;
    $newHistoric->equivalenteeuro = $newValue;

    $dolar->historicos()->save($newHistoric);

    $historic = Historico::where('id', 4)->get();

    return view('Practice16', compact('historic'));
}
```

- Captura:

Historico: [{"id":4,"moneda\_id":1,"equivalenteeuro":0.9,"fecha":"2024-11-10"}]

## Práctica 17

📁 Crear una moneda: dólar, país australia y guardarla con: create(). Mostrar el resultado obtenido haciendo una búsqueda por país: australia. Luego modificarla poniendo en mayúscula Australia usando save(). Toma captura de pantalla que muestre la salida en pantalla solicitada

- Routes:

```
Route::get('/practice17',[Practice17Controller::class, 'createCurrency']);
Route::get('/practice17-modify',[Practice17Controller::class, 'updateCurrency']);
```

- Controller:

```
public function createCurrency(){
    $currency = Moneda::create([
        'pais' => 'australia',
        'nombre' => 'Dolar'
    ]);

    return view('Practice17', compact('currency'));
}

public function updateCurrency(){
    $currency = Moneda::where('pais', 'australia')->first();

    $currency->pais = 'Australia';
    $currency->save();

    return view('Practice17', compact('currency'));
}
```

- Captura:

Currency: {"pais":"australia","nombre":"Dolar","id":2}

Currency: {"id":2,"pais":"Australia","nombre":"Dolar"}

## Práctica 18

📁 Crear entre asignatura y matrícula una relación N:M sin generar entity para la tabla intermedia. Tanto desde matrícula como desde asignatura. Luego mostrar por cada asignatura que personas ( su nombre y su dni ) se han matriculado

- Routes:

```
Route::get('/practice18',[Practice18Controller::class, 'index']);
```

- Controller:

```
public function index(){
    $asignaturas = Asignatura::all();

    foreach ($asignaturas as $asignatura) {
        echo "Asignatura: " . $asignatura->nombre . "\n";
        foreach ($asignatura->matriculas as $matricula) {
            echo json_encode( $matricula->alumno, JSON_UNESCAPED_UNICODE);
        }
    }
}
```

- Captura:

```

Tabnine | Edit | Test | Explain | Document | Ask
public function matriculas(){
    return $this->belongsToMany('App\Models\Asignatura',
                                'asignatura_matricula',
                                'idasignatura',
                                'idmatricula');
}

Tabnine | Edit | Test | Explain | Document | Ask
public function asignaturas()
{
    return $this->belongsToMany(
        'App\Models\Asignatura',
        'asignatura_matricula',
        'idmatricula',
        'asignatura'
    );
}

```

Asignatura: AED Asignatura: BAE Asignatura: DAD modify Asignatura: DPL Asignatura: DSW Asignatura: LND Asignatura: PGL  
 Asignatura: PGV {"dni":"12345678Z","nombre":"Ana","apellidos":"Martín","fechanacimiento":968972400000} Asignatura: PRO

## Práctica 19

📁 Crear un miniformulario donde se introduzca una moneda y un histórico. Utilizar una transacción de tal forma que si al guardar un histórico que no tiene un tipo de cambio numérico sino que el usuario ha introducido texto, se deshaga el guardado de los dos objetos

- Routes:

```

Route::get('/practice19',[Practice19Controller::class, 'index']);
Route::get('/practice19/create',[Practice19Controller::class, 'createMoneda']);

```

- Controller:

```

public function index(){
    return view('Practice19');
}
public function createMoneda(Request $request){
    DB::transaction(function () use ($request) : void{
        $moneda = new Moneda();
        $moneda->pais = $request->get('pais');
        $moneda->nombre = $request->get('nombre');
        $moneda->save();
    });
}

```

```

        $newHistoric = new Historico();
        $newHistoric->moneda()->associate($moneda);
        $newHistoric->fecha = $request->get('fecha');
        $newHistoric->equivalenteeuro = $request->get('equivalenteeuro');
        $newHistoric->save();

        if(!is_numeric($request->get('equivalenteeuro'))){
            DB::rollBack();
        }
    });
}

```

- View:

```

<form action="/practice19/create" method="POST">
    @csrf
    <label for="pais">País:</label>
    <input type="text" id="pais" name="pais" required>
    <br><br>

    <label for="nombre">Nombre de la moneda:</label>
    <input type="text" id="nombre" name="nombre" required>
    <br><br>

    <label for="fecha">Fecha:</label>
    <input type="date" id="fecha" name="fecha" required>
    <br><br>

    <label for="equivalenteeuro">Equivalente en Euro:</label>
    <input type="text" id="equivalenteeuro" name="equivalenteeuro" required>
    <br><br>


    <button type="submit">Guardar</button>
</form>

```

- Captura:

País:

Nombre de la moneda:


Fecha:  

Equivalente en Euro:

Guardar





















País:

Nombre de la moneda:


Fecha:  

Equivalente en Euro:

Guardar

				id	pais	nombre			
<input type="checkbox"/>		Edit		Copy		Delete	2	Australia	Dolar
<input type="checkbox"/>		Edit		Copy		Delete	4	españa	peseta
				id	moneda_id	equivalenteeuro	fecha		
<input type="checkbox"/>		Edit		Copy		Delete	1	1	0.92 2024-11-06
<input type="checkbox"/>		Edit		Copy		Delete	4	1	0.9 2024-11-10
<input type="checkbox"/>		Edit		Copy		Delete	5	1	0.92 2024-11-07
<input type="checkbox"/>		Edit		Copy		Delete	6	4	0.00601 2024-11-29

Práctica 20


 Modificar el Model: Alumno para que tenga soporte y nos convierta el número almacenado en base de datos a una fecha legible por ser humano. Formato: Año-mes-día hora:minutos

- Model:

```
protected $dates = ['fechanacimiento'];
public function setFechaAttribute($dateValue) : void {
    $this->attributes['fechanacimiento'] = (new Carbon($dateValue))->format('Y-m-d H:i');
}

public function getFechaAttribute($dateValue) : string {
    return (new Carbon($dateValue))->format('Y-m-d H:i');
}
```

## Práctica 21

 Busca para los alumnos que incluyan en su nombre: Ana las matrículas hechas después de 2020

- Routes:

```
Route::get('/practice21',[Practice21Controller::class, 'matriculas']);
```

- Controller:


```
public function matriculas(){
    $nombre = 'Ana';
    $matriculas = Matricula::whereHas(
        'alumno',
        function ($query) use ($nombre) {
            $query->where('nombre', $nombre)->where('year', '>', '2020');
        }
    );

    dd($matriculas->get());
}
```

- Captura:



## Práctica 21.1

 Crear los siguiente objetos usando: DB::table Alumno( "Elvira", "Lindo", "35792468Q", 821234400000) Una matrícula para ese alumno Elvira en el año 2024 Y para esa matrícula ponemos en la tabla intermedia entre Matricula y Asignatura le asignamos PRO y LND

- Routes:

```
Route::get('/practice211',[Practice21Controller::class, 'createAlumno']);
```

- Controller:

```
public function createAlumno() {
    DB::table('alumnos')->insert([
        'dni' => '35792468Q',
        'nombre' => 'Elvira',
        'apellidos' => 'Lindo',
        'fechanacimiento' => '821234400000'
    ]);

    DB::table('matriculas')->insert([
        'dni' => '35792468Q',
        'year' => '2024'
    ]);

    $matricula = DB::table('matriculas')->where('dni', '35792468Q')->first();







    DB::table('asignatura_matricula')->insert([
        'idasignatura' => 3,
        'idmatricula' => $matricula->id
    ]);











    DB::table('asignatura_matricula')->insert([
        'idasignatura' => 7,
        'idmatricula' => $matricula->id
    ]);
}
```

- Captura:



<div><div><div></div><div></div><div></div></div></div>				dni	nombre	apellidos	fechanacimiento
<div><div><div></div></div></div>	<div><div><div></div></div></div> Edit	<div><div><div></div></div></div> Copy	<div><div><div></div></div></div> Delete	12312312K	María Luisa	Gutiérrez	821234400000
<div><div><div></div></div></div>	<div><div><div></div></div></div> Edit	<div><div><div></div></div></div> Copy	<div><div><div></div></div></div> Delete	12345678Z	Ana	Martín	968972400000
<div><div><div></div></div></div>	<div><div><div></div></div></div> Edit	<div><div><div></div></div></div> Copy	<div><div><div></div></div></div> Delete	35792468Q	Elvira	Lindo	821234400000
<div><div><div></div></div></div>	<div><div><div></div></div></div> Edit	<div><div><div></div></div></div> Copy	<div><div><div></div></div></div> Delete	87654321X	Marcos	Afonso Jiménez	874278000000

<div><div>↔T↔</div><div>▼</div></div>					id	dni	year
<input type="checkbox"/>	 Edit	 Copy	 Delete	1	12345678Z	2023	
<input type="checkbox"/>	 Edit	 Copy	 Delete	3	35792468Q	2024	

					id	idmatricula	idasignatura		
<input type="checkbox"/>		Edit		Copy		Delete	1	1	2
<input type="checkbox"/>		Edit		Copy		Delete	2	3	3
<input type="checkbox"/>		Edit		Copy		Delete	3	3	7

Práctica 22

📁 Seguir los pasos para incorporar la tabla users en la base de datos de instituto y que se pueda usar la autenticación que nos da: breeze. Hacer una página que devuelva el listado de alumnos y que no se pueda acceder salvo que se haya hecho login/register

- Routes:

```
Route::get('/practice22',[Practice22Controller::class, 'showAlumnos'])->middleware('auth');
```

- Controller:

```
public function showAlumnos() {
    $alumnos = Alumno::all();
    return view('Practice22', compact('alumnos'));
}
```

- View:

```
@auth
  <div>
    <p>Info: {{json_encode($alumnos, JSON_UNESCAPED_UNICODE)}}</p>
  </div>
@endauth
```

Al añadir el middleware auth de breeze automaticamente no podemos acceder sin estar loggeados y efectivamente los devuelve al login. En caso de estar loggeados veremos la lista de alumnos.

- Captura:

```
Info: [{"dni":"12312312K","nombre":"Maria Luisa","apellidos":"Gutiérrez","fechanacimiento":821234400000},
{"dni":"12345678Z","nombre":"Ana","apellidos":"Martin","fechanacimiento":968972400000},{"dni":"35792468Q","nombre":"Elvira","apellidos":"Lindo","fechanacimiento":821234400000},
{"dni":"87654321X","nombre":"Marcos","apellidos":"Afonso Jiménez","fechanacimiento":874278000000}]
```

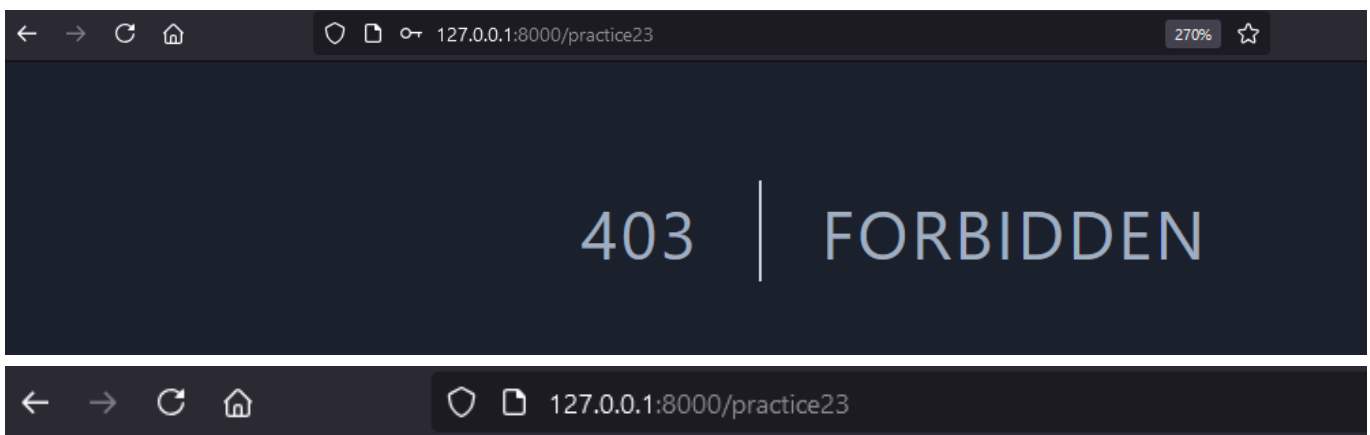
## Práctica 23

📁 Seguir los pasos para crear el middleware rolAdmin y crear una ruta que pase por los middleware auth y rolAdmin Comprobar que un usuario administrador si accede y el otro no

- Routes:

```
Route::get('/practice23', function () {
    dd('soy admin');
})->middleware('auth', 'rolAdmin');
```

- Captura:



```
"soy admin" // routes\web.php:50
```