

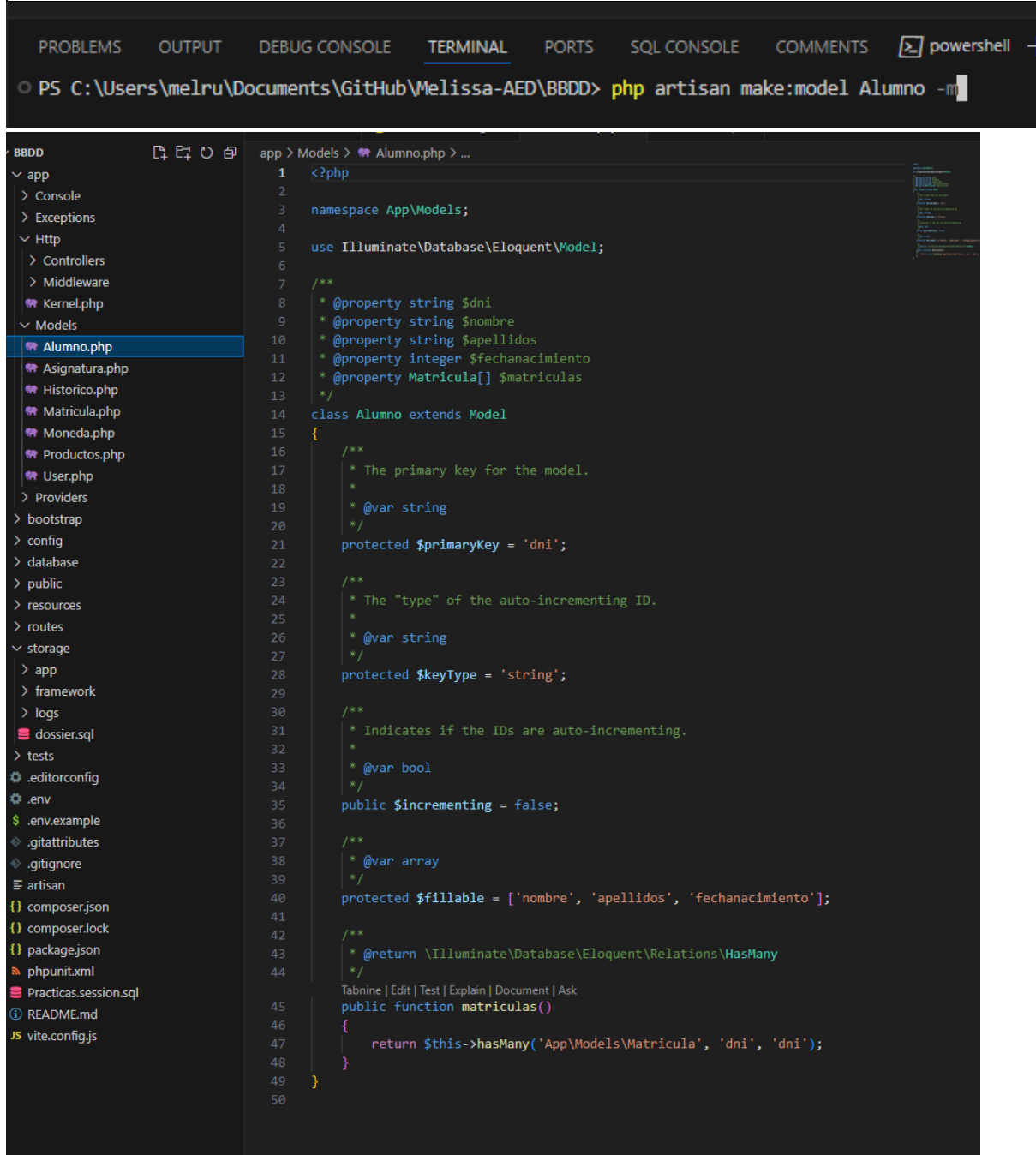
# Dossier Eloquent

# Indice

Práctica 1: .....	3
Práctica 2 .....	4
Práctica 3 .....	5
Práctica 4 .....	5
Práctica 5 .....	6
Práctica 6 .....	8
Práctica 7 .....	9
Práctica 8 .....	10
Práctica 16 .....	11
Practica 17 .....	12
Práctica 18 .....	13
Práctica 19 .....	14
Práctica 20 .....	17
Práctica 21 .....	18
Práctica 21.2 .....	18
Práctica 22 .....	19
Práctica 23 .....	20

## Práctica 1:

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)



The screenshot shows a code editor with a terminal window at the top and a file explorer on the left. The terminal window displays the command: `PS C:\Users\melru\Documents\GitHub\Melissa-AED\BBDD> php artisan make:model Alumno -m`. The file explorer on the left shows the project structure, with the `Models` directory expanded and `Alumno.php` selected. The main editor area shows the code for the `Alumno` model, which extends the `Model` class from `Illuminate\Database\Eloquent`. The code includes properties for `$dni`, `$nombre`, `$apellidos`, `$fechanacimiento`, and `$matriculas`. It also defines the primary key as `$dni` and the key type as `string`. The `fillable` array is set to `['nombre', 'apellidos', 'fechanacimiento']`. The `matriculas` relationship is defined as a `hasMany` relationship with the `Matricula` model.

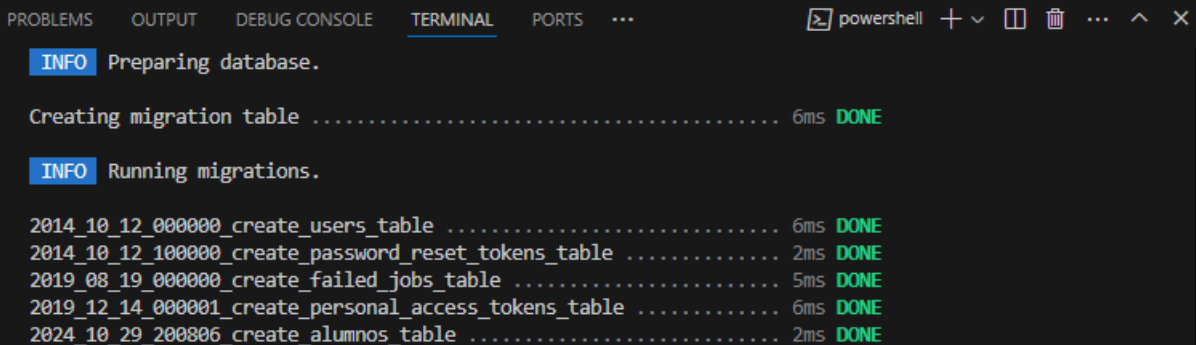
```
1 <?php
2
3 namespace App\Models;
4
5 use Illuminate\Database\Eloquent\Model;
6
7 /**
8  * @property string $dni
9  * @property string $nombre
10  * @property string $apellidos
11  * @property integer $fechanacimiento
12  * @property Matricula[] $matriculas
13  */
14 class Alumno extends Model
15 {
16     /**
17      * The primary key for the model.
18      *
19      * @var string
20      */
21     protected $primaryKey = 'dni';
22
23     /**
24      * The "type" of the auto-incrementing ID.
25      *
26      * @var string
27      */
28     protected $keyType = 'string';
29
30     /**
31      * Indicates if the IDs are auto-incrementing.
32      *
33      * @var bool
34      */
35     public $incrementing = false;
36
37     /**
38      * @var array
39      */
40     protected $fillable = ['nombre', 'apellidos', 'fechanacimiento'];
41
42     /**
43      * @return \Illuminate\Database\Eloquent\Relations\HasMany
44      */
45     public function matriculas()
46     {
47         return $this->hasMany('App\Models\Matricula', 'dni', 'dni');
48     }
49 }
50
```

## Práctica 2

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

```
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->rememberToken();
            $table->timestamps();
        });
    }

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



The screenshot shows a terminal window with a dark background. At the top, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is active), 'PORTS', and a menu icon. The terminal title bar indicates it's a 'powershell' window. The output shows an 'INFO' message 'Preparing database.' followed by a progress bar for 'Creating migration table' which is 6ms and 'DONE'. Another 'INFO' message 'Running migrations.' is shown, followed by a list of migration tasks with their durations and status: '2014\_10\_12\_000000\_create\_users\_table' (6ms, DONE), '2014\_10\_12\_100000\_create\_password\_reset\_tokens\_table' (2ms, DONE), '2019\_08\_19\_000000\_create\_failed\_jobs\_table' (5ms, DONE), '2019\_12\_14\_000001\_create\_personal\_access\_tokens\_table' (6ms, DONE), and '2024\_10\_29\_200806\_create\_alumnos\_table' (2ms, DONE).

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  ...
INFO Preparing database.
Creating migration table ..... 6ms DONE
INFO Running migrations.
2014_10_12_000000_create_users_table ..... 6ms DONE
2014_10_12_100000_create_password_reset_tokens_table ..... 2ms DONE
2019_08_19_000000_create_failed_jobs_table ..... 5ms DONE
2019_12_14_000001_create_personal_access_tokens_table ..... 6ms DONE
2024_10_29_200806_create_alumnos_table ..... 2ms DONE
```

## Práctica 3

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

```
return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('productos', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
            $table->string('nombre',100)->nullable(false);
            $table->float('precio');
            $table->integer('cantidad');
        });
    }

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

## Práctica 4

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

php artisan make:seeder ProductoSeeder

```
public function run(): void
{
    // Generar 10 productos de ejemplo
    for ($i = 0; $i < 10; $i++) {
        DB::table('productos')->insert([
            'nombre' => 'Producto ' . Str::random(5),
            'precio' => rand(10, 500), // Precio entre 10 y 500
            'cantidad' => rand(1, 100), // Cantidad entre 1 y 100
            'created_at' => now(),
        ]);
    }
}
```

```

        'updated_at' => now(),
    ]);
}
}

```

```

PS C:\Users\melru\Documents\GitHub\Melissa-AED\BBDD> php artisan db:seed

INFO Seeding database.

Database\Seeders\ProductoSeeder ..... RUNNING
Database\Seeders\ProductoSeeder ..... 30 ms DONE

PS C:\Users\melru\Documents\GitHub\Melissa-AED\BBDD> 

```

## Práctica 5

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

php artisan krlove:generate:model Asignatura --table-name=asignaturas --no-timestamps

php artisan krlove:generate:model Historico --table-name=historico --no-timestamps

php artisan krlove:generate:model Matricula --table-name=matriculas--no-timestamps

php artisan krlove:generate:model Moneda --table-name=monedas --no-timestamps

```

1  namespace App\Models;
2
3  use Illuminate\Database\Eloquent\Model;
4
5  /**
6   * @property integer $id
7   * @property string $nombre
8   * @property string $curso
9   * @property AsignaturaMatricula[] $asignaturaMatriculas
10  */
11
12  class Asignatura extends Model
13  {
14      /**
15       * @var array
16       */
17      protected $fillable = ['nombre', 'curso'];
18
19      /**
20       * @return \Illuminate\Database\Eloquent\Relations\HasMany
21       */
22      public function asignaturaMatriculas()
23      {
24          return $this->hasMany('App\Models\AsignaturaMatricula', 'idasignatura')
25      }
26  }
27
28

```



## Práctica 6

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

Inserto uno de ejemplo

```
INSERT INTO alumnos (nombre, apellidos, edad, created_at, updated_at)
```

```
VALUES ('Juan', 'Pérez García', 25, NOW(), NOW());
```

Creo la vista y la ruta al controlador

```
class AlumnoController extends Controller
{
    public function show($id)
    {
        // Buscar el alumno por su ID
        $alumno = Alumno::find($id);

        // Si el alumno no se encuentra, podemos redirigir o mostrar un error
        if (!$alumno) {
            return response()->json(['mensaje' => 'Alumno no encontrado'],
404);
        }

        // Pasar el objeto alumno a la vista
        return view('alumno', ['alumno' => $alumno]);
    }
}
```

Vista:

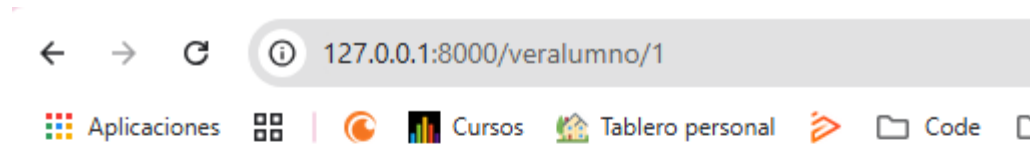
```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Información del Alumno</title>
</head>
<body>
    <h1>Información del Alumno</h1>

    <p><strong>Nombre:</strong> {{ $alumno->nombre }}</p>
    <p><strong>Apellidos:</strong> {{ $alumno->apellidos }}</p>
    <p><strong>Edad:</strong> {{ $alumno->edad }}</p>
</body>
</html>
```



Ruta:

```
Route::get('/veralumno/{id}', [AlumnoController::class, 'show']);
```



## Información del Alumno

**Nombre:** Juan

**Apellidos:** Pérez García

**Edad:** 25

### Práctica 7

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

```
array:1 [▼ // app\Http\Controllers\AlumnoController.php:23  
  0 => array:3 [▶]  
]
```

```
// Habilita el registro de consultas  
DB::connection()->enableQueryLog();  
  
// Realiza la consulta de búsqueda del alumno con el ID proporcionado  
$alumno = Alumno::find($id);  
  
// Obtiene la última consulta ejecutada  
$lastQuery = DB::getQueryLog();
```

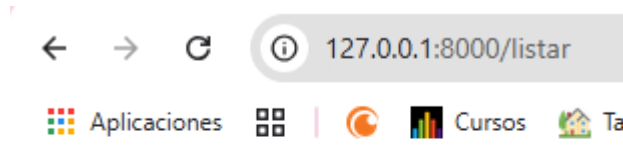
```
// Muestra la consulta en pantalla
dd($lastQuery);

// Retorna la vista con los datos del alumno
return view('alumno.show', ['alumno' => $alumno]);
```

- `DB::connection()->enableQueryLog()`; activa el registro de las consultas realizadas a la base de datos.
- `Alumno::find($id)`; ejecuta la consulta para encontrar al alumno con el ID proporcionado.
- `DB::getQueryLog()`; obtiene el log de todas las consultas realizadas desde que se habilitó el registro.
- `dd($lastQuery)`; muestra la consulta ejecutada en la base de datos.

## Práctica 8

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



Juan Pérez García - Edad: 25

Web.php

```
Route::get('/listar', [AlumnoController::class, 'listar']);
//controller
public function listar()
{
    // Recuperar todos los alumnos de la base de datos
    $alumnos = Alumno::all();

    // Iterar sobre la colección de alumnos y mostrar sus atributos
    foreach ($alumnos as $alumno) {
        // Mostrar el alumno en formato de texto
        echo '<br>' . $alumno->nombre . ' ' . $alumno->apellidos . ' -
Edad: ' . $alumno->edad;
```

```

        // Mostrar los atributos del alumno como un array (opcional)
        // var_dump($alumno->attributesToArray());
    }
}

```

## 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)

```

public function crearHistoricoDolar()
{
    // Obtener la moneda 'Dólar' de la base de datos
    $dolar = Moneda::where('nombre', 'Dólar')->first();

    // Verificar si se encontró la moneda 'Dólar'
    if ($dolar) {
        // Obtener el tipo de cambio actual del dólar a euro (simulamos
que es 1.10)
        $tipoCambioActual = 1.10; // Esto debe ser recuperado
dinámicamente si tienes esta información

        // Calcular el nuevo tipo de cambio restando dos céntimos
        $nuevoTipoCambio = $tipoCambioActual - 0.02;

        // Crear un nuevo objeto 'Historico' con la fecha de pasado mañana
y el nuevo tipo de cambio
        $historicoNuevo = new Historico();
        $historicoNuevo->fecha = Carbon::now()->addDays(2)-
>toDateString(); // Fecha de pasado mañana
        $historicoNuevo->equivalenteeuro = $nuevoTipoCambio; // Tipo de
cambio ajustado
        $historicoNuevo->moneda_id = $dolar->id; // Asociar con la moneda
dólar

        // Guardar el histórico asociado a la moneda dólar
        $dolar->historicos()->save($historicoNuevo);

        // Mostrar el histórico generado
        echo "Histórico generado: " . json_encode($historicoNuevo,
JSON_UNESCAPED_UNICODE);
    } else {
        echo "Moneda Dólar no encontrada.";
    }
}

```

← → ↻ 127.0.0.1:8000/crear-historico-dolar

Aplicaciones Cursos Tablero personal Code Estudio Personal Streaming Curro Gmail YouTube Maps Noticias Traducir GFNOW Cursos en línea

Histórico generado: {"fecha":"2024-11-14","equivalenteeuro":1.08,"moneda\_id":1,"updated\_at":"2024-11-12T12:25:13.000000Z","created\_at":"2024-11-12T12:25:13.000000Z","id":2}

## Practica 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

```
public function practica17()
{
    // 1. Crear la moneda con create()
    Moneda::create([
        'nombre' => 'dólar',
        'pais' => 'Australia',
    ]);

    // 2. Buscar la moneda por país
    $moneda = Moneda::where('pais', 'Australia')->first();

    // 3. Mostrar el resultado de la búsqueda
    echo "Moneda creada: " . json_encode($moneda, JSON_UNESCAPED_UNICODE)
    . "<br>";

    // 4. Modificar el campo 'pais' a mayúsculas
    $moneda->pais = 'AUSTRALIA';

    // 5. Guardar los cambios usando save()
    $moneda->save();

    // 6. Mostrar el resultado después de modificar
    echo "Moneda modificada: " . json_encode($moneda,
    JSON_UNESCAPED_UNICODE) . "<br>";
}
```

← → ↻ 127.0.0.1:8000/practica17

Aplicaciones Cursos Tablero personal Code Estudio Personal Streaming Curro Gmail YouTube Maps Noticias Traducir

Moneda creada: {"id":7,"pais":"Australia","nombre":"dólar","created\_at":"2024-11-12T12:27:28.000000Z","updated\_at":"2024-11-12T12:27:28.000000Z"}

Moneda modificada: {"id":7,"pais":"AUSTRALIA","nombre":"dólar","created\_at":"2024-11-12T12:27:28.000000Z","updated\_at":"2024-11-12T12:27:28.000000Z"}

## 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

```
class CreateAsignaturaMatriculaTable extends Migration
{
    public function up()
    {
        Schema::create('asignatura_matricula', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('idmatricula');
            $table->unsignedBigInteger('idasignatura');
            $table->timestamps();

            $table->foreign('idmatricula')->references('id')->on('matriculas')->onDelete('cascade');
            $table->foreign('idasignatura')->references('id')->on('asignaturas')->onDelete('cascade');
        });
    }

    public function down()
    {
        Schema::dropIfExists('asignatura_matricula');
    }
}
```

Se añade en Asignatura

```
public function alumnos()
{
    return $this->hasManyThrough(
        Alumno::class,
        Matricula::class,
        'idasignatura',
        'id',
        'id',
        'alumno_id'
    );
}
```

Creamos una nueva matricula y la asociamos con asignatura

```
public function crearMatriculaConAsignaturas()
{
    $alumno = Alumno::first();
    if (!$alumno) {
        return "No hay alumnos registrados.";
    }

    $nuevaMatricula = new Matricula();
    $nuevaMatricula->alumno()->associate($alumno);
    $nuevaMatricula->year = 2023;
    $nuevaMatricula->save();

    foreach (Asignatura::all()->take(2) as $asignatura) {
        $nuevaMatricula->asignaturas()->attach($asignatura);
    }

    $nuevaMatricula->refresh();
    return response()->json($nuevaMatricula, JSON_UNESCAPED_UNICODE);
}
```

## 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

Funcion

```
public function guardar(Request $request)
{
    $request->validate([
        'pais' => 'required|string|max:255',
        'nombre' => 'required|string|max:255',
        'equivalenteeuro' => 'required|numeric',
        'fecha' => 'required|date',
    ]);

    try {
        DB::transaction(function () use ($request) {
            // Guardar la moneda
            $moneda = Moneda::create([
                'pais' => $request->pais,
                'nombre' => $request->nombre,
            ]);
        });
    } catch (Exception $e) {
        // Si falla la transacción, se deshace el guardado de los dos objetos
    }
}
```

```

        // Validar si el equivalente es numérico antes de guardar
        if (!is_numeric($request->equivalenteeuro)) {
            throw new \Exception('El equivalente al euro debe ser un
número');
        }

        // Guardar el histórico
        Historico::create([
            'moneda_id' => $moneda->id,
            'equivalenteeuro' => $request->equivalenteeuro,
            'fecha' => $request->fecha,
        ]);
    });

    return redirect()->back()->with('success', 'Moneda e histórico
guardados correctamente.');
```

```

    } catch (\Exception $e) {
        return redirect()->back()->withErrors(['error' => $e-
>getMessage()]);
    }
}

```

#### Formulario:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Formulario Moneda e Histórico</title>
</head>
<body>
    <h1>Guardar Moneda e Histórico</h1>

    @if(session('success'))
        <p style="color: green;">{{ session('success') }}</p>
    @endif

    @if($errors->any())
        <ul style="color: red;">
            @foreach($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    @endif

```

```

<form action="{{ route('moneda.guardar') }}" 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="equivalenteeuro">Equivalente al Euro:</label>
    <input type="text" id="equivalenteeuro" name="equivalenteeuro"
required>
    <br><br>

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

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

Route::get('/moneda', function () {
    return view('formulario');
})->name('moneda.form');

Route::post('/moneda/guardar', [AlumnoController::class, 'guardar'])->
>name('moneda.guardar');
Route::get('/moneda', function () {
    return view('formulario');
})->name('moneda.form');

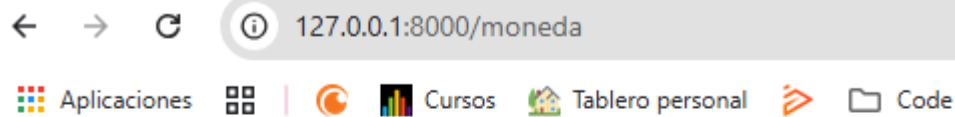
Route::post('/moneda/guardar', [AlumnoController::class, 'guardar'])->
>name('moneda.guardar');
Route::get('/moneda', function () {
    return view('formulario');
})->name('moneda.form');

Route::post('/moneda/guardar', [AlumnoController::class, 'guardar'])->
>name('moneda.guardar');
Route::get('/moneda', function () {
    return view('formulario');
})->name('moneda.form');

```



```
Route::post('/moneda/guardar', [AlumnoController::class, 'guardar'])->name('moneda.guardar')
```



## Guardar Moneda e Histórico

País:

Nombre de la moneda:

Equivalente al Euro:

Fecha:  

### 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

## Práctica 21

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

Se podría hacer con select en consultas nativas o se puede hacer con eloquent directamente, es mejor con eloquent para evitar injections, dicho por la propia documentacion de laravel

```
public function buscarMatriculas()
{
    // Buscar matrículas para alumnos que incluyen "Ana" en su nombre y
    // hechas después de 2020
    $resultados = DB::table('matriculas')
        ->join('alumnos', 'matriculas.dni', '=', 'alumnos.dni')
        ->where('alumnos.nombre', 'LIKE', '%Ana%') // Buscamos "Ana" en el
        nombre
        ->where('matriculas.year', '>', 2020) // Año mayor que 2020
        ->select('matriculas.*', 'alumnos.nombre as alumno_nombre',
        'alumnos.dni as alumno_dni')
        ->get();

    // Mostrar los resultados (por ejemplo, en formato JSON)
    return response()->json($resultados);
}
```

## Práctica 21.2

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

```
public function crearObjetos()
{
    DB::transaction(function () {
        // Crear el alumno Elvira
        $alumnoId = DB::table('alumnos')->insertGetId([
            'nombre' => 'Elvira',
            'apellidos' => 'Lindo',
            'dni' => '35792468Q',
            'created_at' => now(),
            'updated_at' => now(),
            'fecha_nacimiento' => date('Y-m-d H:i:s', 821234400000 /
            1000), // Convertir el timestamp de milisegundos a segundos
        ]);
    });
}
```

```

]);

// Crear la matrícula para Elvira en el año 2024
$matriculaId = DB::table('matriculas')->insertGetId([
    'dni' => '35792468Q',
    'year' => 2024,
    'created_at' => now(),
    'updated_at' => now(),
]);

// Obtener los IDs de las asignaturas PRO y LND
$asignaturas = DB::table('asignaturas')
    ->whereIn('nombre', ['PRO', 'LND'])
    ->pluck('id');

// Insertar en la tabla intermedia asignatura_matricula
foreach ($asignaturas as $asignaturaId) {
    DB::table('asignatura_matricula')->insert([
        'idasignatura' => $asignaturaId,
        'idmatricula' => $matriculaId,
        'created_at' => now(),
        'updated_at' => now(),
    ]);
}
});

return "Objetos creados con éxito.";
}

```

## 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

## 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

```
Route::middleware(['auth'])->group(function () {  
    Route::get('/alumnos', [AlumnoController::class, 'index'])->  
name('alumnos.index');  
});
```

**Explicación:** Solo los usuarios autenticados podrán acceder a la ruta /alumnos. Si no están autenticados, serán redirigidos al formulario de login.