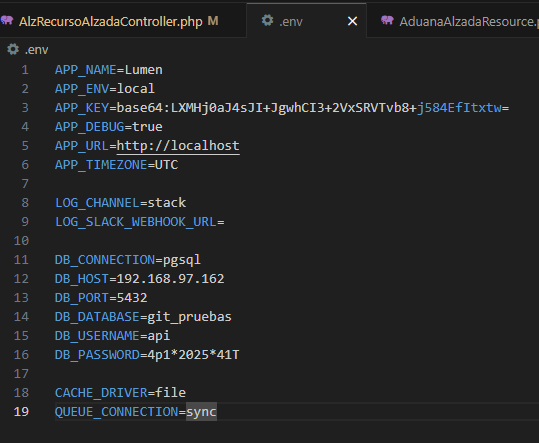
# Creación de una API en lumen

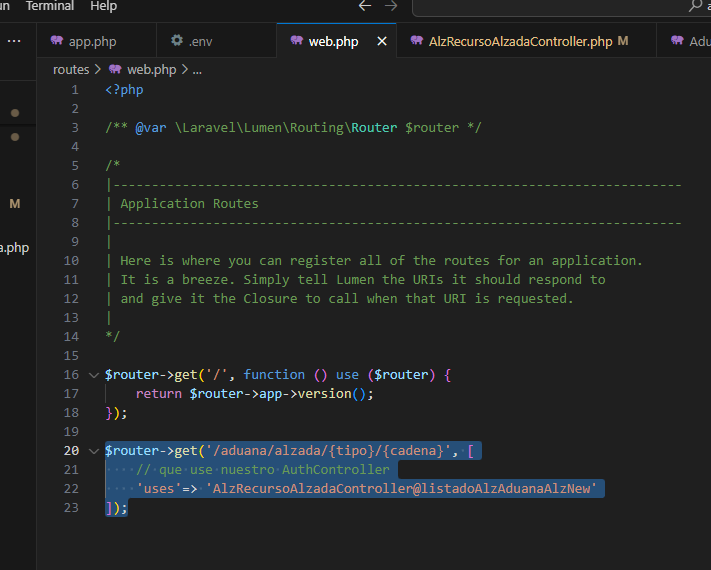
## Credenciales

Primero tenemos que establecer las credenciales para poder acceder a la BD y a sus datos:



## Generación de la ruta

Generamos la ruta para que nos dirija hacia un controlador con un método específico.



El controlador se llama AlzRecursoAlzadaController y la función especifica en listadoAlzAduanaAlzNew.

Dentro del controlador generamos el metodo listadoAlzAduanaAlzNew, al cual le pasamos como parametros las variables tipo y cadena.

<?php

namespace App\Http\Controllers;

use App\Http\Resources\AduanaAlzadaResource;

use App\Models\AlzRecursoAlzada;

use App\Models\parSubadministracionTributaria;

use Illuminate\Http\Request;

class AlzRecursoAlzadaController extends Controller

{

    //

    public function listadoAlzAduanaAlzNew($tipo, $cadena)

    {

        // estamos haciendo un llamado a todos los AlzRecursoAlzada

        // y a las id con su descripcion

        $idAdministracion = 2;

        $recursosAlzada = AlzRecursoAlzada::with([

            'parSubadministracionTributaria' => function ($query) {

                $query->select('id', 'descripcion');

            },

            'alzActosImpugnados' => function ($query) {

                $query->select('id', 'id\_tipo\_acto', 'numero\_acto', 'cite\_acto', 'id\_recurso\_alzada');

            },

            'alzActosImpugnados.parTipoActo' => function ($query) {

                $query->select('id', 'descripcion');

            },

            'alzActuaciones' => function ($query) {

                $query->select('id\_recurso\_alzada', 'id\_tipo\_actuacion', 'fecha\_emision', 'id')

                    ->where('estado', '>', 0)->whereIn('id\_tipo\_actuacion', [3,5,8,9,41,

                    11,19,21,114,500,501,502,503,12,16,56,57,25,31,30,26,28,27,29,61,60,86,58,59,87,85,100

                    ])->orderBy('fecha\_emision');

            },

            'alzActuaciones.parTiposActuaciones' => function ($query) {

                // $query->select('descripcion', 'id');

                $query->select('id', 'descripcion');

            },

            'alzActuaciones.AlzNotificacionesAsig' => function ($query) {

                $query->select('id', 'id\_actuacion', 'fecha\_hora\_notificacion', 'id\_tipo\_demandante')

                    ->where('estado', '>', 0)->orderBy('id\_tipo\_demandante');

            },

            'alzActuaciones.AlzNotificacionesAsig.parTiposDemandantes' => function ($query) {

                $query->select('id', 'descripcion');

            },

            'tipoResolucion' => function ($query) {

                $query->select('id', 'descripcion');

            },

        ])->whereRelation('parSubadministracionTributaria', 'id\_administracion\_tributaria', $idAdministracion)->where('estado', '>', 0);

        if ($tipo == 1) {

            $recursosAlzada = $recursosAlzada->where('numero\_recurso\_alzada', base64\_decode($cadena));

        }

        if ($tipo == 2) {

            $recursosAlzada = $recursosAlzada->where('razon\_social', 'ilike', '%' . base64\_decode($cadena) . '%');

        }

        $recursosAlzada = $recursosAlzada->select('id', 'numero\_recurso\_alzada', 'id\_subadministracion\_tributaria', 'razon\_social', 'numero\_res\_alzada', 'fecha\_respuesta\_administracion', 'fecha\_vencimiento', 'id\_tipo\_resolucion')

            ->get();

        return AduanaAlzadaResource::collection($recursosAlzada);

    }

}

# Modelos

Generamos los modelos de cada una de la tablas de las que queremos obtener datos, dentro de cada uno de los modelos.

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

use Illuminate\Database\Eloquent\Relations\HasMany;

class AlzRecursoAlzada extends Model

{

    // 1 parametricaSubAdministracionTributaria tiene N alzadas

    protected $table = 'alzada.alz\_recursos\_alzada';

    protected function casts(): array

    {

        return [

            'fecha\_vencimiento' => 'datetime:d-m-Y',

            'fecha\_respuesta\_administracion' => 'datetime:d-m-Y',

        ];

    }

    // definimos que un recurso de alzada pertenece a una par. Subadministracion Tributaria

    public function parSubadministracionTributaria(): BelongsTo

    {

        return $this->belongsTo(parSubadministracionTributaria::class, 'id\_subadministracion\_tributaria', 'id');

    }

    //

    public function alzActosImpugnados(): HasMany

    {

        return $this->hasMany(alzActosImpugnados::class, 'id\_recurso\_alzada', 'id');

    }

    //

    public function alzActuaciones(): HasMany

    {

        return $this->hasMany(alzActuaciones::class, 'id\_recurso\_alzada', 'id');

    }

    //

    public function tipoResolucion(): BelongsTo

    {

        return $this->belongsTo(tipoResolucion::class, 'id\_tipo\_resolucion', 'id');

    }

}

Definimos las relaciones como belongsTo y hasMany.

Que es lo que hace el comando with: en vez de consultar N+1 queries solo usamos 2, lo cual es mucho más eficiente.

## Eager Loading

Al acceder a las relaciones de Eloquent como propiedades, los modelos relacionados son cargados con “lazy loading” (El lazy loading, o carga diferida, es una técnica de optimización web que retrasa la carga de elementos (como imágenes, videos o iframes) hasta que son necesarios). Esto significa que los datos de la relación no se cargan hasta que se accede a la propiedad por primera vez. Sin embargo, Eloquent puede cargar las relaciones de forma ansiosa al consultar el modelo principal. La carga ansiosa soluciona el problema de las consultas "N+1". Para ilustrar este problema, considere un modelo de Libro que pertenece a un modelo de Autor:

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

class Book extends Model

{

/\*\*

\* Get the author that wrote the book.

\*/

public function author(): BelongsTo

{

return $this->belongsTo(Author::class);

}

}

Ahora, recuperemos todos los libros y sus autores.

use App\Models\Book;

$books = Book::all();

foreach ($books as $book) {

echo $book->author->name;

}

This loop will execute one query to retrieve all of the books within the database table, then another query for each book in order to retrieve the book's author. So, if we have 25 books, the code above would run 26 queries: one for the original book, and 25 additional queries to retrieve the author of each book.

Thankfully, we can use eager loading to reduce this operation to just two queries. When building a query, you may specify which relationships should be eager loaded using the with method:

$books = Book::with('author')->get();

foreach ($books as $book) {

echo $book->author->name;

}

For this operation, only two queries will be executed - one query to retrieve all of the books and one query to retrieve all of the authors for all of the books:

select \* from books

select \* from authors where id in (1, 2, 3, 4, 5, ...)

Crear el modelo, migración y controlar de una sola vez.  
php artisan make:model Post –mc