LARAVEL (v9.5.2)

# REQUISITOS

php –versión

PHP 8.0.28 (cli) (built: Feb 14 2023 12:12:55) ( ZTS Visual C++ 2019 x64 )

composer –version

Composer version 2.5.5 2023-03-21 11:50:05

# Crear proyecto

CON COMPOSER

composer create-project laravel/laravel mi-proyecto-laravel

CON COMPOSER y el cliente de laravel

composer global require laravel/installer

crear proyecto

laravel new example-app

# Levantar proyecto clonado

composer install

cp .env.example .env

php artisan key:generate

php artisan migrate:fresh --seed

php artisan serve

\*\*\*

php artisan passport:install

php artisan passport:keys

php artisan storage:link

# migración

## Crear archivo de una migración

php artisan make:migration create\_nombreplural\_table

php artisan make:migration create\_clientes\_table

Nota: con el atributo –create= podemos indicar como se llamará la tabla en la base de datos para esa migración.ejecutar migración

php artisan migrate

Para borrar todas las tablas de la base de datos y luego volverlas a crear:

php artisan migrate:refresh

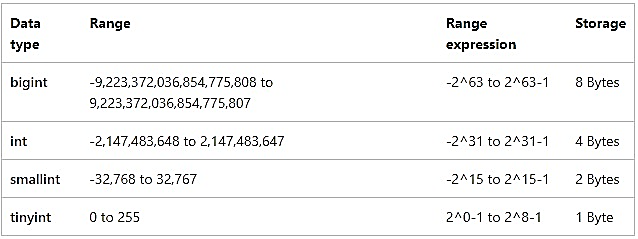
Para eliminar todas las tablas

php artisan migrate:reset

Crear migración con modelo

php artisan make:model tutorial --migration

llenar de campos migración



Creating Tables

Creating Columns

## creación de tabla con campos y relación

    public function up(): void

    {

*Schema*::create('products', function (*Blueprint* $table) {

//id

            $table->id();// primary key bigint(20) AUTO\_INCREMENT UNSIGNED

            //texto

            $table->string('string')->comment('my comment');//varchar(255)  my comment

            $table->text('text')->unique(); // indice unico

            $table->json('json')->default(new *Expression*('(JSON\_ARRAY())'));// longtext Predeterminado json\_array()

            //numero

            $table->decimal('decimal', $precision = 8, $scale = 2)->default(100);//decimal(8,2) 100.00

            $table->double('double', 8, 2);//double(8,2)

            $table->integer('integer');//int(11)

//boolean

            $table->boolean('boolean')->nullable(); //tinyint(1) permite nulo Predeterminado NULL o 0

            //num

            $table->enum('sexo', ['m', 'f']);//enum('m', 'f')

            //fecha y hora

            $table->dateTime('dateTime', $precision = 0);//datetime Predeterminado fecha y hora actual

            $table->date('date');//date "2022-01-10"

            $table->time('time', $precision = 0);//time "23:50:19"

            //`created\_at``updated\_at` de laravel

            $table->timestamps(); //"2023-05-29 04:50:19.000000"

$table->softDeletes(); // deleted\_at

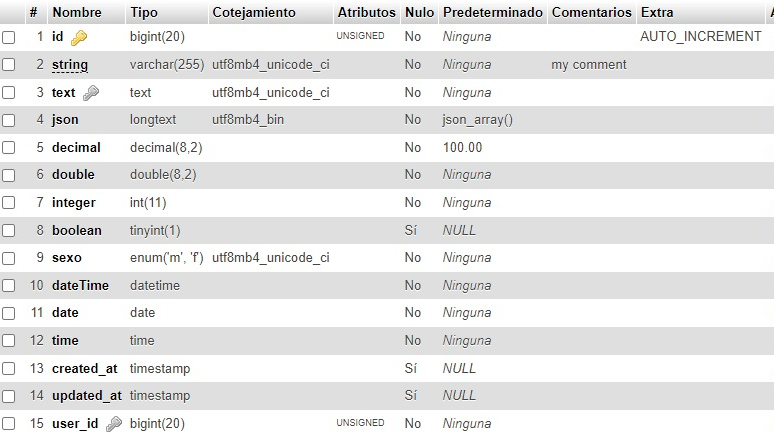
            //Foreign Key Constraints

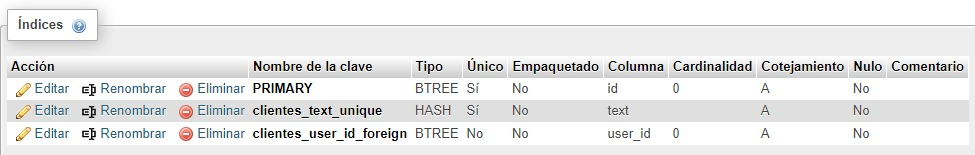
            $table->unsignedBigInteger('user\_id');

            $table->foreign('user\_id')->references('id')->on('users')->onUpdate('cascade')->onDelete('cascade');

        });

    }





$table->unsignedBigInteger('id\_alumno');

$table->foreign("id\_alumno")

->references("id")

->on("alumnos")

->onDelete("cascade")

->onUpdate("cascade");

$table->engine = 'InnoDB';

SQLSTATE[HY000]: General error: 1215 Cannot add foreign key constraint

indicándonos que tenemos un problema de integridad referencial porque la clave foránea no puede ser creada, debido a que la tabla que se está referenciando en la tabla través de category\_id, aún no existe.

https://styde.net/claves-foraneas-e-integridad-referencial-de-datos-en-laravel/

# Factories

Crear factory y seleccionar modelo

php artisan make:factory ProductFactory --model=Product

use *App\Models\Category*;

class ProductFactory extends *Factory*

{

    /\*\*

     \* Define the model's default state.

     \*

     \* @return array<string, mixed>

     \*/

    public function definition(): array

    {

        $category = *Category*::all()->random();

        return [

            'name' => *$this*->faker->sentence(2),

            'description' => *$this*->faker->text(),

            'price' => *$this*->faker->randomElement([9.99,25.99,99.99]),

            'category\_id' => $category->id,

        ];

    }

}

# Seeders

Crear:

php artisan make:seeder nombre\_seeder

use *App\Models\Category*;

use *App\Models\Product*;

class DatabaseSeeder extends *Seeder*

{

    public function run(): void

    {

*Category*::factory(5)->create();

*Product*::factory(25)->create();

    }

}

## Creando solo con el seeders manualmente dentro del método run

*\App\Models\User*::factory()->create([

            'name' => 'Test User',

            'email' => 'test@example.com',

        ]);

ejecutar:

php artisan db:seed

ejecutar seed y migraciones

php artisan migrate –seed

## *hack*

ver tablas de bd

php artisan db:show --counts –views

php artisan db:table users

# modelo

Crear modelo:

php artisan make:model nombre

php artisan model:show User

class Product extends *Model*

{

    use *HasFactory*;

    protected $table = 'products';

    protected $primaryKey = 'id';

    protected $fillable = [

        'id',

        'name',

        'description',

    ];

    protected $casts = [

        'created\_at' => 'date:Y-m-d H:i:s',

        'updated\_at' => 'date:Y-m-d H:i:s',

    ];

    protected $hidden = [

        'updated\_at',

    ];

}

## Fundición o castear data casting

Un tipo de datos que se puede cambiar a otro tipo de datos se puede convertir del tipo de datos de origen al tipo de datos de destino. La conversión de un tipo de datos a otro puede ocurrir implícita o explícitamente. Las funciones de conversión o la especificación CAST (consulte la especificación CAST) se pueden usar para cambiar explícitamente un tipo de datos.

<https://laravel.com/docs/9.x/eloquent-mutators#attribute-casting>

## Accessors & Mutators Atributos del modelo

<https://laravel.com/docs/9.x/eloquent-mutators>

## Generación de clases de modelos

php artisan make:model Flight

crea el modelo con la migración

php artisan make:model Flight -m

ejecutar migracion

php artisan migrate

# Generate a model and a FlightFactory class...

php artisan make:model Flight -f

# Generate a model and a FlightSeeder class...

php artisan make:model Flight -s

# Generate a model and a FlightController class...

php artisan make:model Flight -c

# Generate a model, FlightController resource class, and form request classes...

php artisan make:model Flight --controller --resource --requests

php artisan make:model Flight -crR

# Generate a model and a FlightPolicy class...

php artisan make:model Flight --policy

# Generate a model and a migrati*on, fact*ory, seeder, and controller...

php artisan make:model Flight -mfsc

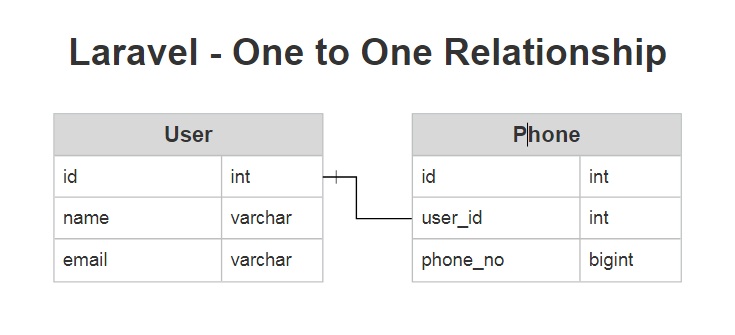
# Shortcut to generate a model, migration, factory, seeder, policy, controller, and form requests...

php artisan make:model Flight --all

# Generate a pivot model...

php artisan make:model Member -p

## RELACION UNO A UNO



### RELACION UNO A UNO

class User extends Model

    public function phone()

    {

        return *$this*->hasOne(*Phone*::class);

    }

### SIN CONVENCIÓN

return $this->hasOne(Phone::class, 'foreign\_key'); // user\_id CLAVE EXTERNA CONVENCIÓN MODELO SINGULAR \_ID

return $this->hasOne(Phone::class, 'foreign\_key', 'local\_key'); //CLaSE // clave externa // $primaryKey

### RELACIÓN UNO A UNO BIDIRECCIONAL

class Phone extends Model

  public function user()

    {

        return *$this*->belongsTo(*User*::class);

    }

### Guardar y borrar uno a uno belongsTo

guardar cuando el modelo es belongsTo

Guardar elemento Si desea asignar un modelo secundario a un nuevo modelo principal belongsTo

$user->phone()->save($phone);

O puede realizar así

$user-> phone()->associate($phone);

$user->save();

$ship->find(1)->captain()->save(new *Captain*(array('name' => 'jean Luc Picard')));

Para eliminar

Para eliminar un modelo principal de un modelo secundario, puede usar el dissociatemétodo. Este método establecerá la clave externa de la relación en null:

$user-> phone()->dissociate();

$user->save();

### SIN CONVENCIÓN

return $this->belongsTo(User::class, 'foreign\_key');

return $this->belongsTo(User::class, 'foreign\_key', 'owner\_key');

## RELACION UNO A MUCHOS

### RELACION UNO A MUCHOS

class Categoria extends Model

  public function libros()

    {

        return *$this*->hasMany(*Libro*::class);

    }

### SIN CONVENCIÓN

 class User

public function facturas()

    {

        return *$this*->hasMany(*Factura*::class, 'cliente\_id', 'id');

    }

 class Factura

public function cliente()

    {

        return *$this*->belongsTo(*User*::class, 'cliente\_id', 'id');

    }

return $this->hasMany(Comment::class, 'foreign\_key');

return $this->hasMany(Comment::class, 'foreign\_key', 'local\_key');

### RELACIÓN UNO A MUCHOS BIDIRECCIONAL

class Libro extends Model

  public function categorias()

    {

        return *$this*->belongsTo(*Categoria*::class);

    }

### GUARDAR

use App\Models\Comment;

use App\Models\Post;

$comment = new Comment(['message' => 'A new comment.']);

$post = Post::find(1);

$post->comments()->save($comment);

        $post->comments()->save($comment);

        $post->refresh();

        // All comments, including the newly saved comment...

        $post->comments;

        $post = *Post*::find(1);

        $post->comments()->saveMany([

            new *Comment*(['message' => 'A new comment.']),

            new *Comment*(['message' => 'Another new comment.']),

        ]);

### CREAR

SE CREA MEDIANTE UN ARREGLO

use *App\Models\Post*;

$post = *Post*::find(1);

$comment = $post->comments()->create([

    'message' => 'A new comment.',

]);

### Asociado BelongsTo

establecerá la clave externa

use *App\Models\Account*;

$account = *Account*::find(10);

$user->account()->associate($account);

$user->save();

Para eliminar un modelo principal de un modelo secundario, puede usar el dissociatemétodo. Este método establecerá la clave externa de la relación en null

    $user->account()->dissociate();

    $user->save();

### SIN CONVENCIÓN

return $this->belongsTo(User::class, 'foreign\_key');

return $this->belongsTo(User::class, 'foreign\_key', 'owner\_key');

## RELACION MUCHOS A MUCHOS (BIDIRECCIONAL)

VA EN ORDEN ALFABÉTICO PRIMERO ROL Y DESPUÉS DE USER ADEMÁS ESTA RELACIÓN DE MUCHOS A MUCHOS V DESPUÉS E L MIGRACIÓN DE LOS DOS MODELOS

php artisan make:migration create\_role\_user\_table

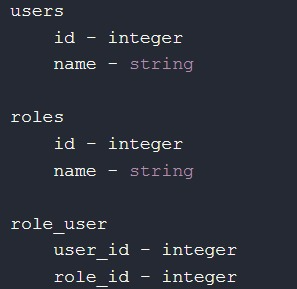


Tabla pivote o intermedia

    $table->unsignedBigInteger('role\_id');

    $table->unsignedBigInteger('user\_id');

    $table->foreign('role\_id')

        ->references('id')->on('roles')

        ->onDelete('cascade');

    $table->foreign('user\_id')

        ->references('id')->on('users')

        ->onDelete('cascade');

class user extends Model

    public function roles() {

        return $this->belongsToMany(Role::class);

    }

class rol extends Model

    public function users() {

        return $this->belongsToMany(User::class);

    }

### SIN CONVENCIÓN

return $this->belongsToMany(Role::class, 'role\_user');

return $this->belongsToMany(Role::class, 'role\_user', 'user\_id', 'role\_id');

Con más campos la tabla pivote

return $this->belongsToMany(Role::class)->withPivot('active', 'created\_by');

con Timestamps no sean nulos

return $this->belongsToMany(Role::class)->withTimestamps();

### Accediendo los datos de la tabla pivote

    use App\Models\User;

    $user = User::find(1);

    foreach ($user->roles as $role) {

        echo $role->pivot->created\_at;

    }

Personalización del pivot nombre del atributo

    En el modelo

return *$this*->belongsToMany(*Podcast*::class)

    ->as('proles')

    ->withTimestamps();

### Accediendo los datos de la tabla pivote Personalizado

       foreach ($user->roles as $role) {

        echo $role->proles->created\_at;

    }

### Agregando modelos una tabla pivote se puede usar attach y sync

   $user = User::find(1);

   $user->roles()->attach(1);

   $user->roles()->attach([1, 2, 3]);

   $user->roles()->sync([1, 2]);

Añadiendo a la tabla pivoteel id y la cantidad

Al adjuntar una relación a un modelo, también puede pasar una matriz de datos adicionales para insertarlos en la tabla intermedia:

                $Factura->productos()->attach([

                    $producto["id"] => ['cantidad' => $producto["cantidad"]]

                ]);

### Eliminar

// Detach a single role from the user...

$user->roles()->detach($roleId);

// Detach all roles from the user...

$user->roles()->detach();

    $user = *User*::find(1);

    $user->roles()->detach([1, 2, 3]);

    $user->roles()->attach([

        1 => ['expires' => $expires],

        2 => ['expires' => $expires],

    ]);

### Attach vs sync

$user->roles()->sync([1, 2, 3]);

También puede pasar valores de tabla intermedios adicionales con los ID:

$user->roles()->sync([1 => ['expires' => true], 2, 3]);

Si desea insertar los mismos valores de la tabla intermedia con cada uno de los ID de modelo sincronizados, puede usar el syncWithPivotValues método:

$user->roles()->syncWithPivotValues([1, 2, 3], ['active' => true]);

La attach función solo agrega registros a la tabla dinámica.

La sync función reemplaza los registros actuales con los nuevos registros. Esto es muy útil para actualizar un modelo.

**Ejemplo:**

Suponiendo que tiene una publicación creada que tiene muchas etiquetas adjuntas donde las ID de las etiquetas son **[1,2,3]** .

Y el usuario tiene la capacidad de actualizar la publicación y sus etiquetas.

El usuario le enviará las nuevas ID de etiquetas [3,4,5].

Si usa la syncfunción, las nuevas etiquetas de la publicación serán solo **[3,4,5]** .

Si usa la attachfunción, las nuevas etiquetas de la publicación serán **[1,2,3,4,5]** .

### Toggle AGREGAR Y QUITAR

Si la identificación dada está actualmente adjunta, se separará. Asimismo, si actualmente se encuentra desvinculado, se adjuntará

    $user->roles()->toggle([1, 2, 3]);

//También puede pasar valores de tabla intermedios adicionales con los ID:

    $user->roles()->toggle([

        1 => ['expires' => true],

        2 => ['expires' => true],

    ]);

### Actualización de un registro en la tabla intermedia

Si necesita actualizar una fila existente en la tabla intermedia de su relación, puede usar el updateExistingPivotmétodo. Este método acepta la clave externa del registro intermedio y una matriz de atributos para actualizar:

    $user = *User*::find(1);

    $user->roles()->updateExistingPivot($roleId, [

        'active' => false,

    ]);

## Mostrar relaciones

$products = *Product*::with('category')->get();

return response()->json($products);

return *User*::where('id',$request->id)->with([

            'facturas' => [

                'formasdepago',

                'productos',

            ],

])->get();

$Factura->load(['formasdepago', 'cliente', 'productos']);

## CONSULTAS

    $user->posts()

    ->where('active', 1)

    ->orWhere('votes', '>=', 100)

    ->get();

select \*

from posts

where user\_id = ? and active = 1 or votes >= 100

   En la mayoría de las situaciones, debe usar grupos lógicos para agrupar las comprobaciones condicionales entre paréntesis:  
Tenga en cuenta que la agrupación lógica ha agrupado correctamente las restricciones y la consulta permanece restringida a un usuario específico:

use *Illuminate\Database\Eloquent\Builder*;

    $user->posts()

            ->where(function (*Builder* $query) {

                return $query->where('active', 1)

                             ->orWhere('votes', '>=', 100);

            })

            ->get();

    select \* from posts

    where user\_id = ? and (active = 1 or votes >= 100)

## Hack:

### Inspección de modelos

php artisan model:show User

<https://styde.net/laravel-6-doc-eloquent-relaciones/#many-to-many>

## método with() versus load()

<https://stackoverflow.com/questions/26005994/laravel-with-method-versus-load-method>

Polimorfismo de una tabla crear dos perfiles

<http://novate.co.uk/using-laravel-polymorphic-relationships-for-different-user-profiles/>

<https://laracasts.com/discuss/channels/laravel/best-practices-for-different-user-types>

<https://www.laravelia.com/post/laravel-one-to-onehasone-eloquent-relationship-tutorial>

## belongsTo

guardar cuando el modelo es belongsTo

Guardar elemento Si desea asignar un modelo secundario a un nuevo modelo principal belongsTo

$user->phone()->save($phone);

O puede realizar así

$user-> phone()->associate($phone);

$user->save();

Para eliminar

Para eliminar un modelo principal de un modelo secundario, puede usar el dissociatemétodo. Este método establecerá la clave externa de la relación en null:

$user-> phone()->dissociate();

$user->save();

guardar cuando el modelo es BelongsToMany

Está muy bien detallado todos los requisitos, me gusta la oferta, pero por problema de salud yo no puedo trabajar muchos días consecutivos por lo que me apego más a lo remoto o hibrido, por ejemplo , que sea de ir un día a oficina por semana o dos saltándose días.

Estoy abierto a cualquier oportunidad hibrida, si sale 100% remoto mucho mejor y haga match con lo que busca el cliente, me escribas de nuevo ☺

<https://serversideup.net/managing-pivot-data-with-laravel-eloquent/>

<https://laravel.com/docs/9.x/eloquent-relationships>

tinker

<https://norvicsoftware.com/eloquent-orm-en-laravel-8/>

relación

<https://www.itsolutionstuff.com/post/laravel-9-one-to-one-relationship-exampleexample.html>

# CONTROLADORES Y RUTAS

## Crear CONTROLADORES

php artisan controller:make PhotoController

php artisan make:controller PhotoController --api

php artisan make:controller Api\\ProductController --model=Product

php artisan make:controller PhotoController --api --model=Product

use *App\Models\Categoria*;

public function update(*Request* $request, $id){

    //$name = $request->name;

}

## Crear RUTAS

Route::apiResource('products', miController::class);

Route::put('/user/{id}', [UserController::class, 'update']);

Recursos anidados

Route::resource('photos.comments', PhotoCommentController::class);

/photos/{photo}/comments/{comment}

## Acciones manejadas por el controlador de recursos

|  |  |  |  |
| --- | --- | --- | --- |
| **Verb** | **Path** | **Action** | **Route Name** |
| GET | /resource | index | resource.index |
| GET | /resource/create | create | resource.create |
| POST | /resource | store | resource.store |
| GET | /resource/{resource} | show | resource.show |
| GET | /resource/{resource}/edit | edit | resource.edit |
| PUT/PATCH | /resource/{resource} | update | resource.update |
| DELETE | /resource/{resource} | destroy | resource.destroy |

php artisan controller:make PhotoController --only=index,show

php artisan make:controller PhotoController --resource

php artisan controller:make PhotoController --except=index

*use App\Http\Controllers\UserController;*

*Route::*resource*(*'photos'*,* PhotoController*::*class*);*

*Route*::resource('photo', 'PhotoController',

                array('only' => array('index', 'show')));

*Route*::resource('photo', 'PhotoController',

                array('except' => array('create', 'store', 'update', 'destroy')));

*Route*::get('user/{id}', [*miController*::class, 'index']);

## Ver rutas

php artisan route:list

# REQUEST

## OBTENER VALOR DEL REQUEST

$name = $request->name;

query cadena de consulta de la url ejemplo www.sitiodeejemplo.net/pagina.php?name=valor1&nombredevalor2=valor2tar sql

$name = $request->query('name', 'Helen');

solicitudes JSON

$name = $request->input('user.name');

$name = $request->string('name')->trim();

$archived = $request->boolean('archived');

$input = $request->all();

$input = $request->only(['username', 'password']);

$input = $request->only('username', 'password');

$input = $request->except('credit\_card');

$file = $request->file('photo'); //<https://laravel.com/docs/9.x/requests#retrieving-uploaded-files>

Determinar si la entrada está presente

        if ($request->has('name')) {

            //Determinar si la entrada está presente

        }

## Crear Solicitud de formulario de validación

app/Http/Requests

php artisan make:request StorePostRequest

**authorize** este es un método opcional donde colocamos la lógica para la autorización del usuario que devolverá true si el usuario está autorizado para hacer la solicitud y false en caso contrario. Si se decide dejar la lógica de autorización en otra parte de la aplicación, simplemente se devuelve true de esta manera

    public function authorize()

    {

        return true;

    }

        public function rules(): array

        {

            return [

            'title' => 'required|unique:posts|max:255',

            'formasdepago\_id' => ['required', 'numeric', 'min:1','regex:/^\d+$/','exists:App\Models\Formasdepago,id'],

'cliente\_id' => ['required', 'numeric', 'min:1', 'regex:/^\d+$/', 'exists:App\Models\User,id'],

'productos' => 'required|array|min:1',

'productos.\*.id' => 'required|numeric|min:1|distinct',

'productos.\*.cantidad' => 'required|numeric|gt:0|min:1',

            ];

        }

## Personalizar mensajes de validación

public function messages**()**

**{**

return **[**

'name.required' =**>** 'El :attribute es obligatorio.',

'price.required' =**>** 'Añade un :attribute al producto',

'price.min' =**>** 'El :attribute debe ser mínimo 0'

**]**;

**}**

public function attributes**()**

**{**

return **[**

'name' =**>** 'nombre del producto',

'price' =**>** 'precio de venta',

**]**;

**}**

Uso en el controlador

use *App\Http\Requests\StorePostRequest*;

public function store(*StorePostRequest* $request): *RedirectResponse* {

    // Retrieve the validated input data...

    $validated = $request->validated();

    // Retrieve a portion of the validated input data...

    $validated = $request->safe()->only(['name', 'email']);

    $validated = $request->safe()->except(['name', 'email']);

}

## Validación manual en el mismo controlador

    public function store(*Request* $request)

    {

        $validador = *Validator*::make($request->all(), [

            'nombre' => 'required',

        ]);

        if ($validador->fails()) {

            $errores = $validador->errors();

            return response()->json(['errores' => $errores], 422);

        }

        $datavalidada = $validador->validated();

        $categoria = *Categoria*::create($datavalidada);

        return response()->json($categoria, 201);

    }

# RESPONSE JSON RESOURCE

Al crear una API, es posible que necesite una capa de transformación que se encuentre entre sus modelos Eloquent y las respuestas JSON que en realidad se devuelven a los usuarios de su aplicación. Por ejemplo, puede que desee mostrar ciertos atributos para un subconjunto de usuarios y no para otros, o puede que desee incluir siempre ciertas relaciones en la representación JSON de sus modelos. Las clases de recursos de Eloquent le permiten transformar de forma expresiva y sencilla sus modelos y colecciones de modelos en JSON.

Por supuesto, siempre puede convertir modelos o colecciones de Eloquent a JSON utilizando sus toJsonmétodos; sin embargo, los recursos de Eloquent brindan un control más granular y sólido sobre la serialización JSON de sus modelos y sus relaciones.

<https://laraveljsonapi.io/docs/>

## JsonResource

Para traer un solo response de la entidad **api/user/{id}**

php artisan make:resource UserResource

return new UserResource(User::findOrFail($id));

            return [

                'id' => *$this*->id,

                'name' => *$this*->name,

                'email' => *$this*->email,

                'created\_at' => *$this*->created\_at,

                'updated\_at' => *$this*->updated\_at,

            ];

Para traer una colección response de la entidad **api/users**

## ResourceCollection

php artisan make:resource UserCollection

return UserResource::collection(User::all());

        return [

            'data' => *$this*->collection,

            'links' => [

                'self' => 'link-value',

            ],

        ];

## Traer una JsonResource con un colección

   use App\Http\Resources\PostResource;

  return [

            'id' => *$this*->id,

            'name' => *$this*->name,

            'email' => *$this*->email,

            'posts' => *PostResource*::collection(*$this*->posts),

            'created\_at' => *$this*->created\_at,

            'updated\_at' => *$this*->updated\_at,

        ];

<https://laravel.com/docs/9.x/eloquent-resources#data-wrapping>

# Realizar diagrama de la bd online

<https://dbdiagram.io/d>

# como eliminar todas las tablas MySQL

En el anterior artículo veíamos [**como eliminar todas las tablas de una base de datos MySQL**](https://www.backtheweb.com/como-eliminar-todas-las-tablas-de-una-base-de-datos-mysql). Sin embargo, si usas **Laravel** dispones de un comando para tal propósito.

php artisan db:wipe

crear controlador indicando el modelo con –m y –r indica q cree los métodos basicos de un crud

php artisan make:controller PacienteController -r -m Paciente

php artisan make:controller ProductController --resource --model=Product

ver tablas de bd

php artisan db:show --counts –views

php artisan db:table users

ver rutas

php artisan route:list -c

Link;

<https://bluuweb.github.io/tutorial-laravel/bases-datos/#eloquent>

<https://bluuweb.github.io/tutorial-laravel/db-relacional/#diseno-db>

<https://richos.gitbooks.io/laravel-5/content/capitulos/chapter6.html>

<https://laravel.com/docs/5.8/eloquent-relationships#the-save-method>

relaciones guardar

<https://laravel.com/docs/10.x/eloquent-relationships#the-save-method>

diagrama

<https://app.diagrams.net/>

video pa un diagram entidad relacion

<https://www.youtube.com/watch?v=TKuxYHb-Hvc&ab_channel=LucidSoftwareEspa%C3%B1ol>

<https://www.youtube.com/watch?v=SkjaSwvsB20&ab_channel=GYPZ>

<https://www.youtube.com/watch?v=AArIcStS0TU&t=146s&ab_channel=TodoCode>

# CARBON MANIPULACIÓN DE FECHAS

https://blog.codmind.com/manipulacion-de-fechas-con-laravel-con-carbon/

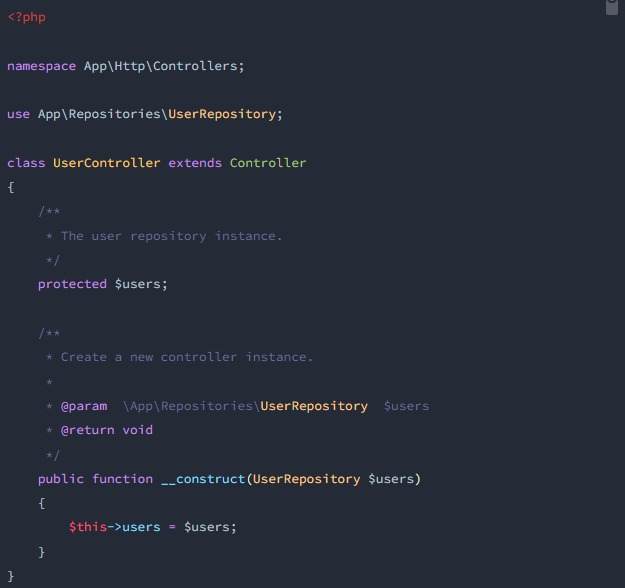
# tinker

php artisan tinker

$c = new App\Models\Categoria;

# Repositorio

## Inyección de dependencia y controladores



# json Configurar cabecera de petición

Content-Type : application/json

Accept : application/json

# sql

use Illuminate\Support\Facades\DB;

$deleted = *DB*::delete('delete from users');

$affected = *DB*::insert('insert into categorias (nombre) values (?)', ['Terror']);

$affected = *DB*::update('update categorias set nombre = "nuevo terror" where nombre = ?',['Terror']);

$data = *DB*::select('select \* from categorias where id = :id', ['id' => 4]);

$data1 = *DB*::select('select \* from categorias');

# Database: Query Builder

El get método devuelve una **Illuminate\Support\Collectioninstancia** que contiene los resultados de la consulta donde cada resultado es una instancia del **stdClass objeto PHP**. Puede acceder al valor de cada columna accediendo a la columna como una propiedad del objeto:

    use *Illuminate\Support\Facades\DB*;

    $users = *DB*::table('users')->get();

    foreach ($users as $user) {

        echo $user->name;

    }

# Obtener SQL generado por Laravel

|  |
| --- |
|  |
|  | *DB::table("ventas")* |
|  | *->join("productos\_vendidos", "productos\_vendidos.id\_venta", "=", "ventas.id")* |
|  | *->where("ventas.created\_at", ">=", $haceMeses)* |
|  | *->where("ventas.created\_at", "<=", $hoy)* |
|  | *->where("ventas.id\_cliente", "=", $idCliente)* |
|  | *->where("productos\_vendidos.codigo\_barras", "=", $codigoBarras)* |
|  | *->toSql();* |

*select \* from `ventas` inner join `productos\_vendidos` on `productos\_vendidos`.`id\_venta` = `ventas`.`id` where `ventas`.`created\_at` >= ? and `ventas`.`created\_at` <= ? and `ventas`.`id\_cliente` = ? and `productos\_vendidos`.`codigo\_barras` = ?*

# CARBON HORA Y FECHA

        $mytime = Carbon::now('America/Guayaquil');

        $ingreso->fecha\_hora=$mytime->toDateTimeString();

# Mutators & Casting

# Links útil

https://dev.to/dalelantowork/laravel-8-api-resources-for-beginners-2cpa