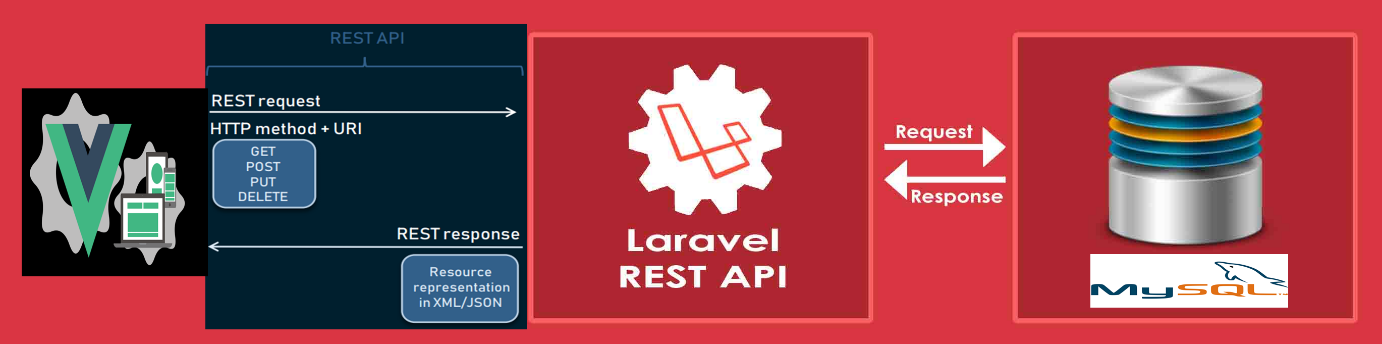
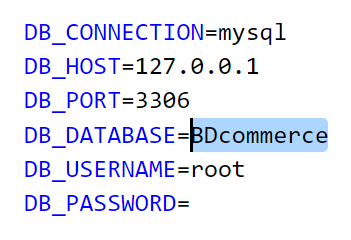
Atelier 2 : Mise en place d’une API



# Connexion à la base de données

Faire une connexion à la base de données en ajoutant le nom de la base de données, dans le fichier de configuration .env de votre projet :





# Cas de la catégorie

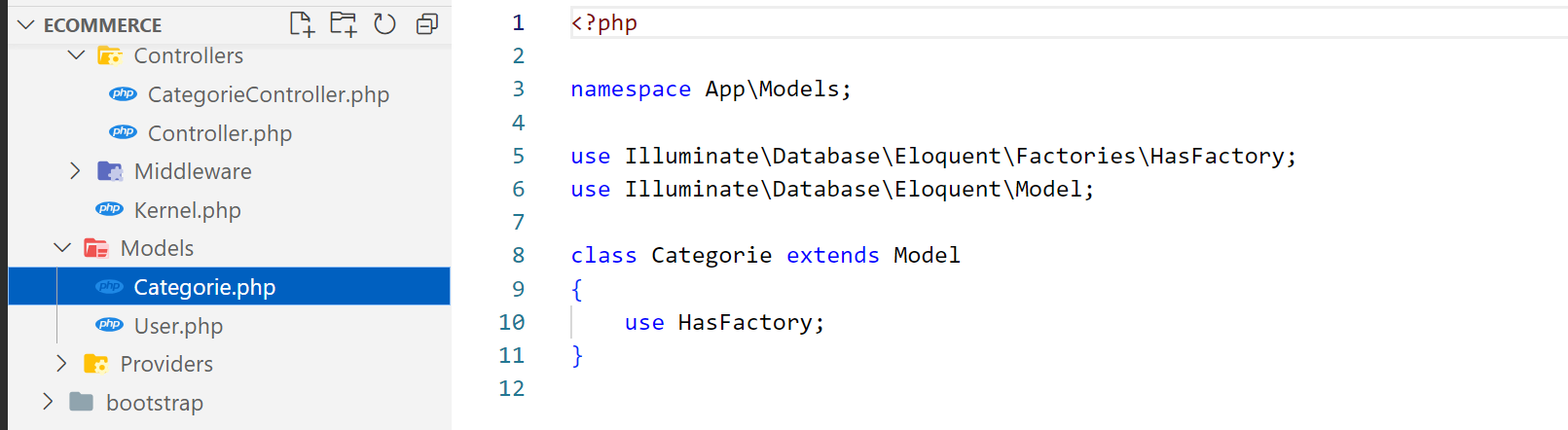
## Créer Model, migration et controller

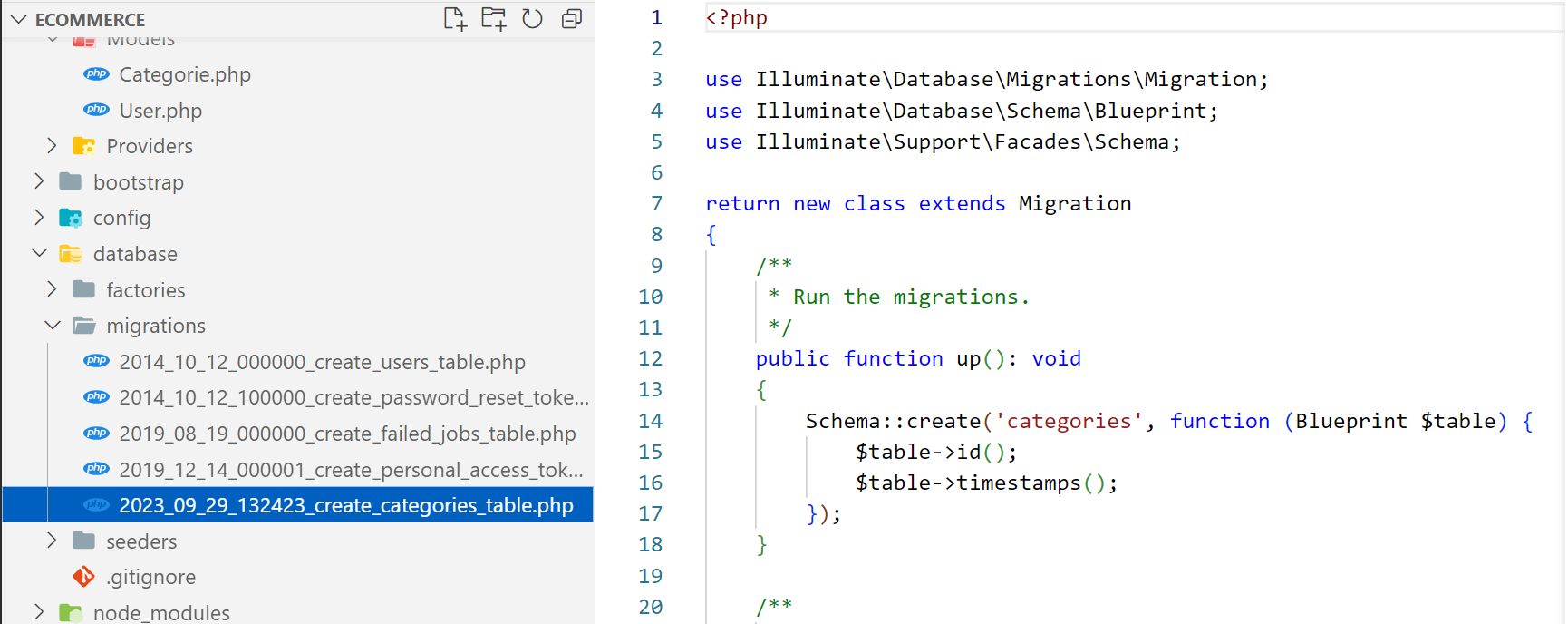
Taper la commande :

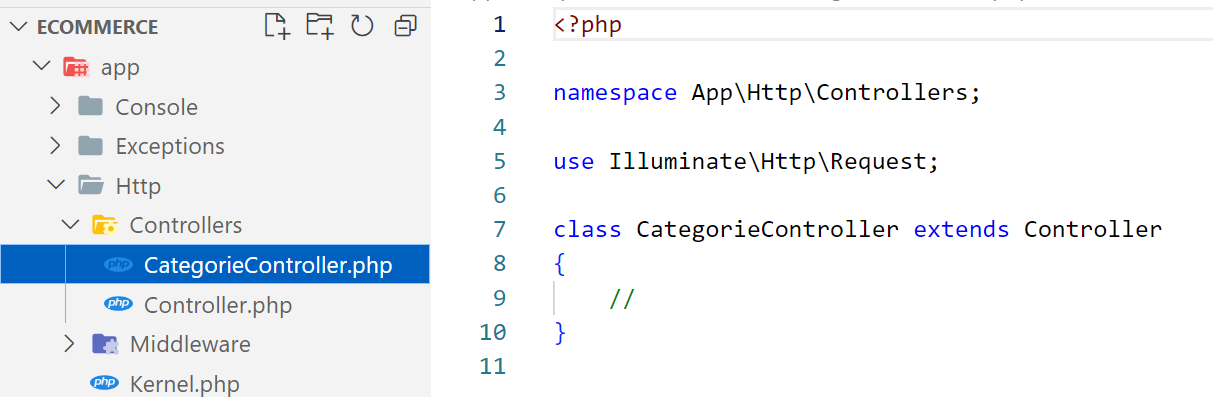
php artisan make:model Categorie -mc

Cette commande créera un Model Categorie, une migration pour la table categories et CategorieController

Fichiers générés :







## Modifier la migration de catégorie

Sous le dossier database/migrations



 public function up()

    {

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

            $table->id();

            $table->string('nomcategorie');

            $table->string('imagecategorie');

            $table->timestamps();

        });

    }

## Modifier le model de catégorie

Dans Models/Categorie.php



class Categorie extends Model

{

    use HasFactory;

    protected $fillable = [

        'nomcategorie','imagecategorie'

    ];

}

## Modifier le controller de catégorie

Dans Http/Controllers/CategorieController.php



<?php

namespace App\Http\Controllers;

use App\Models\Categorie;

use Illuminate\Http\Request;

class CategorieController extends Controller

{

    public function index()

    {

        $categories = Categorie::all()->toArray();

        return array\_reverse($categories);

    }

    public function store(Request $request)

    {

        $categorie = new Categorie([

            'nomcategorie' => $request->input('nomcategorie'),

            'imagecategorie' => $request->input('imagecategorie')

        ]);

        $categorie->save();

        return response()->json('Catégorie créée !');

    }

    public function show($id)

    {

        $categorie = Categorie::find($id);

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

    }

    public function update(Request $request, $id)

    {

        $categorie = Categorie::find($id);

        $categorie->update($request->all());

        return response()->json('Catégorie MAJ !');

    }

    public function destroy($id)

    {

        $categorie = Categorie::find($id);

        $categorie->delete();

        return response()->json('Catégorie supprimée !');

    }

}

## Mettre en place les routes

Dans routes/api.php



use App\Http\Controllers\CategorieController;

Route::middleware('api')->group(function () {

    Route::resource('categories', CategorieController::class);

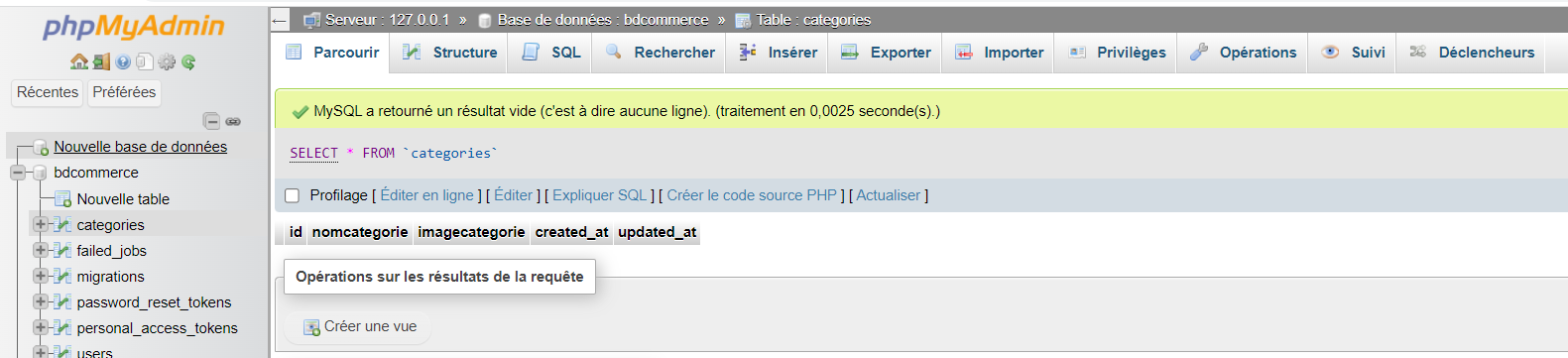
});

## Exécuter la migration

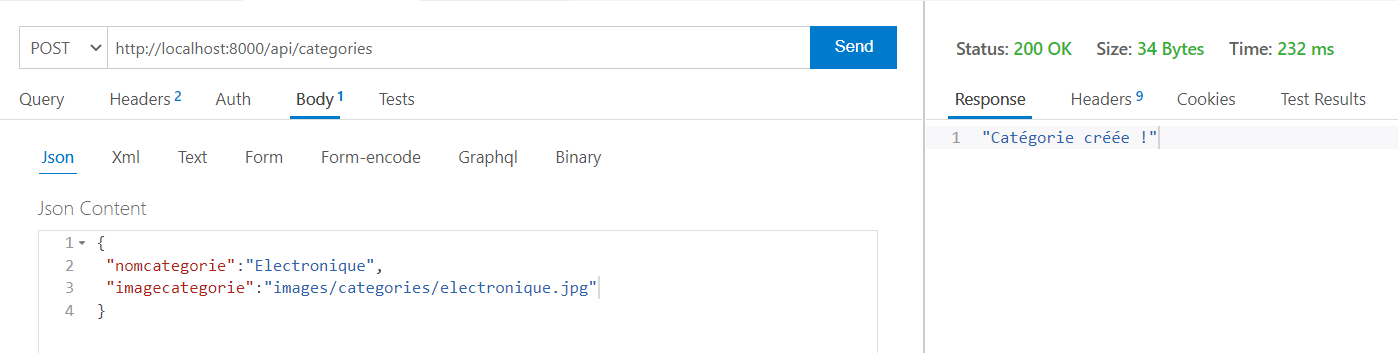
Exécuter la commande :

php artisan migrate

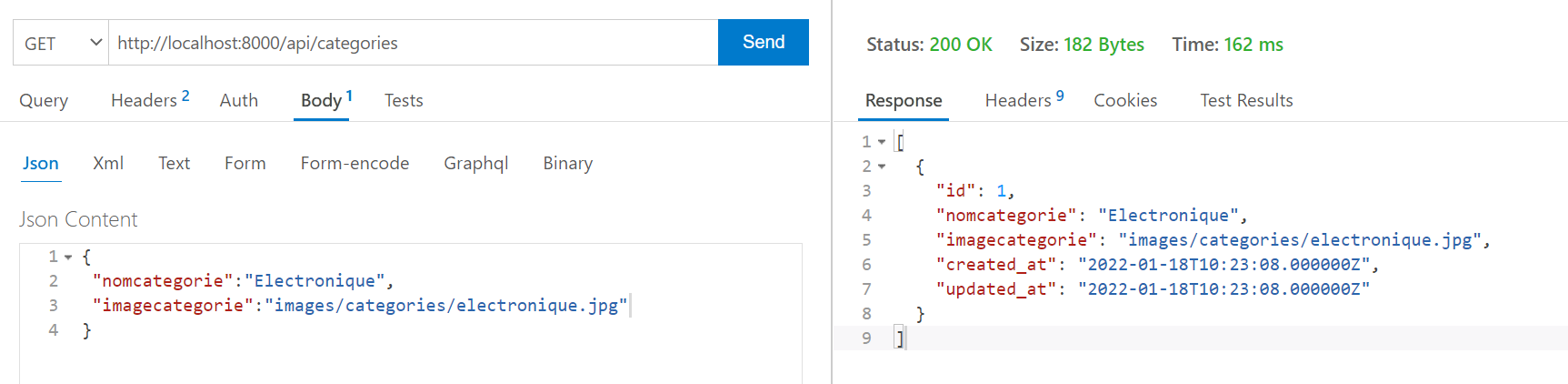


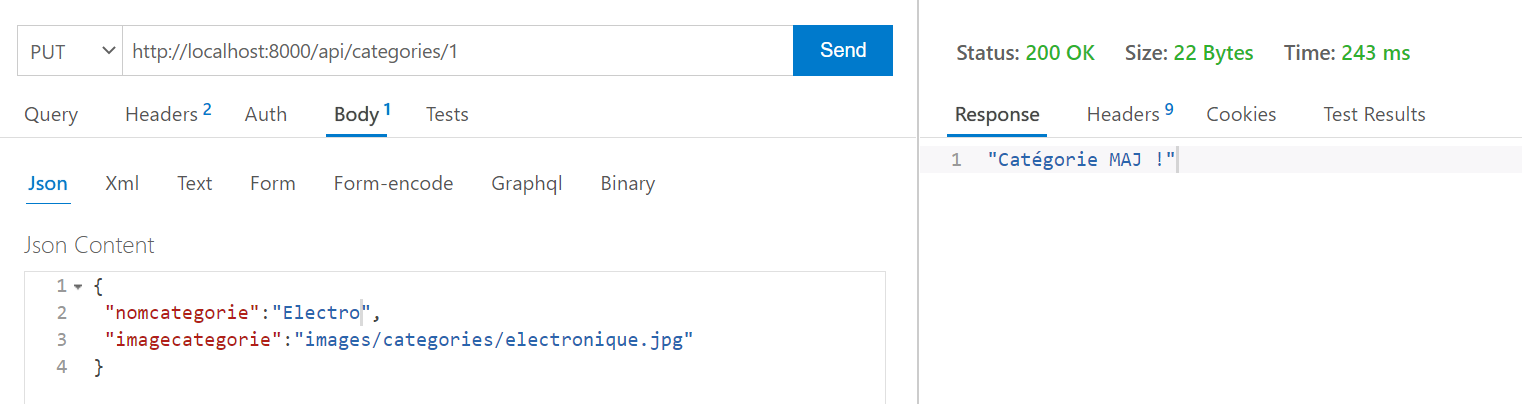


http://localhost:8000/api/categories

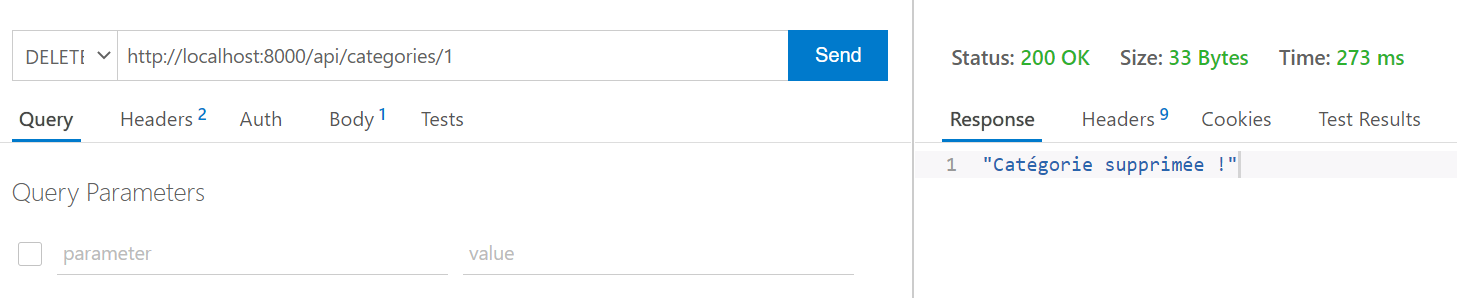


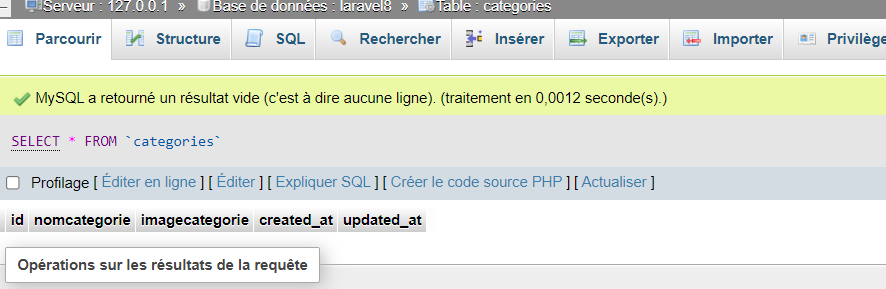


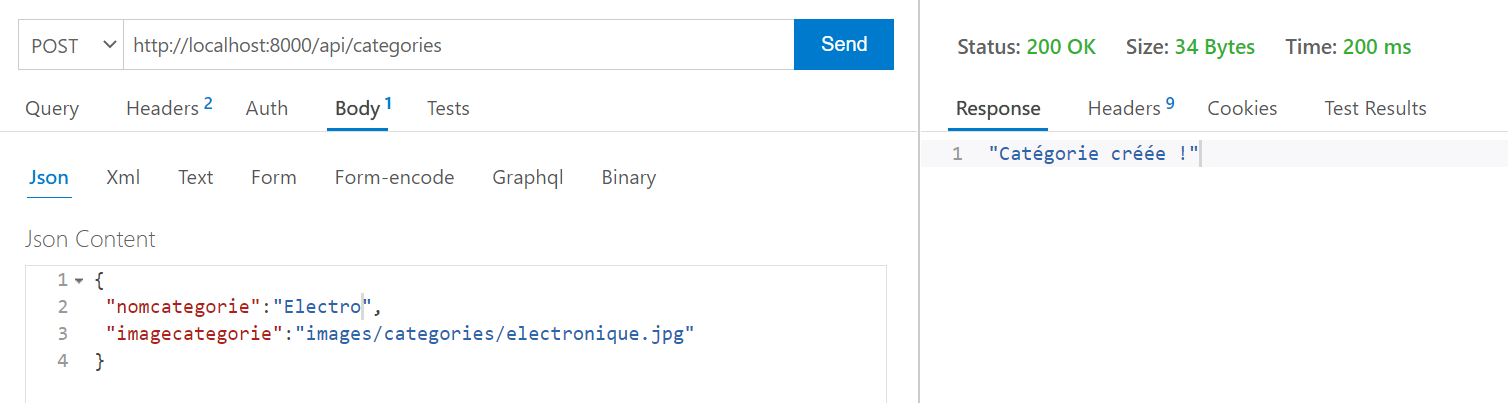












# Relation one to Many dans Laravel

Une relation de un à plusieurs indique que chaque enregistrement de la table principale (celle contenant la clé primaire) peut avoir 0, 1 ou plusieurs enregistrement dans la table secondaire (celle contenant la clé étrangère). Inversement, chaque enregistrement de la table secondaire est relié à un et exactement un enregistrement dans la table principale.

La méthode hasMany() définit une relation de un à plusieurs dans le modèle de la table principale de la relation.

# Cas de la sous catégorie

## Génération de la sous catégorie

On peut créer un modèle, un contrôleur de ressources (lié au modèle) ainsi que la migration dans une seule commande :

php artisan make:model Scategorie -mc

## Modifier la migration de la sous catégorie

Sous le dossier database/migrations



  public function up()

    {

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

            $table->id();

            $table->string('nomscategorie');

            $table->string('imagescategorie');

            $table->unsignedBigInteger('categorieID');

            $table->foreign('categorieID')

                ->references('id')

                ->on('categories')

                ->onDelete('restrict')

                ->onUpdate('restrict');

            $table->timestamps();

        });

    }

Dans la table scategories on déclare une clé étrangère (foreign) nommée categorieID qui référence (references) la colonne id dans la table (on) categories. En cas de suppression (onDelete) ou de modification (onUpdate) on a une restriction (restrict).

En mettant restrict on empêche la suppression d’une catégorie qui a des sous catégories. On doit donc commencer par supprimer ces sous catégories avant de le supprimer lui-même. On dit que la base assure l’intégrité référentielle. Elle n’acceptera pas non plus qu’on utilise pour categorieID une valeur qui n’existe pas dans la table categories.

Une autre possibilité est cascade à la place de restrict. Dans ce cas si vous supprimez une catégorie ça supprimera en cascade les sous catégories de cette catégorie.

C’est une option qui est rarement utilisée parce qu’elle peut s’avérer dangereuse, surtout dans une base comportant de multiples tables en relation. Mais c’est aussi une stratégie très efficace parce que c’est le moteur de la base de données qui se charge de gérer les enregistrements en relation, vous n’avez ainsi pas à vous en soucier au niveau du code.

## Modifier le model de la sous catégorie

Dans Models/Scategorie.php



<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Scategorie extends Model

{

    use HasFactory;

    protected $fillable = [

        'nomscategorie','imagescategorie','categorieID'

    ];

    public function categories()

    {

        return $this->belongsTo(Categorie::class,"categorieID");

    }

}

On a défini une relation pour permettre à une sous catégorie d'accéder à sa catégorie parente. Pour définir l'inverse d'une relation hasMany, définissez une méthode de relation sur le modèle enfant qui appelle la méthode belongsTo.

## Modifier le controller de la sous catégorie

Dans Http/Controllers/ScategorieController.php



<?php

namespace App\Http\Controllers;

use App\Models\Scategorie;

use Illuminate\Http\Request;

class ScategorieController extends Controller

{

    public function index()

    {

        $scategories = Scategorie**::with('categories')**->get()->toArray();

        return array\_reverse($scategories);

    }

    public function store(Request $request)

    {

        $scategorie = new Scategorie([

            'nomscategorie' => $request->input('nomscategorie'),

            'imagescategorie' => $request->input('imagescategorie'),

            'categorieID' => $request->input('categorieID'),

        ]);

        $scategorie->save();

        return response()->json('S/Categorie créée !');

    }

    public function show($id)

    {

        $scategorie = Scategorie::find($id);

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

    }

    public function update(Request $request, $id)

    {

        $scategorie = Scategorie::find($id);

        $scategorie->update($request->all());

        return response()->json('S/Catégorie MAJ !');

    }

    public function destroy($id)

    {

        $scategorie = Scategorie::find($id);

        $scategorie->delete();

        return response()->json('Scategorie supprimée !');

    }

    public function showSCategorieByCAT($idcat)

    {

$Scategorie= Scategorie::where('categorieID', $idcat)->with('categories')->get()->toArray();

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

    }

}

En plus des méthodes CRUD, nous ajoutons une méthode qui permet d’avoir la liste des sous catégories pour une catégorie donnée (showSCategorieByCAT).

## Mettre en place les routes



Dans routes/api.php ajouter :

use App\Http\Controllers\ScategorieController;

Route::middleware('api')->group(function () {

            Route::resource('scategories', ScategorieController::class);

});

Route::get('/scat/{idcat}', [ScategorieController::class,'showSCategorieByCAT']);

Une route supplémentaire a été ajoutée qui permettra d’appeler le méthode showSCategorieByCAT du controller de sous catégories.

# Modification du model de catégorie



Dans la classe de Models/Categories.php, la relation hasMany doit être définies comme suit dans la classe :

    public function scategories()

    {

        return $this->hasMany(Scategorie::class ,"categorieID");

    }

Puisque hasMany() peut retrouver plus d'un enregistrement, il est d'usage de mettre le ou les mots définissant la relation au pluriel.

Si vous ne spécifiez pas de manière explicite le nom de la table dans un modèle, Laravel le déduit à partir du nom du modèle en le mettant au pluriel et en mettant la première lettre en minuscule. Donc avec le modèle Article il en conclut que la table s’appelle articles par exemple.

Les deux méthodes mises en place (belongsTo dans Scategories et hasMany dans Categories) permettent de récupérer facilement un enregistrement lié.

## Exécuter la migration

Exécuter la commande :

php artisan migrate

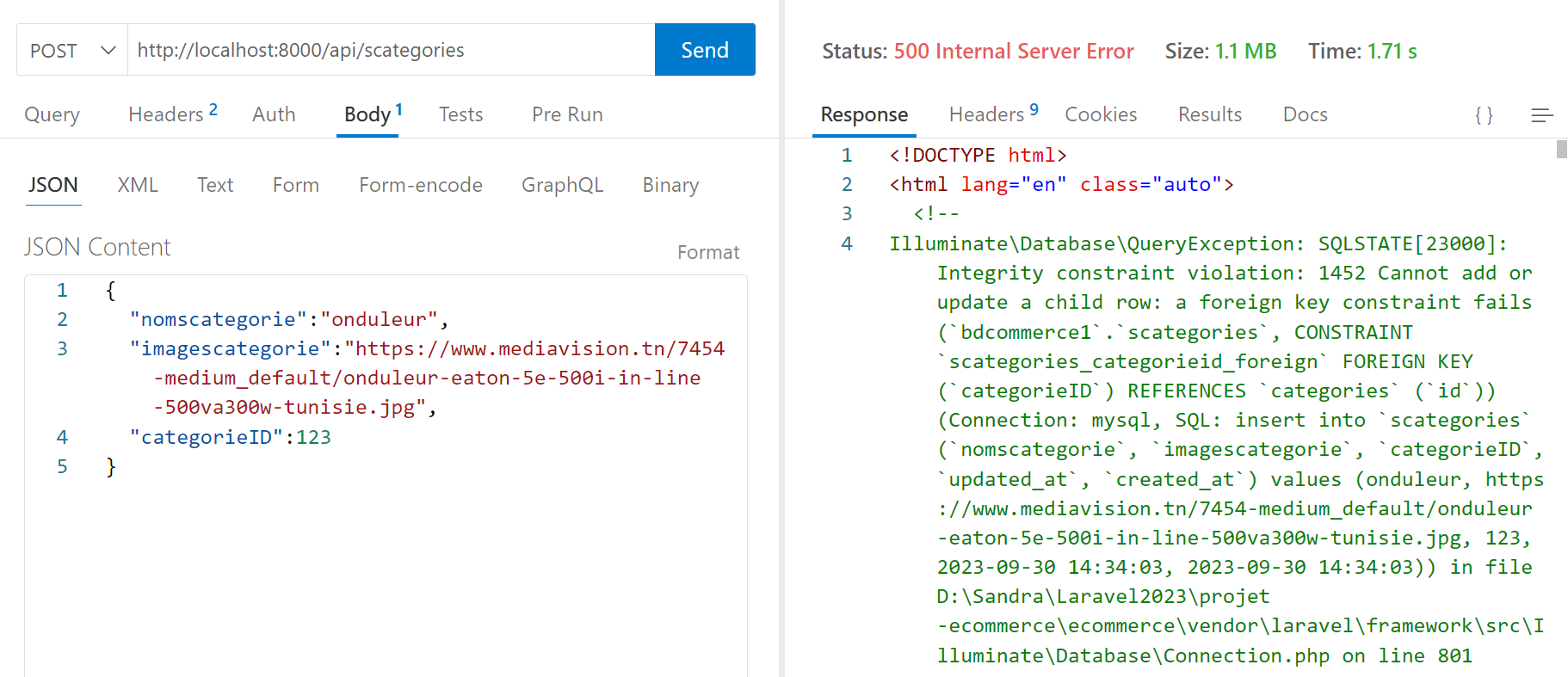
## Exécuter et faire le test de l’api

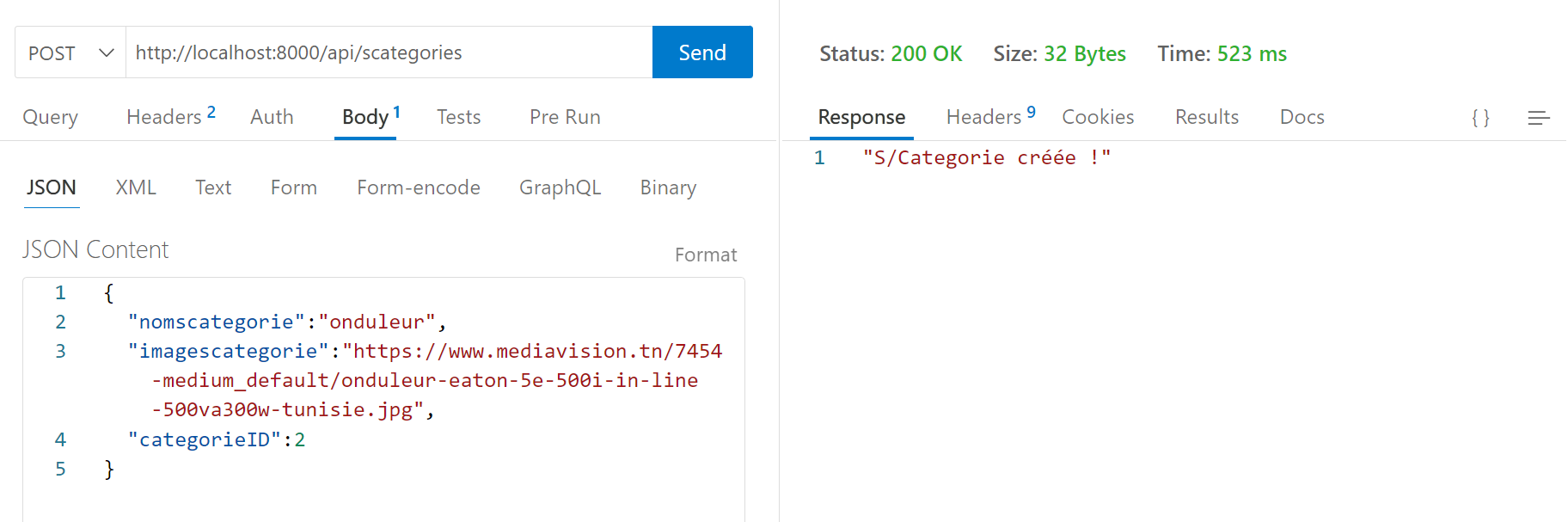
Exécuter la commande :

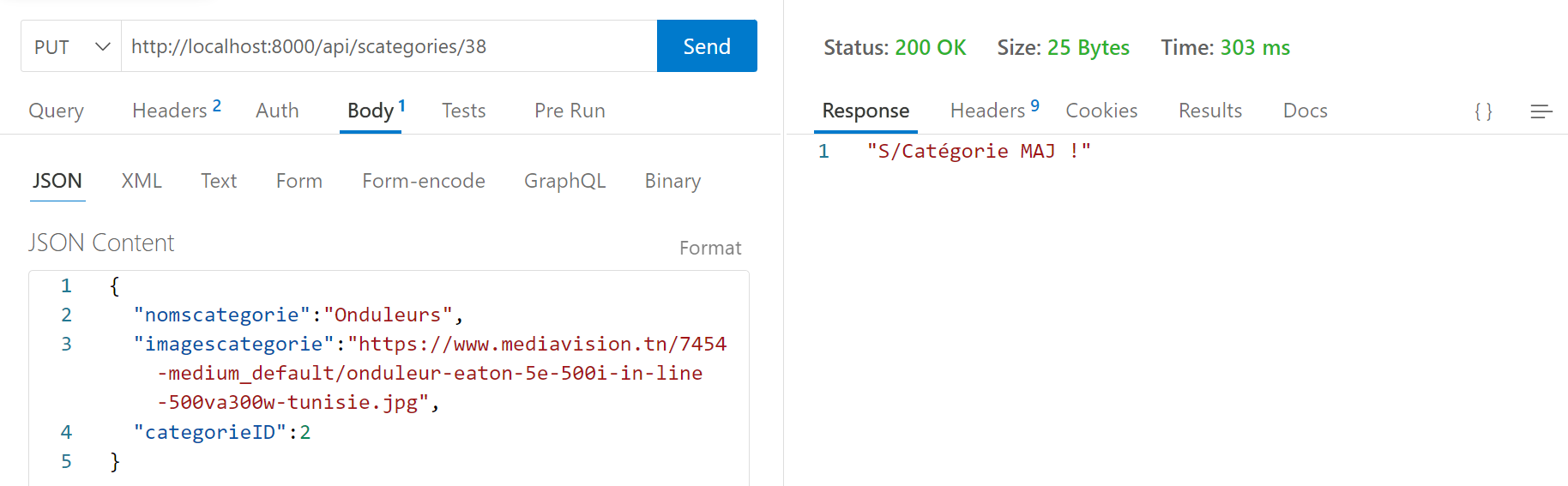
php artisan serve

<http://localhost:8000/api/scategories>

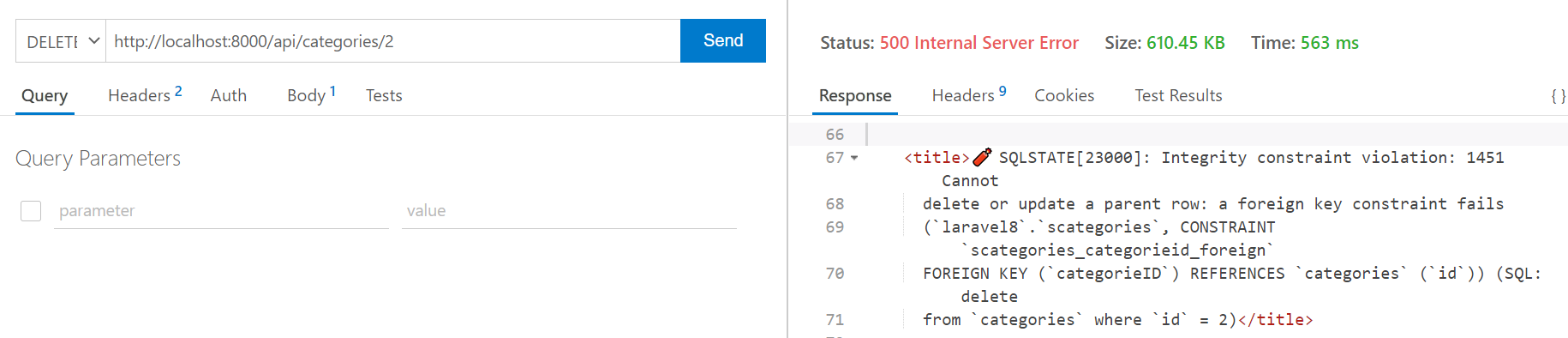
on donne une valeur inexistante pour la catégorie pour tester la contrainte d’intégrité



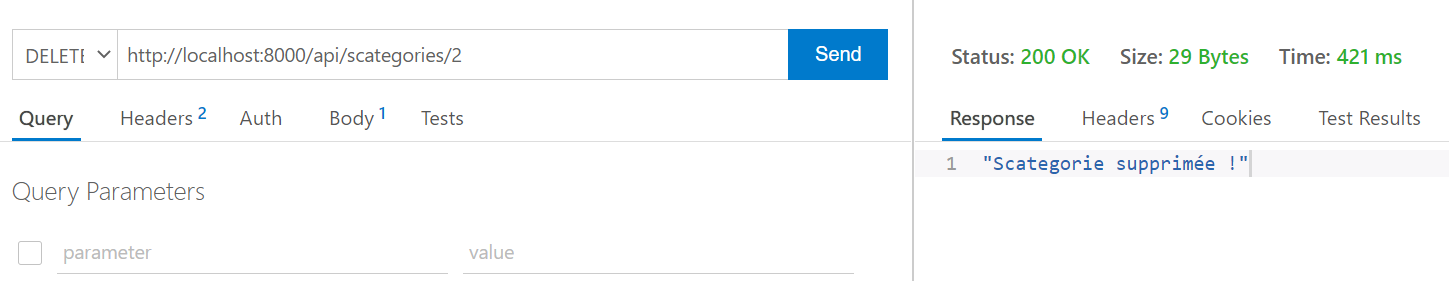




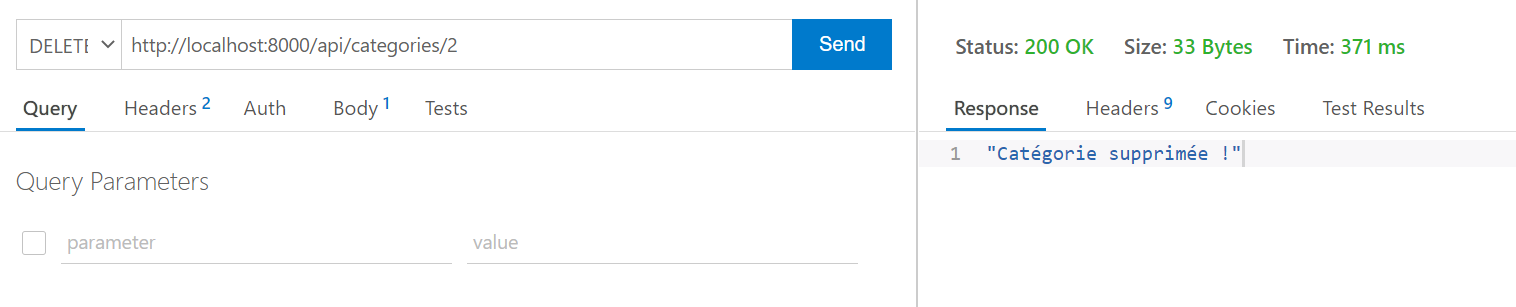
En essayant de supprimer la catégorie. Test de la contrainte d’intégrité.



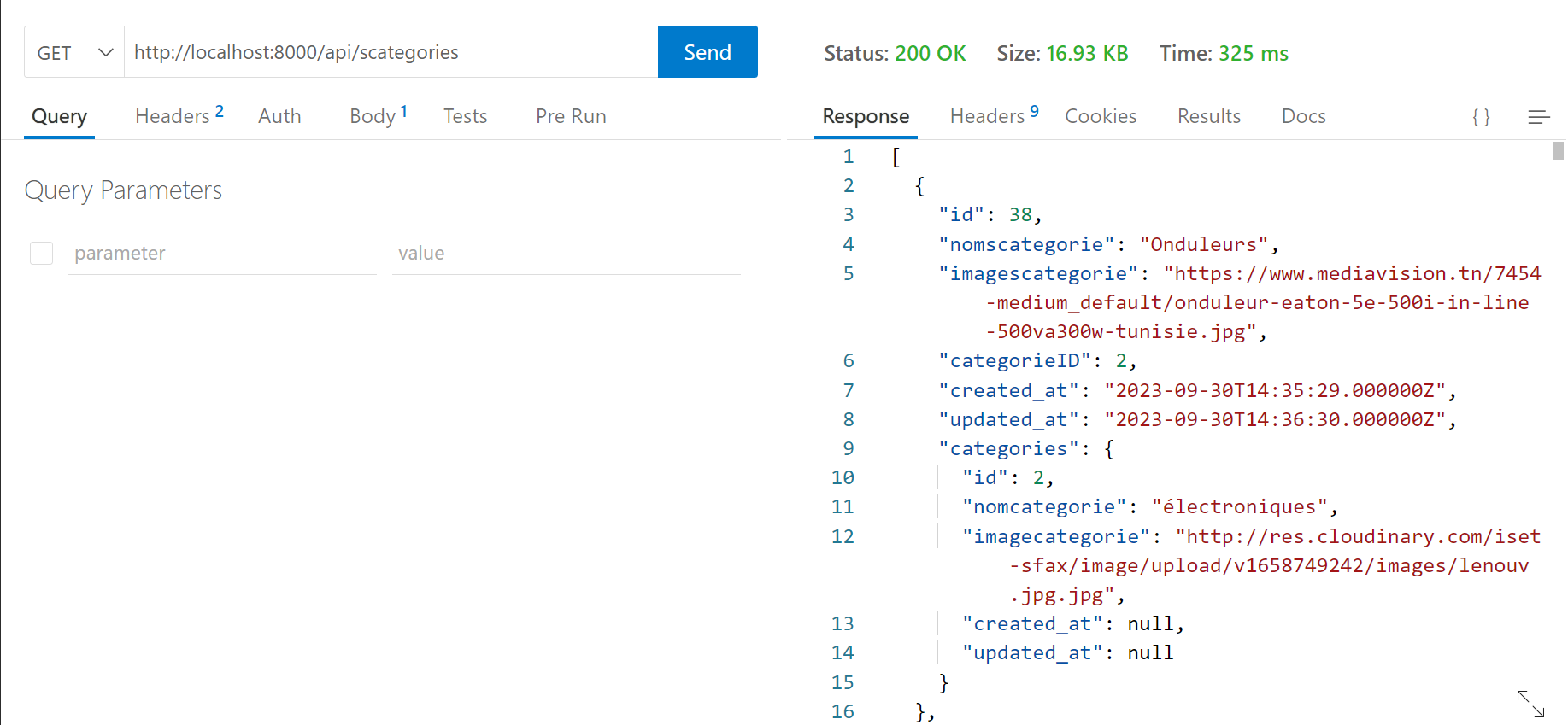
Supprimer la sous catégorie faisant référence :



Maintenant c’est possible de supprimer la catégorie :

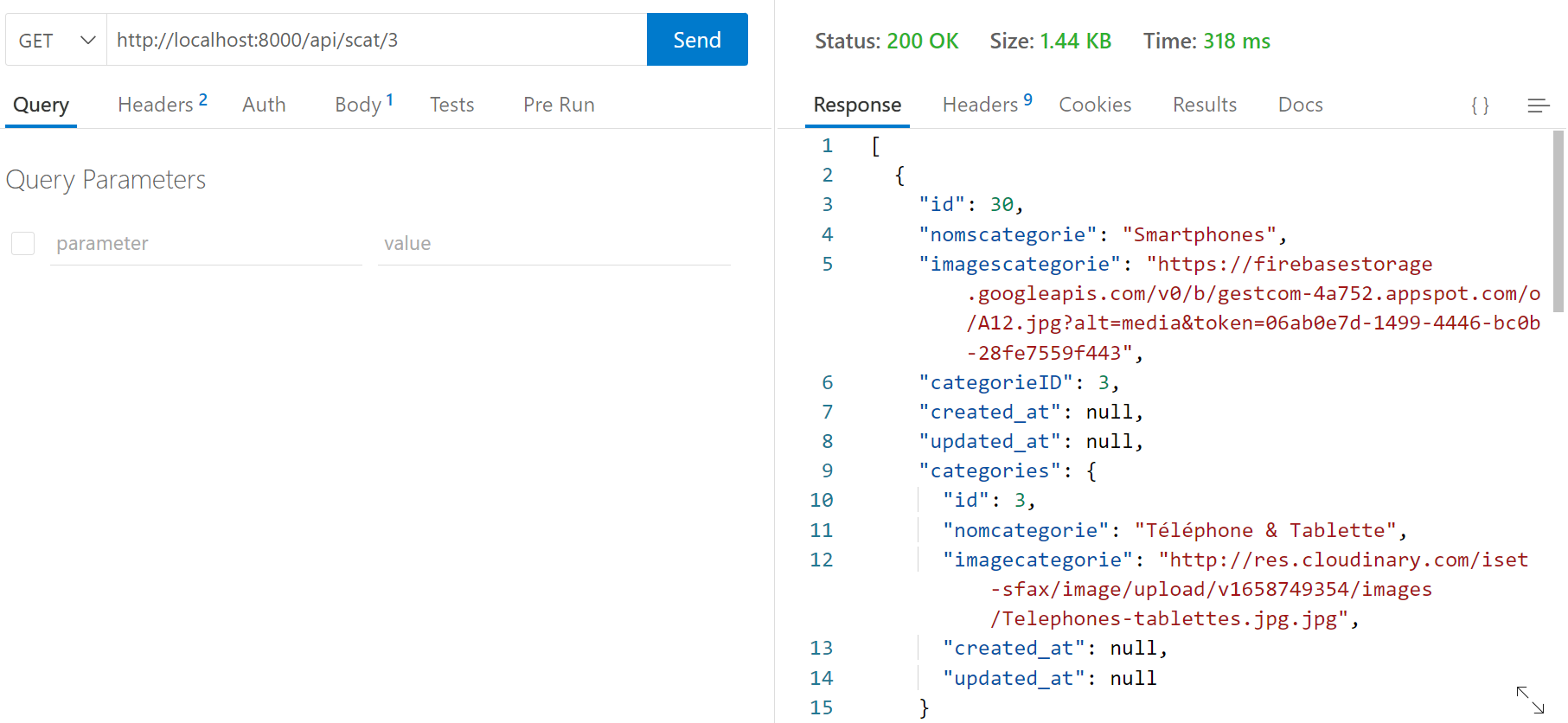


Afficher tous les enregistrements :



Test de la requête qui affiche les sous catégories d’une catégorie donnée.

<http://localhost:8000/api/scat/3>



# Cas de l’article

## Génération de l’article

On peut créer un modèle, un contrôleur de ressources (lié au modèle) ainsi que la migration dans une seule commande :

php artisan make:model Article -mc

## Modifier la migration de l’article

Sous le dossier database/migrations



 public function up()

    {

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

            $table->id();

            $table->string('designation',100)->unique();

            $table->string('marque');

            $table->string('reference');

            $table->string('qtestock');

            $table->string('prix');

            $table->string('imageart');

            $table->unsignedBigInteger('scategorieID');

            $table->foreign('scategorieID')

                ->references('id')

                ->on('scategories')

                ->onDelete('restrict')

                ->onUpdate('restrict');

            $table->timestamps();

        });

    }

## Modifier le model de l’article

Dans Models/Article.php

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Article extends Model

{

    use HasFactory;

    protected $fillable = [

        'designation','marque','reference','qtestock','prix',

        'imageart','scategorieID'

        ];

public function scategories()

        {

        return $this->belongsTo(Scategorie::class,"scategorieID");

        }

}

## Modifier le controller de l’article

Dans Http/Controllers/ArticleController.php



<?php

namespace App\Http\Controllers;

use App\Models\Article;

use Illuminate\Http\Request;

class ArticleController extends Controller

{

      public function index()

    {

        $articles = Article::with('scategories')->get()->toArray();

        $res = array\_reverse($articles);

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

    }

        public function store(Request $request)

        {

            $article = new Article();

            $article->designation= $request->input('designation');

            $article->marque= $request->input('marque');

            $article->reference= $request->input('reference');

            $article->qtestock= $request->input('qtestock');

            $article->prix= $request->input('prix');

            $article->imageart= $request->input('imageart');

            $article->scategorieID= $request->input('scategorieID');

            $article->save();

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

        }

        public function show($id)

        {

            $article= Article::find($id);

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

        }

        public function update($id, Request $request)

        {

            $article = Article::find($id);

            $article->update($request->all());

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

        }

        public function destroy($id)

        {

            $article = Article::find($id);

            $article->delete();

            return response()->json(['message' => 'Article deleted successfully']);

        }

}

## Mettre en place les routes



Dans routes/api.php ajouter :

use App\Http\Controllers\ArticleController;

Route::middleware('api')->group(function () {

    Route::resource('articles', ArticleController::class);

});

## Modification du model de la sous catégorie



Dans la classe de Models/Scategories.php, la relation hasMany doit être définies comme suit dans la classe :

    public function article()

    {

        return $this->hasMany(Article::class,"scategorieID");

    }

## Exécuter la migration

Exécuter la commande :

php artisan migrate

### 

## Exécuter et faire le test de l’api

Exécuter la commande :

php artisan serve

<http://localhost:8000/api/articles>

