Laravel Nested Resource CRUD

The essentials

- Functional perspective
- Nested resource routes
- Creating the controller
- Creating views

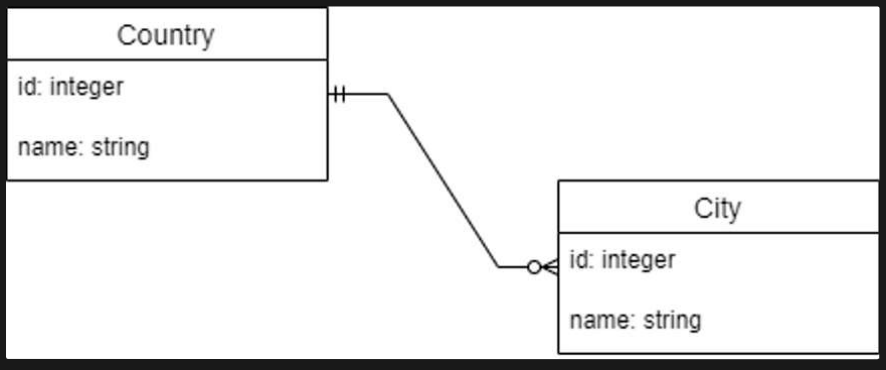
Digging deeper

Further reading

The essentials

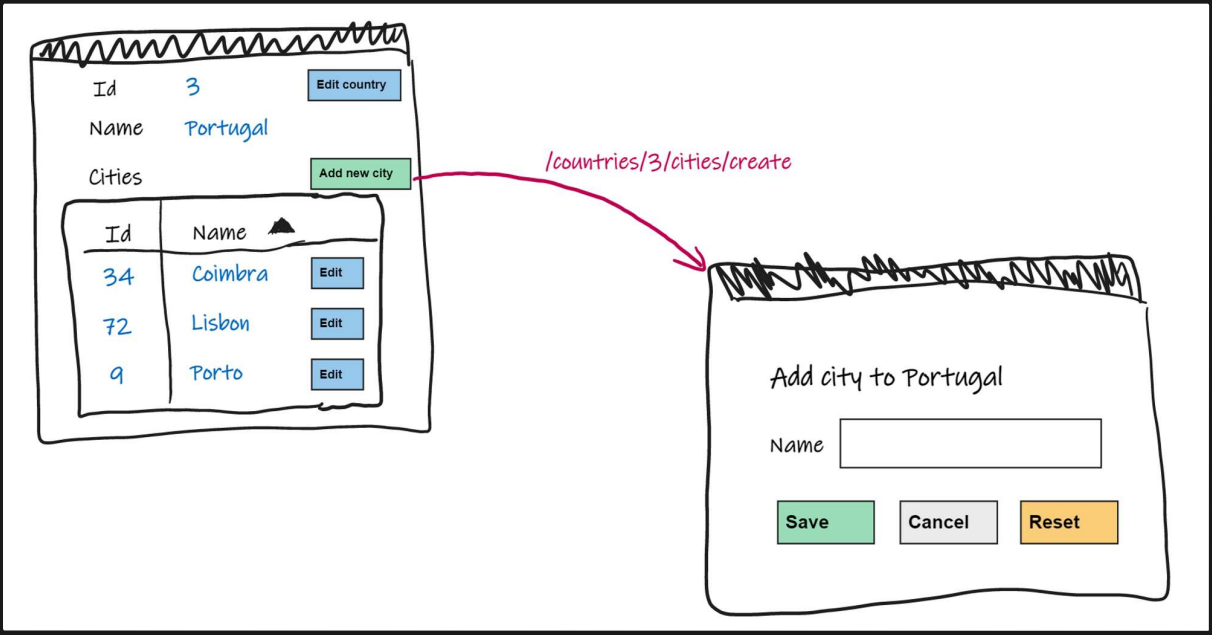
Nested Resource CRUD is a UI design pattern that is useful when users need to manage related resources. For example, a country can have many cities, an order can have many orderlines, a blog post can have many comments, and each comment can have many reactions. This page explains the ideas behind this pattern and how to implement this in Laravel.

In a nested resource CRUD, some CRUD pages of a related resource are combined in pages of the parent resource. For example, given the following two entities that are related:



Functional perspective

A country consists of cities and a city belongs to one country. In a nested resource CRUD design to manage these resources, the Country show (and/or edit) page includes an index-like section that shows a list of Cities that belong to that country and a create button that navigates to a form where the user can add a city to the given country:



Notice that, in the example above, a URL that opens the create page of a city incorporates the ID of the related country.

Nested resource routes

Laravel supports nested resources which allow you to define relationships between multiple models in a single request. This is often used when dealing with parent and child models, such as a User having multiple Posts. Nested resources can also be used to define complex routes that include multiple models.

Using nested resources, you can create routes that represent the relationship between the models. This makes writing complex applications easier and allows you to quickly access the data you need. To create a nested resource, you must define the parent resource and the nested resources in the `routes/web.php` file. For example, to create a nested resource for the countries and cities example, you would define the following:

```
Route::resource('countries.cities', CountryCityController::class);
```

This route defines all the necessary routes for the City resource nested within the Country resource. The routes created by this command will have the prefix `/countries/{country}/city`. This allows you to access the cities belonging to a specific country.

Creating the controller

You can create a nested resource controller using the Artisan command line tool. This command creates all the necessary routes for the nested resource, allowing for complex applications to be quickly developed. Such a command should look like this:

```
php artisan make:controller CountryCityController -r --model=City --
parent=Country
```

The generated controllers all have an extra parameter: the parent model. For example, the show method:

```
/** * Display the specified resource. */ public function show(Country
$country, City $city) { // TODO return a proper view }
```

The parent model can - of course - be passed to a view, for example:

```
/** * Show the form for creating a new resource. */ public function
create(Country $country) { return view('countries.cities.create', [
'country' => $country ]); }
```

It then can be used to generate proper routes or render for example the title of the country where to add a city to:

```
<x-layout> <div class="box"> <form action="{{
route('countries.cities.store', $country) }}" method="POST"> @csrf <h1
class="title is-4">Create a New City to {{ $country->name }}</h1> <br> //
...
```

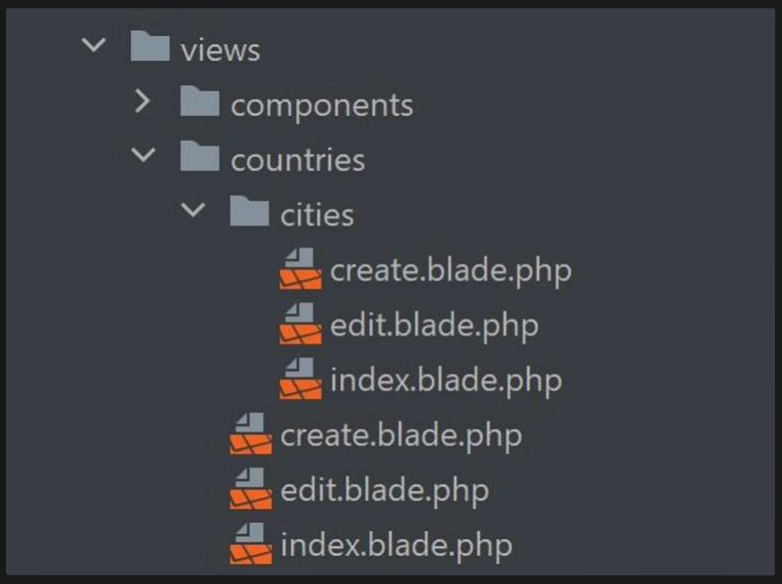
The corresponding store method also has the same parameter added, which can be used to associate the new city to:

```
/** * Store a newly created resource in storage. */ public function
store(Request $request, Country $country) { $validated = $request-
>validate([ 'name' => 'required' ]); $city = $country->cities()-
>create($validated); return redirect()->route('countries.show', $country)
->with('success', "$city->title is created successfully"); }
```

It can also be used to redirect to a proper page, as the example above shows.

Creating views

It is a good practice to store nested resources views also in a nested manner, like:



In this example, it is chosen that the `cities.show` view is not present because it completely overlaps with the `countries.cities.index` view. These views can also be generated with an Artisan command:

```
php artisan make:view countries.cities.create
```

Digging deeper

If you have a controller for the parent resource and a nested controller you will find that the parent's `show()` method and nested `index()` route may overlap from a functional perspective. The nested index usually shows details of the parent resource (which is the same as what a show method of the parent controller should do) together with the list of the related nested resource. You should consider this overlap in your functional design, and decide if one of them may be removed or dealt with in another way.

Using this technique, you can define nested resources as deeply as you need. You can also define multiple levels of nesting, such as a User having many Posts, and each Post having many Comments.

Further reading

For more information on how to use nested resources in Laravel, please refer to the official documentation here.