CONTROLLERS

1.1. Introduction

Thay vì định nghĩa các logic xử lý yêu cầu dưới dạng các closures trong file `route`, trong ứng dụng thực tế sẽ có rất nhiều `route`, nếu như logic mà đặt hết trong `route` thì file này sẽ rất lớn và code lúc này sẽ trở nên phức tạp. Vì vậy nên tổ chức lại các hành động này bằng cách sử dụng các lớp "controller". Controllers có thể nhóm logic xử lý yêu cầu liên quan vào một lớp chung. Ví dụ: một lớp `UserController` có thể xử lý tất cả các yêu cầu đến liên quan đến người dùng, bao gồm hiển thị, tạo, cập nhật và xóa người dùng. Theo mặc định, controllers được lưu trữ trong thư mục `app/Http/Controllers`.

1.2. Writing Controllers

Basic Controllers

Hãy xem một ví dụ về controller cơ bản. Chú ý rằng controller được kế thừa từ lớp controller trong thư mục: `App\Http\Controllers\Controller`:

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\User;


class UserController extends Controller
{
    /**
     * Show the profile for a given user.
     *
     * @param  int  $id
     * @return \Illuminate\View\View
     */
    public function show($id)
    {
        return view('user.profile', [
            'user' => User::findOrFail($id)
        ]);
    }
}
```

Xác định một route đến phương thức trong controller như sau:

```
use App\Http\Controllers\UserController;

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

Khi một yêu cầu đến khớp với route URI đã chỉ định, phương thức `show` trong `App\Http\Controllers\UserController` sẽ được gọi và các tham số của route sẽ được truyền cho phương thức.

Lưu ý: Controllers không bắt buộc kế thừa từ lớp cơ sở `App\Http\Controllers\Controller`. Tuy nhiên, khi đó sẽ không có quyền truy cập vào các tính năng như `middleware` và các phương thức `authorize`.

Single Action Controllers

Nếu một controller action phức tạp, có thể dành toàn bộ lớp controller cho action đó. Để thực hiện điều này, định nghĩa một phương thức `__invoke` trong controller:

```
<?php

namespace App\Http\Controllers;
use App\Http\Controllers\Controller;
use App\Models\User;

class ProvisionServer extends Controller
{
    /**
     * Provision a new web server.
     *
     * @return \Illuminate\Http\Response
     */
    public function __invoke()
    {
        // ...
    }
}
```

Khi đăng ký các routes cho single action controllers, không cần chỉ định phương thức trong controller. Thay vào đó, chỉ cần chuyển tên của controller cho router:

```
use App\Http\Controllers\ProvisionServer;
Route::post('/server', ProvisionServer::class);
```

Có thể tạo một invokable controller bằng cách sử dụng tùy chọn `--invokable` của lệnh Artisan `make:controller`:

```
php artisan make:controller ProvisionServer --invokable
```

Controller stubs có thể được tùy chỉnh bằng cách sử dụng stub publishing.

1.3. Controller Middleware

Middleware có thể được gán cho các routes của controller trong các route:

```
Route::get('profile', [UserController::class, 'show'])->middleware('auth');
```

Hoặc, có thể chỉ định `middleware` trong phương thức khởi tạo của controller. Sử dụng phương thức `middleware` trong phương thức khởi tạo của controller, có thể gán middleware cho các hành động của controller:

```
class UserController extends Controller
{
    /**
     * Instantiate a new controller instance.
     *
     * @return void
     */
    public function __construct()
    {
        $this->middleware('auth');
        $this->middleware('log')->only('index');
        $this->middleware('subscribed')->except('store');
    }
}
```

Controllers cũng cho phép đăng ký middleware bằng cách sử dụng closure. Điều này cung cấp một cách thuận tiện để xác định middleware nội tuyến (inline) cho single controller mà không cần định nghĩa toàn bộ lớp middleware:

```
$this->middleware(function ($request, $next) {
    return $next($request);
});
```

1.4. Resource Controllers

Nếu mỗi Eloquent model trong ứng dụng như một "resource", thì thông thường, nên thực hiện các nhóm hành động giống nhau đối với từng resource trong ứng dụng. Ví dụ: giả sử ứng dụng chứa một model `Photo` và một model `Movie`. Người dùng có thể `create`, `read`, `update` hoặc `delete` các resources này.

Do trường hợp sử dụng phổ biến này, Laravel resource routing chỉ định các route create, read, update và delete ("CRUD") cho controller bằng một dòng mã. Để bắt đầu, sử dụng tùy chọn `-resource` trong câu lệnh `make:controller` Artisan để tạo nhanh controller để xử lý các hành động này:

```
php artisan make:controller PhotoController --resource
```

Lệnh này sẽ tạo một controller tại `app/Http/Controllers/PhotoController.php`. Controller sẽ chứa một phương thức cho mỗi hoạt động resource có sẵn. Tiếp theo, đăng ký một route resource trỏ đến controller:

```
use App\Http\Controllers\PhotoController;
Route::resource('photos', PhotoController::class);
```

Khai báo một single route này tạo ra nhiều routes để xử lý nhiều hành động khác nhau trên resource. Controller được tạo sẽ có sẵn các phương thức cho mỗi hành động này. Có thể xem tổng quan về các route bằng cách dùng lệnh Artisan `route:list`.

Có thể đăng ký nhiều resource controllers cùng một lúc bằng cách truyền một mảng đến phương thức `resources`:

```
Route::resources([
    'photos' => PhotoController::class,
    'posts' => PostController::class,
]);
```

Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	`/photos`	index	photos.index
GET	`/photos/create`	create	photos.create
POST	`/photos`	store	photos.store
GET	`/photos/{photo}`	show	photos.show
GET	`/photos/{photo}/edit`	edit	photos.edit
PUT/PATCH	`/photos/{photo}`	update	photos.update
DELETE	`/photos/{photo}`	destroy	photos.destroy

Customizing Missing Model Behavior

Thông thường, một 404 HTTP response sẽ được tạo nếu implicitly bound resource model không được tìm thấy. Tuy nhiên, có thể tùy chỉnh hành vi này bằng cách gọi phương thức `missing` khi định nghĩa resource route. Phương thức `missing` chấp nhận một closure sẽ được gọi nếu không thể tìm thấy một implicitly bound model cho bất routes nào của resource:

```
use App\Http\Controllers\PhotoController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Redirect;

Route::resource('photos', PhotoController::class)
        ->missing(function (Request $request) {
            return Redirect::route('photos.index');
        });
```

Specifying The Resource Model

Nếu đang sử dụng route model binding và muốn các phương thức của bộ điều khiển tài nguyên type-hint một model instance, sử dụng tùy chọn `--model` khi tạo controller:

```
php artisan make:controller PhotoController --resource --model=Photo
```

Partial Resource Routes

Khi khai báo một resource route, có thể chỉ định một tập hợp con các hành động mà controller sẽ xử lý thay vì tập hợp đầy đủ các hành động mặc định:

```
use App\Http\Controllers\PhotoController;

Route::resource('photos', PhotoController::class)->only([
    'index', 'show'
]);

Route::resource('photos', PhotoController::class)->except([
    'create', 'store', 'update', 'destroy'
]);
```

API Resource Routes

Khi khai báo các resource routes sẽ được sử dụng bởi các API, thông thường sẽ muốn loại trừ các routes trình bày các HTML templates chẳng hạn như `create` và `edit` sửa. Để thuận tiện, có thể sử dụng phương thức `apiResource` để tự động loại trừ hai routes như sau:

```
use App\Http\Controllers\PhotoController;

Route::apiResource('photos', PhotoController::class);
```

Có thể đăng ký nhiều API resource controllers cùng một lúc bằng cách truyền một mảng tới phương thức `apiResources`:

```
use App\Http\Controllers\PhotoController;
use App\Http\Controllers\PostController;

Route::apiResources([
    'photos' => PhotoController::class,
    'posts' => PostController::class,
]);
```

Để nhanh chóng tạo bộ điều khiển tài nguyên API không bao gồm các phương thức tạo hoặc chỉnh sửa, hãy sử dụng công tắc --api khi thực hiện lệnh make: controller:

```
php artisan make:controller PhotoController --api
```

Nested Resources

Đôi khi cần định nghĩa các routes đến một nested resource. Ví dụ: photo resource có thể có nhiều comments có thể được đính kèm vào photo. Để lồng các resource controllers, sử dụng ký hiệu "dấu chấm" trong khai báo route:

```
use App\Http\Controllers\PhotoCommentController;

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

Route này sẽ đăng ký một nested resource có thể được truy cập bằng các URI như sau:

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

Scoping Nested Resources

Tính năng implicit model binding của Laravel có thể tự động điều chỉnh các ràng buộc lồng nhau sao cho child model đã phân giải được xác nhận là thuộc về parent model. Bằng cách sử dụng phương thức `scoped` vi khi định nghĩa nested resource, có thể kích hoạt tính năng tự động xác định phạm vi như là instruct Laravel, trường child resource nên được truy xuất. Để biết thêm thông tin về cách thực hiện điều này, vui lòng xem tài liệu về các scoping resource routes.

Shallow Nesting

Thông thường, không hoàn toàn cần thiết phải có cả ID cha và ID con trong URI vì ID con đã là một unique identifier. Khi sử dụng các unique identifier, chẳng hạn như auto-incrementing primary keys để xác định mô hình trong các phân đoạn URI, có thể chọn sử dụng "shallow nesting":

```
use App\Http\Controllers\CommentController;

Route::resource('photos.comments', CommentController::class)->shallow();
```

Định nghĩa route này sẽ xác định các routes sau:

Verb	URI	Action	Route Name
GET	`/photos/{photo}/comments`	index	photos.comments.index
GET	`/photos/{photo}/comments/create`	create	photos.comments.create
POST	`/photos/{photo}/comments`	store	photos.comments.store
GET	`/comments/{comment}`	show	comments.show
GET	`/comments/{comment}/edit`	edit	comments.edit
PUT/PATCH	`/comments/{comment}`	update	comments.update
DELETE	`/comments/{comment}`	destroy	comments.destroy

Naming Resource Routes

Theo mặc định, tất cả các controller actions đều có tên route; Tuy nhiên, có thể override các tên này bằng cách truyền một mảng `names` với các tên route mong muốn:

```
use App\Http\Controllers\PhotoController;

Route::resource('photos', PhotoController::class)->names([
    'create' => 'photos.build'
]);
```

Naming Resource Route Parameters

Theo mặc định, `Route::resource` sẽ tạo các tham số route cho các resource routes dựa trên phiên bản "singularized" của resource name. Có thể dễ dàng override điều này trên mỗi resource basis bằng cách sử dụng phương thức `parameters`. Mảng được truyền vào phương thức `parameters` phải là một mảng kết hợp của tên resource và tên tham số:

```
use App\Http\Controllers\AdminUserController;

Route::resource('users', AdminUserController::class)->parameters([
    'users' => 'admin_user'
]);
```

Ví dụ trên tạo URI cho route `show` của resource:

```
/users/{admin_user}
```

Scoping Resource Routes

Tính năng scoped implicit model binding của Laravel có thể tự động điều chỉnh các ràng buộc lồng nhau sao cho child model đã phân giải được xác nhận là thuộc về parent model. Bằng cách sử dụng phương thức `scoped` khi định nghĩa nested resource, có thể bật tính năng tự động xác định phạm vi như instruct Laravel, trường child resource sẽ được truy xuất bằng:

```
use App\Http\Controllers\PhotoCommentController;

Route::resource('photos.comments', PhotoCommentController::class)->scoped([
    'comment' => 'slug',
]);
```

Route này sẽ đăng ký một nested resource có phạm vi (scoped) có thể được truy cập bằng các URI như sau:

```
/photos/{photo}/comments/{comment:slug}
```

Khi sử dụng ràng buộc ngầm có khóa tùy chỉnh làm tham số nested route, Laravel sẽ tự động phân phạm vi truy vấn để truy xuất nested model bởi parent của nó bằng cách sử dụng các quy ước để đoán tên mối quan hệ trên parent. Trong trường hợp này, giả sử `Photo` model có một mối quan hệ (relationship) có tên là `comments` (số nhiều của tên tham số route) có thể được sử dụng để truy xuất `Comment` model.

Localizing Resource URIs

Theo mặc định, `Route::resource` sẽ tạo các resource URIs bằng cách sử dụng các động từ. Nếu cần bản địa hóa các động từ hành động `create` và `edit`, có thể sử dụng phương thức `Route::resourceVerbs`. Điều này có thể được thực hiện khi bắt đầu phương thức `boot` trong `App\Providers\RouteServiceProvider`:

```
/**
 * Define your route model bindings, pattern filters, etc.
 *
 * @return void
 */
```

```
public function boot()
{
    Route::resourceVerbs([
        'create' => 'crear',
        'edit' => 'editar',
    ]);

    // ...
}
```

Khi các động từ đã được tùy chỉnh, đăng ký resource route như `Route::resource('fotos', PhotoController::class)` sẽ tạo ra các URI sau:

```
/fotos/crear

/fotos/{foto}/editar
```

Supplementing Resource Controllers

Nếu cần thêm các tuyến bổ sung vào resource controller ngoài tập hợp các resource routes mặc định, định nghĩa các routes đó trước khi gọi phương thức `Route::resource`; nếu không, các tuyến được định nghiac bởi phương thức `resource` có thể vô tình được ưu tiên hơn các tuyến bổ sung:

```
use App\Http\Controller\PhotoController;

Route::get('/photos/popular', [PhotoController::class, 'popular']);
Route::resource('photos', PhotoController::class);
```

1.5. Dependency Injection & Controllers

Constructor Injection

Laravel service container được sử dụng để giải quyết tất cả các Laravel controllers. Do đó, có thể type-hint bất kỳ phụ thuộc nào mà controller có thể cần trong phương thức khởi tạo (constructor) của nó. Các phần phụ thuộc đã khai báo sẽ tự động được giải quyết và đưa vào controller instance:

```
<?php
```

```php
namespace App\Http\Controllers;

use App\Repositories\UserRepository;

class UserController extends Controller
{
    /**
     * The user repository instance.
     */
    protected $users;

    /**
     * Create a new controller instance.
     *
     * @param  \App\Repositories\UserRepository  $users
     * @return void
     */
    public function __construct(UserRepository $users)
    {
        $this->users = $users;
    }
}
```

Method Injection

Ngoài việc chèn hàm tạo, có thể nhập các phụ thuộc type-hint vào các phương thức của v. Một trường hợp sử dụng phổ biến để injection phương thức là đưa instance `Illuminate\Http\Request` vào các phương thức controller:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
```

```
class UserController extends Controller
{
    /**
     * Store a new user.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $name = $request->name;

        //
    }
}
```

Nếu phương thức controller cũng đang mong đợi đầu vào từ một tham số tuyến, hãy liệt kê các đối số tuyến sau các phần phụ thuộc khác. Ví dụ: nếu route được định nghĩa:

```
use App\Http\Controllers\UserController;

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

Vẫn có thể type-hint `Illuminate\Http\Request` và truy cập tham số `id` bằng cách định nghĩa phương thức controller như sau:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
```

```
    /**
     * Update the given user.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  string  $id
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request, $id)
    {
        //
    }
}
```