Course - Laravel Framework

Tài nguyên API

Khi xây dựng một API, bạn có thể cần một tầng chuyển đổi nằm giữa các model Eloquent của bạn và các phản hồi JSON thực sự được trả lại cho người dùng ứng dụng của bạn.

Tags: api resources, tai nguyen api, laravel

Giới thiệu

Khi xây dựng một API, bạn có thể cần một tầng chuyển đổi (tranformer) nằm giữa model Eloquent và phản hồi JSON response thực sự sẽ được trả lại cho người dùng ứng dụng của bạn.

Ví dụ: bạn có thể muốn hiển thị những thuộc tính nào đó cho một tập hợp người dùng con chứ không phải những người khác hoặc bạn có thể muốn luôn đưa vào các mối quan hệ nhất định trong phần trình bày JSON của các model của bạn. Các class tài nguyên của Eloquent cho phép bạn chuyển đổi các model và collection các model của mình thành dữ liệu JSON một cách rõ ràng và dễ dàng.

Tất nhiên, bạn luôn có thể chuyển đổi các model hoặc collection Eloquent sang JSON bằng cách sử dụng các phương thức toJson của chúng; tuy nhiên, tài nguyên Eloquent cung cấp nhiều kỹ thuật chi tiết và mạnh mẽ hơn đối với việc mã hóa serialize JSON của các model của bạn và các mối quan hệ của chúng.

Tạo ra tài nguyên

Để tạo một class tài nguyên, bạn có thể sử dụng lệnh Artisan make:resource. Theo mặc định, tài nguyên sẽ được đặt trong thư mục `app/Http/Resources` của ứng dụng của bạn. Tài nguyên sẽ mở rộng class `Illuminate\Http\Resources\Json\JsonResource`:

```
php artisan make:resource UserResource
```

Collection các tài nguyên (resource collections)

Ngoài việc tạo tài nguyên nhằm biến đổi (transform) các model riêng lẻ, bạn có thể tạo tài nguyên để chịu trách nhiệm biến đổi (transform) collection các model. Điều này cho phép các phản hồi JSON response của bạn chứa thêm các liên kết và thông tin meta khác có liên quan đến toàn bộ collection của một tài nguyên nào đó.

Để tạo một collection tài nguyên, bạn nên sử dụng thêm tùy chọn `--collection` khi tạo tài nguyên. Hoặc, đưa thêm từ `Collection` vào trong tên tài nguyên để cho Laravel biết rằng nó nên tạo một tài nguyên là collection. Tài nguyên collection này mở rộng từ class `Illuminate\Http\Resources\Json\ResourceCollection`:

```
php artisan make:resource User --collection
```

```
php artisan make:resource UserCollection
```

Khái nhiệm tổng quát

> Đây là tổng quan cấp cao về tài nguyên và collection tài nguyên. Chúng tôi rất khuyến khích bạn đọc các phần khác của tài liệu này để hiểu sâu hơn về khả năng tùy chỉnh và sức mạnh mà các tài nguyên cung cấp cho bạn.

Trước khi đi sâu vào tất cả các tùy chọn có sẵn cho bạn khi viết tài nguyên, trước tiên chúng ta hãy xem xét phần cấp cao về cách tài nguyên được sử dụng trong Laravel. Một class tài nguyên sẽ đại diện cho một model đơn lẽ nào đó cần được biến đổi (transform) thành cấu trúc JSON. Ví dụ, đây là một class tài nguyên `UserResource` đơn giản:

```php
<?php

namespace App\Http\Resources;
use Illuminate\Http\Resources\Json\JsonResource;

class UserResource extends JsonResource
{
  /**
   * Transform the resource into an array.
   *
   * @param  \Illuminate\Http\Request  $request
   * @return array
   */
  public function toArray($request)
  {
    return [
      'id' => $this->id,
      'name' => $this->name,
      'email' => $this->email,
      'created_at' => $this->created_at,
      'updated_at' => $this->updated_at,
    ];
  }
}
```

Mọi class tài nguyên sẽ khai báo qua một phương thức `toArray` để trả về mảng thuộc tính sẽ được biến đổi thành dữ liệu JSON khi tài nguyên được trả về dưới dạng phản hồi

response từ phương pháp route hoặc controller.

Lưu ý rằng chúng ta có thể truy cập các thuộc tính của model trực tiếp từ biến `$this`. Điều này được cho là do một lớp tài nguyên sẽ tự động truy cập thuộc tính và phương thức proxy xuống model bên dưới để truy cập. Sau khi tài nguyên được khai báo, nó có thể được trả về từ một route hoặc collection. Tài nguyên chấp nhận thể hiện model cơ bản thông qua constructor của nó:

```
use App\Http\Resources\UserResource;
use App\Models\User;

Route::get('/user/{id}', function ($id) {
  return new UserResource(User::findOrFail($id));
});
```

Collections tài nguyên

Nếu bạn đang trả về một tập hợp các tài nguyên hoặc một phản hồi có phân trang, bạn nên sử dụng phương thức collection được cung cấp bởi class tài nguyên của bạn khi tạo đối tượng tài nguyên trong route hoặc controller của bạn:

```
use App\Http\Resources\UserResource;
use App\Models\User;

Route::get('/users', function () {
  return UserResource::collection(User::all());
});
```

Lưu ý rằng điều này không cho phép bổ sung bất kỳ dữ liệu meta tùy chỉnh nào có thể cần được trả lại cùng với bộ sưu tập của bạn. Nếu bạn muốn tùy chỉnh phản hồi thu thập tài nguyên, bạn có thể tạo một tài nguyên chuyên dụng để đại diện cho tập hợp:

```
php artisan make:resource UserCollection
```

Khi lớp thu thập tài nguyên đã được tạo, bạn có thể dễ dàng xác định bất kỳ siêu dữ liệu nào cần được đưa vào phản hồi:

```php
<?php

namespace App\Http\Resources;
use Illuminate\Http\Resources\Json\ResourceCollection;

class UserCollection extends ResourceCollection
{
  /**
   * Transform the resource collection into an array.
   *
   * @param  \Illuminate\Http\Request  $request
   * @return array
   */
  public function toArray($request)
  {
    return [
      'data' => $this->collection,
      'links' => [
        'self' => 'link-value',
      ],
    ];
  }
}
```

Sau khi khai báo collection tài nguyên, nó có thể được trả về từ một route hoặc collection:

```php
use App\Http\Resources\UserCollection;
use App\Models\User;

Route::get('/users', function () {
  return new UserCollection(User::all());
});
```

Bảo lưu khóa collection

Khi trả về một collection tài nguyên từ một route, Laravel sẽ đặt lại các khóa của collection

để chúng theo thứ tự số đếm. Tuy nhiên, bạn có thể thêm thuộc tính `preserveKeys` vào class tài nguyên của mình để cho các khóa gốc của một collection tài nguyên biết liệu có nên được giữ nguyên hay không:

```php
<?php

namespace App\Http\Resources;
use Illuminate\Http\Resources\Json\JsonResource;

class UserResource extends JsonResource
{
  /**
   * Indicates if the resource's collection keys should be preserved.
   *
   * @var bool
   */
  public $preserveKeys = true;
}
```

Khi thuộc tính `preserveKeys` được đặt thành `true`, các khóa bộ sưu tập sẽ được giữ nguyên khi bộ sưu tập được trả về từ một tuyến đường hoặc bộ điều khiển:

```php
use App\Http\Resources\UserResource;
use App\Models\User;

Route::get('/users', function () {
  return UserResource::collection(User::all()->keyBy->id);
});
```

Tùy chỉnh class tài nguyên bên dưới

Thông thường, thuộc tính `$this->collection` của collection tài nguyên được tự động thao tác với kết quả của việc bố trí từng mục của tập hợp với class tài nguyên đơn lẻ của nó. Class tài nguyên đơn lẻ được giả định là tên class của collection mà không có phần `Collection` theo sau của tên class. Ngoài ra, tùy thuộc vào sở thích cá nhân của bạn, class tài nguyên đơn lẽ có thể có hoặc không có hậu tố với `Resource`.

Ví dụ: `UserCollection` sẽ cố gắng bố trí các đối tượng người dùng đã cho vào tài nguyên `UserResource`. Để điều chỉnh hành vi này, bạn có thể ghi đè thuộc tính `$collects` của collection tài nguyên của mình:

```php
<?php

namespace App\Http\Resources;
use Illuminate\Http\Resources\Json\ResourceCollection;

class UserCollection extends ResourceCollection
{
  /**
   * The resource that this resource collects.
   *
   * @var string
   */
  public $collects = Member::class;
}
```

Viết tài nguyên

> Nếu bạn chưa đọc khái niệm tổng quát, bạn rất nên đọc nó trước khi tiếp tục với tài liệu này.

Về bản chất, tài nguyên rất đơn giản. Chúng chỉ cần biến đổi một model đã cho thành một mảng. Vì vậy, mỗi tài nguyên chứa một phương thức `toArray` để chuyển các thuộc tính của model của bạn thành một mảng thân thiện với API có thể được trả về từ các route hoặc controller của ứng dụng của bạn:

```php
<?php

namespace App\Http\Resources;
use Illuminate\Http\Resources\Json\JsonResource;

class UserResource extends JsonResource
{
```

```
  /**
   * Transform the resource into an array.
   *
   * @param  \Illuminate\Http\Request  $request
   * @return array
   */
  public function toArray($request)
  {
    return [
      'id' => $this->id,
      'name' => $this->name,
      'email' => $this->email,
      'created_at' => $this->created_at,
      'updated_at' => $this->updated_at,
    ];
  }
}
```

Khi tài nguyên đã được khai báo, thì nó có thể được trả về trực tiếp từ một route hoặc controller:

```
use App\Http\Resources\UserResource;
use App\Models\User;

Route::get('/user/{id}', function ($id) {
  return new UserResource(User::findOrFail($id));
});
```

Các mối quan hệ

Nếu bạn muốn đưa thêm các tài nguyên liên quan vào trong phản hồi response của mình, thì bạn có thể thêm chúng vào mảng được trả về bởi phương thức `toArray` của tài nguyên của bạn. Trong ví dụ này, chúng ta sẽ sử dụng phương thức `collection` của tài nguyên `PostResource` để thêm các bài đăng trên blog của người dùng vào *phản hồi tài nguyên* (resource response):

```php
use App\Http\Resources\PostResource;


/**
 * Transform the resource into an array.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return array
 */
public function toArray($request)
{
  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,
  ];
}
```

> Nếu bạn chỉ muốn đưa các mối quan hệ vào khi chúng đã được tải, thì hãy xem tài liệu về *mối quan hệ có điều kiện* (conditional relationships).

Các collection tài nguyên

Trong khi các tài nguyên biến đổi một model đơn lẻ thành một mảng, thì các collection tài nguyên sẽ biến đổi một collection các model thành một mảng. Tuy nhiên, không nhất thiết phải tạo class collection tài nguyên cho từng model của bạn vì tất cả các tài nguyên đều cung cấp phương thức `collection` để tạo tập hợp tài nguyên "đặc biệt" một cách nhanh chóng:

```php
use App\Http\Resources\UserResource;
use App\Models\User;

Route::get('/users', function () {
  return UserResource::collection(User::all());
});
```

Tuy nhiên, nếu bạn cần tùy chỉnh dữ liệu meta được trả về với collection, thì cần phải xác định collection tài nguyên của riêng bạn:

```php
<?php

namespace App\Http\Resources;
use Illuminate\Http\Resources\Json\ResourceCollection;

class UserCollection extends ResourceCollection
{
  /**
   * Transform the resource collection into an array.
   *
   * @param  \Illuminate\Http\Request  $request
   * @return array
   */
  public function toArray($request)
  {
    return [
      'data' => $this->collection,
      'links' => [
        'self' => 'link-value',
      ],
    ];
  }
}
```

Giống như các tài nguyên đơn lẻ, collection tài nguyên có thể được trả lại trực tiếp từ các

route hoặc controller:

```
use App\Http\Resources\UserCollection;
use App\Models\User;

Route::get('/users', function () {
  return new UserCollection(User::all());
});
```

Gói bọc dữ liệu

Theo mặc định, tài nguyên ngoài cùng của bạn được gói bọc trong một khóa `data` khi phản hồi response của tài nguyên được chuyển đổi thành JSON. Vì vậy, ví dụ, một phản hồi thu thập tài nguyên điển hình trông giống như sau:

```
{
  "data": [
    {
      "id": 1,
      "name": "Eladio Schroeder Sr.",
      "email": "therese28@example.com",
    },
    {
      "id": 2,
      "name": "Liliana Mayert",
      "email": "evandervort@example.com",
    }
  ]
}
```

Nếu bạn muốn sử dụng khóa tự tạo thay vì `data`, thì bạn có thể khai báo thuộc tính attribute `$wrap` trên class tài nguyên:

```
<?php

namespace App\Http\Resources;
```

```php
use Illuminate\Http\Resources\Json\JsonResource;

class UserResource extends JsonResource
{
  /**
   * The "data" wrapper that should be applied.
   *
   * @var string
   */
  public static $wrap = 'user';
}
```

Nếu bạn muốn vô hiệu hóa việc bọc gói tài nguyên ngoài cùng, bạn nên gọi phương thức `withoutWrapping` trên class cơ sở `Illuminate\Http\Resources\Json\JsonResource`. Thông thường, bạn nên gọi phương thức này từ `AppServiceProvider` hoặc một nhà cung cấp dịch vụ khác được tải theo mọi yêu cầu đối với ứng dụng của bạn:

```php
<?php

namespace App\Providers;
use Illuminate\Http\Resources\Json\JsonResource;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
  /**
   * Register any application services.
   *
   * @return void
   */
  public function register()
  {
    //
  }

  /**
```

```
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        JsonResource::withoutWrapping();
    }
}
```

> **Chú ý:** Phương thức `withoutWrapping` chỉ ảnh hưởng đến phản hồi ngoài cùng và sẽ không xóa các khóa `data` mà bạn thêm theo cách thủ công vào collection tài nguyên của riêng mình.

Gói bọc các tài nguyên được lồng vào nhau

Bạn có toàn quyền tự do để xác định cách mà các mối quan hệ của tài nguyên của bạn được gói bọc. Nếu bạn muốn tất cả các collection tài nguyên được gói bọc trong từ khóa `data`, bất kể chúng lồng vào nhau như thế nào, bạn nên tạo một class collection tài nguyên cho mỗi tài nguyên và trả về collection trong từ khóa `data`.

Bạn có thể tự hỏi liệu điều này có khiến tài nguyên ngoài cùng của bạn bị gói bọc trong hai khóa `data` hay không. Đừng lo lắng, Laravel sẽ không bao giờ để tài nguyên của bạn vô tình bị gói bọc gấp đôi lần lên, vì vậy bạn không cần phải lo lắng về cấp độ lồng ghép của collection tài nguyên mà bạn đang biến đổi:

```
<?php

namespace App\Http\Resources;
use Illuminate\Http\Resources\Json\ResourceCollection;

class CommentsCollection extends ResourceCollection
{
    /**
     * Transform the resource collection into an array.
     *
     * @param  \Illuminate\Http\Request  $request
```

```
     * @return array
     */
    public function toArray($request)
    {
        return ['data' => $this->collection];
    }
}
```

Gói bọc dữ liệu và phân trang

Khi trả về các collection được phân trang thông qua phản hồi response của tài nguyên, Laravel sẽ gói bọc dữ liệu tài nguyên của bạn trong từ khóa `data` ngay cả khi phương thức `withoutWrapping` đã được gọi. Điều này là do các câu trả lời được phân trang luôn chứa các khóa `meta` và `links` cùng với thông tin về trạng thái phân trang:

```
{
    "data": [
        {
            "id": 1,
            "name": "Eladio Schroeder Sr.",
            "email": "therese28@example.com",
        },
        {
            "id": 2,
            "name": "Liliana Mayert",
            "email": "evandervort@example.com",
        }
    ],
    "links":{
        "first": "http://example.com/pagination?page=1",
        "last": "http://example.com/pagination?page=1",
        "prev": null,
        "next": null
    },
    "meta":{
        "current_page": 1,
        "from": 1,
```

```
    "last_page": 1,
    "path": "http://example.com/pagination",
    "per_page": 15,
    "to": 10,
    "total": 10
  }
}
```

Phân trang

Bạn có thể truyền một đối tượng phân trang Laravel tới phương thức `collection` của một tài nguyên hoặc vào một collection tài nguyên tự tạo:

```
use App\Http\Resources\UserCollection;
use App\Models\User;

Route::get('/users', function () {
  return new UserCollection(User::paginate());
});
```

Các câu trả lời được phân trang luôn chứa các khóa `meta` và `links` với thông tin về trạng thái của trình phân trang:

```
{
  "data": [
    {
      "id": 1,
      "name": "Eladio Schroeder Sr.",
      "email": "therese28@example.com",
    },
    {
      "id": 2,
      "name": "Liliana Mayert",
      "email": "evandervort@example.com",
    }
  ],
```

```
  "links":{
    "first": "http://example.com/pagination?page=1",
    "last": "http://example.com/pagination?page=1",
    "prev": null,
    "next": null
  },
  "meta":{
    "current_page": 1,
    "from": 1,
    "last_page": 1,
    "path": "http://example.com/pagination",
    "per_page": 15,
    "to": 10,
    "total": 10
  }
}
```

Các thuộc tính điều kiện

Đôi khi bạn có thể muốn chỉ gói bọc một thuộc tính attribute trong phản hồi tài nguyên nếu một điều kiện nào đó được đáp ứng. Ví dụ: bạn có thể chỉ muốn đưa một giá trị vào nếu người dùng hiện tại là "quản trị viên" (administrator). Laravel sẽ cho nhiều phương pháp trợ giúp khác nhau để hỗ trợ bạn trong tình huống này. Phương thức `when` có thể được sử dụng để thêm thuộc tính attribute có điều kiện vào phản hồi tài nguyên:

```
use Illuminate\Support\Facades\Auth;


/**
 * Transform the resource into an array.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return array
 */
public function toArray($request)
{
  return [
    'id' => $this->id,
```

```
    'name' => $this->name,
    'email' => $this->email,
    'secret' => $this->when(Auth::user()->isAdmin(), 'secret-value'),
    'created_at' => $this->created_at,
    'updated_at' => $this->updated_at,
  ];
}
```

Trong ví dụ này, khóa `secret` sẽ chỉ được trả lại trong phản hồi tài nguyên cuối cùng nếu phương thức `isAdmin` của người dùng đã xác thực trả về `true`. Nếu phương thức trả về `false`, khóa `recret` sẽ bị xóa khỏi phản hồi tài nguyên trước khi nó được gửi đến máy khách. Phương thức `when` cho phép bạn khai báo rõ ràng các tài nguyên của mình mà không cần dùng đến các câu lệnh điều kiện khi xây dựng mảng.

Phương thức `when` cũng chấp nhận một hàm xử lý làm đối số thứ hai của nó, cho phép bạn chỉ tính toán giá trị kết quả nếu điều kiện đã cho là `true`:

```
'secret' => $this->when(Auth::user()->isAdmin(), function () {
  return 'secret-value';
}),
```

Hợp nhất các thuộc tính cho điều kiện

Đôi khi bạn có thể có một số thuộc tính attribute chỉ nên được đưa vào phản hồi tài nguyên dựa trên cùng một điều kiện. Trong trường hợp này, bạn có thể sử dụng phương thức `mergeWhen` để chỉ bao gồm các thuộc tính attribute trong phản hồi khi điều kiện đã cho là `true`:

```
/**
 * Transform the resource into an array.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return array
 */
public function toArray($request)
{
  return [
```

```
            'id' => $this->id,
            'name' => $this->name,
            'email' => $this->email,
            $this->mergeWhen(Auth::user()->isAdmin(), [
                'first-secret' => 'value',
                'second-secret' => 'value',
            ]),
            'created_at' => $this->created_at,
            'updated_at' => $this->updated_at,
        ];
    }
```

Một lần nữa, nếu điều kiện đã cho là `false`, thì các thuộc tính attribute này sẽ bị xóa khỏi phản hồi tài nguyên trước khi nó được gửi đến máy khách.

> **Chú ý:** Phương thức `mergeWhen` không nên được sử dụng trong các mảng kết hợp các khóa chuỗi và số đếm. Hơn nữa, nó không nên được sử dụng trong các mảng có các khóa số đếm không được sắp xếp theo thứ tự.

Các mối quan hệ có điều kiện

Ngoài việc tải có điều kiện các thuộc tính attribute, thì bạn có thể đưa vào các mối quan hệ có điều kiện trên các phản hồi tài nguyên của mình dựa trên việc mối quan hệ đã được tải trên model hay chưa. Điều này cho phép controller của bạn quyết định mối quan hệ nào nên được tải trên model và tài nguyên của bạn có thể dễ dàng lấy chúng chỉ khi chúng thực sự đã được tải. Cuối cùng, điều này giúp bạn dễ dàng tránh các sự cố truy vấn "N + 1" trong tài nguyên của mình.

Phương thức `whenLoaded` có thể được sử dụng để tải một mối quan hệ có điều kiện. Để tránh tải các mối quan hệ một cách không cần thiết, phương thức này chấp nhận tên của mối quan hệ thay vì chính mối quan hệ:

```
use App\Http\Resources\PostResource;

/**
 * Transform the resource into an array.
 *
 * @param  \Illuminate\Http\Request  $request
```

```
     * @return array
     */
    public function toArray($request)
    {
      return [
        'id' => $this->id,
        'name' => $this->name,
        'email' => $this->email,
        'posts' => PostResource::collection($this->whenLoaded('posts')),
        'created_at' => $this->created_at,
        'updated_at' => $this->updated_at,
      ];
    }
```

Trong ví dụ này, nếu mối quan hệ chưa được tải, khóa `posts` sẽ bị xóa khỏi phản hồi tài nguyên trước khi nó được gửi đến máy khách.

Thông tin *trục trung gian (pivot)* có điều kiện

Ngoài việc lấy vào có điều kiện thông tin mối quan hệ trong phản hồi tài nguyên của bạn, bạn có thể lấy thêm dữ liệu từ các bảng trung gian của nhiều mối quan hệ một cách có điều kiện bằng cách sử dụng phương thức `whenPivotLoaded`. Phương thức `whenPivotLoaded` chấp nhận tên của bảng tổng hợp làm đối số đầu tiên của nó. Đối số thứ hai phải là một hàm xử lý trả về giá trị được trả về nếu thông tin trục có sẵn trên mô hình:

```
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return array
     */
    public function toArray($request)
    {
      return [
        'id' => $this->id,
        'name' => $this->name,
```

```
    'expires_at' => $this->whenPivotLoaded('role_user', function () {
      return $this->pivot->expires_at;
    }),
  ];
}
```

Nếu mối quan hệ của bạn đang sử dụng *model bảng trung gian tùy chỉnh*, thì bạn có thể truyền một đối tượng của model bảng trung gian làm đối số đầu tiên cho phương thức `whenPivotLoaded`:

```
'expires_at' => $this->whenPivotLoaded(new Membership, function () {
  return $this->pivot->expires_at;
}),
```

Nếu bảng trung gian của bạn đang sử dụng một *công cụ truy cập* (accessor) không phải là `pivot`, thì bạn có thể sử dụng phương thức `whenPivotLoadedAs`:

```
/**
 * Transform the resource into an array.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return array
 */
public function toArray($request)
{
  return [
    'id' => $this->id,
    'name' => $this->name,
    'expires_at' => $this->whenPivotLoadedAs('subscription', 'role_user', function () {
      return $this->subscription->expires_at;
    }),
  ];
}
```

Thêm vào dữ liệu meta

Một số tiêu chuẩn API JSON yêu cầu bổ sung dữ liệu meta vào các phản hồi response của collection tài nguyên và tài nguyên của bạn. Điều này thường bao gồm những thứ như `links` đến tài nguyên hoặc các tài nguyên liên quan hoặc dữ liệu meta về chính tài nguyên đó. Nếu bạn cần trả về dữ liệu meta bổ sung về một tài nguyên, hãy đưa nó vào phương thức `toArray` của bạn. Ví dụ: bạn có thể lấy thêm thông tin `link` khi biến đổi collection tài nguyên:

```
/**
 * Transform the resource into an array.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return array
 */
public function toArray($request)
{
  return [
    'data' => $this->collection,
    'links' => [
      'self' => 'link-value',
    ],
  ];
}
```

Khi trả lại dữ liệu meta bổ sung từ các tài nguyên của bạn, bạn sẽ không bao giờ phải lo lắng về việc vô tình ghi đè các khóa `links` hoặc `meta` được Laravel tự động thêm vào khi trả lại các phản hồi được phân trang. Bất kỳ `links` bổ sung nào bạn xác định sẽ được hợp nhất với các liên kết được cung cấp bởi chức năng phân trang.

Cấp độ top của dữ liệu meta

Đôi khi bạn có thể muốn chỉ đưa vào một số dữ liệu meta nào đó với phản hồi tài nguyên nếu tài nguyên đó là tài nguyên ngoài cùng được trả về. Thông thường, điều này bao gồm thông tin meta về toàn bộ phản hồi. Để tạo dữ liệu meta này, hãy thêm phương thức `with` vào class tài nguyên của bạn. Phương thức này sẽ trả về một mảng dữ liệu meta được đưa vào phản hồi tài nguyên chỉ khi tài nguyên là tài nguyên ngoài cùng được biến đổi:

```
<?php
```

```php
namespace App\Http\Resources;
use Illuminate\Http\Resources\Json\ResourceCollection;

class UserCollection extends ResourceCollection
{
  /**
   * Transform the resource collection into an array.
   *
   * @param  \Illuminate\Http\Request  $request
   * @return array
   */
  public function toArray($request)
  {
    return parent::toArray($request);
  }

  /**
   * Get additional data that should be returned with the resource array.
   *
   * @param  \Illuminate\Http\Request  $request
   * @return array
   */
  public function with($request)
  {
    return [
      'meta' => [
        'key' => 'value',
      ],
    ];
  }
}
```

Thêm vào dữ liệu meta khi cấu trúc tài nguyên

Bạn cũng có thể thêm dữ liệu cấp cao nhất khi xây dựng các đối tượng tài nguyên trong route hoặc controller của mình. Phương thức `additional`, có sẵn trên tất cả các tài nguyên, chấp nhận một mảng dữ liệu cần được thêm vào phản hồi tài nguyên:

```
return (new UserCollection(User::all()->load('roles')))
  ->additional(['meta' => [
    'key' => 'value',
  ]]);
```

Các phản hồi tài nguyên

Như bạn đã đọc qua rồi, các tài nguyên có thể được trả lại trực tiếp từ các route và controller:

```
use App\Http\Resources\UserResource;
use App\Models\User;

Route::get('/user/{id}', function ($id) {
  return new UserResource(User::findOrFail($id));
});
```

Tuy nhiên, đôi khi bạn có thể cần phải tùy chỉnh phản hồi HTTP response sẽ gửi đi trước khi nó được gửi đến máy khách. Có hai cách để thực hiện điều này. Đầu tiên, bạn có thể xâu chuỗi phương thức `response` vào tài nguyên. Phương thức này sẽ trả lại một đối tượng `Illuminate\Http\JsonResponse`, cho phép bạn toàn quyền kiểm soát các tiêu đề header của phản hồi:

```
use App\Http\Resources\UserResource;
use App\Models\User;

Route::get('/user', function () {
  return (new UserResource(User::find(1)))
    ->response()
    ->header('X-Value', 'True');
});
```

Ngoài ra, bạn có thể khai báo phương thức `withResponse` trong chính tài nguyên đó. Phương thức này sẽ được gọi khi tài nguyên được trả về dưới dạng tài nguyên ngoài cùng trong một phản hồi response:

```php
<?php

namespace App\Http\Resources;
use Illuminate\Http\Resources\Json\JsonResource;

class UserResource extends JsonResource
{
  /**
   * Transform the resource into an array.
   *
   * @param  \Illuminate\Http\Request  $request
   * @return array
   */
  public function toArray($request)
  {
    return [
      'id' => $this->id,
    ];
  }

  /**
   * Customize the outgoing response for the resource.
   *
   * @param  \Illuminate\Http\Request  $request
   * @param  \Illuminate\Http\Response  $response
   * @return void
   */
  public function withResponse($request, $response)
  {
    $response->header('X-Value', 'True');
  }
}
```