Course - Laravel Framework

Phân quyền truy cập

Ngoài việc luôn cung cấp các service xác thực, Laravel cũng cung cấp một cách đơn giản để phân quyền cho các hành động của người dùng đối với một tài nguyên nhất định.

Tags: phan quyen truy cap, laravel 8

Giới thiệu

Ngoài việc luôn cung cấp các service xác thực người dùng, Laravel cũng cung cấp một cách đơn giản để phân quyền cho các hành động của người dùng đối với một tài nguyên nhất định. Ví dụ: ngay cả khi người dùng được xác thực, họ có thể không được phép cập nhật hoặc xóa một số model Eloquent hoặc record cơ sở dữ liệu do ứng dụng của bạn quản lý. Các tính năng ủy quyền của Laravel cung cấp một cách dễ dàng, có tổ chức để quản lý các loại kiểm tra phân quyền này.

Laravel cung cấp hai cách chính để ủy quyền hành động: gates và chính sách. Hãy nghĩ về các gates và chính sách như các route và controller. Gates cung cấp một cách tiếp cận đơn giản, dựa trên hàm xử lý để phân quyền trong khi các chính sách nhìn như những controller, logic nhóm xung quanh một model hoặc tài nguyên cụ thể. Trong tài liệu này, trước tiên chúng ta sẽ khám phá các gates và sau đó xem xét các chính sách.

Bạn không cần phải lựa chọn giữa việc sử dụng độc quyền các gates hoặc sử dụng riêng các chính sách khi xây dựng một ứng dụng. Hầu hết các ứng dụng rất có thể sẽ chứa một số hỗn hợp các gates và chính sách, và điều đó hoàn toàn ổn! Gates áp dụng nhiều nhất cho các hành động không liên quan đến bất kỳ model hoặc tài nguyên nào, chẳng hạn như xem dashboard dành cho quản trị viên. Ngược lại, các chính sách nên được sử dụng khi bạn muốn phân quyền một hành động cho một model hoặc tài nguyên cụ thể.

Gates

Viết gates

> **Chú ý:** Cổng là một cách tuyệt vời để tìm hiểu những điều cơ bản về các tính năng ủy quyền của Laravel; tuy nhiên, khi xây dựng các ứng dụng Laravel mạnh mẽ, bạn nên cân nhắc sử dụng các chính sách để tổ chức các quy tắc phân quyền của mình.

Các cổng chỉ đơn giản là các chốt xác định xem người dùng có được phép thực hiện một hành động nhất định hay không. Thông thường, các cổng được định nghĩa trong phương thức `boot` của class `App\Providers\AuthServiceProvider` bằng cách sử dụng mặt tiền Gates. Gates luôn nhận một đối tượng người dùng làm đối số đầu tiên của họ và có thể tùy chọn nhận các đối số bổ sung, chẳng hạn như một model Eloquent có liên quan.

Trong ví dụ này, chúng tôi sẽ xác định một cổng để xác định xem người dùng có thể cập nhật một mô hình `App\Models\Post` nhất định hay không. Gates sẽ thực hiện điều này bằng cách so sánh `id` của người dùng với `user_id` của người dùng đã tạo bài đăng:

```php
use App\Models\Post;
use App\Models\User;
use Illuminate\Support\Facades\Gate;

/**
 * Register any authentication / authorization services.
 *
 * @return void
 */
public function boot()
{
  $this->registerPolicies();

  Gate::define('update-post', function (User $user, Post $post) {
    return $user->id === $post->user_id;
  });
}
```

Giống như controller, các gates cũng có thể được xác định bằng cách sử dụng mảng class callback:

```php
use App\Policies\PostPolicy;
use Illuminate\Support\Facades\Gate;

/**
 * Register any authentication / authorization services.
 *
 * @return void
 */
public function boot()
{
  $this->registerPolicies();

  Gate::define('update-post', [PostPolicy::class, 'update']);
}
```

Phân quyền các action

Để phân quyền cho một action bằng cách sử dụng các gates, bạn nên sử dụng các phương thức allows hoặc denies được cung cấp bởi facade Gate. Lưu ý rằng bạn không bắt buộc phải truyền người dùng hiện đã được xác thực đến các phương thức này. Laravel sẽ tự động xử lý việc đưa người dùng vào hàm xử lý gate. Thông thường, bạn nên gọi các phương thức phân quyền gate trong controller của ứng dụng trước khi thực hiện một action mà có yêu cầu phân quyền:

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\Post;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Gate;

class PostController extends Controller
{
  /**
   * Update the given post.
   *
   * @param  \Illuminate\Http\Request  $request
   * @param  \App\Models\Post  $post
   * @return \Illuminate\Http\Response
   */
  public function update(Request $request, Post $post)
  {
    if (! Gate::allows('update-post', $post)) {
      abort(403);
    }

    // Update the post...
  }
}
```

Nếu bạn muốn xác định xem một người dùng khác với người dùng hiện đã được xác thực

có được phân quyền để thực hiện một action hay không, thì bạn có thể sử dụng phương thức `forUser` trên facade `Gate`:

```
if (Gate::forUser($user)->allows('update-post', $post)) {
  // The user can update the post...
}

if (Gate::forUser($user)->denies('update-post', $post)) {
  // The user can't update the post...
}
```

Cho phép hoặc bỏ qua các trường hợp ngoại lệ

Nếu bạn muốn cố gắng phân quyền một hành động và tự động ném một `Illuminate\Auth\Access\AuthorizationException` nếu người dùng không được phép thực hiện action đã cho, bạn có thể sử dụng phương thức `authorize` của facade `Gate`. Các đối tượng của `AuthorizationException` sẽ được tự động chuyển đổi thành phản hồi HTTP 403 bởi trình xử lý exception của Laravel:

```
Gate::authorize('update-post', $post);

// The action is authorized...
```

Các trường hợp bổ sung

Các phương thức gate cho phép khả năng (allows, denies, check, any, none, authorize, can, cannot) và các chỉ thị phân quyền Blade (@can, @cannot, @canany) có thể nhận một mảng làm đối số thứ hai của chúng. Các phần tử mảng này được truyền dưới dạng tham số cho quá trình hàm xử lý gate và có thể được sử dụng cho trường hợp bổ sung khi đưa ra quyết định phân quyền:

```
use App\Models\Category;
use App\Models\User;
use Illuminate\Support\Facades\Gate;

Gate::define('create-post', function (User $user, Category $category, $pinned) {
```

```
  if (! $user->canPublishToGroup($category->group)) {
    return false;
  } elseif ($pinned && ! $user->canPinPosts()) {
    return false;
  }

  return true;
});

if (Gate::check('create-post', [$category, $pinned])) {
  // The user can create the post...
}
```

Phản hồi gate

Cho đến nay, chúng ta mới chỉ kiểm tra các gate trả về giá trị boolean đơn giản. Tuy nhiên, đôi khi bạn có thể muốn trả lại phản hồi chi tiết hơn, bao gồm cả thông báo lỗi. Để làm như vậy, bạn có thể trả về `Illuminate\Auth\Access\Response` từ gate của bạn:

```
use App\Models\User;
use Illuminate\Auth\Access\Response;
use Illuminate\Support\Facades\Gate;

Gate::define('edit-settings', function (User $user) {
  return $user->isAdmin
    ? Response::allow()
    : Response::deny('You must be an administrator.');
});
```

Ngay cả khi bạn trả lại một phản hồi phân quyền từ gate của mình, phương thức `Gate::allows` vẫn sẽ trả về một giá trị `boolean` đơn giản; tuy nhiên, bạn có thể sử dụng phương thức `Gate::inspect` để nhận được phản hồi phân quyền đầy đủ do gate trả về:

```
$response = Gate::inspect('edit-settings');

if ($response->allowed()) {
```

```
    // The action is authorized...
} else {
    echo $response->message();
}
```

Khi sử dụng phương thức `Gate::authorize`, phương thức này sẽ đưa ra một `AuthorizationException` nếu action không được phân quyền, thông báo lỗi được cung cấp bởi phản hồi phân quyền sẽ được truyền tới phản hồi HTTP:

```
Gate::authorize('edit-settings');

// The action is authorized...
```

Kiểm tra gate chặn

Đôi khi, bạn có thể muốn cấp tất cả các khả năng cho một người dùng cụ thể. Bạn có thể sử dụng phương thức `before` để khai báo một hàm xử lý được chạy trước tất cả các lần kiểm tra phân quyền khác:

```
use Illuminate\Support\Facades\Gate;

Gate::before(function ($user, $ability) {
    if ($user->isAdministrator()) {
        return true;
    }
});
```

Nếu hàm `before` trả về một kết quả khác rỗng thì kết quả đó sẽ được coi là kết quả của việc kiểm tra phân quyền.

Bạn có thể sử dụng phương thức `after` để xác định một hàm xử lý sẽ được thực thi sau tất cả các lần kiểm tra phân quyền khác:

```
Gate::after(function ($user, $ability, $result, $arguments) {
    if ($user->isAdministrator()) {
        return true;
```

```
    }
});
```

Tương tự như phương thức `before`, nếu sau khi hàm xử lý trả về một kết quả khác rỗng thì kết quả đó sẽ được coi là kết quả của việc kiểm tra phân quyền.

Phân quyền trực tiếp

Đôi khi, bạn có thể muốn xác định xem người dùng hiện đã được xác thực có được phép thực hiện một action nào đó hay không mà không cần viết một gate chuyên dụng tương ứng với action đó. Laravel cho phép bạn thực hiện các loại kiểm tra phân quyền này thông qua các phương thức `Gate::allowIf` và `Gate::denyIf`:

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

Gate::allowIf(fn ($user) => $user->isAdministrator());

Gate::denyIf(fn ($user) => $user->banned());
```

Nếu action không được phân quyền hoặc nếu không có người dùng nào hiện được xác thực, Laravel sẽ tự động đưa ra exception `Illuminate\Auth\Access\AuthorizationException`. Các đối tượng của `AuthorizationException` được tự động truyền đổi thành phản hồi HTTP 403 bởi trình xử lý exception của Laravel.

Tạo chính sách

Tạo chính sách

Chính sách là các class tổ chức logic phân quyền xung quanh một model cụ thể hoặc tài nguyên cụ thể. Ví dụ: nếu ứng dụng của bạn là blog, bạn có thể có model `App\Models\Post` và một `App\Policies\PostPolicy` tương ứng để cho phép người dùng thực hiện các action như tạo hoặc cập nhật bài đăng.

Bạn có thể tạo một chính sách bằng lệnh Artisan `make:policy`. Chính sách đã tạo sẽ được đặt trong thư mục `app/Policies`. Nếu thư mục này không tồn tại trong ứng dụng của bạn, Laravel sẽ tạo nó cho bạn:

```
php artisan make:policy PostPolicy
```

Lệnh `make:policy` sẽ tạo ra một class chính sách trống. Nếu bạn muốn tạo một class với các phương thức policy được làm sẵn liên quan đến việc xem, tạo, cập nhật và xóa tài nguyên, bạn có thể cung cấp thêm tùy chọn `--model` khi thực hiện lệnh:

```
php artisan make:policy PostPolicy --model=Post
```

Đăng ký các chính sách

Khi class chính sách đã được tạo, nó cần được đăng ký. Đăng ký các chính sách là cách chúng ta có thể thông báo cho Laravel chính sách nào sẽ sử dụng khi cho phép các action chống lại một loại model nhất định.

`App\Providers\AuthServiceProvider` đi kèm với các ứng dụng Laravel mới có chứa một thuộc tính chính sách ánh xạ các model Eloquent của bạn với các chính sách tương ứng của chúng. Việc đăng ký một chính sách sẽ hướng dẫn Laravel sử dụng chính sách nào khi cho phép các action chống lại một model Eloquent nhất định:

```
<?php

namespace App\Providers;

use App\Models\Post;
use App\Policies\PostPolicy;
use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;
use Illuminate\Support\Facades\Gate;

class AuthServiceProvider extends ServiceProvider
{
  /**
   * The policy mappings for the application.
   *
   * @var array
   */
  protected $policies = [
    Post::class => PostPolicy::class,
  ];
```

```
    /**
     * Register any application authentication / authorization services.
     *
     * @return void
     */
    public function boot()
    {
        $this->registerPolicies();

        //
    }
}
```

Tự động khám phá chính sách

Thay vì đăng ký chính sách model theo cách thủ công, Laravel có thể tự động khám phá các chính sách miễn là model và chính sách tuân theo các quy ước đặt tên Laravel tiêu chuẩn. Cụ thể, các chính sách phải nằm trong thư mục *Policies* tại hoặc phía trên thư mục chứa các models của bạn. Vì vậy, ví dụ: các model có thể được đặt trong thư mục *app/Models* trong khi các chính sách có thể được đặt trong thư mục *app/Policies*. Trong trường hợp này, Laravel sẽ kiểm tra các chính sách trong *app/Models/Policies* rồi đến *app/Policies*. Ngoài ra, tên chính sách phải khớp với tên model và có hậu tố `Policy`. Vì vậy, một model `User` sẽ tương ứng với một class chính sách `UserPolicy`.

Nếu bạn muốn xác định logic khám phá chính sách của riêng mình, bạn có thể đăng ký lệnh gọi lại khám phá chính sách tùy chỉnh bằng cách sử dụng phương thức `Gate::guessPolicyNamesUsing`. Thông thường, phương thức này sẽ được gọi từ phương thức `boot` của `AuthServiceProvider` của ứng dụng của bạn:

```
use Illuminate\Support\Facades\Gate;

Gate::guessPolicyNamesUsing(function ($modelClass) {
    // Return the name of the policy class for the given model...
});
```

Chú ý: Bất kỳ chính sách nào được phát hiện rõ ràng trong `AuthServiceProvider`

của bạn sẽ được ưu tiên hơn bất kỳ chính sách nào có khả năng được phát hiện tự động.

Viết Chính sách

Các phương thức chính sách

Khi class chính sách đã được đăng ký, bạn có thể thêm các phương thức cho mỗi action tương ứng mà nó cho phép. Ví dụ: hãy tạo một phương thức `update` trên `PostPolicy` của chúng ta để xác định xem một đối tượng `App\Models\User` nào đó có thể cập nhật một đối tượng `App\Models\Post` cụ thể hay không.

Phương thức `update` sẽ nhận một đối tượng `User` và một đối tượng `Post` làm đối số của nó và phải trả về `true` hoặc `false` cho biết liệu người dùng có được phép cập nhật model `Post` đã cho hay không. Vì vậy, trong ví dụ này, chúng ta sẽ xác minh rằng `id` của người dùng khớp với `user_id` trên bài đăng:

```php
<?php

namespace App\Policies;

use App\Models\Post;
use App\Models\User;

class PostPolicy
{
  /**
   * Determine if the given post can be updated by the user.
   *
   * @param  \App\Models\User  $user
   * @param  \App\Models\Post  $post
   * @return bool
   */
  public function update(User $user, Post $post)
  {
    return $user->id === $post->user_id;
  }
```

```
}
```

Bạn có thể tiếp tục khai báo các phương thức bổ sung trên chính sách nếu cần cho các action khác mà chính sách cho phép. Ví dụ: bạn có thể xác định các phương thức `view` hoặc `delete` trong chính sách để cho phép các action tương ứng trong model `Post`, nhưng hãy nhớ rằng bạn có thể tự do đặt tên cho các phương thức trên chính sách của mình bất kỳ tên nào bạn thích miễn là nó tương đồng với tên action trong model mà bạn đang thao tác.

Nếu bạn đã sử dụng tùy chọn `--model` khi tạo chính sách của mình với lệnh Artisan CLI, thì nó sẽ chứa sẵn thêm các phương thức cho các action điển hình như `viewAny`, `view`, `create`, `update`, `delete`, `restore` và `forceDelete`.

> Tất cả các chính sách được giải quyết thông qua service container của Laravel, cho phép bạn nhập gợi ý bất kỳ thư viện cần thiết nào trong constructor của chính sách để chúng tự động được đưa vào.

Các phản hồi của chính sách

Cho đến nay, chúng ta mới chỉ kiểm tra các phương thức trên chính sách trả về các giá trị boolean đơn giản. Tuy nhiên, đôi khi bạn có thể muốn trả lại phản hồi chi tiết hơn, bao gồm cả thông báo lỗi. Để làm như vậy, bạn có thể trả về một đối tượng `Illuminate\Auth\Access\Response` từ phương thức chính sách của mình:

```
use App\Models\Post;
use App\Models\User;
use Illuminate\Auth\Access\Response;

/**
 * Determine if the given post can be updated by the user.
 *
 * @param  \App\Models\User  $user
 * @param  \App\Models\Post  $post
 * @return \Illuminate\Auth\Access\Response
 */
public function update(User $user, Post $post)
{
  return $user->id === $post->user_id
    ? Response::allow()
```

```
        : Response::deny('You do not own this post.');
}
```

Khi trả về một phản hồi phân quyền từ chính sách của bạn, phương thức `Gate::allows` vẫn sẽ trả về một giá trị boolean đơn giản; tuy nhiên, bạn có thể sử dụng phương thức `Gate::inspect` để nhận được phản hồi phân quyền đầy đủ do gate trả về:

```
use Illuminate\Support\Facades\Gate;

$response = Gate::inspect('update', $post);

if ($response->allowed()) {
    // The action is authorized...
} else {
    echo $response->message();
}
```

Khi sử dụng phương thức `Gate::authorize`, phương thức này sẽ ném ra một `AuthorizationException` nếu action không được phép truy cập, thông báo lỗi được cung cấp bởi phản hồi phân quyền sẽ được truyền tới phản hồi HTTP:

```
Gate::authorize('update', $post);

// The action is authorized...
```

Phương thức không có model

Một số phương thức chính sách chỉ nhận được một đối tượng của người dùng hiện đã được xác thực. Tình huống này thường xảy ra nhất khi ủy quyền các action tạo "create". Ví dụ: nếu bạn đang tạo blog, bạn có thể muốn xác định xem người dùng có được phép tạo bất kỳ bài đăng nào hay không. Trong những tình huống này, phương thức policy của bạn chỉ nên nhận được một đối tượng người dùng:

```
/**
 * Determine if the given user can create posts.
 *
```

```
     * @param  \App\Models\User  $user
     * @return bool
     */
    public function create(User $user)
    {
        return $user->role == 'writer';
    }
```

Người dùng khách

Theo mặc định, tất cả các gate và chính sách sẽ tự động trả về false nếu yêu cầu HTTP request được gửi đến không được khởi tạo bởi người dùng đã xác thực. Tuy nhiên, bạn có thể cho phép các bài kiểm tra phân quyền này đi qua các gate và chính sách của bạn bằng cách khai báo "optional" hoặc cung cấp giá trị mặc định null cho định nghĩa đối số người dùng:

```php
<?php

namespace App\Policies;

use App\Models\Post;
use App\Models\User;

class PostPolicy
{
  /**
   * Determine if the given post can be updated by the user.
   *
   * @param  \App\Models\User  $user
   * @param  \App\Models\Post  $post
   * @return bool
   */
  public function update(?User $user, Post $post)
  {
    return optional($user)->id === $post->user_id;
  }
}
```

Bộ lọc (filter) trong chính sách

Đối với một số người dùng nhất định, bạn có thể muốn phân quyền tất cả các action trong một chính sách nhất định. Để thực hiện điều này, hãy xác định một phương thức `before` trên chính sách. Phương thức `before` sẽ được thực thi trước bất kỳ phương thức nào khác trong chính sách, tạo cơ hội cho bạn phân quyền action trước khi phương thức chính sách mà bạn dự định thực sự được gọi. Tính năng này được sử dụng phổ biến nhất để phân quyền cho quản trị viên ứng dụng thực hiện bất kỳ action nào:

```php
use App\Models\User;

/**
 * Perform pre-authorization checks.
 *
```

```
     * @param  \App\Models\User  $user
     * @param  string  $ability
     * @return void|bool
     */
    public function before(User $user, $ability)
    {
        if ($user->isAdministrator()) {
            return true;
        }
    }
```

Nếu bạn muốn từ chối tất cả các bài kiểm tra phân quyền cho một loại người dùng cụ thể thì bạn có thể trả về `false` từ phương thức `before`. Nếu trả về `null`, thì bài kiểm tra phân quyền sẽ chuyển sang phương thức trên chính sách.

> **Chú ý:** Phương thức `before` của một class chính sách sẽ không được gọi nếu class đó không chứa một phương thức có tên khớp với tên của khả năng đang được kiểm tra.

Phân quyền các Actions bằng các chính sách

Thông qua model User

Mô hình `App\Models\User` được đưa sẵn vào trong ứng dụng Laravel của bạn bao gồm hai phương thức hữu ích để cho phép các action: `can` và `cannot`. Các phương thức `can` và `cannot` nhận tên của action bạn muốn cho phép và model có liên quan. Ví dụ: hãy xác định xem người dùng có được phép cập nhật một model `App\Models\Post` nhất định hay không. Thông thường, điều này sẽ được thực hiện trong phương thức controller:

```
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\Post;
use Illuminate\Http\Request;
```

```
class PostController extends Controller
{
  /**
   * Update the given post.
   *
   * @param  \Illuminate\Http\Request  $request
   * @param  \App\Models\Post  $post
   * @return \Illuminate\Http\Response
   */
  public function update(Request $request, Post $post)
  {
    if ($request->user()->cannot('update', $post)) {
      abort(403);
    }

    // Update the post...
  }
}
```

Nếu một chính sách được đăng ký cho mô hình đã cho, phương thức `can` sẽ tự động gọi chính sách thích hợp và trả về kết quả boolean. Nếu không có chính sách nào được đăng ký cho model, phương thức `can` sẽ cố gắng gọi Gate dựa trên hàm xử lý mà khớp với tên action đã cho.

Các action mà không đòi hỏi các model

Hãy nhớ rằng, một số action có thể tương ứng với các phương thức chính sách như `create` không yêu cầu đối tượng model. Trong những tình huống này, bạn có thể truyền một tên class cho phương thức `can`. Tên class sẽ được sử dụng để xác định chính sách nào sẽ sử dụng khi phân quyền hành động:

```
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\Post;
```

```php
use Illuminate\Http\Request;

class PostController extends Controller
{
  /**
   * Create a post.
   *
   * @param  \Illuminate\Http\Request  $request
   * @return \Illuminate\Http\Response
   */
  public function store(Request $request)
  {
    if ($request->user()->cannot('create', Post::class)) {
      abort(403);
    }

    // Create the post...
  }
}
```

Thông qua các helper về controller

Ngoài các phương thức hữu ích được cung cấp cho mô hình `App\Models\User`, Laravel cung cấp một phương thức `authorize` hữu ích cho bất kỳ controller nào của bạn, mở rộng class gốc `App\Http\Controllers\Controller`.

Giống như phương thức `can`, phương thức này chấp nhận tên của action bạn muốn cho phép và mô hình có liên quan. Nếu action không được phân quyền, phương thức `authorize` sẽ đưa ra một exception `Illuminate\Auth\Access\AuthorizationException` mà trình xử lý exception Laravel sẽ tự động chuyển đổi thành phản hồi HTTP response với mã trạng thái 403:

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
```

```php
use App\Models\Post;
use Illuminate\Http\Request;

class PostController extends Controller
{
  /**
   * Update the given blog post.
   *
   * @param  \Illuminate\Http\Request  $request
   * @param  \App\Models\Post  $post
   * @return \Illuminate\Http\Response
   *
   * @throws \Illuminate\Auth\Access\AuthorizationException
   */
  public function update(Request $request, Post $post)
  {
    $this->authorize('update', $post);

    // The current user can update the blog post...
  }
}
```

Các action mà không đòi hỏi model

Như đã thảo luận trước đây, một số phương thức chính sách như create không yêu cầu đối tượng model. Trong những tình huống này, bạn nên truyền một tên class cho phương thức authorize. Tên class sẽ được sử dụng để xác định chính sách nào sẽ sử dụng khi phân quyền action:

```php
use App\Models\Post;
use Illuminate\Http\Request;

/**
 * Create a new blog post.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
```

```
     *
     * @throws \Illuminate\Auth\Access\AuthorizationException
     */
    public function create(Request $request)
    {
        $this->authorize('create', Post::class);

        // The current user can create blog posts...
    }
```

Phân quyền cho controller tài nguyên

Nếu bạn đang sử dụng controller tài nguyên, bạn có thể sử dụng phương thức `authorizeResource` trong constructor của controller. Phương thức này sẽ đính kèm các khai báo middleware có thể thích hợp vào các phương thức của controller tài nguyên.

Phương thức `authorizeResource` chấp nhận tên class của model làm đối số đầu tiên của nó và tên của tham số route/request sẽ chứa ID của model làm đối số thứ hai của nó. Bạn nên đảm bảo controller tài nguyên của mình được tạo bằng tùy chọn `--model` để nó có đầy đủ phương thức và phần khai báo kiểu cần thiết:

```
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Models\Post;
use Illuminate\Http\Request;

class PostController extends Controller
{
    /**
     * Create the controller instance.
     *
     * @return void
     */
    public function __construct()
```

```
    {
        $this->authorizeResource(Post::class, 'post');
    }
}
```

Các phương thức controller sau sẽ được ánh xạ tới phương thức chính sách tương ứng của chúng. Khi các requests được chuyển đến phương thức controller đã cho, phương thức chính sách tương ứng sẽ tự động được gọi trước khi phương thức controller được thực thi:

Controller Method	Policy Method
index	viewAny
show	view
create	create
store	create
edit	update
update	update
destroy	delete

> Bạn có thể sử dụng lệnh `make:policy` với tùy chọn `--model` để nhanh chóng tạo một class chính sách cho một model nhất định: `php artisan make:policy PostPolicy --model=Post`.

Thông qua Middleware

Laravel bao gồm một middleware có thể authorize các hành động trước khi yêu cầu gửi đến thậm chí đến được các routes hoặc controller của bạn. Theo mặc định, middleware `Illuminate\Auth\Middleware\Authorize` được gán khóa can trong class `App\Http\Kernel` của bạn. Hãy cùng khám phá một ví dụ về việc sử dụng middleware có thể để cho phép người dùng có thể cập nhật một bài đăng:

```
use App\Models\Post;
```

```
Route::put('/post/{post}', function (Post $post) {
  // The current user may update the post...
})->middleware('can:update,post');
```

Trong ví dụ này, chúng ta đang truyền hai đối số của middleware can. Đầu tiên là tên của action mà chúng ta muốn cho phép và thứ hai là tham số route mà chúng ta muốn truyền cho phương thức chính sách. Trong trường hợp này, vì chúng ta đang sử dụng liên kết model ngầm định, nên model App\Models\Post sẽ được truyền đến phương thức chính sách. Nếu người dùng không được phép thực hiện hành động đã cho, một phản hồi HTTP với mã trạng thái 403 sẽ được middleware trả về.

Để thuận tiện, bạn cũng có thể đính kèm middleware `can` vào route của mình bằng phương pháp `can:`

```
use App\Models\Post;

Route::put('/post/{post}', function (Post $post) {
  // The current user may update the post...
})->can('update', 'post');
```

Các Action không đòi hỏi các model

Một lần nữa, một số phương thức chính sách như `create` không yêu cầu đối tượng model. Trong những tình huống này, bạn có thể truyền tên class cho middleware. Tên class sẽ được sử dụng để xác định chính sách nào sẽ sử dụng khi phân quyền action:

```
Route::post('/post', function () {
  // The current user may create posts...
})->middleware('can:create,App\Models\Post');
```

Việc chỉ định toàn bộ tên lớp trong một định nghĩa phần mềm trung gian chuỗi có thể trở nên cồng kềnh. Vì lý do đó, bạn có thể chọn đính kèm phần mềm trung gian có thể vào tuyến đường của mình bằng phương pháp có thể:

```
use App\Models\Post;

Route::post('/post', function () {
```

```
    // The current user may create posts...
})->can('create', Post::class);
```

Thông qua Blade template

Khi viết các mẫu Blade template, bạn có thể muốn hiển thị một phần của trang chỉ khi người dùng được phép thực hiện một action nhất định. Ví dụ: bạn có thể muốn hiển thị form biểu mẫu cập nhật cho một bài đăng trên blog chỉ khi người dùng thực sự có thể cập nhật bài đăng đó. Trong trường hợp này, bạn có thể sử dụng lệnh `@can` và `@cannot`:

```
@can('update', $post)
    <!-- The current user can update the post... -->
@elsecan('create', App\Models\Post::class)
    <!-- The current user can create new posts... -->
@else
    <!-- ... -->
@endcan

@cannot('update', $post)
    <!-- The current user cannot update the post... -->
@elsecannot('create', App\Models\Post::class)
    <!-- The current user cannot create new posts... -->
@endcannot
```

Các lệnh này là các phím tắt thuận tiện để viết các câu lệnh `@if` và `@unless`. Các câu lệnh `@can` và `@cannot` ở trên tương đương với các câu lệnh sau:

```
@if (Auth::user()->can('update', $post))
    <!-- The current user can update the post... -->
@endif

@unless (Auth::user()->can('update', $post))
    <!-- The current user cannot update the post... -->
@endunless
```

Bạn cũng có thể xác định xem người dùng có được phép thực hiện bất kỳ action nào từ một

loạt các action nào đó hay không. Để thực hiện điều này, hãy sử dụng chỉ thị `@canany`:

```
@canany(['update', 'view', 'delete'], $post)
  <!-- The current user can update, view, or delete the post... -->
@elsecanany(['create'], \App\Models\Post::class)
  <!-- The current user can create a post... -->
@endcanany
```

Các action không đòi hỏi các model

Giống như hầu hết các phương thức phân quyền khác, bạn có thể truyền tên lớp cho các chỉ thị `@can` và `@cannot` nếu action không yêu cầu đối tượng model:

```
@can('create', App\Models\Post::class)
  <!-- The current user can create posts... -->
@endcan

@cannot('create', App\Models\Post::class)
  <!-- The current user can't create posts... -->
@endcannot
```

Khi phân quyền các action bằng chính sách, bạn có thể truyền một mảng làm đối số thứ hai cho các hàm và hàm xử lý phân quyền khác nhau. Phần tử đầu tiên trong mảng sẽ được sử dụng để xác định chính sách nào sẽ được gọi, trong khi phần tử còn lại của mảng được chuyển dưới dạng tham số cho phương thức chính sách và có thể được sử dụng cho ngữ cảnh bổ sung khi đưa ra quyết định phân quyền. Ví dụ: hãy xem qua phần khai báo phương thức `PostPolicy` sau đây, nó có chứa tham số `$category` bổ sung:

```
/**
 * Determine if the given post can be updated by the user.
 *
 * @param  \App\Models\User  $user
 * @param  \App\Models\Post  $post
 * @param  int  $category
 * @return bool
 */
```

```
public function update(User $user, Post $post, int $category)
{
  return $user->id === $post->user_id &&
    $user->canUpdateCategory($category);
}
```

Khi cố gắng xác định xem người dùng đã xác thực có thể cập nhật một bài đăng nhất định hay không, chúng ta có thể gọi phương thức chính sách này như sau:

```
/**
 * Update the given blog post.
 *
 * @param  \Illuminate\Http\Request  $request
 * @param  \App\Models\Post  $post
 * @return \Illuminate\Http\Response
 *
 * @throws \Illuminate\Auth\Access\AuthorizationException
 */
public function update(Request $request, Post $post)
{
  $this->authorize('update', [$post, $request->category]);

  // The current user can update the blog post...
}
```