Course - Laravel Framework

Lưu trữ tập tin

Laravel cung cấp một hệ thống tập tin mạnh mẽ nhờ package Flysystem PHP tuyệt vời của Frank de Jonge. Laravel Flysystem sẽ cung cấp các driver đơn giản để làm việc với các hệ thống tập tin cục bộ, cũng như SFTP và Amazon S3.

Tags: laravel upload file, laravel file storage, laravel

Giới thiệu

Laravel cung cấp một hệ thống tập tin mạnh mẽ nhờ package Flysystem PHP tuyệt vời của Frank de Jonge. Tính năng Laravel Flysystem cung cấp các driver đơn giản để làm việc với các hệ thống tập tin local, SFTP và Amazon S3. Không những thế, việc chuyển đổi giữa các tùy chọn lưu trữ này giữa local machine và server thành phẩm của bạn rất đơn giản vì API được giữ nguyên cho mỗi hệ thống.

Cấu hình

Tập tin cấu hình hệ thống tập tin của Laravel được đặt tại *config/filesystems.php*. Trong tập tin này, bạn có thể định cấu hình tất cả các "đĩa" hệ thống tập tin của mình. Mỗi đĩa đại diện cho một driver lưu trữ và vị trí lưu trữ cụ thể. Các cấu hình mẫu cho từng driver được hỗ trợ được bao gồm trong tập tin cấu hình để bạn có thể sửa đổi cấu hình để phản ánh tùy chọn lưu trữ và thông tin đăng nhập của mình.

Driver lưu trữ `local` tương tác với các tập tin được lưu trữ cục bộ trên máy chủ chạy ứng dụng Laravel trong khi driver `s3` được sử dụng để ghi vào dịch vụ lưu trữ đám mây S3 của Amazon.

> Bạn có thể cấu hình bao nhiêu đĩa tùy thích và thậm chí có thể có nhiều đĩa sử dụng cùng một driver.

Lưu trữ cục bộ

Khi sử dụng driver `local`, tất cả các hoạt động tập tin đều liên quan đến thư mục `root` được xác định trong tập tin cấu hình `filesystems` của bạn. Mặc định, giá trị này được đặt thành thư mục *storage/app*. Do đó, phương thức sau sẽ ghi vào *storage/app/example.txt*:

```
use Illuminate\Support\Facades\Storage;

Storage::disk('local')->put('example.txt', 'Contents');
```

Truy cập công khai

Đĩa `public` có trong tập tin cấu hình `filesystems` của ứng dụng của bạn dành cho các tập tin sẽ có thể truy cập công khai. Mặc định, đĩa public sử dụng driver local và lưu trữ các tập tin của nó trong `storage/app/public`.

Để làm cho các tập tin này có thể truy cập được từ web, bạn nên tạo một liên kết tượng trưng từ `public/storage` đến `storage/app/public`. Việc sử dụng quy ước thư mục này sẽ giữ các tập tin có thể truy cập công khai của bạn trong một thư mục có thể dễ dàng chia sẻ trên các lần triển khai khi sử dụng các hệ thống triển khai không tốn thời gian như Envoyer.

Để tạo liên kết tượng trưng, bạn có thể sử dụng lệnh Artisan `storage:link`:

```
php artisan storage:link
```

Khi tệp đã được lưu trữ và liên kết tượng trưng đã được tạo, bạn có thể tạo URL cho tập tin bằng hàm trợ giúp `asset`:

```
echo asset('storage/file.txt');
```

Bạn có thể định cấu hình các liên kết tượng trưng bổ sung trong tập cấu hình `filesystems` của mình. Mỗi liên kết được cấu hình sẽ được tạo khi bạn chạy lệnh `storage:link`:

```
'links' => [
  public_path('storage') => storage_path('app/public'),
  public_path('images') => storage_path('app/images'),
],
```

Các yêu cầu cho driver

Các composer packages

Trước khi sử dụng trình điều khiển S3 hoặc SFTP, bạn sẽ cần cài đặt các package thích hợp thông qua trình quản lý gói Composer:

- Amazon S3:`composer require --with-all-dependencies league/flysystem-aws-s3-v3 "^1.0"`
- SFTP:`composer require league/flysystem-sftp "~1.0"`

Ngoài ra, bạn có thể chọn cài đặt bộ điều hợp được lưu trong bộ nhớ cache để tăng hiệu suất:

- CachedAdapter:`composer require league/flysystem-cached-adapter "~1.0"`

Cấu hình driver cho S3

Thông tin cấu hình driver S3 nằm trong tập tin cấu hình *config/filesystems.php* của bạn. Tập tin này chứa một mảng cấu hình ví dụ cho driver S3. Bạn có thể tự do sửa đổi mảng này với cấu hình S3 và thông tin đăng nhập của riêng bạn. Để thuận tiện, các biến môi trường này phù hợp với quy ước đặt tên được AWS CLI sử dụng.

Cấu hình driver cho FTP

Tính năng Flysystem của Laravel hoạt động hiệu quả với FTP; tuy nhiên, cấu hình mẫu của nó không được đưa sẵn vào trong tập tin cấu hình *filesystems.php* mặc định của framework. Nếu bạn cần cấu hình hệ thống tập tin FTP, bạn có thể sử dụng ví dụ cấu hình bên dưới:

```
'ftp' => [
  'driver' => 'ftp',
  'host' => env('FTP_HOST'),
  'username' => env('FTP_USERNAME'),
  'password' => env('FTP_PASSWORD'),

  // Optional FTP Settings...
  // 'port' => env('FTP_PORT', 21),
  // 'root' => env('FTP_ROOT'),
  // 'passive' => true,
  // 'ssl' => true,
  // 'timeout' => 30,
],
```

Cấu hình driver cho SFTP

Tính năng Flysystem của Laravel cũng hoạt động hiệu quả với SFTP; tuy nhiên, cấu hình mẫu của nó không được đưa sẵn trong tập tin cấu hình *filesystems.php* mặc định của framework. Nếu bạn cần cấu hình hệ thống tập tin SFTP, bạn có thể sử dụng ví dụ cấu hình bên dưới:

```
'sftp' => [
```

```
    'driver' => 'sftp',
    'host' => env('SFTP_HOST'),

    // Settings for basic authentication...
    'username' => env('SFTP_USERNAME'),
    'password' => env('SFTP_PASSWORD'),

    // Settings for SSH key based authentication with encryption password...
    'privateKey' => env('SFTP_PRIVATE_KEY'),
    'password' => env('SFTP_PASSWORD'),

    // Optional SFTP Settings...
    // 'port' => env('SFTP_PORT', 22),
    // 'root' => env('SFTP_ROOT'),
    // 'timeout' => 30,
],
```

Hệ thống tập tin tương thích của Amazon S3

Mặc định, tập tin cấu hình `filesystems` của ứng dụng của bạn có chứa cấu hình đĩa cho s3. Ngoài việc sử dụng đĩa này để tương tác với Amazon S3, bạn có thể sử dụng nó để tương tác với bất kỳ dịch vụ lưu trữ tệp tương thích nào của S3 như MinIO hoặc DigitalOcean Spaces.

Thông thường, sau khi cập nhật thông tin đăng nhập của đĩa để khớp với thông tin đăng nhập của dịch vụ bạn định sử dụng, bạn chỉ cần cập nhật giá trị của tùy chọn cấu hình `url`. Giá trị của tùy chọn này thường được xác định thông qua biến môi trường `AWS_ENDPOINT`:

```
'endpoint' => env('AWS_ENDPOINT', 'https://minio:9000'),
```

Bộ nhớ đệm cache

Để kích hoạt bộ nhớ đệm cache cho một đĩa nhất định, bạn có thể thêm chỉ thị `cache` vào các tùy chọn cấu hình của đĩa. Tùy chọn `cache` phải là các cài đặt bộ nhớ đệm cache chứa tên của đĩa, thời hạn expire với thời gian tính bằng giây và tiền tố bộ nhớ cache `prefix`:

```
's3' => [
  'driver' => 's3',

  // Other Disk Options...

  'cache' => [
    'store' => 'memcached',
    'expire' => 600,
    'prefix' => 'cache-prefix',
  ],
],
```

Lấy các phiên bản đĩa

Facade `Storage` có thể được sử dụng để tương tác với bất kỳ đĩa nào đã được cấu hình của bạn. Ví dụ: bạn có thể sử dụng phương thức `put` trên facade này để lưu hình đại diện trên đĩa mặc định. Nếu bạn gọi các phương thức trên facade `Storage` mà không gọi phương thức `disk` trước, phương thức sẽ tự động được chuyển vào đĩa mặc định:

```
use Illuminate\Support\Facades\Storage;

Storage::put('avatars/1', $content);
```

Nếu ứng dụng của bạn tương tác với nhiều đĩa, bạn có thể sử dụng phương thức `disk` trên facade `Storage` để làm việc với các tập tin trên một đĩa cụ thể:

```
Storage::disk('s3')->put('avatars/1', $content);
```

Đĩa theo yêu cầu

Đôi khi bạn có thể muốn tạo một đĩa trong thời gian chạy bằng cách sử dụng một cấu hình nhất định mà cấu hình đó thực sự không có trong tập tin cấu hình `filesystems` của ứng dụng của bạn. Để thực hiện điều này, bạn có thể chuyển một mảng cấu hình cho phương thức `build` của facade `Storage`:

```php
use Illuminate\Support\Facades\Storage;

$disk = Storage::build([
  'driver' => 'local',
  'root' => '/path/to/root',
]);

$disk->put('image.jpg', $content);
```

Truy xuất tập tin

Phương thức `get` này có thể được sử dụng để truy xuất nội dung của tập tin. Nội dung chuỗi thô của tập tin sẽ được trả về bởi phương thức. Hãy nhớ rằng tất cả các đường dẫn tập tin phải được chỉ định liên quan đến vị trí "gốc" của đĩa:

```php
$contents = Storage::get('file.jpg');
```

Phương thức `exists` này có thể được sử dụng để xác định xem tập tin có tồn tại trên đĩa hay không:

```php
if (Storage::disk('s3')->exists('file.jpg')) {
  // ...
}
```

Phương thức `missing` này có thể được sử dụng để xác định xem tập tin có bị thiếu trong đĩa hay không:

```php
if (Storage::disk('s3')->missing('file.jpg')) {
  // ...
}
```

Tải tập tin

Phương thức `download` này có thể được sử dụng để tạo response mà sẽ buộc trình duyệt

của người dùng tải xuống tập tin theo đường dẫn nhất định. Phương thức `download` chấp nhận tên tập tin làm đối số thứ hai cho phương thức, sẽ xác định tên tập tin mà người dùng muốn nhìn thấy sau khi hoàn tất việc tải xuống. Cuối cùng, bạn có thể truyền một mảng tiêu đề HTTP header làm đối số thứ ba cho phương thức:

```
return Storage::download('file.jpg');

return Storage::download('file.jpg', $name, $headers);
```

URL tập tin

Bạn có thể sử dụng phương thức `url` này để lấy URL cho một tập tin nhất định. Nếu bạn đang sử dụng driver cho `local`, cái này thường sẽ chỉ thêm trước vào đường dẫn */storage* đã cho và trả về một URL tương đối cho tập. Nếu bạn đang sử dụng driver `s3`, URL từ xa đủ điều kiện sẽ được trả về:

```
use Illuminate\Support\Facades\Storage;

$url = Storage::url('file.jpg');
```

Khi sử dụng driver `local`, tất cả các tập tin có thể truy cập công khai phải được đặt trong thư mục *storage/app/public*. Hơn nữa, bạn nên tạo một liên kết tượng trưng trỏ *public/storage* đến thư mục *storage/app/public*.

> **Chú ý:** Khi sử dụng driver `local`, giá trị trả về của `url` không được mã hóa URL. Vì lý do này, chúng tôi khuyên bạn nên luôn lưu trữ tập tin của mình bằng cách sử dụng tên mà sẽ tạo ra URL hợp lệ nhất có thể.

URL tạm thời

Sử dụng phương thức `temporaryUrl` này, bạn có thể tạo các URL tạm thời cho các tập tin được lưu trữ bằng driver `s3`. Phương thức này chấp nhận một đường dẫn và một đối tượng `DateTime` chỉ định khi nào URL sẽ hết hạn:

```
use Illuminate\Support\Facades\Storage;
```

```
$url = Storage::temporaryUrl(
  'file.jpg', now()->addMinutes(5)
);
```

Nếu bạn cần chỉ định các tham số yêu cầu S3 bổ sung, bạn có thể truyền mảng tham số yêu cầu làm đối số thứ ba cho phương thức `temporaryUrl`:

```
$url = Storage::temporaryUrl(
  'file.jpg',
  now()->addMinutes(5),
  [
    'ResponseContentType' => 'application/octet-stream',
    'ResponseContentDisposition' => 'attachment; filename=file2.jpg',
  ]
);
```

Nếu bạn cần tùy chỉnh cách tạo URL tạm thời cho một đĩa lưu trữ cụ thể, bạn có thể sử dụng phương thức `buildTemporaryUrlsUsing` này. Ví dụ: điều này có thể hữu ích nếu bạn có controller cho phép bạn tải xuống các tập tin được lưu trữ qua đĩa thường không hỗ trợ URL tạm thời. Thông thường, phương thức này nên được gọi từ phương thức `boot` của nhà service provider:

```
<?php
namespace App\Providers;
use Illuminate\Support\Facades\Storage;
use Illuminate\Support\Facades\URL;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
  /**
   * Bootstrap any application services.
   *
   * @return void
   */
  public function boot()
  {
```

```
    Storage::disk('local')->buildTemporaryUrlsUsing(function ($path, $expiration, $options) {
      return URL::temporarySignedRoute(
        'files.download',
        $expiration,
        array_merge($options, ['path' => $path])
      );
    });
  }
}
```

Tùy chỉnh máy chủ URL

Nếu bạn muốn xác định trước máy chủ cho các URL được tạo bằng cách sử dụng facade `Storage`, bạn có thể thêm tùy chọn `url` vào mảng cấu hình của đĩa:

```
'public' => [
  'driver' => 'local',
  'root' => storage_path('app/public'),
  'url' => env('APP_URL').'/storage',
  'visibility' => 'public',
],
```

Meta của tập tin

Ngoài việc đọc và ghi tập tin, Laravel cũng có thể cung cấp thông tin về chính tập tin đó. Ví dụ, phương thức `size` này có thể được sử dụng để lấy kích thước của tập tin bằng đơn vị đo là byte:

```
use Illuminate\Support\Facades\Storage;

$size = Storage::size('file.jpg');
```

Phương thức `lastModified` trả về dấu thời gian UNIX của lần cuối cùng tập tin được sửa đổi:

```php
$time = Storage::lastModified('file.jpg');
```

Đường dẫn của tập tin

Bạn có thể sử dụng phương thức `path` này để lấy đường dẫn cho một tập tin nhất định. Nếu bạn đang sử dụng driver `local`, điều này sẽ trả về đường dẫn tuyệt đối đến tập tin. Nếu bạn đang sử dụng driver `s3`, thì phương thức này sẽ trả về đường dẫn tương đối đến tập tin trong nhóm S3:

```php
use Illuminate\Support\Facades\Storage;

$path = Storage::path('file.jpg');
```

Lưu trữ tập tin

Phương thức `put` này có thể được sử dụng để lưu trữ nội dung tập tin trên đĩa. Bạn cũng có thể truyền một PHP resource vào phương thức `put` này, phương thức này sẽ sử dụng hỗ trợ luồng cơ bản của Flysystem. Hãy nhớ rằng, tất cả các đường dẫn tập tin phải được chỉ định liên quan đến vị trí "gốc" được cấu hình cho đĩa:

```php
use Illuminate\Support\Facades\Storage;

Storage::put('file.jpg', $contents);

Storage::put('file.jpg', $resource);
```

Phát trực tuyến tự động

Truyền tập tin vào bộ nhớ giúp giảm đáng kể việc sử dụng bộ nhớ. Nếu bạn muốn Laravel tự động quản lý việc truyền trực tuyến một tập tin nhất định đến vị trí lưu trữ của bạn, bạn có thể sử dụng phương thức `putFile` hoặc `putFileAs`. Phương thức này chấp nhận một đối tượng `Illuminate\Http\File` hoặc một đối tượng `Illuminate\Http\UploadedFile` và sẽ tự động truyền tập tin đến vị trí mong muốn của bạn:

```php
use Illuminate\Http\File;
```

```php
use Illuminate\Support\Facades\Storage;

// Automatically generate a unique ID for filename...
$path = Storage::putFile('photos', new File('/path/to/photo'));

// Manually specify a filename...
$path = Storage::putFileAs('photos', new File('/path/to/photo'), 'photo.jpg');
```

Có một số điều quan trọng cần lưu ý về phương thức `putFile` này. Lưu ý rằng chúng ta chỉ chỉ định tên thư mục chứ không phải tên tập tin. Mặc định, phương thức `putFile` sẽ tạo một ID duy nhất để làm tên tập tin. Phần mở rộng của tập tin sẽ được xác định bằng cách kiểm tra kiểu MIME của tập tin. Đường dẫn đến tập tin sẽ được trả về theo phương thức `putFile` để bạn có thể lưu trữ đường dẫn, bao gồm tên tập tin đã tạo, trong cơ sở dữ liệu của mình.

Các phương thức `putFile` và `putFileAs` cũng chấp nhận một đối số để chỉ định "khả năng hiển thị" của tập tin được lưu trữ. Điều này đặc biệt hữu ích nếu bạn đang lưu trữ tập tin trên đĩa đám mây chẳng hạn như Amazon S3 và muốn tập tin có thể truy cập công khai qua các URL được tạo:

```php
Storage::putFile('photos', new File('/path/to/photo'), 'public');
```

Thêm trước & Thêm bổ sung vào Tập tin

Các phương thức `prepend` và `append` sẽ cho phép bạn ghi vào đầu hoặc cuối tập tin:

```php
Storage::prepend('file.log', 'Prepended Text');

Storage::append('file.log', 'Appended Text');
```

Sao chép & di chuyển tập tin

Phương thức `copy` có thể được sử dụng để sao chép tập tin hiện có sang vị trí mới trên đĩa, trong khi phương thức `move` lại được sử dụng để đổi tên hoặc di chuyển tập tin hiện tại sang vị trí mới:

```
Storage::copy('old/file.jpg', 'new/file.jpg');

Storage::move('old/file.jpg', 'new/file.jpg');
```

Tải upload tập tin

Trong các ứng dụng web, một trong những trường hợp sử dụng phổ biến nhất để lưu trữ tập tin là lưu trữ các tập tin do người dùng upload lên như ảnh và tài liệu. Laravel giúp bạn dễ dàng lưu trữ các tập tin đã tải lên uploaded bằng phương thức `store` trên đối tượng tập tin đã tải lên. Hãy gọi phương thức `store` với đường dẫn mà bạn muốn để lưu trữ tập tin đã tải lên:

```
<?php
namespace App\Http\Controllers;
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;

class UserAvatarController extends Controller
{
  /**
   * Update the avatar for the user.
   *
   * @param  \Illuminate\Http\Request  $request
   * @return \Illuminate\Http\Response
   */
  public function update(Request $request)
  {
    $path = $request->file('avatar')->store('avatars');

    return $path;
  }
}
```

Có một số điều quan trọng cần lưu ý về ví dụ này. Lưu ý rằng chúng ta chỉ chỉ định tên thư mục, không phải tên tập tin. Mặc định, thì phương thức `store` sẽ tạo ra một ID duy nhất để làm tên tập tin. Phần mở rộng của tập tin sẽ được xác định bằng cách kiểm tra kiểu

MIME của tập tin. Đường dẫn đến tập tin sẽ được trả về theo phương thức `store` để bạn có thể lưu trữ đường dẫn, bao gồm tên tập tin đã tạo, trong cơ sở dữ liệu của mình.

Bạn cũng có thể gọi phương thức `putFile` trên facade `Storage` để thực hiện thao tác lưu trữ tập tin giống như ví dụ trên:

```
$path = Storage::putFile('avatars', $request->file('avatar'));
```

Chỉ định tên tập tin

Nếu bạn không muốn dùng tên tập tin tự động mà sẽ gán cho tập tin đã được lưu trữ của mình, bạn có thể sử dụng phương thức `storeAs`, nó chấp nhận đường dẫn tên tập tin và đĩa (tùy chọn) làm các đối số của nó:

```
$path = $request->file('avatar')->storeAs(
  'avatars', $request->user()->id
);
```

Bạn cũng có thể sử dụng phương thức `putFileAs` trên facade `Storage`, phương thức này sẽ thực hiện thao tác lưu trữ tập tin giống như ví dụ trên:

```
$path = Storage::putFileAs(
  'avatars', $request->file('avatar'), $request->user()->id
);
```

> **Chú ý:** Những ký tự unicode không thể in và không hợp lệ sẽ tự động bị xóa khỏi đường dẫn tập tin. Do đó, bạn có thể muốn làm sạch đường dẫn tập tin của mình trước khi truyền chúng vào các phương thức lưu trữ tập tin của Laravel. Đường dẫn tập tin được quy chuẩn hóa bằng phương thức `League\Flysystem\Util::normalizePath`.

Cài đặt đĩa lưu trữ

Mặc định, phương thức `store` của tập tin được tải lên sẽ sử dụng đĩa mặc định của bạn. Nếu bạn muốn chỉ định một đĩa khác, hãy truyền tên đĩa làm đối số thứ hai cho phương thức `store`:

```
$path = $request->file('avatar')->store(
  'avatars/'.$request->user()->id, 's3'
);
```

Nếu bạn đang sử dụng phương thức `storeAs`, bạn có thể chuyển tên đĩa làm đối số thứ ba cho phương thức:

```
$path = $request->file('avatar')->storeAs(
  'avatars',
  $request->user()->id,
  's3'
);
```

Thông tin tập tin được tải lên khác

Nếu bạn muốn lấy tên gốc và phần mở rộng của tập tin đã tải lên, bạn có thể làm như vậy bằng cách sử dụng các phương thức `getClientOriginalName` và `getClientOriginalExtension`:

```
$file = $request->file('avatar');

$name = $file->getClientOriginalName();
$extension = $file->getClientOriginalExtension();
```

Tuy nhiên, hãy nhớ rằng các phương thức `getClientOriginalName` và `getClientOriginalExtension` được coi là không an toàn, vì tên tập tin và phần mở rộng có thể bị giả mạo bởi người dùng độc hại. Vì lý do này, bạn thường muốn dùng các phương thức `hashName` và `extension` để lấy tên và phần mở rộng cho việc upload tập tin đã cho:

```
$file = $request->file('avatar');

// Generate a unique, random name...
$name = $file->hashName();
```

```php
// Determine the file's extension based on
// the file's MIME type...
$extension = $file->extension();
```

Mức độ công khai tập tin

Trong tính năng Flysystem của Laravel, "visibility" là sự ẩn dụ hóa nói đến quyền đối với tập tin trên nhiều nền tảng. Các tập tin có thể được khai báo với `public` hoặc `private`. Khi một tập tin được khai báo `public`, là bạn đang chỉ ra rằng tập tin sẽ có thể truy cập được đối với những người khác. Ví dụ: khi sử dụng driver lưu trữ S3, bạn có thể truy xuất URL cho các tập tin `public`.

Bạn có thể đặt chế độ công khai khi ghi tập tin thông qua phương thức `put`:

```php
use Illuminate\Support\Facades\Storage;

Storage::put('file.jpg', $contents, 'public');
```

Nếu tập tin đã được lưu trữ, thì việc công khai tập tin có thể được thiết lập và truy xuất thông qua các phương thức `getVisibility` và `setVisibility`:

```php
$visibility = Storage::getVisibility('file.jpg');

Storage::setVisibility('file.jpg', 'public');
```

Khi tương tác với các tập tin được tải lên, bạn có thể sử dụng các phương thức `storePublicly` và `storePubliclyAs` để lưu trữ tập tin đã tải lên với chế độ `public`:

```php
$path = $request->file('avatar')->storePublicly('avatars', 's3');

$path = $request->file('avatar')->storePubliclyAs(
  'avatars',
  $request->user()->id,
  's3'
);
```

Tập tin local & chế độ công khai

Khi sử dụng driver `local`, chế độ `public` sẽ chuyển thành quyền số 0755 đối với thư mục và quyền số 0644 đối với tập tin. Bạn có thể sửa đổi ánh xạ quyền trong tập tin cấu hình `filesystems` của ứng dụng của mình:

```
'local' => [
  'driver' => 'local',
  'root' => storage_path('app'),
  'permissions' => [
    'file' => [
      'public' => 0644,
      'private' => 0600,
    ],
    'dir' => [
      'public' => 0755,
      'private' => 0700,
    ],
  ],
],
```

Xóa tệp

Phương thức `delete` chấp nhận một tên tập tin hoặc một mảng tập tin để xóa:

```
use Illuminate\Support\Facades\Storage;

Storage::delete('file.jpg');

Storage::delete(['file.jpg', 'file2.jpg']);
```

Nếu cần, bạn có thể chỉ định đĩa mà tập tin sẽ được xóa khỏi:

```php
use Illuminate\Support\Facades\Storage;

Storage::disk('s3')->delete('path/file.jpg');
```

Thư mục

Nhận tất cả các tệp trong một thư mục

Phương thức `files` sẽ trả về một mảng tất cả các tập tin trong một thư mục nào đó. Nếu bạn muốn truy xuất danh sách tất cả các tập tin trong một thư mục nào đó bao gồm tất cả các thư mục con, thì bạn có thể sử dụng phương thức `allFiles`:

```php
use Illuminate\Support\Facades\Storage;

$files = Storage::files($directory);

$files = Storage::allFiles($directory);
```

Nhận tất cả các thư mục trong một thư mục

Phương thức `directories` trả về một mảng tất cả các thư mục trong một thư mục nào đó. Ngoài ra, bạn có thể sử dụng phương thức `allDirectories` này để lấy danh sách tất cả các thư mục trong một thư mục nào đó và tất cả các thư mục con của chúng:

```php
$directories = Storage::directories($directory);

$directories = Storage::allDirectories($directory);
```

Tạo một thư mục

Phương thức `makeDirectory` sẽ tạo một thư mục nào đó, bao gồm bất kỳ thư mục con nào cần thiết:

```
Storage::makeDirectory($directory);
```

Xóa thư mục

Cuối cùng, phương thức `deleteDirectory` này có thể được sử dụng để xóa một thư mục và tất cả các tập tin của nó:

```
Storage::deleteDirectory($directory);
```

Hệ thống tập tin tự tạo

Tính năng Flysystem của Laravel cung cấp hỗ trợ cho một số "driver" bổ sung; tuy nhiên, Flysystem không bị giới hạn ở những điều này và có các bộ điều hợp cho nhiều hệ thống lưu trữ khác. Bạn có thể tạo một driver tự tạo nếu bạn muốn sử dụng một trong những bộ điều hợp bổ sung này trong ứng dụng Laravel của mình.

Để tạo một hệ thống tập tin tự tạo, bạn sẽ cần một bộ điều hợp Flysystem. Hãy thêm bộ điều hợp Dropbox do cộng đồng phát triển vào dự án của chúng ta:

```
composer require spatie/flysystem-dropbox
```

Tiếp theo, bạn có thể đăng ký driver theo phương thức `boot` của một trong những service provider của ứng dụng của bạn . Để thực hiện điều này, bạn nên sử dụng phương thức `extend` của facade `Storage`:

```
<?php

namespace App\Providers;

use Illuminate\Support\Facades\Storage;
use Illuminate\Support\ServiceProvider;
use League\Flysystem\Filesystem;
use Spatie\Dropbox\Client as DropboxClient;
use Spatie\FlysystemDropbox\DropboxAdapter;

class AppServiceProvider extends ServiceProvider
```

```
{
  /**
   * Register any application services.
   *
   * @return void
   */
  public function register()
  {
    //
  }

  /**
   * Bootstrap any application services.
   *
   * @return void
   */
  public function boot()
  {
    Storage::extend('dropbox', function ($app, $config) {
      $client = new DropboxClient(
        $config['authorization_token']
      );

      return new Filesystem(new DropboxAdapter($client));
    });
  }
}
```

Đối số đầu tiên của phương thức `extend` là tên của driver và đối số thứ hai là một hàm nặc danh nhận các biến `$app` và `$config`. Hàm này sẽ phải trả về một đối tượng của `League\Flysystem\Filesystem`. Biến `$config` chứa các giá trị được xác định trong tập tin *config/filesystems.php* cho đĩa được chỉ định.

Khi bạn đã tạo và đăng ký nhà service provider của tiện ích mở rộng, bạn có thể sử dụng driver dropbox trong tập tin cấu hình *config/filesystems.php* của mình.