Course - Laravel Framework

Làm việc với Request

Class Illuminate\Http\Request của Laravel cung cấp một đối tượng để tương tác với HTTP request hiện tại đang được ứng dụng của bạn xử lý cũng như truy xuất đầu vào (input), cookie và tập tin đính kèm đã được gửi cùng với request.

Tags: request, laravel

Giới Thiệu

Class `Illuminate\Http\Request` của Laravel cung cấp một đối tượng để tương tác với HTTP request hiện tại đang được ứng dụng của bạn xử lý cũng như truy xuất đầu vào (input), cookie và tập tin đính kèm đã được gửi cùng với request.

Tương tác với Request

Truy cập vào request

Để có được một phiên bản đối tượng HTTP request hiện tại thông qua DI, bạn nên khai kiểu của class `Illuminate\Http\Request` trên hàm nặc danh của route hoặc phương thức trên controller của bạn. Đối tượng request sẽ tự động được đưa vào bởi *service container* của Laravel:

```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->input('name');

    //
  }
}
```

Như đã đề cập, bạn cũng có thể khai kiểu class `Illuminate\Http\Request` khi đóng tuyến. Vùng chứa dịch vụ sẽ tự động đưa yêu cầu đến vào vùng đóng khi nó được thực thi:

```php
use Illuminate\Http\Request;

Route::get('/', function (Request $request) {
  //
});
```

DI và Tham số route

Nếu phương thức *controller* của bạn cũng đang chờ đợi đầu vào từ một tham route, bạn nên liệt kê các tham số route sau các thư viện của bạn. Ví dụ: nếu route của bạn được khai báo như code bên dưới:

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

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

Bạn vẫn có thể khai kiểu cho class `Illuminate\Http\Request` và truy cập tham số route `id` của bạn bằng cách khai báo phương thức controller của bạn như sau:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
  /**
   * Update the specified user.
   *
   * @param  \Illuminate\Http\Request  $request
   * @param  string  $id
   * @return \Illuminate\Http\Response
   */
  public function update(Request $request, $id)
```

```
  {
    //
  }
}
```

Đường dẫn và phương thức của request

Đối tượng `Illuminate\Http\Request` cung cấp nhiều phương thức để kiểm tra HTTP request gửi đến và được mở rộng từ class `Symfony\Component\HttpFoundation\Request`. Chúng tôi sẽ thảo luận về một số phương thức quan trọng nhất dưới đây.

Lấy đường dẫn của request

Phương thức `path` trả về thông tin đường dẫn của request. Vì vậy, nếu request gửi đến nắm đường dẫn *http://example.com/foo/bar*, thì phương thức `path` sẽ trả về `"foo/bar"`:

```
$uri = $request->path();
```

Kiểm tra đường dẫn hoặt route của request

Phương thức `is` cho phép bạn xác minh đường dẫn request gửi đến có khớp với một pattern đã cho. Bạn có thể sử dụng ký tự * làm ký tự chọn tất cả (wildcard), khi vận dụng phương pháp này:

```
if ($request->is('admin/*')) {
  //
}
```

Phương thức `routeIs` dùng để kiểm tra route bằng tên route. Với phương thức `routeIs`, bạn có thể xác định được xem request gửi đến có khớp với một route đã có hay không bằng tham số tên route trong phương thức `routeIs`:

```
if ($request->routeIs('admin.*')) {
  //
}
```

Lấy URI của request

Để lấy URL đầy đủ trong request gửi đến, bạn có thể sử dụng phương thức `url` hoặc `fullUrl`. Phương thức `url` sẽ trả về URL mà không có chuỗi truy vấn (những thông tin đằng sau dấu chấm ? sẽ không thấy được), trong khi phương thức `fullUrl` bao gồm cả chuỗi truy vấn (những thông tin đằng sau dấu chấm ? có thể thấy được):

```
$url = $request->url();

$urlWithQueryString = $request->fullUrl();
```

Nếu bạn muốn nối dữ liệu chuỗi truy vấn vào URL hiện tại, bạn có thể gọi phương thức `fullUrlWithQuery`. Phương thức này sẽ merge mảng biến chuỗi truy vấn mà bạn muốn với chuỗi truy vấn hiện tại:

```
$request->fullUrlWithQuery(['type' => 'phone']);
```

Lấy method của request gửi đến

Request gửi đến có nhiều method, POST, GET, PUT, và DELETE và nhiều hơn nữa, để biết được method nào được request nắm giữ, thì khi đó phương thức `method` sẽ trả về HTTP method trong request. Bạn có thể sử dụng phương thức `isMethod` để xác minh HTTP method có khớp với một chuỗi đại diện cho method mà bạn đang muốn kiểm tra:

```
$method = $request->method();

if ($request->isMethod('post')) {
  //
}
```

Header của request

Bạn có thể lấy thông tin header của request từ đối tượng `Illuminate\Http\Request` bằng cách sử dụng phương thức `header`. Nếu không có header trong request, thì `null` sẽ được trả về. Tuy nhiên, phương thức `header` chấp nhận tham số thứ hai như một đối số tùy chọn sẽ được trả về nếu header không có trong request:

```php
$value = $request->header('X-Header-Name');

$value = $request->header('X-Header-Name', 'default');
```

Phương thức `hasHeader` có thể được sử dụng để xác định xem request có chứa một header đang yêu cầu hay không:

```php
if ($request->hasHeader('X-Header-Name')) {
  //
}
```

Khi bạn muốn lấy thông tin header xác thực Authorization, thì phương thức `bearerToken` có thể được sử dụng để lấy mã token bên trong header này. Nếu không có header này, thì một chuỗi trống sẽ được trả về:

```php
$token = $request->bearerToken();
```

Địa chỉ IP của request

Phương thức `ip` có thể được sử dụng để truy xuất địa chỉ IP của người truy cập đã thực hiện yêu cầu đối với ứng dụng của bạn:

```php
$ipAddress = $request->ip();
```

Content type trong request

Laravel cung cấp một số phương thức để kiểm tra các loại nội dung được yêu cầu của request gửi đến thông qua header là Accept. Đầu tiên, phương thức `getAcceptableContentTypes` sẽ trả về một mảng chứa tất cả các kiểu nội dung (content type) được request chấp nhận:

```php
$contentTypes = $request->getAcceptableContentTypes();
```

Phương thức `accept` cho nhập một mảng các kiểu nội dung (content type) và trả về `true`

nếu bất kỳ kiểu nội dung nào trong mảng được request chấp nhận. Nếu không, `false` sẽ được trả về:

```
if ($request->accepts(['text/html', 'application/json'])) {
  // ...
}
```

Bạn có thể sử dụng phương pháp `prefers` để xác định loại nội dung (content type) nào trong số các loại nội dung liệt kê được request ưu tiên nhất. Nếu không có loại nội dung nào được request chấp nhận, thì giá trị `null` sẽ được trả về:

```
$preferred = $request->prefers(['text/html', 'application/json']);
```

Vì nhiều ứng dụng chỉ phân phát HTML hoặc JSON, nên bạn có thể sử dụng phương thức `expectsJson` để nhanh chóng xác định xem request gửi đến có đang đợi phản hồi JSON hay không:

```
if ($request->expectsJson()) {
  // ...
}
```

PSR-7 trong request

Tiêu chuẩn PSR-7 chỉ định các interface cho các thông điệp HTTP, bao gồm các request và response. Nếu bạn muốn lấy một phiên bản của PSR-7 request thay cho Laravel request, trước tiên bạn sẽ cần cài đặt một vài thư viện mới. Laravel sử dụng component *Symfony HTTP Message Bridge* để chuyển đổi các request và response điển hình của Laravel thành các bản thực thi mới tương thích với PSR-7:

```
composer require symfony/psr-http-message-bridge
composer require nyholm/psr7
```

Một khi bạn đã cài đặt các thư viện này, bạn có thể nhận được các PSR-7 request bằng cách khai kiểu cho request interface trên hàm nặc danh của route hoặc phương thức của controller trong ứng dụng của bạn:

```php
use Psr\Http\Message\ServerRequestInterface;

Route::get('/', function (ServerRequestInterface $request) {
  //
});
```

Nếu bạn trả về một PSR-7 response từ một route hoặc controller, thì cho dù là gì đi chăng nữa, nó cũng sẽ tự động được chuyển đổi ngược trở lại thành một phiên bản Laravel response và sẽ được framework hiển thị.

Dữ liệu đầu vào

Lấy dữ liệu đầu vào

Cách lấy tất cả dữ liệu đầu vào

Bạn có thể truy xuất tất cả dữ liệu đầu vào của request gửi đến dưới dạng một mảng dữ liệu bằng cách sử dụng phương thức `all`. Phương pháp này có thể được áp dụng cho bất kỳ request gửi đến ứng dụng của bạn, nào là HTML form, XHR fetch:

```php
$input = $request->all();
```

Sử dụng phương thức `collect`, bạn có thể truy xuất tất cả dữ liệu đầu vào của request gửi đến ứng dụng dưới dạng collection:

```php
$input = $request->collect();
```

Phương thức `collect` cũng cho phép bạn truy xuất một tập hợp con của dữ liệu đầu vào dưới dạng một collection:

```php
$request->collect('users')->each(function ($user) {
  // ...
});
```

Cách lấy một giá trị đầu vào cụ thể

Chỉ bằng một vài phương thức đơn giản, bạn có thể truy cập tất cả thông tin người dùng được gửi từ `Illuminate\Http\Request` mà không cần lo lắng về HTTP method nào đã được kèm theo request. Bất kể HTTP method là gì, phương thức `input` có thể được sử dụng để truy xuất thông tin đầu vào của người dùng:

```
$name = $request->input('name');
```

Bạn có thể truyền một giá trị mặc định làm đối số thứ hai cho phương thức `input`. Giá trị này sẽ được trả về nếu giá trị đầu vào được yêu cầu không có trong request:

```
$name = $request->input('name', 'Sally');
```

Khi làm việc với các HTML form có chứa đầu vào là mảng, hãy sử dụng ký hiệu dấu chấm (.) để truy cập các mảng:

```
$name = $request->input('products.0.name');

$names = $request->input('products.*.name');
```

Bạn có thể gọi phương thức `input` mà không có bất kỳ đối số nào để truy xuất tất cả các giá trị đầu vào dưới dạng một mảng kiểu từ khóa:

```
$input = $request->input();
```

Cách lấy input từ chuỗi truy vấn

Trong khi phương thức `input` truy xuất các giá trị từ toàn bộ payload của request (bao gồm cả chuỗi truy vấn), thì phương thức `query` sẽ chỉ truy xuất các giá trị từ chuỗi truy vấn:

```
$name = $request->query('name');
```

Nếu dữ liệu giá trị chuỗi truy vấn được yêu cầu không có, thì đối số thứ hai cho phương

thức này sẽ được trả về:

```
$name = $request->query('name', 'Helen');
```

Bạn có thể gọi phương thức `query` mà không có bất kỳ đối số nào để truy xuất tất cả các giá trị chuỗi truy vấn dưới dạng một mảng kiểu từ khóa:

```
$query = $request->query();
```

Cách lấy dữ liệu JSON

Khi dữ liệu JSON được gửi đến ứng dụng của bạn, thì bạn có thể truy cập dữ liệu JSON thông qua phương thức `input` miễn là header *Content-Type* của request được đặt đúng theo giá trị `"application/json"`. Bạn thậm chí có thể sử dụng cú pháp dấu chấm (`.`) để truy xuất các giá trị được lồng trong các mảng JSON:

```
$name = $request->input('user.name');
```

Cách lấy các giá trị kiểu boolean

Khi xử lý các phần tử HTML như checkbox, ứng dụng của bạn có thể nhận các giá trị là các chuỗi. Ví dụ: `"true"` hoặc `"on"`. Để thuận tiện, bạn có thể sử dụng phương thức `boolean` để truy xuất các giá trị này dưới dạng boolean. Phương thức `boolean` trả về `true` cho các giá trị là `1`, `"1"`, `true`, `"true"`, `"on"` và `"yes"`. Tất cả các giá trị khác sẽ trả về `false`:

```
$archived = $request->boolean('archived');
```

Cách lấy giá trị ngày tháng

Để thuận tiện, các giá trị đầu vào có chứa ngày/giờ có thể được truy xuất dưới dạng các đối tượng Carbon bằng cách sử dụng phương thức `date`. Nếu request không chứa giá trị đầu vào nào khớp với tên yêu cầu, thì giá trị `null` sẽ được trả về:

```
$birthday = $request->date('birthday');
```

Đối số thứ hai và thứ ba trong phương thức `date` có thể được sử dụng để mô tả định dạng thời gian và múi giờ:

```
$elapsed = $request->date('elapsed', '!H:i', 'Europe/Madrid');
```

Nếu giá trị đầu vào có nhưng có định dạng không hợp lệ, thì một lỗi không hợp lệ `InvalidArgumentException` sẽ được đưa ra; do đó, bạn được khuyên nên kiểm tra đầu vào trước khi gọi phương thức `date`.

Cách lấy dữ liệu theo kiểu thuộc tính động

Bạn cũng có thể truy cập thông tin đầu vào của người dùng bằng cách sử dụng các thuộc tính động trên đối tượng `Illuminate\Http\Request`. Ví dụ: nếu một trong các HTML form của ứng dụng của bạn chứa field là `name`, thì bạn có thể truy cập giá trị của field này như sau:

```
$name = $request->name;
```

Khi sử dụng thuộc tính động, trước tiên Laravel sẽ tìm kiếm giá trị của thuộc tính trong payload của request. Nếu nó không xuất hiện, thì Laravel sẽ tiếp tục tìm kiếm field trong các tham số của route thích hợp.

```
route::get('/get/{name}', ...);

$name = $request->name;
```

Cách lấy một phần dữ liệu đầu vào

Nếu bạn cần truy xuất một phần nào đó của dữ liệu đầu vào, thì bạn có thể sử dụng phương thức `only` và `except`. Cả hai phương thức này đều chấp nhận một mảng (kiểu array) hoặc một danh sách động (kiểu list) các đối số yêu cầu:

```php
$input = $request->only(['username', 'password']);
$input = $request->only('username', 'password');
$input = $request->except(['credit_card']);
$input = $request->except('credit_card');
```

> **Chú ý:** Phương thức `only` trả về dữ liệu cho tất cả các cặp khóa/giá trị mà bạn yêu cầu; tuy nhiên, nó sẽ không trả về dữ liệu cho những cặp khóa/giá trị không tồn tại trong request.

Sự tồn tại của dữ liệu đầu vào

Bạn có thể sử dụng phương thức `has` để xác định xem dữ liệu đầu vào có tồn tại trong request hay không. Phương thức `has` trả về `true` nếu dữ kiện có trong request:

```php
if ($request->has('name')) {
  //
}
```

Khi được truyền vào một mảng, phương thức `has` sẽ kiểm tra xem tất cả các giá trị đang yêu cầu có tồn tại hay không:

```php
if ($request->has(['name', 'email'])) {
  //
}
```

Phương thức `whenHas` sẽ thực hàm nặc danh đã khai báo nếu có dữ kiện trong request:

```php
$request->whenHas('name', function ($input) {
  //
});
```

Một hàm nặc danh thứ hai có thể được truyền vào cho phương thức `whenHas` để thực thi nếu dữ kiện đang yêu cầu không có trong request:

```php
$request->whenHas('name', function ($input) {
  // The "name" value is present...
}, function () {
  // The "name" value is not present...
});
```

Phương thức `hasAny` trả về `true` nếu bất kỳ giá trị nào được yêu cầu tồn tại trong request:

```php
if ($request->hasAny(['name', 'email'])) {
  //
}
```

Nếu bạn muốn xác định xem một giá trị có tồn tại trong request và không trống hay không, thì bạn có thể sử dụng phương thức `filled`:

```php
if ($request->filled('name')) {
  //
}
```

Phương thức `whenFilled` sẽ thực thi hàm nặc danh đã truyền vào nếu một giá trị có tồn tại trong request và không trống:

```php
$request->whenFilled('name', function ($input) {
  //
});
```

Một hàm nặc danh thứ hai có thể được truyền vào phương thức `whenFilled` để thực thi nếu giá trị đang yêu cầu bị bỏ trống:

```php
$request->whenFilled('name', function ($input) {
  // The "name" value is filled...
}, function () {
  // The "name" value is not filled...
});
```

Để kiểm tra xem một khóa nào đó bị thiếu trong request hay không, thì bạn có thể sử dụng phương thức `missing`:

```
if ($request->missing('name')) {
  //
}
```

Dữ liệu bổ sung

Đôi khi bạn có thể cần phải thêm dữ liệu bổ sung theo cách thủ công vào dữ liệu đầu vào hiện có trên request. Để thực hiện điều này, bạn có thể sử dụng phương thức `merge`:

```
$request->merge(['votes' => 0]);
```

Phương thức `mergeIfMissing` có thể được sử dụng để bổ sung dữ liệu cho request, nếu các dữ liệu yêu cầu chưa tồn tại trong request:

```
$request->mergeIfMissing(['votes' => 0]);
```

Dữ liệu đã gửi

Laravel cho phép bạn trữ lại (cache) dữ liệu đã gửi từ một request khi phản hồi cho người dùng. Tính năng này đặc biệt hữu ích để điền lại các HTML form sau khi phát hiện lỗi kiểm tra đầu vào. Tuy nhiên, nếu bạn đang sử dụng các tính năng kiểm tra đầu vào có sẵn trong Laravel, thì có thể bạn sẽ không cần sử dụng trực tiếp bằng tay các phương pháp trữ đầu vào qua session flash, vì một số công cụ tích hợp của Laravel sẽ tự động gọi chúng.

Phương pháp flash vào session

Phương thức `flash` trên lớp `Illuminate\Http\Request` sẽ flash (chép ngắn hạn) dữ liệu đầu vào hiện tại vào trong session, để nó có thể truy cập trong lần gửi request tiếp theo của người dùng đối với ứng dụng:

```
$request->flash();
```

Bạn cũng có thể sử dụng các phương thức `flashOnly` và `flashExcept` để flash một phần dữ liệu yêu cầu vào trong session. Các phương thức này rất hữu ích khi loại bỏ các thông tin nhạy cảm như mật khẩu ra khỏi session:

```
$request->flashOnly(['username', 'email']);

$request->flashExcept('password');
```

Flash dữ liệu sau khi redirect

Vì bạn thường xuyên flash dữ liệu đầu vào cho session và sau đó chuyển hướng đến trang trước đó; do đó, Laravel cho bạn dễ dàng tạo flash lưu trữ với trang chuyển hướng bằng phương thức `withInput`:

```
return redirect('form')->withInput();

return redirect()->route('user.create')->withInput();

return redirect('form')->withInput(
  $request->except('password')
);
```

Cách lấy dữ liệu đã nhập

Để truy xuất đầu vào đã flash (lưu ngắn hạn) từ request trước đó, hãy gọi phương thức `old` trên đối tượng `Illuminate\Http\Request`. Phương thức `old` sẽ lấy dữ liệu đầu vào đã flash trước đó từ session:

```
$username = $request->old('username');
```

Laravel cũng cung cấp một hàm trợ giúp toàn cục `old`. Nếu bạn đang hiển thị dữ liệu đầu vào đã gửi trong Blade template, thì sẽ thuận tiện hơn khi sử dụng hàm toàn cục `old` để tái hiện lại HTML form. Nếu không có dữ liệu nào cho field đã cho, thì giá trị `null` sẽ được điền vào:

```
<input type="text" name="username" value="{{ old('username') }}">
```

Dữ liệu Cookie

Cách lấy dữ liệu từ cookie

Tất cả các cookie được tạo bởi framework Laravel đều được mã hóa và đánh dấu bằng mã xác thực, có nghĩa là chúng sẽ bị coi là không hợp lệ nếu đã bị người dùng thay đổi. Để truy xuất giá trị cookie từ request, hãy sử dụng phương thức `cookie` trên đối tượng `Illuminate\Http\Request`:

```
$value = $request->cookie('name');
```

Quy chuẩn dữ liệu đầu vào

Mặc định, Laravel có sẵn middleware `App\Http\Middleware\TrimStrings` và `App\Http\Middleware\ConvertEmptyStringsToNull` trong ngăn xếp middleware tổng bộ của ứng dụng của bạn. Các middleware này được liệt kê trong ngăn xếp middleware tổng bộ bởi class `App\Http\Kernel`. Các middleware này sẽ tự động trim (cắt bỏ phần thừa) tất cả các field là chuỗi được gửi kèm theo request, cũng như chuyển đổi bất kỳ field nào là chuỗi trống thành giá trị `null`. Điều này cho phép bạn không phải lo lắng về việc quy chuẩn hóa giá trị trong các route và controller của bạn.

Nếu bạn muốn vô hiệu hóa chức năng này, bạn có thể xóa hai middleware này ra khỏi ngăn xếp middleware tổng bộ của ứng dụng. Bây giờ, bạn có thể thực hiện bằng cách xóa chúng khỏi thuộc tính `$middleware` của class `App\Http\Kernel`.

Dữ liệu FILE

Cách lấy các tập tin upload

Bạn có thể truy xuất các tập tin đã được upload từ đối tượng `Illuminate\Http\Request` bằng cách sử dụng phương thức `file` hoặc sử dụng thuộc tính động. Phương thức `file` trả về một đối tượng của class `Illuminate\Http\UploadedFile`, class này mở rộng class `SplFileInfo` của PHP và cung cấp nhiều phương thức để tương tác với tập tin đã upload:

```php
$file = $request->file('photo');

$file = $request->photo;
```

Bạn có thể kiểm tra xem một tập tin có được upload hay không bằng phương thức `hasFile`:

```php
if ($request->hasFile('photo')) {
  //
}
```

Kiểm tra upload thành công

Ngoài việc kiểm tra xem tập tin có tồn tại hay không, bạn còn có thể xác định xem có vấn đề gì không khi tải tập tin lên với phương thức `isValid`:

```php
if ($request->file('photo')->isValid()) {
  //
}
```

Đường dẫn và phần mở rộng

Class `UploadedFile` cũng chứa các phương thức để truy cập đường dẫn đủ điều kiện của tập tin và phần mở rộng của nó. Phương thức `extension` sẽ phân tích phần mở rộng của tập tin dựa trên nội dung của nó. Tiện ích này có thể khác với tiện ích của người dùng:

```php
$path = $request->photo->path();

$extension = $request->photo->extension();
```

Các phương thức khác

Có nhiều phương thức khác có sẵn trên các đối tượng UploadedFile. Tham khảo tài liệu API của class để biết thêm thông tin về các phương thức này.

Lưu trữ các file đã upload

Để lưu trữ tập tin đã upload, bạn thường sẽ sử dụng một trong các hệ thống filesystem đã được cấu hình của mình. Class `UploadedFile` có phương thức `store` sẽ di chuyển tập tin đã tải lên vào một trong các đĩa của bạn, có thể là một vị trí trên hệ thống filesystem cục bộ của bạn hoặc một vị trí lưu trữ đám mây như Amazon S3.

Phương thức `store` cho truyền đường dẫn nơi mà tập tin sẽ được lưu trữ bên trong thư mục root của ứng dụng. Đường dẫn này không được chứa tên tập tin, vì một ID thống nhất sẽ tự động được cấp phát để dùng làm tên tập tin.

Phương thức `store` cũng cho phép truyền một đối số thứ hai tùy chọn cho tên của đĩa sẽ được sử dụng để lưu trữ tập tin. Phương thức sẽ trả về đường dẫn của tập tin liên quan đến root ứng dụng trên ổ đĩa:

```
$path = $request->photo->store('images');

$path = $request->photo->store('images', 's3');
```

Nếu bạn không muốn tên tập tin được tạo tự động, bạn có thể sử dụng phương thức `storeAs`, phương thức này cho truyền đường dẫn, tên tập tin và tên đĩa làm đối số của nó:

```
$path = $request->photo->storeAs('images', 'filename.jpg');

$path = $request->photo->storeAs('images', 'filename.jpg', 's3');
```

> Để biết thêm thông tin về lưu trữ tập tin trong Laravel, hãy xem toàn bộ tài liệu lưu trữ tập tin.

Cấu hình Proxy tin cậy

Khi chạy các ứng dụng của bạn mà trên hệ thống cân bằng tải mà sẽ chấm dứt chứng chỉ TLS/SSL, bạn có thể nhận thấy ứng dụng của mình đôi khi không tạo ra liên kết HTTPS khi sử dụng hàm tổng cục `url`. Điều này thường là do ứng dụng của bạn đang chuyển tiếp truy cập từ bộ cân bằng tải của bạn trên cổng 80 và không biết rằng nó sẽ tạo ra các liên kết an ninh.

Để giải quyết vấn đề này, bạn có thể sử dụng middleware `App\Http\Middleware`

`\TrustProxies` có trong ứng dụng Laravel của bạn, cho phép bạn nhanh chóng tùy chỉnh bộ cân bằng tải hoặc proxy mà ứng dụng của bạn nên tin cậy. Các proxy đáng tin cậy của bạn nên được liệt kê dưới dạng một mảng trên thuộc tính `$proxies` của middleware này. Ngoài việc định cấu hình proxy đáng tin cậy, bạn có thể định cấu hình header các proxy đáng tin cậy `$header`:

```php
<?php

namespace App\Http\Middleware;

use Illuminate\Http\Middleware\TrustProxies as Middleware;
use Illuminate\Http\Request;

class TrustProxies extends Middleware
{
    /**
     * The trusted proxies for this application.
     *
     * @var string|array
     */
    protected $proxies = [
        '192.168.1.1',
        '192.168.1.2',
    ];

    /**
     * The headers that should be used to detect proxies.
     *
     * @var int
     */
    protected $headers = Request::HEADER_X_FORWARDED_FOR | Request::HEADER_X_FORWARDED_HOST | Rec
}
```

> Nếu bạn đang sử dụng service cân bằng tải AWS Elastic, giá trị `$headers` của bạn phải là `Request::HEADER_X_FORWARDED_AWS_ELB`. Để biết thêm thông tin về các hằng số có thể được sử dụng trong thuộc tính `$headers`, hãy xem tài liệu của Symfony về proxy đáng tin cậy (trusting proxies).

Tin cậy toàn bộ proxy

Nếu bạn đang sử dụng service trên Amazon AWS hoặc nhà cung cấp bộ cân bằng tải khác, bạn có thể không biết địa chỉ IP thực sự của bộ cân bằng của mình. Trong trường hợp này, bạn có thể sử dụng `*` để tin cậy tất cả các proxy:

```
/**
 * The trusted proxies for this application.
 *
 * @var string|array
 */
protected $proxies = '*';
```

Cấu hình host tin cậy

Mặc định, Laravel sẽ phản hồi tất cả các request mà nó nhận được bất kể nội dung nào của header `Host` của HTTP request. Ngoài ra, giá trị của tiêu đề `Host` sẽ được sử dụng khi tạo ra URL tuyệt đối cho ứng dụng của bạn trong một web request.

Thông thường, bạn nên cấu hình máy chủ web của mình, chẳng hạn như Nginx hoặc Apache, sao cho chỉ gửi các request đến ứng dụng của bạn khi nó khớp với một tên máy chủ cụ thể. Tuy nhiên, nếu bạn không thể tùy chỉnh trực tiếp máy chủ web của mình cũng như không thể hướng dẫn Laravel chỉ phản hồi với một số tên máy chủ nhất định, thì bạn có thể thử thực hiện bằng cách bật middleware `App\Http\Middleware\TrustHosts` cho ứng dụng của mình.

Middleware `TrustHosts` đã được liệt kê trong ngăn xếp `$middleware` tổng bộ của ứng dụng của bạn; tuy nhiên, bạn cần phải bỏ comment để nó hoạt động. Trong phương thức `hosts` của middleware này, bạn có thể khai báo tên máy chủ mà ứng dụng của bạn sẽ phản hồi. Các request gửi đến có header giá trị `Host` khác sẽ bị từ chối:

```
/**
 * Get the host patterns that should be trusted.
 *
 * @return array
 */
public function hosts()
{
```

```
    return [
        'laravel.test',
        $this->allSubdomainsOfApplicationUrl(),
    ];
}
```

Phương thức trợ giúp `allSubdomainsOfApplicationUrl` sẽ trả về một biểu thức chính quy khớp với tất cả các miền phụ trong giá trị cấu hình `app.url` của ứng dụng của bạn. Phương thức trợ giúp này cung cấp một cách thuận tiện để thông qua tất cả các miền phụ của ứng dụng của bạn khi xây dựng một ứng dụng sử dụng miền phụ kiểu tự chọn (wildcard subdomain).