Course - Laravel Framework

Xác thực người dùng

Nhiều ứng dụng web cung cấp cách để người dùng của họ xác thực bằng ứng dụng và "đăng nhập". Việc triển khai tính năng này trong các ứng dụng web có thể là một nỗ lực phức tạp và tiềm ẩn nhiều rủi ro. Vì lý do này, Laravel sẽ cung cấp cho bạn các công cụ mà bạn cần để triển khai xác thực người dùng nhanh chóng, an toàn và dễ dàng.

Tags: authentication, xac thuc nguoi dung, laravel 8

Giới thiệu

Nhiều ứng dụng web cung cấp cách để người dùng của họ xác thực bằng ứng dụng và "đăng nhập". Việc triển khai tính năng này trong các ứng dụng web có thể là một nỗ lực phức tạp và tiềm ẩn nhiều rủi ro. Vì lý do này, Laravel sẽ cung cấp cho bạn các công cụ mà bạn cần để triển khai xác thực người dùng nhanh chóng, an toàn và dễ dàng.

Về cốt lõi, các cơ chế xác thực người dùng của Laravel được tạo thành từ các "guards" và "providers". Các guard xác định cách người dùng được xác thực cho mỗi request. Ví dụ: Laravel cung cấp guard trên phiên nhằm duy trì trạng thái đăng nhập bằng cách sử dụng cookie và lưu trữ session.

Các provider xác định cách người dùng được truy xuất từ bộ nhớ cố định của bạn. Laravel hỗ trợ truy xuất người dùng bằng Eloquent và chương trình tạo truy vấn CSDL. Tuy nhiên, bạn có thể tự do xác định các provider bổ sung khi cần thiết cho các ứng dụng của mình.

Tập tin cấu hình xác thực người dùng của ứng dụng của bạn được đặt tại *config/auth.php*. Tập tin này chứa một số tùy chọn được ghi lại một cách đầy đủ để điều chỉnh hoạt động của các service xác thực người dùng của Laravel.

> Guards và Providers không nên bị nhầm lẫn với "roles" và "permission". Để tìm hiểu thêm về cách phân quyền các hành động của người dùng thông qua phân quyền, vui lòng tham khảo tài liệu phân quyền người dùng.

Các công cụ để bắt đầu

Bạn muốn bắt đầu nhanh ư? Vậy thì hãy cài đặt bộ công cụ nhanh của Laravel trong ứng dụng Laravel mới. Sau khi migrate CSDL của bạn, hãy đưa trình duyệt của bạn đến */register* hoặc bất kỳ URL nào khác đã được chỉ định trong ứng dụng của bạn có liên quan đến xác thực người dùng. Bộ công cụ nhanh này sẽ chăm lo cho toàn bộ hệ thống xác thực người của bạn!

Ngay cả khi bạn chọn không sử dụng bộ công cụ nhanh trong ứng dụng Laravel cuối cùng của mình, thì việc cài đặt bộ công cụ nhanh Laravel Breeze cũng có thể là một cơ hội tuyệt vời để tìm hiểu cách triển khai tất cả chức năng xác thực người dùng của Laravel trong một dự án Laravel thực tế. Vì Laravel Breeze sẽ tạo controller, các route và view liên quan đến xác thực người dùng cho bạn, bạn có thể kiểm tra code trong các tập tin này để tìm hiểu cách các tính năng xác thực người dùng của Laravel có thể được triển khai.

Cơ sở dữ liệu

Mặc định, Laravel có một model `App\Models\User` được đặt trong thư mục *app/Models*. Model này có thể được sử dụng với driver xác thực Eloquent mặc định. Nếu ứng dụng của bạn không sử dụng Eloquent, bạn có thể sử dụng provider xác thực người dùng trên cơ sở dữ liệu qua việc sử dụng chương trình tạo truy vấn CSDL của Laravel.

Khi xây dựng schema cơ sở dữ liệu cho model `App\Models\User`, hãy đảm bảo cột "password" có độ dài ít nhất 60 ký tự. Tất nhiên, các bản migrate bảng người dùng được đặt trong các ứng dụng Laravel mới cũng đã tạo ra một cột vượt quá độ dài này.

Ngoài ra, bạn nên xác minh rằng bảng người dùng của bạn (hoặc tương đương) có chứa một cột chuỗi "remember_token", nó là nullable, và dài 100 ký tự. Cột này sẽ được sử dụng để lưu trữ token cho người dùng chọn tùy chọn "remember me" khi đăng nhập vào ứng dụng của bạn. Một lưu ý khác nữa là, Các migrate bảng người dùng mặc định được đặt trong các ứng dụng Laravel mới đã chứa sẵn cột này.

Tổng quan về hệ sinh thái

Laravel cung cấp một số gói liên quan đến xác thực. Trước khi tiếp tục, chúng tôi sẽ xem xét hệ sinh thái xác thực chung trong Laravel và thảo luận về mục đích dự định của từng gói.

Trước tiên, hãy xem xét cách thức mà chương trình xác thực người dùng hoạt động. Khi sử dụng trình duyệt web, người dùng sẽ cung cấp tên người dùng và mật khẩu của họ với các form đăng nhập. Nếu những thông tin này là chính xác, thì ứng dụng sẽ lưu trữ thông tin về người dùng đã được xác thực trong session được cấp cho mỗi người dùng trình duyệt.

Cookie trong trình duyệt có chứa ID session, nó cho phép các request tiếp theo đến ứng dụng có thể liên kết người dùng đúng với session được cấp phát. Sau khi nhận được cookie này, ứng dụng sẽ lấy dữ liệu session trên server dựa theo ID session này, lưu ý rằng thông tin xác thực đã được lưu trữ trong session và sẽ coi người dùng là "đã được xác thực".

Khi một dịch vụ từ xa (remote service) nào đó cần được xác thực trước khi truy cập API, thì cookie thường không được sử dụng để xác thực vì không có trình duyệt web. Thay vào đó, remote service sẽ gửi token API đến API theo mỗi request. Ứng dụng có thể xác thực token được gửi đến dựa trên bảng API token hợp lệ và "xác thực" request đang được thực hiện bởi người dùng đang được liên kết với API token đó.

Service xác thực tích hợp trong Laravel

Laravel bao gồm các service xác thực và session được tích hợp sẵn thường được truy cập với facade `Auth` và `Session`. Các tính năng này cung cấp tính xác thực người dùng dựa trên cookie cho các request được khởi tạo từ trình duyệt web. Chúng cung cấp các phương

thức cho phép bạn xác minh thông tin đăng nhập của người dùng và sau đó xác thực người dùng. Ngoài ra, các service này sẽ tự động lưu trữ dữ liệu xác thực phù hợp trong session của người dùng và cấp cookie liên quan người dùng. Trong tài liệu này có thảo luận về cách sử dụng các service này.

Bộ công cụ nhanh

Như đã thảo luận trong tài liệu này, bạn có thể tương tác với các service xác thực này theo cách thủ công để xây dựng class xác thực riêng cho ứng dụng của bạn. Tuy nhiên, để giúp bạn bắt đầu nhanh hơn, chúng tôi đã phát hành các package miễn phí cung cấp khung sườn mạnh mẽ, hiện đại của toàn bộ class xác thực. Các gói này là *Laravel Breeze*, *Laravel Jetstream* và *Laravel Fortify*.

Laravel Breeze gồm một sự thực thi đơn giản, tối thiểu cho tất cả các tính năng xác thực của Laravel, bao gồm đăng nhập, đăng ký, đặt lại mật khẩu, xác minh email và xác nhận mật khẩu. Phần view của Laravel Breeze bao gồm các template Blade đơn giản được tạo style bằng Tailwind CSS. Để bắt đầu, hãy xem tài liệu về bộ công cụ nhanh trong ứng dụng của Laravel.

Laravel Fortify là một phần mềm xác thực headless cho Laravel triển khai nhiều tính năng mà sẽ được tìm thấy trong tài liệu này, bao gồm xác thực dựa trên cookie cũng như các tính năng khác như xác thực two-factor và xác minh email. *Fortify* cung cấp phần mềm xác thực cho *Laravel Jetstream* hoặc có thể được sử dụng độc lập kết hợp với *Laravel Sanctum* để cung cấp xác thực người dùng cho một SPA cần xác thực với Laravel.

Laravel Jetstream là một bộ công cụ nhanh cho ứng dụng, nó mạnh mẽ khi sử dụng và hiển thị các serivce xác thực của *Laravel Fortify* với giao diện người dùng đẹp, hiện đại được cung cấp bởi Tailwind CSS, Livewire và/hoặc Inertia.js. *Laravel Jetstream* bao gồm hỗ trợ tùy chọn cho xác thực two-factor, hỗ trợ nhóm, quản lý session trình duyệt, quản lý hồ sơ và tích hợp sẵn với *Laravel Sanctum* để cung cấp xác thực API token. Các service xác thực API của Laravel sẽ được thảo luận bên dưới.

Service cho API xác thực của Laravel

Laravel cung cấp hai package tùy chọn để hỗ trợ bạn quản lý API token và xác thực các request được thực hiện với API token: Passport và Sanctum. Xin lưu ý rằng những thư viện này và thư viện xác thực dựa trên cookie tích hợp của Laravel không loại trừ lẫn nhau. Các thư viện này chủ yếu tập trung vào xác thực API token trong khi các service xác thực tích hợp tập trung vào việc xác thực trình duyệt dựa trên cookie. Nhiều ứng dụng sẽ sử dụng cả service xác thực dựa trên cookie tích hợp của Laravel và một trong các package xác thực API của Laravel.

Passport

Passport là service provider xác thực OAuth2, cung cấp nhiều "kiểu hộ trợ" OAuth2 cho phép bạn phát hành nhiều loại token khác nhau. Nói chung, đây là một package mạnh mẽ và phức tạp để xác thực API. Tuy nhiên, hầu hết các ứng dụng không yêu cầu các tính năng phức tạp được cung cấp bởi thông số kỹ thuật OAuth2, điều này có thể gây nhầm lẫn cho cả người dùng và nhà phát triển. Ngoài ra, các nhà phát triển trong lịch sử đã nhầm lẫn về cách xác thực ứng dụng SPA hoặc ứng dụng di động bằng cách sử dụng provider xác thực OAuth2 như Passport.

Sanctum

Để đối phó với sự phức tạp của OAuth2 và sự nhầm lẫn của nhà phát triển, chúng tôi bắt đầu xây dựng một package xác thực đơn giản hơn, hợp lý hơn có thể xử lý cả web request của bên thứ nhất từ trình duyệt web và yêu cầu API thông qua token. Mục tiêu này đã được thực hiện với việc phát hành *Laravel Sanctum*, nên được coi là package xác thực được ưu tiên và đề xuất cho các ứng dụng sẽ cung cấp giao diện người dùng web của bên thứ nhất ngoài API hoặc sẽ được cung cấp bởi ứng dụng một trang (SPA) tồn tại tách biệt với ứng dụng Laravel phụ trợ hoặc các ứng dụng cung cấp ứng dụng di động.

Laravel Sanctum là một gói xác thực web/API kết hợp có thể quản lý toàn bộ quy trình xác thực ứng dụng của bạn. Điều này có thể thực hiện được vì khi các ứng dụng dựa trên Sanctum nhận được request, trước tiên, Sanctum sẽ xác định xem request có bao gồm cookie có session nào tham chiếu đến session được xác thực hay không. Sanctum thực hiện điều này bằng cách gọi các service xác thực tích hợp sẵn của Laravel mà chúng ta đã thảo luận trước đó. Nếu request không được xác thực qua cookie có session, Sanctum sẽ kiểm tra request đối với API token. Nếu có API token, Sanctum sẽ xác thực request bằng token đó. Để tìm hiểu thêm về quy trình này, vui lòng tham khảo tài liệu "cách hoạt động" của Sanctum.

Laravel Sanctum là package API mà chúng tôi đã chọn để đưa vào bộ công cụ nhanh cho ứng dụng *Laravel Jetstream* vì chúng tôi tin rằng nó phù hợp nhất với phần lớn các nhu cầu xác thực của ứng dụng web.

Tóm tắt & Chọn stack cho bạn

Tóm lại, nếu ứng dụng của bạn sẽ được truy cập bằng trình duyệt và bạn đang xây dựng một ứng dụng Laravel nguyên khối, ứng dụng của bạn sẽ sử dụng các dịch vụ xác thực tích hợp sẵn của Laravel.

Tiếp theo, nếu ứng dụng của bạn cung cấp API sẽ được sử dụng bởi các bên thứ ba, bạn sẽ

chọn giữa Passport hoặc Sanctum để cung cấp API token xác thực cho ứng dụng của mình. Nói chung, Sanctum nên được ưu tiên khi có thể vì nó là một giải pháp đơn giản, hoàn chỉnh để xác thực API, xác thực SPA và xác thực di động, bao gồm hỗ trợ cho "scopes" hoặc "abilities".

Nếu bạn đang xây dựng một ứng dụng một trang (SPA) sẽ được hỗ trợ bởi phần mềm phụ trợ Laravel, bạn nên sử dụng *Laravel Sanctum*. Khi sử dụng Sanctum, bạn sẽ cần phải triển khai thủ công các route xác thực phụ trợ của riêng mình hoặc sử dụng *Laravel Fortify* như một service phụ trợ xác thực headless cung cấp các route và controller cho các tính năng như đăng ký, đặt lại mật khẩu, xác minh email, v.v.

Passport có thể được chọn khi ứng dụng của bạn hoàn toàn cần tất cả các tính năng được cung cấp bởi yêu cầu kỹ thuật OAuth2.

Và, nếu bạn muốn bắt đầu nhanh chóng, chúng tôi vui mừng giới thiệu *Laravel Jetstream* như một cách nhanh chóng để bắt đầu một ứng dụng Laravel mới đã sử dụng stack xác thực ưu tiên của chúng tôi gồm các service xác thực tích hợp sẵn của Laravel lẫn *Laravel Sanctum*.

Xác thực người dùng nhanh

> **Chú ý:** Phần này của tài liệu thảo luận về việc xác thực người dùng thông qua bộ công cụ nhanh cho ứng dụng Laravel, bao gồm khung sườn giao diện người dùng để giúp bạn bắt đầu nhanh chóng. Nếu bạn muốn tích hợp trực tiếp với các hệ thống xác thực của Laravel, hãy xem tài liệu về cách xác thực người dùng theo cách thủ công.

Cài đặt một bộ công cụ nhanh do bạn chọn

Trước tiên, bạn nên cài đặt một bộ công cụ nhanh cho ứng dụng Laravel. Các bộ công cụ hiện tại của chúng tôi là, *Laravel Breeze* và *Laravel Jetstream*, chúng sẽ cung cấp các điểm khởi đầu được thiết kế đẹp mắt để tích hợp tính năng xác thực vào ứng dụng Laravel mới của bạn.

Laravel Breeze là một sự thực thi đơn giản, tối thiểu của tất cả các tính năng xác thực của Laravel, bao gồm đăng nhập, đăng ký, đặt lại mật khẩu, xác minh email và xác nhận mật khẩu. Phần view của *Laravel Breeze* được tạo thành từ các template Blade đơn giản được tạo style với Tailwind CSS. Breeze cũng cung cấp tùy chọn khung sườn dựa trên mặc định như sử dụng Vue hoặc React.

Laravel Jetstream là một bộ khởi động nhanh xác thực cho ứng dụng mạnh mẽ hơn, bao gồm

hỗ trợ tạo khung cho ứng dụng của bạn với *Livewire* hoặc *Inertia.js* và *Vue*. Ngoài ra, *Jetstream* có hỗ trợ tùy chọn cho xác thực two-factor, quản lý nhóm, quản lý hồ sơ, quản lý session trình duyệt, hỗ trợ API qua *Laravel Sanctum*, xóa tài khoản, v.v.

Truy xuất dữ liệu xác thực người dùng

Sau khi cài đặt bộ công cụ xác thực và cho phép người dùng đăng ký và xác thực với ứng dụng của bạn, bạn thường sẽ cần phải tương tác với người dùng hiện đã được xác thực. Trong khi xử lý một request được gửi đến, bạn có thể truy cập vào người dùng đã xác thực thông qua phương thức `user` của facade `Auth`:

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

// Retrieve the currently authenticated user...
$user = Auth::user();

// Retrieve the currently authenticated user's ID...
$id = Auth::id();
```

Ngoài ra, sau khi người dùng được xác thực, bạn có thể truy cập người dùng đã được xác thực với đối tượng `Illuminate\Http\Request`. Hãy nhớ rằng, các class khi được khai báo kiểu sẽ tự động được đưa vào các phương thức của controller của bạn. Bằng cách nhập khai báo kiểu cho đối tượng `Illuminate\Http\Request`, thì bạn sẽ có thể có được quyền truy cập thuận tiện vào người dùng đã xác thực từ bất kỳ phương thức nào của controller nào trong ứng dụng của bạn thông qua phương thức `user` trên đối tượng request:

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class FlightController extends Controller
{
  /**
   * Update the flight information for an existing flight.
```

```
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function update(Request $request)
    {
        // $request->user()
    }
}
```

Xác định xem người dùng hiện tại có được xác thực hay không

Để xác định xem người dùng thực hiện yêu cầu HTTP gửi đến có được xác thực hay không, bạn có thể sử dụng phương thức `check` trên facade `Auth`. Phương thức này sẽ trả về `true` nếu người dùng được xác thực:

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

if (Auth::check()) {
    // The user is logged in...
}
```

Mặc dù có thể xác định xem người dùng có được xác thực hay không bằng cách sử dụng phương thức check, thông thường bạn sẽ sử dụng phần mềm trung gian để xác minh rằng người dùng được xác thực trước khi cho phép người dùng truy cập vào các routes/controller nhất định. Để tìm hiểu thêm về điều này, hãy xem tài liệu về bảo mật các route.

Bảo mật route

Middleware của route có thể được sử dụng để chỉ ưu tiên cho phép người dùng đã xác thực truy cập vào một route nào đó. Laravel cung cấp middleware `auth`, tham chiếu đến class `Illuminate\Auth\Middleware\Authenticate`. Vì middleware này đã được đăng ký trong nhân HTTP kernel của ứng dụng của bạn, tất cả những gì bạn cần việc cần làm là đính kèm middleware vào một khai báo route:

```
Route::get('/flights', function () {
```

```
  // Only authenticated users may access this route...
})->middleware('auth');
```

Chuyển hướng người dùng khách

Khi middleware `auth` phát hiện người dùng chưa được xác thực, nó sẽ chuyển hướng người dùng đến route có tên `login`. Bạn có thể sửa đổi điểm đến này bằng cách sửa chữa cập nhật chức năng `redirectTo` trong tập tin *app/Http/Middleware/Authenticate.php* của ứng dụng:

```
/**
 * Get the path the user should be redirected to.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return string
 */
protected function redirectTo($request)
{
    return route('login');
}
```

Khai báo lớp bảo mật

Khi đính kèm middleware `auth` vào một route, bạn cũng có thể chỉ định "lớp bảo mật" nào sẽ được sử dụng để xác thực người dùng. Lớp bảo mật được chỉ định phải tương ứng với một trong các khóa trong mảng liệt kê lớp bảo vệ của tập tin cấu hình *auth.php* của bạn:

```
Route::get('/flights', function () {
  // Only authenticated users may access this route...
})->middleware('auth:admin');
```

Hạn chế lần đăng nhập sai

Nếu bạn đang sử dụng bộ công cụ nhanh *Laravel Breeze* hoặc *Laravel Jetstream*, thì cơ chế rate limit sẽ tự động được áp dụng cho các lần đăng nhập. Theo mặc định, người dùng sẽ không thể đăng nhập trong một phút nếu họ không cung cấp thông tin đăng nhập chính xác sau

nhiều lần thử. Cơ chế throttling lọc unique đối với username/email và IP của họ.

> Nếu bạn muốn áp dụng rate limit các route khác trong ứng dụng của mình, hãy xem tài liệu giới hạn tỷ lệ.

Xác thực người dùng theo cách thủ công

Bạn không bắt buộc phải sử dụng xác thực người dùng được kèm với bộ khởi động nahnh của Laravel. Nếu bạn chọn không sử dụng các tính năng này, bạn sẽ cần quản lý xác thực người dùng bằng cách sử dụng trực tiếp các class xác thực người dùng Laravel.

Chúng ta sẽ truy cập các service xác thực của Laravel thông qua facade `Auth`, vì vậy chúng ta cần đảm bảo chèn facade `Auth` ở đầu class. Tiếp theo, hãy kiểm tra phương thức `attempt`. Phương thức `attempt` thường được sử dụng để xử lý các các lần xác thực từ form "đăng nhập" ứng dụng của bạn. Nếu xác thực thành công, bạn nên tạo lại session của người dùng để ngăn chặn việc cố định session:

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class LoginController extends Controller
{
  /**
   * Handle an authentication attempt.
   *
   * @param  \Illuminate\Http\Request  $request
   * @return \Illuminate\Http\Response
   */
  public function authenticate(Request $request)
  {
    $credentials = $request->validate([
      'email' => ['required', 'email'],
      'password' => ['required'],
    ]);
```

```
    if (Auth::attempt($credentials)) {
      $request->session()->regenerate();

      return redirect()->intended('dashboard');
    }

    return back()->withErrors([
      'email' => 'The provided credentials do not match our records.',
    ])->onlyInput('email');
  }
}
```

Phương thức `attempt` chấp nhận một mảng các cặp khóa/giá trị làm đối số đầu tiên của nó. Các giá trị trong mảng sẽ được sử dụng để tìm người dùng trong bảng cơ sở dữ liệu của bạn. Vì vậy, trong ví dụ trên, người dùng sẽ được truy xuất bằng giá trị của cột `email`. Nếu người dùng được tìm thấy, mật khẩu băm được lưu trữ trong cơ sở dữ liệu sẽ được đem đi so sánh với giá trị `password` được truyền cho phương thức thông qua mảng. Bạn không nên băm giá trị mật khẩu trong request đã gửi đến, vì framework sẽ tự động băm giá trị trước khi so sánh nó với mật khẩu đã băm trong cơ sở dữ liệu. Một session đã xác thực sẽ được bắt đầu cho người dùng nếu hai mật khẩu băm khớp nhau.

Hãy nhớ rằng, các service xác thực của Laravel sẽ truy xuất người dùng từ cơ sở dữ liệu của bạn dựa trên cấu hình "provider" bảo vệ xác thực của bạn. Trong tập tin cấu hình *config/auth.php* mặc định, nhà cung cấp người dùng Eloquent được chỉ định và nó được hướng dẫn sử dụng mô hình *App\Models\User* khi truy xuất người dùng. Bạn có thể thay đổi các giá trị này trong tập tin cấu hình của mình dựa trên nhu cầu của ứng dụng của bạn.

Phương thức `attempt` sẽ trả về `true` nếu xác thực thành công. Nếu không, `false` sẽ được trả về.

Phương thức `intended` được cung cấp bởi trình điều hướng của Laravel, nó sẽ chuyển hướng người dùng đến URL mà họ đang cố gắng truy cập trước khi bị middleware xác thực chặn. Một URI dự phòng có thể được cấp cho phương thức này trong trường hợp không có đích đến dự kiến.

Chỉ định các điều kiện bổ sung

Nếu muốn, bạn cũng có thể thêm các điều kiện truy vấn bổ sung vào truy vấn xác thực ngoài email và mật khẩu của người dùng. Để thực hiện điều này, chúng ta có thể chỉ cần

thêm các điều kiện truy vấn vào mảng được truyền cho phương thức attempt. Ví dụ: chúng ta có thể xác minh rằng người dùng được đánh dấu là "active":

```
if (Auth::attempt(['email' => $email, 'password' => $password, 'active' => 1])) {
  // Authentication was successful...
}
```

> **Chú ý:** Trong những ví dụ này, email không phải là một tùy chọn bắt buộc, nó chỉ được dùng làm ví dụ. Bạn nên sử dụng bất kỳ tên cột nào tương ứng với "username" trong bảng cơ sở dữ liệu của bạn.

Phương thức `attemptWhen`, nhận một hàm nặc danh làm đối số thứ hai của nó, có thể được sử dụng để thực hiện kiểm tra mở rộng người dùng tiềm năng trước khi thực sự xác thực người dùng. Hàm này nhận được người dùng tiềm năng và phải trả về `true` hoặc `false` để cho biết liệu người dùng có thể được xác thực hay không:

```
if (Auth::attemptWhen([
  'email' => $email,
  'password' => $password,
], function ($user) {
  return $user->isNotBanned();
})) {
  // Authentication was successful...
}
```

Truy cập các đối tượng guard cụ thể

Thông qua phương thức `guard` của facade `Auth`, bạn có thể chỉ định đối tượng guard nào bạn muốn sử dụng khi xác thực người dùng. Điều này cho phép bạn quản lý xác thực cho các phần riêng biệt của ứng dụng bằng cách sử dụng các model hoặc bảng người dùng có thể xác thực hoàn toàn riêng biệt.

Tên của guard được truyền vào phương thức `guard` phải tương ứng với một trong các lớp bảo vệ được cấu hình trong tập tin cấu hình *auth.php* của bạn:

```
if (Auth::guard('admin')->attempt($credentials)) {
  // ...
```

```
}
```

Ghi nhớ đăng nhập

Nhiều ứng dụng web cung cấp hộp kiểm "remember me" trên biểu mẫu form đăng nhập của họ. Nếu bạn muốn cung cấp chức năng "remember me" trong ứng dụng của mình, bạn có thể truyền một giá trị boolean làm đối số thứ hai cho phương thức `attempt`.

Khi giá trị này là `true`, Laravel sẽ giữ cho người dùng được xác thực vô thời hạn hoặc cho đến khi họ đăng xuất theo cách thủ công. Bảng `users` của bạn phải bao gồm cột `remember_token`, cột này sẽ được sử dụng để lưu trữ mã thông báo "remember me". Phần cấu tạo (migration) của bảng `users` được đi kèm với các ứng dụng Laravel mới tạo đã chứa sẵn cột này:

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

if (Auth::attempt(['email' => $email, 'password' => $password], $remember)) {
  // The user is being remembered...
}
```

Các phương thức xác thực khác

Xác thực một đối tượng người dùng

Nếu bạn cần đặt một đối tượng người dùng đang có nào đó làm người dùng hiện được xác thực, thì bạn có thể chuyển cá thể người dùng đó cho phương thức login của facade Auth. Đối tượng người dùng nhất định phải là bản thực thi của contract `Illuminate\Contracts\Auth\Authenticatable`. Model `App\Models\User` có trong Laravel đã triển khai interface này. Phương thức xác thực này hữu ích khi bạn đã có đối tượng người dùng hợp lệ, chẳng hạn như ngay sau khi người dùng đăng ký với ứng dụng của bạn:

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

Auth::login($user);
```

Bạn có thể truyền một giá trị boolean làm đối số thứ hai cho phương thức đăng nhập. Giá trị này cho biết chức năng "remember me" có được chờ đợi cho session được xác thực hay không. Hãy nhớ rằng, điều này có nghĩa là session sẽ được xác thực vô thời hạn hoặc cho đến khi người dùng đăng xuất ứng dụng theo cách thủ công:

```
Auth::login($user, $remember = true);
```

Nếu cần, bạn có thể chỉ định đối tượng guard để xác thực trước khi gọi phương thức login:

```
Auth::guard('admin')->login($user);
```

Xác thực người dùng bằng ID

Để xác thực người dùng bằng khóa chính của bản ghi cơ sở dữ liệu của họ, bạn có thể sử dụng phương thức `loginUsingId`. Phương thức này chấp nhận khóa chính của người dùng mà bạn muốn xác thực:

```
Auth::loginUsingId(1);
```

Bạn có thể truyền một giá trị boolean làm đối số thứ hai cho phương thức `loginUsingId`. Giá trị này cho biết chức năng "remember me" có được mong chờ cho session được xác thực hay không. Hãy nhớ rằng, điều này có nghĩa là session sẽ được xác thực vô thời hạn hoặc cho đến khi người dùng đăng xuất ứng dụng theo cách thủ công:

```
Auth::loginUsingId(1, $remember = true);
```

Xác thực người dùng một lần

Bạn có thể sử dụng phương thức once để xác thực người dùng với ứng dụng cho một yêu cầu duy nhất. Không có session hoặc cookie nào sẽ được sử dụng khi gọi phương thức này:

```
if (Auth::once($credentials)) {
  //
}
```

Xác thực HTTP Basic

Xác thực HTTP basic cung cấp một cách nhanh chóng để xác thực người dùng ứng dụng của bạn mà không cần thiết lập trang "đăng nhập" chuyên dụng. Để bắt đầu, hãy đính kèm middleware `auth.basic` vào một route. Middleware `auth.basic` được bao gồm trong khuôn khổ Laravel, vì vậy bạn không cần phải xác định nó:

```
Route::get('/profile', function () {
  // Only authenticated users may access this route...
})->middleware('auth.basic');
```

Khi middleware đã được đính kèm vào route, thì bạn sẽ tự động được nhắc nhập thông tin đăng nhập khi truy cập route trong trình duyệt của mình. Theo mặc định, middleware `auth.basic` sẽ giả định cột `email` trên bảng cơ sở dữ liệu người dùng của bạn là "username" của người dùng.

Lưu ý về FastCGI

Nếu bạn đang sử dụng PHP FastCGI và Apache để phục vụ ứng dụng Laravel của mình, xác thực HTTP cơ bản có thể không hoạt động chính xác. Để khắc phục những sự cố này, các dòng sau có thể được thêm vào tệp *.htaccess* của ứng dụng của bạn:

```
RewriteCond %{HTTP:Authorization} ^(.+)$
RewriteRule .* - [E=HTTP_AUTHORIZATION:%{HTTP:Authorization}]
```

Xác thực HTTP basic với stateless

Bạn cũng có thể sử dụng xác thực HTTP basic mà không cần đặt cookie định danh người dùng trong session. Điều này đặc biệt hữu ích khi bạn chọn sử dụng xác thực HTTP để xác thực các yêu cầu tới API ứng dụng của mình. Để thực hiện điều này, hãy khai báo một middleware gọi phương thức `onceBasic`. Nếu không có phản hồi nào được trả lại bởi phương thức `onceBasic`, yêu cầu có thể được truyền thêm vào ứng dụng:

```
<?php

namespace App\Http\Middleware;
```

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

class AuthenticateOnceWithBasicAuth
{
  /**
   * Handle an incoming request.
   *
   * @param  \Illuminate\Http\Request  $request
   * @param  \Closure  $next
   * @return mixed
   */
  public function handle($request, $next)
  {
    return Auth::onceBasic() ?: $next($request);
  }

}
```

Tiếp theo, đăng ký middleware route và đính kèm nó vào một route bạn muốn:

```php
Route::get('/api/user', function () {
  // Only authenticated users may access this route...
})->middleware('auth.basic.once');
```

Đăng xuất

Để đăng xuất người dùng khỏi ứng dụng của bạn theo cách thủ công, bạn có thể sử dụng phương thức `logout` được cung cấp bởi facade `Auth`. Thao tác này sẽ xóa thông tin xác thực khỏi session của người dùng để các yêu cầu tiếp theo không được xác thực.

Ngoài việc gọi phương thức logout, bạn nên làm mất hiệu lực session của người dùng và tạo lại mã thông báo CSRF của họ. Sau khi đăng xuất người dùng, bạn thường sẽ chuyển hướng người dùng đến thư mục gốc của ứng dụng của bạn:

```php
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;
```

```php
/**
 * Log the user out of the application.
 *
 * @param  \Illuminate\Http\Request  $request
 * @return \Illuminate\Http\Response
 */
public function logout(Request $request)
{
  Auth::logout();

  $request->session()->invalidate();

  $request->session()->regenerateToken();

  return redirect('/');
}
```

Vô hiệu hóa session trên các thiết bị khác

Laravel cũng cung cấp cơ chế làm mất hiệu lực và "đăng xuất" session của người dùng đang hoạt động trên các thiết bị khác mà không làm mất hiệu lực session trên thiết bị hiện tại của họ. Tính năng này thường được sử dụng khi người dùng đang thay đổi hoặc cập nhật mật khẩu của họ và bạn muốn làm mất hiệu lực các session trên các thiết bị khác trong khi vẫn xác thực thiết bị hiện tại.

Trước khi bắt đầu, bạn nên đảm bảo rằng middleware `Illuminate\Session\Middleware\AuthenticateSession` hiện diện và chưa được nhận xét trong nhóm middleware web `App\Http\Kernel` của bạn:

```php
'web' => [
  // ...
  \Illuminate\Session\Middleware\AuthenticateSession::class,
  // ...
],
```

Sau đó, bạn có thể sử dụng phương thức `logoutOtherDevices` được cung cấp bởi Auth.

Phương thức này yêu cầu người dùng xác nhận mật khẩu hiện tại của họ, mật khẩu mà ứng dụng của bạn phải chấp nhận thông qua biểu mẫu đầu vào:

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

Auth::logoutOtherDevices($currentPassword);
```

Khi phương thức `logoutOtherDevices` được gọi, các session khác của người dùng sẽ bị vô hiệu hoàn toàn, có nghĩa là họ sẽ bị "đăng xuất" khỏi tất cả các lớp bảo vệ mà họ đã được xác thực trước đó.

Mật khẩu xác nhận

Trong khi xây dựng ứng dụng của mình, đôi khi bạn có thể có các hành động yêu cầu người dùng xác nhận mật khẩu của họ trước khi hành động được thực hiện hoặc trước khi người dùng được điều hướng đến một khu vực nhạy cảm của ứng dụng. Laravel bao gồm middleware được tích hợp sẵn để làm cho quá trình này trở nên dễ dàng. Việc triển khai tính năng này sẽ yêu cầu bạn khai báo hai route: một route để hiển thị dạng hiển thị nhằm yêu cầu người dùng xác nhận mật khẩu của họ và một route khác để xác nhận rằng mật khẩu hợp lệ và chuyển hướng người dùng đến đích dự kiến của họ.

> Tài liệu sau đây thảo luận về cách tích hợp trực tiếp với các tính năng xác nhận mật khẩu của Laravel; tuy nhiên, nếu bạn muốn bắt đầu nhanh hơn, bộ dụng cụ nhanh của ứng dụng Laravel sẽ luôn hỗ trợ cho tính năng này!

Cấu hình

Sau khi xác nhận mật khẩu của họ, người dùng sẽ không được yêu cầu xác nhận lại mật khẩu của họ trong ba giờ. Tuy nhiên, bạn có thể cấu hình khoảng thời gian trước khi người dùng được nhắc nhập lại mật khẩu của họ bằng cách thay đổi giá trị của giá trị cấu hình `password_timeout` trong tệp cấu hình *config/auth.php* của ứng dụng của bạn.

Phân tuyến (Routing)

Form để xác nhận mật khẩu

Đầu tiên, chúng tôi sẽ khai báo một route để hiển thị trang web yêu cầu người dùng xác nhận mật khẩu của họ:

```php
Route::get('/confirm-password', function () {
  return view('auth.confirm-password');
})->middleware('auth')->name('password.confirm');
```

Như bạn có thể mong đợi, view được trả về bởi route này phải có một biểu mẫu chứa trường mật khẩu. Ngoài ra, hãy đưa văn bản vào view để giải thích rằng người dùng đang vào vùng được bảo vệ của ứng dụng và phải xác nhận mật khẩu của họ.

Xác nhận mật khẩu

Tiếp theo, chúng tôi sẽ khai báo một route sẽ xử lý yêu cầu biểu mẫu từ view "xác nhận mật khẩu". Route này sẽ chịu trách nhiệm xác thực mật khẩu và chuyển hướng người dùng đến đích dự kiến của họ:

```php
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Redirect;

Route::post('/confirm-password', function (Request $request) {
  if (! Hash::check($request->password, $request->user()->password)) {
    return back()->withErrors([
      'password' => ['The provided password does not match our records.']
    ]);
  }

  $request->session()->passwordConfirmed();

  return redirect()->intended();
})->middleware(['auth', 'throttle:6,1']);
```

Trước khi tiếp tục, hãy xem xét route này chi tiết hơn. Đầu tiên, trường mật khẩu của yêu cầu được xác định là thực sự khớp với mật khẩu của người dùng đã xác thực. Nếu mật khẩu hợp lệ, chúng tôi cần thông báo cho session của Laravel rằng người dùng đã xác nhận mật khẩu của họ. Phương thức mật khẩu đã xác nhận sẽ đặt một dấu thời gian trong phiên của người dùng mà Laravel có thể sử dụng để xác định thời điểm người dùng xác nhận mật khẩu của họ lần cuối. Cuối cùng, chúng tôi có thể chuyển hướng người dùng đến đích dự kiến của họ.

Bảo vệ các route

Bạn nên đảm bảo rằng bất kỳ route nào thực hiện một hành động yêu cầu xác nhận mật khẩu gần đây đều được gán cho middleware là `password.confirm`. Middleware này được bao gồm trong cài đặt mặc định của Laravel và sẽ tự động lưu trữ điểm đến dự kiến của người dùng trong session để người dùng có thể được chuyển hướng đến vị trí đó sau khi xác nhận mật khẩu của họ. Sau khi lưu trữ đích dự kiến của người dùng trong session, middleware sẽ chuyển hướng người dùng đến đường dẫn có tên `password.confirm`:

```
Route::get('/settings', function () {
  // ...
})->middleware(['password.confirm']);

Route::post('/settings', function () {
  // ...
})->middleware(['password.confirm']);
```

Thêm vào đối tượng guard tự tạo

Bạn có thể tạo đối tượng guard xác thực của riêng mình bằng cách sử dụng phương thức `extend` trên facade `Auth`. Bạn nên thực hiện cuộc gọi đến phương thức `extend` trong một nhà cung cấp dịch vụ. Vì Laravel đã được cung cấp với `AuthServiceProvider`, nên chúng ta có thể đặt mã trong nhà cung cấp đó:

```
<?php

namespace App\Providers;

use App\Services\Auth\JwtGuard;
use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;
use Illuminate\Support\Facades\Auth;

class AuthServiceProvider extends ServiceProvider
{
  /**
   * Register any application authentication / authorization services.
   *
```

```
     * @return void
     */
    public function boot()
    {
        $this->registerPolicies();

        Auth::extend('jwt', function ($app, $name, array $config) {
            // Return an instance of Illuminate\Contracts\Auth\Guard...

            return new JwtGuard(Auth::createUserProvider($config['provider']));
        });
    }
}
```

Như bạn có thể thấy trong ví dụ trên, lệnh callback được chuyển đến phương thức extend sẽ trả về một bản thực thi của `Illuminate\Contracts\Auth\Guard`. Interface này chứa một số phương thức mà bạn sẽ cần triển khai để tự tạo một đối tượng guard. Sau khitự tạo một đối tượng guard của bạn, bạn có thể tham chiếu đến nó trong cấu hình `guard` của tập tin cấu hình *auth.php* của bạn:

```
'guards' => [
    'api' => [
        'driver' => 'jwt',
        'provider' => 'users',
    ],
],
```

Đối tượng guard request với hàm

Cách đơn giản nhất để triển khai hệ thống xác thực dựa trên yêu cầu HTTP tùy chỉnh là sử dụng phương thức `Auth::viaRequest`. Phương thức này cho phép bạn nhanh chóng khai báo quy trình xác thực của mình bằng một hàm duy nhất.

Để bắt đầu, hãy gọi phương thức `Auth::viaRequest` trong phương thức `boot` của `AuthServiceProvider` của bạn. Phương thức `viaRequest` chấp nhận tên driver xác thực làm đối số đầu tiên của nó. Tên này có thể là bất kỳ chuỗi nào mô tả đối tượng guard của bạn. Đối số thứ hai được truyền cho phương thức phải là một hàm mà sẽ nhận yêu cầu

HTTP đến và trả về một đối tượng người dùng hoặc, nếu xác thực không thành công, thì nó trả `null`:

```
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

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

  Auth::viaRequest('custom-token', function (Request $request) {
    return User::where('token', $request->token)->first();
  });
}
```

Khi driver xác thực tự tạo của bạn đã được xác định, bạn có thể định cấu hình nó làm driver trong cấu hình `guards` của tập tin cấu hình *auth.php* của bạn:

```
'guards' => [
  'api' => [
    'driver' => 'custom-token',
  ],
],
```

Tạo provider người dùng tự tạo

Nếu bạn không sử dụng cơ sở dữ liệu quan hệ truyền thống để lưu trữ người dùng của mình, bạn sẽ cần mở rộng Laravel với nhà cung cấp người dùng xác thực của riêng bạn. Chúng tôi sẽ sử dụng phương thức provider trên facade Auth để tự tạo provider người dùng. Provider người dùng sẽ trả về một bản thực thi `Illuminate\Contracts`

`\Auth\UserProvider`:

```
<?php

namespace App\Providers;

use App\Extensions\MongoUserProvider;
use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;
use Illuminate\Support\Facades\Auth;

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

    Auth::provider('mongo', function ($app, array $config) {
      // Return an instance of Illuminate\Contracts\Auth\UserProvider...

      return new MongoUserProvider($app->make('mongo.connection'));
    });
  }
}
```

Sau khi bạn đã đăng ký provider bằng phương thức `provider`, bạn có thể chuyển sang provider người dùng mới trong tập tin cấu hình *auth.php* của mình. Trước tiên, hãy xác định provider sử dụng driver mới của bạn:

```
'providers' => [
  'users' => [
    'driver' => 'mongo',
  ],
```

```
],
```

Cuối cùng, bạn có thể tham chiếu provider này trong cấu hình đối tượng guard của mình:

```
'guards' => [
  'web' => [
    'driver' => 'session',
    'provider' => 'users',
  ],
],
```

Contract trong provider người dùng

Các `Illuminate\Contracts\Auth\UserProvider` sẽ chịu trách nhiệm tìm nạp `Illuminate\Contracts\Auth\Authenticatable` từ một hệ thống lưu trữ liên tục, chẳng hạn như MySQL, MongoDB, v.v. Hai interface này cho phép các cơ chế xác thực Laravel tiếp tục hoạt động bất kể cách mà dữ liệu người dùng được lưu trữ hoặc loại class nào được sử dụng để đại diện cho người dùng được xác thực:

Hãy xem qua contract `Illuminate\Contracts\Auth\UserProvider`:

```
<?php

namespace Illuminate\Contracts\Auth;

interface UserProvider
{
  public function retrieveById($identifier);
  public function retrieveByToken($identifier, $token);
  public function updateRememberToken(Authenticatable $user, $token);
  public function retrieveByCredentials(array $credentials);
  public function validateCredentials(Authenticatable $user, array $credentials);
}
```

Hàm `retrieveById` thường nhận một khóa đại diện cho người dùng, chẳng hạn như ID tự động tăng dần từ cơ sở dữ liệu MySQL. Việc triển khai `Authenticatable` khớp với ID

phải được truy xuất và trả về bằng phương thức này.

Hàm `retrieveByToken` sẽ truy xuất người dùng bằng mã định danh `$identifier` của họ và `$token` "remember me", thường được lưu trữ trong cột cơ sở dữ liệu như `remember_token`. Như với phương thức trước đó, phương thức này sẽ trả về việc triển khai `Authenticatable` với giá trị token phù hợp.

Phương thức `updateRememberToken` sẽ cập nhật `remember_token` của đối tượng `$user` với `$token` mới. Token mới được chỉ định cho người dùng khi cố gắng xác thực thành công "remember me" hoặc khi người dùng đăng xuất.

Phương thức `retrieveByCredentials` sẽ nhận mảng thông tin xác thực được truyền vào phương thức `Auth::attempt` cố gắng xác thực với một ứng dụng. Sau đó, phương thức sẽ "truy vấn" thông tin người dùng được lưu trữ có khớp với các thông tin đăng nhập đó. Thông thường, phương thức này sẽ chạy một truy vấn với điều kiện "where" để tìm kiếm record người dùng có "username" khớp với giá trị của `$credentials['username']`. Phương thức sẽ trả về một bản thực thi của `Authenticatable`. Phương thức này không nên cố gắng thực hiện bất kỳ xác thực hoặc giám định mật khẩu nào.

Phương thức `validateCredentials` sẽ so sánh `$user` đã cho với `$credentials` đăng nhập để xác thực người dùng. Ví dụ: phương pháp này thường sẽ sử dụng phương thức `Hash::check` để so sánh giá trị của `$user->getAuthPassword()` với giá trị của `$credentials['password']`. Phương thức này sẽ trả về `true` hoặc `false` cho biết mật khẩu có hợp lệ hay không.

Contract xác thực được

Bây giờ chúng ta đã khám phá từng phương thức trên `UserProvider`, hãy xem qua contract `Authenticatable`. Hãy nhớ rằng, các provider người dùng phải trả về các bản thực thi của interface này từ các phương thức `retrieveById`, `retrieveByToken`, và `retrieveByCredentials`:

```php
<?php

namespace Illuminate\Contracts\Auth;

interface Authenticatable
{
  public function getAuthIdentifierName();
```

```
    public function getAuthIdentifier();
    public function getAuthPassword();
    public function getRememberToken();
    public function setRememberToken($value);
    public function getRememberTokenName();
}
```

Interface này đơn giản. Phương thức `getAuthIdentifierName` phải trả về tên của trường "khóa chính" của người dùng và phương thức `getAuthIdentifier` sẽ trả về "khóa chính" của người dùng. Khi sử dụng back-end MySQL, đây có thể là khóa chính tự động tăng dần được gán cho bản ghi người dùng. Phương thức `getAuthPassword` sẽ trả về mật khẩu đã băm của người dùng.

Interface này cho phép hệ thống xác thực hoạt động với bất kỳ lớp "người dùng" nào, bất kể ORM hoặc class trừu tượng lưu trữ bạn đang sử dụng. Theo mặc định, Laravel bao gồm một class `App\Models\User` trong thư mục `app/Models` để cài đặt interface này.

Events

Laravel gửi nhiều sự kiện khác nhau trong quá trình xác thực. Bạn có thể đính kèm các chương trình theo dõi vào các sự kiện này trong `EventServiceProvider` của mình:

```
/**
 * The event listener mappings for the application.
 *
 * @var array
 */
protected $listen = [
    'Illuminate\Auth\Events\Registered' => [
        'App\Listeners\LogRegisteredUser',
    ],

    'Illuminate\Auth\Events\Attempting' => [
        'App\Listeners\LogAuthenticationAttempt',
    ],

    'Illuminate\Auth\Events\Authenticated' => [
        'App\Listeners\LogAuthenticated',
```

```
    ],

    'Illuminate\Auth\Events\Login' => [
        'App\Listeners\LogSuccessfulLogin',
    ],

    'Illuminate\Auth\Events\Failed' => [
        'App\Listeners\LogFailedLogin',
    ],

    'Illuminate\Auth\Events\Validated' => [
        'App\Listeners\LogValidated',
    ],

    'Illuminate\Auth\Events\Verified' => [
        'App\Listeners\LogVerified',
    ],

    'Illuminate\Auth\Events\Logout' => [
        'App\Listeners\LogSuccessfulLogout',
    ],

    'Illuminate\Auth\Events\CurrentDeviceLogout' => [
        'App\Listeners\LogCurrentDeviceLogout',
    ],

    'Illuminate\Auth\Events\OtherDeviceLogout' => [
        'App\Listeners\LogOtherDeviceLogout',
    ],

    'Illuminate\Auth\Events\Lockout' => [
        'App\Listeners\LogLockout',
    ],

    'Illuminate\Auth\Events\PasswordReset' => [
        'App\Listeners\LogPasswordReset',
    ],
];
```