# Bài 12
# Bảo mật ứng dụng Web

Module: BOOTCAMP WEB-BACKEND DEVELOPMENT WITH LARAVEL 2.1

# Mục tiêu

- Biết được các nguy cơ mất an toàn thông tin
- Mô tả được các mô hình bảo mật thông dụng
- Phân biệt được https và http về phương diện bảo mật
- Giải thích được mô hình bảo mật bằng passowrd
- Triển khai được cơ chế bảo mật sử dụng password
- Triển khai được cơ chế phân quyền cơ bản sử dụng role

# Mục tiêu

- Trình bày được khái niệm Authentication
- Trình bày được khái niệm Authorization
- Triển khai được Laravel Authentication
- Triển khai được Laravel Authorization
- Sử dụng được Gates
- Sử dụng được Policy
- Sử dụng được băm
- Thao tác được với password

# Nguy cơ mất an toàn thông tin

- Phân quyền và xác thực người dùng
- Tấn công bởi các phần mềm độc hại
- Mất, hỏng, sửa đổi thông tin
- Mất an toàn thông tin do sử dụng Email, mạng xã hội
- Mất an toàn thông tin đối với Website

# Cơ chế bảo mật ứng dụng

- Xác thực (Authentication)
- Phân quyền (Authorization)
- Phân quyền hệ thống tệp tin
- Quản lý tài khoản
- Sử dụng SSL để truyền dữ liệu
- Mã hoá
- Firewalls

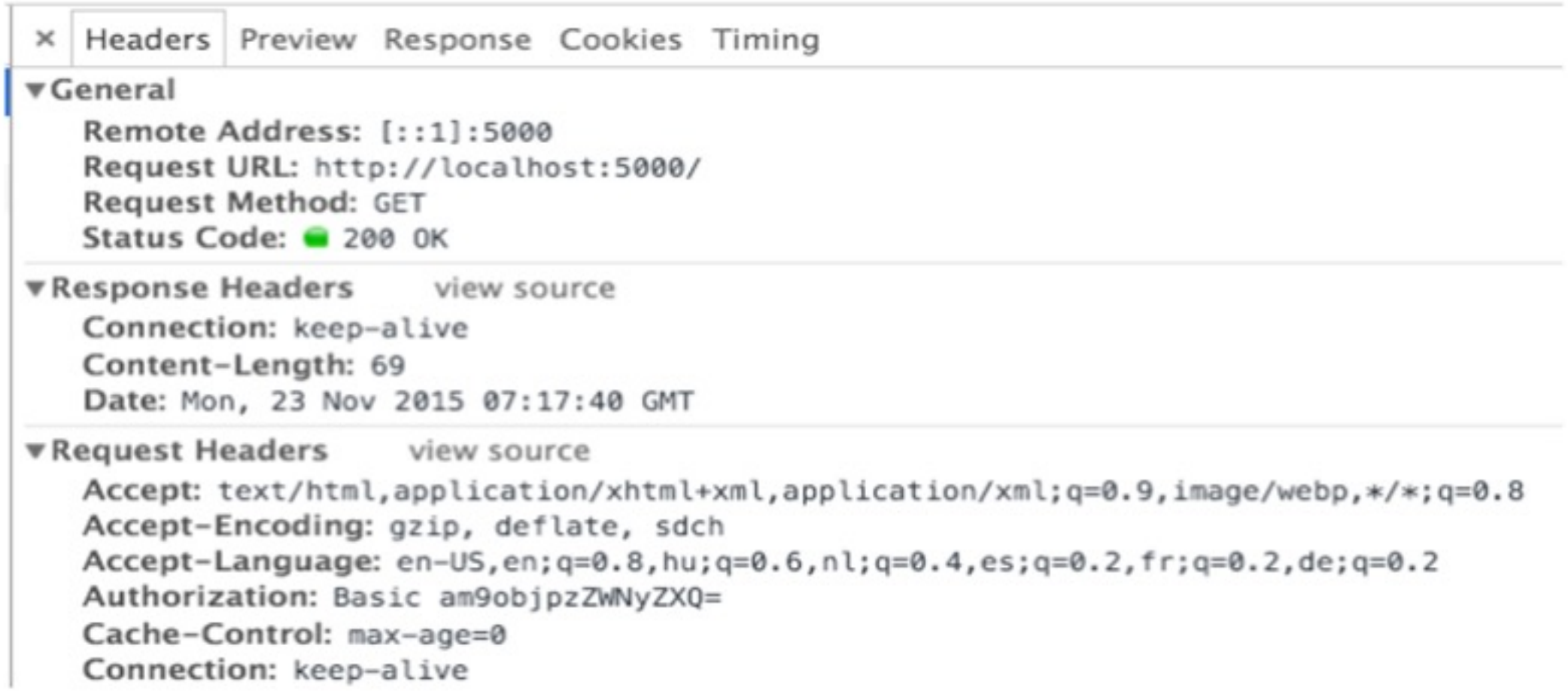
# Các cơ chế xác thực

- HTTP Basic
- Cookies
- Tokens
- Signature
- One-time password

# HTTP Basic

- Xác thực HTTP Basic là một phương thức để client cung cấp username và password khi thực hiện yêu cầu.
- Đây là cách đơn giản nhất có thể để thực thi kiểm soát truy cập vì nó không yêu cầu cookie, session hoặc bất kỳ thứ gì khác.
- Để sử dụng HTTP Basic, client phải gửi tiêu đề Cấp phép (Authorization) cùng với mọi yêu cầu mà nó thực hiện. Username và password không được mã hóa, nhưng được xây dựng theo cách:
  - Username và password được nối vào một chuỗi duy nhất: username:password
  - Chuỗi này được mã hóa với Base64
  - Từ khóa Basic  được đặt trước giá trị được mã hóa này

# Ví dụ quan sát thấy trong Chrome



# Hạn chế của HTTP Basic

- username và password được gửi cùng với mọi yêu cầu, có khả năng hiển thị chúng - ngay cả khi được gửi qua kết nối an toàn.
- Được kết nối với SSL / TLS, nếu trang web sử dụng mã hóa yếu hoặc kẻ tấn công có thể phá vỡ nó, tên người dùng và mật khẩu sẽ được hiển thị ngay lập tức.
- Không có cách nào để đăng xuất người dùng bằng cách sử dụng xác thực cơ bản.
- Yêu cầu thay đổi mật khẩu thường xuyên.

# Cookies

- Để sử dụng cookie cho mục đích xác thực, có một vài nguyên tắc chính cần tuân theo:
  - Luôn sử dụng HttpOnly cookies
    - Để giảm thiểu khả năng tấn công XSS luôn sử dụng cờ HttpOnly khi thiết lập cookie. Bằng cách này, họ sẽ không hiển thị trong document.cookies.
  - Luôn sử dụng cookie đã ký
    - Với cookie đã ký, máy chủ có thể biết cookie có được khách hàng sửa đổi hay không.

# Ví dụ trong Chrome

- Xem máy chủ thiết lập cookie

▼General
Remote Address: 185.63.147.10:443
Request URL: https://www.linkedin.com/nhome/?trk=hb_signin
Request Method: GET
Status Code: 200 OK

▼Response Headers
cache-control: no-cache, no-store
content-encoding: gzip
content-type: text/html; charset=utf-8
date: Mon, 23 Nov 2015 07:25:35 GMT
expires: Thu, 01 Jan 1970 00:00:00 GMT
pragma: no-cache
server: Play
set-cookie: lidc="b=TB16:g=272:u=1:i=1448263535:t=1448349780:s=AQHtWo_H6eKMTC-ysMUDV4m_j19p94Ha"; Expires=Tue, 24 Nov 2015 07:23:00 GMT; domain=.linkedin.com; Path=/

- Sau đó, tất cả các yêu cầu sử dụng tập hợp cookie cho miền đã cho:

Request Headers
:host: www.linkedin.com
:method: GET
:path: /home?trk=nav_responsive_tab_home
:scheme: https
:version: HTTP/1.1
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
accept-encoding: gzip, deflate, sdch
accept-language: en-US,en;q=0.8,hu;q=0.6,nl;q=0.4,es;q=0.2,fr;q=0.2,de;q=0.2
cookie: bcookie="v=2&8b44d748-8d82-46a4-8577-36c00172794b"; bscookie="v=1&201511230722494b3c36f8-d94e-4796-8b49-a5371e6804d1AQEWRbrseJHuyZDHtFipryvvTHp0Sh7o"; L1e=108b50b8; _gat=1; L1c=506114f5; visit="v=1&M"; _ga=GA1.2.96221765.1448263372; oz_props_fetch_size1_undefined=undefined; wutan=i4YA1rjunkpRvR+nkTLFo1Tc8VA9+AOYr0i6eMU+s84=; sl="v=1&i-LQe"; li_at=AQEDARuYmFkFnETwAAABUTM7yS8AAAFRM6mmL04AHZ3tRDqVEG7yio6z6WHQZWKXW0KcCuaX8VpT1gUf8RaKq1YzYoXazo47r3u4tP28mIy5bTs3GLWoym5Xg9XQpqeT3lwdy3Ps5pLNICgTwOjCsoDa; JSESSIONID="ajax:8745093687238714270"; liap=true; lidc="b=TB16:g=272:u=1:i=1448263557:t=1448349780:s=AQH_zUY2E78NFlUbZOHCzlfOatKxlxxy"; RT=s=1448263559420&r=https%3A%2F%2Fwww.linkedin.com%2Fnhome%2F%3Ftrk%3Dhb_signin; share_setting=PUBLIC; _lipt=0_1bmJJGSdPtMK4q9DLkNulXXW5-iWZ3vLKJSmIXmUDL9otamnhKTL_Tp4xsiTWowWeXz07fMN_z7blHodRWXvgr9M72nA7ucghFT4sQG3E6fD3sF9zVmVHVynVYd9lijIAtUYh3Vz8BfPe0oLeRwdL8; lang="v=2&lang=en-us"; sdsc=1%3A1SZM1shxDNbLt36wZwCgPgvN58iw%3D

# Hạn chế của sử dụng Cookie

- Cần phải nỗ lực nhiều hơn để giảm thiểu các cuộc tấn công CSRF
- Không tương thích với REST - vì nó đưa vào một trạng thái thành một giao thức không trạng thái

# Tokens

- JWT (JSON Web Token) là 1 tiêu chuẩn mở (RFC 7519) định nghĩa cách thức truyền tin an toàn giữa các thành viên bằng 1 đối tượng JSON
- JWT bao gồm ba phần:
  - Tiêu đề, chứa loại mã thông báo và thuật toán băm
  - Tải trọng, chứa các xác nhận quyền sở hữu
  - Chữ ký, có thể được tính như sau nếu bạn chọn HMAC SHA256:

    HMACSHA256( base64UrlEncode(header) + **"."** + base64UrlEncode(payload), secret)
- Nhược điểm
  - Cần phải nỗ lực nhiều hơn để giảm thiểu các cuộc tấn công XSS.

# Signature (Chữ ký)

- Chữ ký số khóa công khai là mô hình sử dụng các kỹ thuật mật mã để gắn với mỗi người sử dụng một cặp khóa công khai - bí mật và qua đó có thể ký các văn bản điện tử cũng như trao đổi các thông tin mật. Khóa công khai thường được phân phối thông qua chứng thực khóa công khai.
- Bao gồm 2 quá trình:
  - Tạo chữ ký
  - Kiểm tra chữ ký.

# One-Time Passwords

- Thuật toán One-Time passwords tạo mật khẩu một lần với bí mật dùng chung và thời gian hoặc bộ đếm hiện tại:
  - Thuật toán Time-based One-time Password dựa trên thời gian hiện tại.
  - Thuật toán HMAC-based One-time Password dựa trên bộ đếm.
- Các phương thức này được sử dụng trong các ứng dụng tận dụng xác thực hai yếu tố: người dùng nhập username và password, sau đó cả máy chủ lẫn máy khách đều tạo mật khẩu một lần.

# Certificate (Chứng chỉ xác thực)

- Hệ thống chứng thực là một hạ tầng an ninh mạng được xây dựng trên một hạ tầng cơ sở khóa công khai (PKI) cung cấp các giải pháp đảm bảo an toàn cho các hoạt động (gọi chung là giao dịch) thông qua mạng.
- Hệ thống chứng thực cung cấp các dịch vụ đảm bảo an toàn cho các giao dịch thông qua mạng. Các dịch vụ cơ bản mà một hệ thống chứng thực cung cấp bao gồm:
  - Dịch vụ xác thực: nhằm xác định xem ai đang giao dịch với mình.
  - Dịch vụ bảo mật: đảm bảo tính bí mật của thông tin, người không có thẩm quyền không thể đọc được nội dung của thông tin.
  - Dịch vụ toàn vẹn: khẳng định thông tin có bị thay đổi hay không.
  - Dịch vụ chống chối bỏ: cung cấp các bằng chứng chống lại việc chối bỏ một hành động đã thực hiện hay đã diễn ra
  - Như vậy sử dụng hệ thống chứng thực sẽ đảm bảo, bí mật, toàn vẹn cho thông tin được truyền qua mạng, xác thực được người dùng và chống chối bỏ các hành động hay sự kiện đã xảy ra.

# Certificate (Chứng chỉ xác thực)

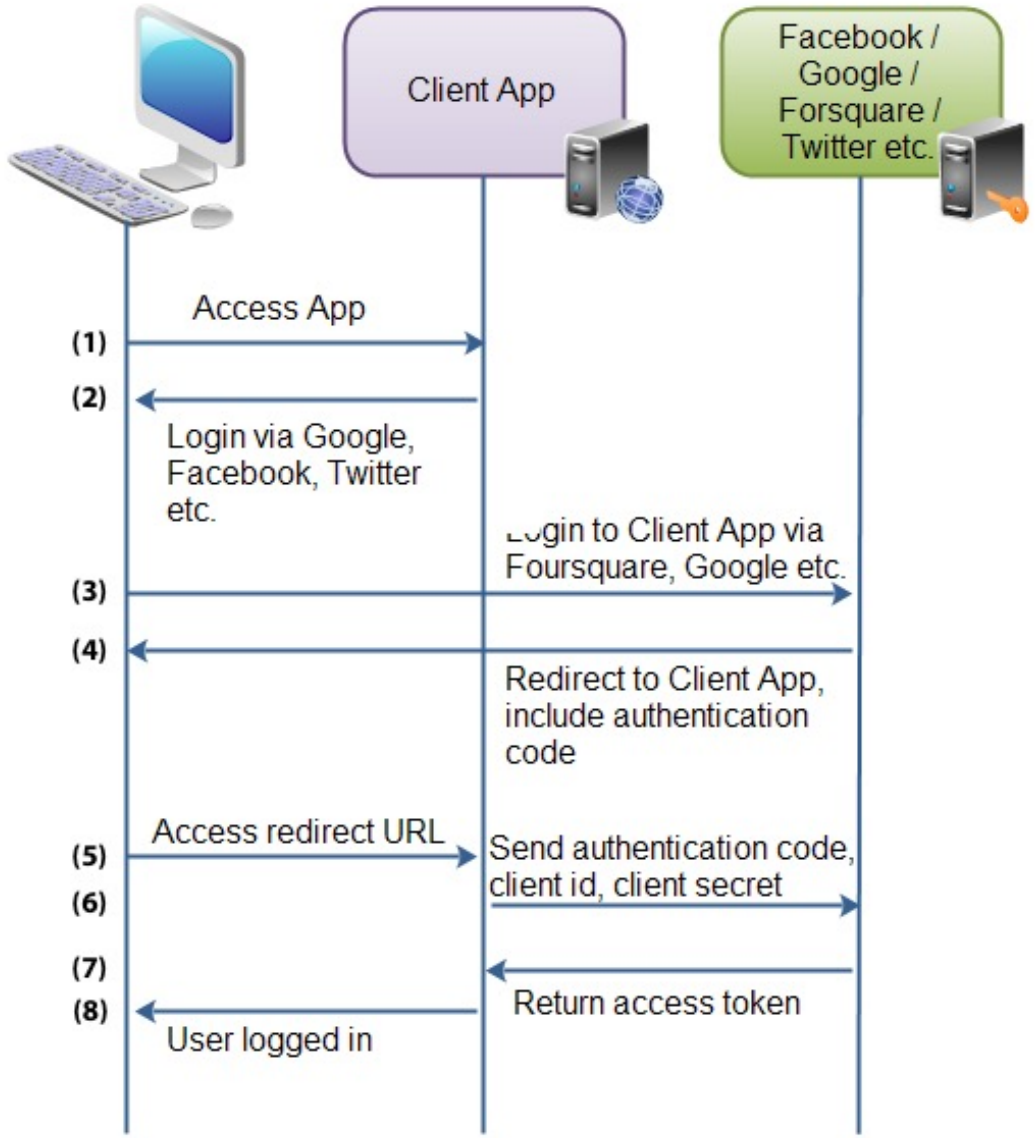
- Hệ thống chứng thực gồm 2 thành phần:
  - Thành phần thực hiện các nhiệm vụ về quản lý chứng thư số như: đăng ký và phát hành, thu hồi ... chứng thư số.
  - Thành phần thực hiện chức năng xác định xem một chứng thư số có hợp lệ hay không
- Cơ quan chứng thực (Certification Authority - CA) có thẩm quyền cấp phát, thu hồi, quản lý chứng thư số cho các thực thể thực hiện các giao dịch an toàn. Cơ quan chứng thực là một thành phần chính của hệ thống chứng thực.
- Một số ứng dụng hệ thống chứng thực
  - Thuế điện tử (E-Tax Filing)
  - Thanh toán trực tuyến (E-Payment)
  - Bảo mật email
  - ...

# OAuth2

- OAuth là viết tắt của Open Authorization là một giao thức mở nhằm cung cấp cách thức đơn giản và tiêu chuẩncho các thao tác xác thực an toàn trong các ứng dụng trên web, mobile & desktop.
- **OAuth** đã ra mắt thế hệ thứ 2, gọi là **OAuth2**, là 1 phiên bản nâng cấp hoàn thiện hơn của **OAuth** đời đầu tiên.

# OAuth2

- Cơ chế của OAuth



# Laravel Authentication

# Authentication và Authorization

- Authentication (Chứng thực) và Authorization (Phân quyền) là 2 cơ chế quan trọng của các ứng dụng phân tán
- Authentication là quá trình xác định thực thể đang giao tiếp với hệ thống
- Authentication trả lời cho câu hỏi “Anh là ai?”
- Authorization là quá trình xem xét để cấp các quyền truy cập phù hợp cho thực thể
- Authorization diễn ra sau Authentication
- Authorization trả lời cho câu hỏi “Anh được phép làm gì?”

# Laravel Authentication

- Laravel hỗ trợ authentication trở nên dễ dàng
- Cấu hình của Laravel Authentication được đặt trong *config/auth.php*
- 2 thành phần:
  - Guards: Cách người dùng được chứng thực trong từng request
  - Providers: Cách lưu trữ và truy xuất thông tin người dùng
- Ví dụ Guards: Lưu trữ trạng thái đăng nhập trên *session*
- Ví dụ Providers: Lưu trữ thông tin người dùng sử dụng Eloquent
- Có thể tự định nghĩa Guards và Providers mới

# Authentication Controller

- Laravel có sẵn controller để xử lý authentication
- Lưu trữ trong *App\Http\Controllers\Auth:*
  - RegisterController: Xử lý thao tác đăng ký người dung
  - LoginController: Xử lý thao tác đăng nhập
  - ForgotPasswordController: Xử lý thao tác gửi email để cấp lại password
  - ResetPasswordController: Xử lý thao tác cấp lại password

# Khởi tạo route cho Authentication

- Sử dụng câu lệnh *php artisan*:
  *php artisan make:auth*
- Nên sử dung câu lệnh này trên một ứng dung hoàn toàn mới
- Các thành phần được khởi tạo:
  - Layout View
  - Registration View
  - Login View
  - Các route (đăng nhập, đăng ký, quên mật khẩu)
  - HomeController

# Bỏ tính năng đăng ký

- Có thể bỏ tính năng đăng ký người dùng mới nếu không cần đến
- Các thao tác
  - Xoá RegisterController
  - Thay đổi route: *Auth::routes(['register' => false]);*

# Tuỳ biến trường username

- Mặc định, Laravel sử dụng trường email để xác thực
- Có thể tuỳ biến bằng cách tuỳ chỉnh phương thức username() của LoginController
- Ví dụ:

```
public function username() {
    return 'username’;
}
```

# Lấy về thông tin người dùng

- Có thể lấy về id hoặc toàn bộ thông tin người dùng đã đăng nhập
- Ví dụ:

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


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


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

# Kiểm tra trạng thái đăng nhập

- Kiểm tra trạng thái đăng nhập của người dùng bằng phương thức check()
- Ví dụ:

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

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

# Kiểm soát quyền truy cập route

- Sử dung middleware auth để kiểm soát quyền truy cập route
- Ví dụ, sử dụng trong route:

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

- Hoặc sử dung trong controller:

```
public function __construct() {
    $this->middleware('auth’);
}
```

# Tuỳ biến Authentication

# Đăng nhập thủ công

- Kiểm tra credentials sử dung phương thức attempt()

```
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Auth;

class LoginController extends Controller {
    public function authenticate(Request $request) {
        $credentials = $request->only('email', 'password’);
        if (Auth::attempt($credentials)) {
            // Authentication passed...
            return redirect()->intended('dashboard’);
        }
    }
}
```

# Logout thủ công

- Sử dung phương thức logout() của Auth:

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

Auth::logout();
```

# Laravel Authorization

# Laravel Authorization

- Laravel hỗ trợ authorization trở nên đơn giản
- Có thể sử dụng 2 cơ chế:
  - Gates: đơn giản, đơn lẻ, nhanh chóng
  - Policies: hệ thống, bài bản
- Có thể sử dụng kết hợp cả gates và policies trong cùng một ứng dụng

# Gates

- Gates xác định liệu một người dùng có đủ thẩm quyền để thực hiện một hành động nào đó
- Gates được định nghĩa trong *App\Providers\AuthServiceProvider*
- Ví dụ, người dùng chỉ được cập nhật các post của mình:

```
public function boot() {
    $this->registerPolicies();
    Gate::define('update-post', function ($user, $post) {
        return $user->id == $post->user_id;
    });
}
```

# Định nghĩa Gates

- Có thể định nghĩa Gates sử dung lớp và phương thức riêng
- Ví dụ:

```
public function boot() {
    $this->registerPolicies();
    Gate::define('update-post', 'App\Policies\PostPolicy@update’);
}
```

  - Trong đó, phương thức *update()* của lớp *PostPolicy* sẽ xác định quyền truy cập cho gate *‘update-post’*

# Resource Gates

- Có thể định nghĩa nhiều Gates sử dụng phương thức resource
- Ví dụ:

```
Gate::resource('posts', 'App\Policies\PostPolicy');
```

  - Tương đương với:

```
Gate::define('posts.view', 'App\Policies\PostPolicy@view');
Gate::define('posts.create', 'App\Policies\PostPolicy@create');
Gate::define('posts.update', 'App\Policies\PostPolicy@update');
Gate::define('posts.delete', 'App\Policies\PostPolicy@delete');
```

# Cấp quyền cho các action

- Có thể sử dụng phương thức allows() và denies() để cấp quyền cho các action
- Ví dụ:

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

# Can thiệp vào Gates

- Sử dụng phương thức before() và after() để can thiệp vào quá trình kiểm tra của Gates
- Ví dụ:

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

- và:

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

# Policy

# Policy

- Policy là cơ chế authorization trong đó nhóm các nghiệp vụ liên quan đến một model để dễ kiểm soát
- Ví dụ, kiểm soát quyền thao tác với Post thông qua PostPolicy
- Có thể khởi tạo policy bằng câu lệnh php artisan:

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

  - Hoặc:

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

# Đăng ký Policy

- Đăng ký trong mảng *policies* của *AuthServiceProvider*
- Ví dụ:

```php
namespace App\Providers;

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

class AuthServiceProvider extends ServiceProvider {
    protected $policies = [
        Post::class => PostPolicy::class,
    ];
    public function boot() {
        $this->registerPolicies();
    }
}
```

# Tạo policy

- Ví dụ:

```php
namespace App\Policies;

use App\User;

use App\Post;


class PostPolicy {

    public function update(User $user, Post $post) {

        return $user->id === $post->user_id;

    }

}
```

# Cấp quyền sử dụng policy

- Sử dụng phương thức *cant* của đối tượng *user* để cấp quyền
  - Ví dụ:

```
if ($user->can('update', $post)) {
    //
}
```

- Sử dụng phương thức *cant* của đối tượng user để từ chối cấp quyền
  - Ví dụ:

```
if ($user->cant('update', $post)) {
    //
}
```

# Cấp quyền thông qua route

- Sử dụng middleware *can* để cấp quyền thông qua route
- Ví dụ:

```
use App\Post;

Route::put('/post/{post}', function (Post $post) {

// The current user may update the post...

})->middleware('can:update,post');
```

# Cấp quyền thông qua controller

- Sử dụng phương thức *authorize* của controller để cấp quyền
- Ví dụ:

```php
namespace App\Http\Controllers;

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

class PostController extends Controller {
    public function update(Request $request, Post $post) {
        $this->authorize('update', $post);
        // The current user can update the blog post...
    }
}
```

# Cấp quyền trong Blade template

- Sử dụng directive @can và @cannot để cấp quyền
- Ví dụ:

```
@can('update', $post)
<!-- The Current User Can Update The Post -->
@elsecan('create', App\Post::class)
<!-- The Current User Can Create New Post -->
@endcan


@cannot('update', $post)
<!-- The Current User Can't Update The Post -->
@elsecannot('create', App\Post::class)
<!-- The Current User Can't Create New Post -->
@endcannot
```

# Mã hoá và băm

# Cấu hình mã hoá Laravel

- Laravel cung cấp tiện ích mã hoá sử dung AES-256 và AES-128
- Cần chỉ định key trong file config/app.php
- Có thể sử dung câu lệnh php artisan để sinh key:

```
php artisan key:generate
```

# Mã hoá

- Sử dụng phương thức encrypt để mã hoá một giá trị
- Ví dụ:

```
public function storeSecret(Request $request, $id) {
    $user = User::findOrFail($id);
    $user->fill([
        'secret' => encrypt($request->secret)
    ])->save();
}
```

# Giải mã

- Sử dụng phương thức decrypt để giải mã một giá trị đã được mã hoá
- Ví dụ:

```
use Illuminate\Contracts\Encryption\DecryptException;


try {
    $decrypted = decrypt($encryptedValue);
} catch (DecryptException $e) {
    //
}
```

# Băm (hashing)

- Băm là một cơ chế mã hoá một chiều
- Không thể giải mã một giá trị đã được băm
- Băm thường được sử dụng để thao tác với password
- Laravel hỗ trợ thao tác với băm dễ dàng
- Cấu hình băm ở trong *config/hashing.php*

# Băm password

- Sử dụng phương thức make của façade Hash để băm password
- Ví dụ:

```
use Illuminate\Support\Facades\Hash;


public function update(Request $request) {
    $request->user()->fill([
        'password' => Hash::make($request->newPassword)
    ])->save();
}
```

# Kiểm tra password

- Sử dụng phương thức check() của façade Hash để kiểm tra giá trị băm của password
- Ví dụ:

```
if (Hash::check('plain-text', $hashedPassword)) {
    // The passwords match...
}
```

# Tóm tắt bài học

# Hướng dẫn

- Hướng dẫn làm bài thực hành và bài tập
- Chuẩn bị bài tiếp: ***Deployment***