CHƯƠNG 5

XÂY DỰNG WEB ĐỘNG VỚI FRAMEWORK MÃ NGUỒN MỞ

thanhlv (thanh.cntt.dhv@gmail.com)

📖 Nội dung giảng dạy

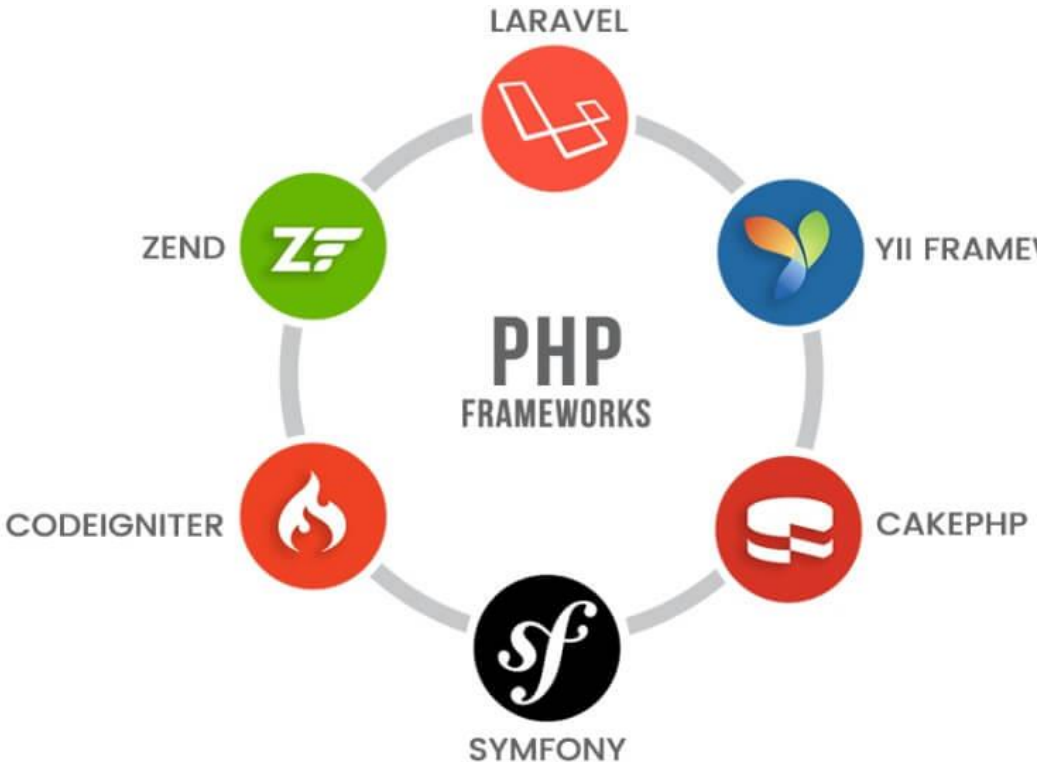
5.1. Giới thiệu framework mã nguồn mở
5.2. Cấu trúc
5.3. Mô hình MVC
5.4. Route
5.5. View
5.6. Controller
5.7. Model
5.8. Làm việc với Database
5.9. Collection
5.10. HTTP Request
5.11. Validation
5.12. Authentication

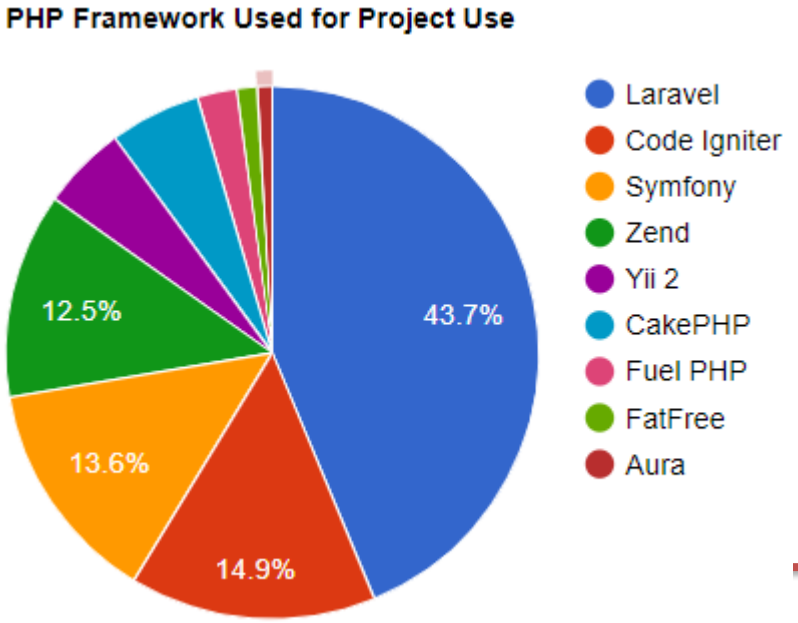
5.1. Giới thiệu framework mã nguồn mở

- PHP Framework:
  - Là các nền tảng open source viết bằng PHP, được tạo ra để giúp phát triển các ứng dụng web nhanh chóng hơn. Các PHP Framework thường ứng dụng mô hình MVC, OOP.
  - Đội ngũ phát triển framework cũng tạo sẵn các class, quy ước cấu trúc folder, tên biến và các chức năng cơ bản. Nhờ đó giúp tiết kiệm được thời gian triển khai dự án, code sẽ ổn định hơn, hạn chế viết lại mã.
  - Hiện có nhiều PHP framework như Laravel, CakePHP, CodeIgnitor, Yii, Kohana, Phalcon, Zend, Slim, PHPixie…

5.1. Giới thiệu framework mã nguồn mở

- Các PHP Framework phổ biến nhất hiện nay:





5.1. Giới thiệu framework mã nguồn mở

- Laravel là một PHP Framework, có mã nguồn mở, miễn phí, được xây dựng nhằm hỗ trợ phát triển ứng dụng web, theo mô hình MVC (Model, Controller, View)
- Ra đời lần đầu tiên vào năm 2011, cho đến nay, Laravel đã được phát triển đến phiên bản 6.x (09/2019), 7.x (03/2020), 8.x (09/2020) với nhiều cải tiến
- Là PHP Framework được sử dụng phổ biến nhất trên thế giới

5.1. Giới thiệu framework mã nguồn mở

- Ưu điểm:
  - Theo mô hình MVC
  - Kiến trúc thống nhất, khoa học và cực đơn giản
  - Sử dụng các tính năng mới nhất của PHP
  - Nguồn tài nguyên lớn và sẵn có
  - Tích hợp với dịch vụ mail
  - Tốc độ xử lý nhanh
  - Dễ sử dụng
  - Tính bảo mật cao
  - Sử dụng composer để quản lí các gói mở rộng
  - Nhiều chức năng hay: artisan, mirgration, seeder…

5.1. Giới thiệu framework mã nguồn mở

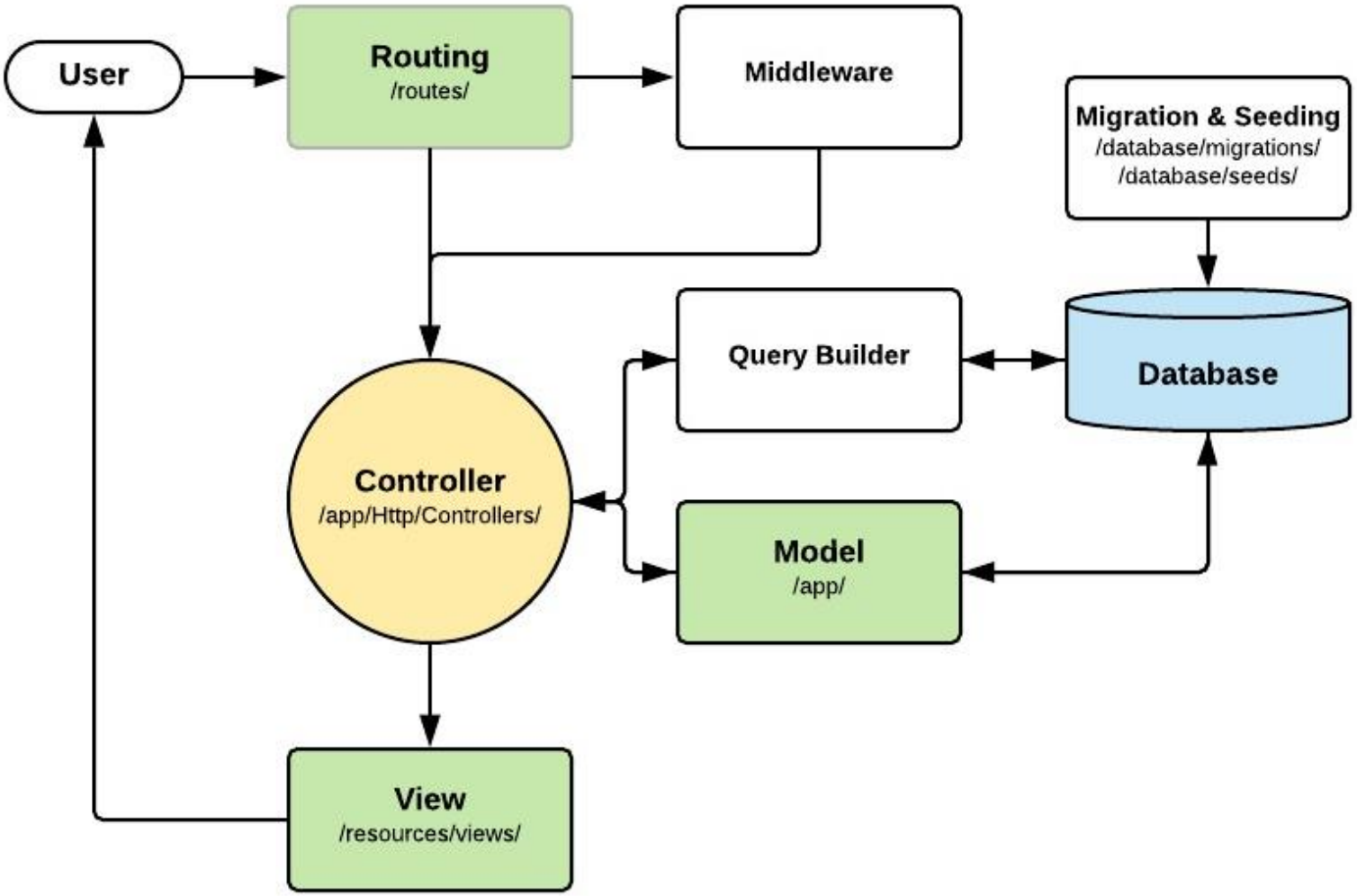
- Laravel:

5.1. Giới thiệu framework mã nguồn mở

- Các tính năng hay của Laravel:
  - Composer: sử dụng để nâng cấp, cài đặt…
  - Eloquent ORM: thao tác với cú pháp đẹp mắt và đơn giản.
  - Restful API: hỗ trợ biến Laravel thành một web service API.
  - Artisan: cung cấp các lệnh cần thiết để phát triển ứng dụng.
  - View: giúp code rõ ràng, dễ đọc hơn.
  - Migrations: hỗ trợ tạo các trường trong cơ sở dữ liệu, thêm các cột trong bảng, tạo mối quan hệ giữa các bảng, hỗ trợ quản lý cơ sở dữ liệu.
  - Authentication: cung cấp sẵn các tính năng đăng nhập, đăng ký, quên mật khẩu…
  - Unit Testing: hỗ trợ test lỗi để sửa chữa.

5.1. Giới thiệu framework mã nguồn mở

- Luồng xử lý của Laravel:



5.1. Giới thiệu framework mã nguồn mở

- Luồng xử lý của Laravel:

Folder / File	Mô tả
User	User gửi yêu cầu.
Routing	Yêu cầu từ User sẽ được Routing điều hướng: - Tới Controller để xử lý yêu cầu. - Tới Middleware để kiểm tra điều hướng cần thiết. - Tới thẳng View nếu không cần xử lý.
Middleware	Middleware có thể xem như bộ lọc HTTP từ request, VD như dùng để kiểm tra, xác thực người dùng đăng nhập vào hệ thống. Sau khi kiểm tra, xác thực xong sẽ trả về Controller để tiếp tục xử lý.
Controller	Controller được xem như trung tâm điều khiển của hệ thống, tất cả thao tác xử lý nên được thực hiện ở đây. - Controller kết nối với Database thông qua điều khiển từ Model. Hoặc có thể thao tác trực tiếp tới Database thông qua các Query Builder - Kết quả xử lý sẽ được trả về view.

5.1. Giới thiệu framework mã nguồn mở

- Luồng xử lý của Laravel:

Folder / File	Mô tả
Model	Khi có yêu cầu từ Controller, Model sẽ tương tác với Database và trả kết quả về Controller, một số trường hợp cần thiết thì Model cũng có thể trả thẳng kết quả về View.
Database	Lưu trữ và chứa dữ liệu.
Migration & Seeding	- Migration, được dùng để tạo Database. - Seeding, được dùng để tạo dữ liệu ảo cho Database.
Query Builder	Câu truy vấn Database.
View	View nhận dữ liệu xử lý từ Controller (hoặc Model, Routing), hiển thị kết quả cho người dùng.

5.1. Giới thiệu framework mã nguồn mở

- Composer:
  - Composer là một chương trình open source miễn phí, nó giúp quản lý các gói thư viện trong ứng dụng web.
  - Nhờ Composer mà người lập trình sẽ tiết kiệm nhiều thời gian và công sức trong việc bổ sung/cập nhật/gỡ bỏ các gói thư viện trong project.
  - Khi cần cài thư viện nào, chỉ cần khai báo, Composer sẽ tự động tải chúng về thông qua một server cộng đồng.
  - Download: http://getcomposer.org/
  - Các lệnh thường dùng:
    - Xem version: composer –V
    - Cập nhật Composer: composer selfupdate
    - Tìm các gói thư viện: composer search từ_khóa
    - Bổ sung thư viện vào dự án: composer require tên_thư_viện
    - Xóa thư viện khỏi dự án: composer remove tên_thư_viện

5.1. Giới thiệu framework mã nguồn mở

- Artisan:
  - Artisan là một tiện ích có sẵn trong Laravel và rất hữu ích, chạy trong command line. Nó cung cấp nhiều chức năng giúp tự động hóa công việc, giúp tiết kiệm thời gian và công sức khi khi xử lý các công việc mang tính thủ công.
    - Ví dụ: chạy các công việc ngầm theo kiểu hàng đợi (queue job), sử dụng cache, các việc xử lý database như migrate, tạo dữ liệu test.
  - Artisan cũng có thể giúp tạo các model, controller, event… Các việc liên quan đến bảo mật như cài đặt xác thực người dùng, sinh key mã hóa, các việc liên quan đến sử dụng OAuth2.

5.1. Giới thiệu framework mã nguồn mở

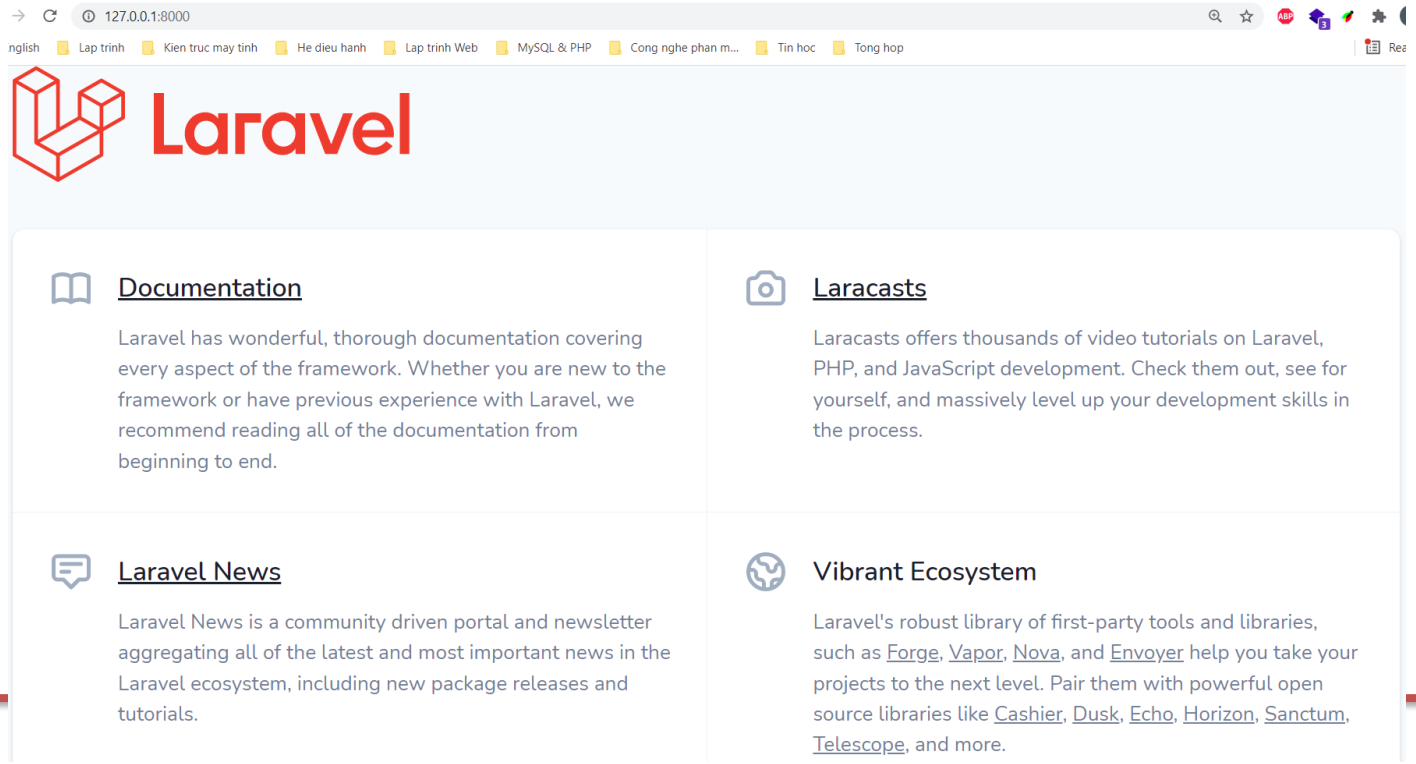
- Artisan: Các lệnh thường dùng
  - Xem các lệnh trong artisan: php artisan list
  - Kiểm tra version của laravel: php artisan --version
  - Đưa ứng dụng về chế độ bảo trì: php artisan down
  - Đưa ứng dụng hoạt động trở lại: php artisan up
  - Liệt kê các route: php artisan route:list
  - Tạo controller: php artisan make:controller “tên_Controller”
  - Tạo key mới: php artisan key:generate
  - Tạo model: php artisan make:model “tên_Model”
  - Chạy project: php artisan serve
    - nếu muốn chạy port khác (ví dụ 8080) thì dùng lệnh: php artisan serve --port=8080.
  - Biên dịch trước tất cả view: php artisan view:cache (các file cache view được lưu trong \storage\framework\views).
  - Xóa các view cache: php artisan view:clear

5.1. Giới thiệu framework mã nguồn mở

- Cài đặt Laravel (sử dụng Composer) :
  - Thông qua Laravel Installer (PHP >= 7.3.0):
    - Tải Laravel installer: composer global require laravel/installer
    - Tạo project: laravel new “project_name”
  - Thông qua Composer Create-Project:
    - Tạo project: composer create-project laravel/laravel --prefer-dist “project_name”
  - Thông qua Composer.phar:
    - Tải source code của Laravel
    - Đến thư mục chứa code, chạy lệnh: composer.phar install
    - Chạy lệnh php artisan key:generate để kích hoạt application key
  - Cài bằng framework: Laragon (all in one)
  - Cài đặt các gói: composer require package_name (VD: illuminate/html, laravel/ui...)

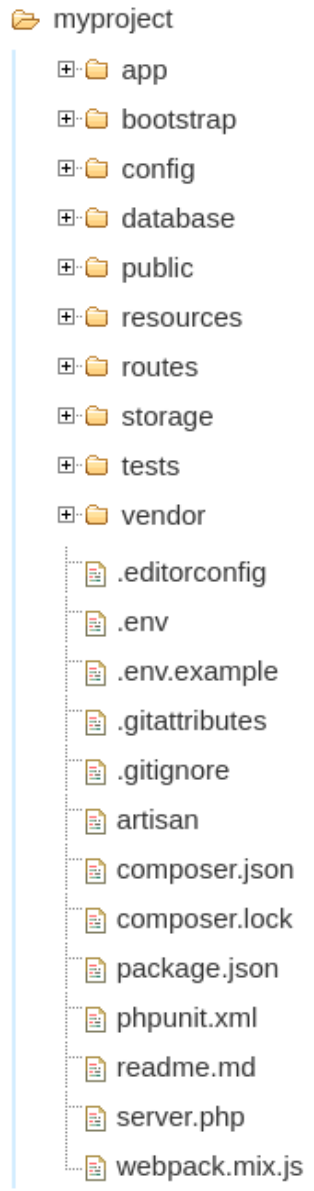
5.1. Giới thiệu framework mã nguồn mở

- Chạy Laravel:
  - Vào thư mục Laravel, chạy lệnh: php artisan serve
  - Gõ địa chỉ server website vào trình duyệt
- Giao diện trang chủ mặc định:



5.2. Cấu trúc

Folder/File	Mô tả
app	Chứa các thư mục, các tập tin php, thư viện, models
app/Console	Chứa các tập tin định nghĩa các câu lệnh trên artisan
app/Exceptions	Chứa các tập tin quản lý, điều hướng lỗi
app/Http/Controllers	Chứa các controller của project
app/Http/Middleware	Chứa các tập tin lọc và ngăn chặn các requests
app/Providers	Chứa các file thực hiện việc khai báo service và bind vào trong Service Container
bootstrap	Chứa những file khởi động của framework và những file cấu hình auto loading, route, và file cache
config	Chứa tất cả những file cấu hình
database	Chứa các file định nghĩa các cột bảng dữ liệu để tạo ra các dữ liệu mẫu



5.2. Cấu trúc

Folder/File	Mô tả
database/migrations	Chứa các file tạo và chỉnh sửa dữ liệu
database/seeds	Thư mục seeds, chứa các file tạo dữ liệu thêm vào CSDL
public	Chứa file index.php giống như cổng cho tất cả các request vào project, bên trong thư mục còn chứa file JavaScript, và CSS
resources	Chứa những file view và raw, các file biên soạn như LESS, SASS, hoặc JavaScript. Ngoài ra còn chứa tất cả các file lang trong project
resources/views	Chứa các file view xuất giao diện người dùng
routes	Chứa tất cả các điều khiển route (đường dẫn) trong project. Chứa các file route sẵn có: web.php, channels.php, api.php, và console.php
routes/api.php	file api.php, điều khiển các route của ứng dụng, như route của ứng dụng User (đăng ký, đăng nhập...)
routes/web.php	file web.php, điều khiển các route của view, như route của trang top, sản phẩm...

5.2. Cấu trúc

Folder/File	Mô tả
storage	Chứa các tập tin hệ thống cache, session
tests	Thư mục tests, chứa những file tests, như PHPUnit test
vendor	Thư mục vendor, chứa các thư viện của Composer
.env	file .env, chứa các config chính của Laravel
artisan	file thực hiện lệnh của Laravel
.gitattributes, .gitignore	File dành cho xử lý git
composer.json, composer.lock, composer-setup.php	File của Composer
package.js	file package.js, chứa các package cần dùng cho projects
phpunit.xml	file phpunit.xml, xml của phpunit dùng để testing project
webpack.mix.js	file webpack.mix.js, file dùng để build các webpack

5.2. Cấu trúc

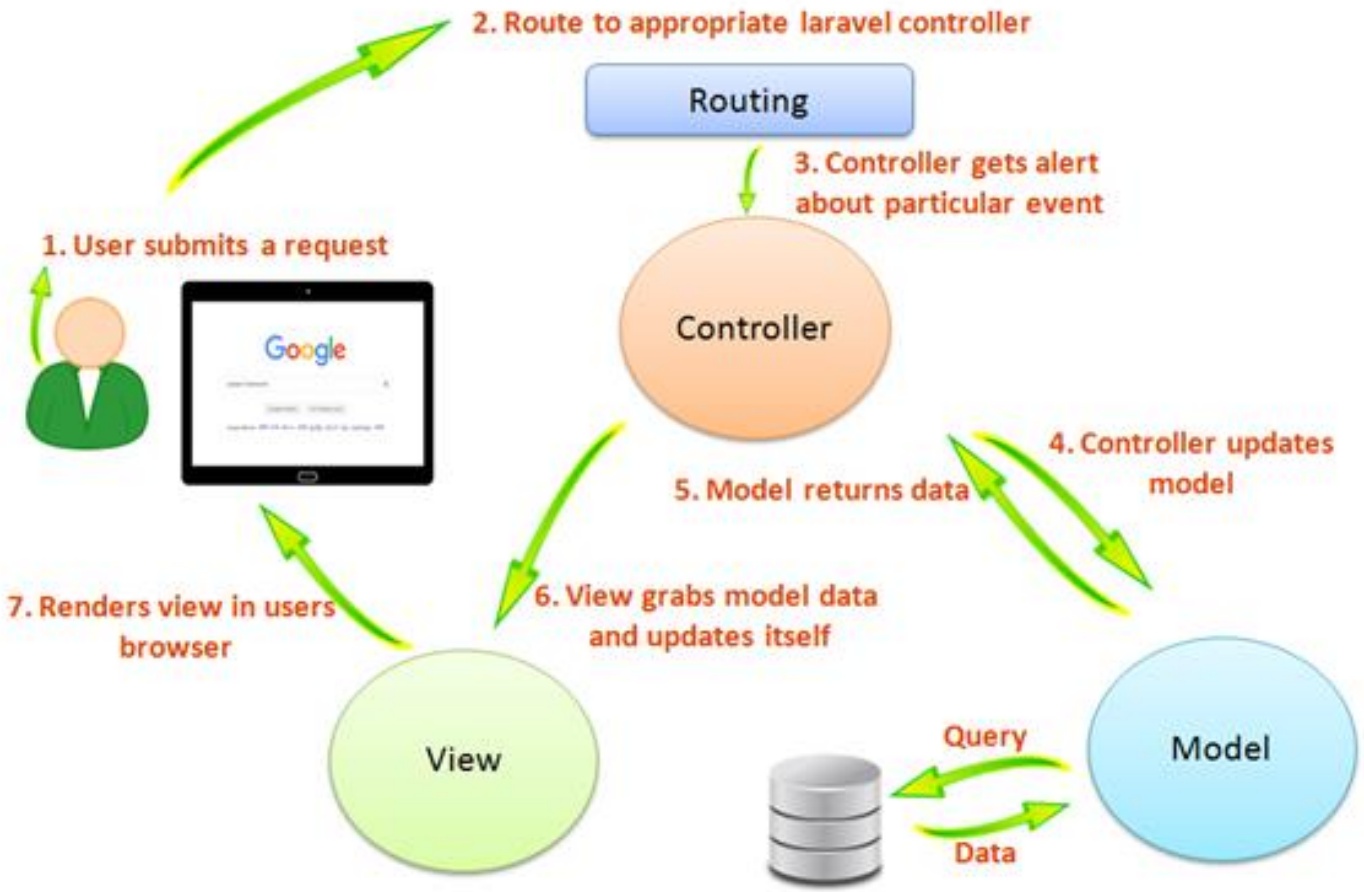
- Các thông số cơ bản (trong file .env)
  - APP_URL: địa chỉ website, bổ sung vào folder website để có địa chỉ đúng.
  - APP_KEY: 1 chuỗi được tạo ngẫu nhiên khi cài đặt laravel, dùng cho việc mã hóa và giải mã các biến cookie.
  - DB_HOST: địa chỉ của server chạy MySQL.
  - DB_PORT: cổng port của server MySQL (thường để mặc định là 3306).
  - DB_DATABASE: tên database sẽ kết nối tới.
  - DB_USERNAME: username để kết nối vào database.
  - DB_PASSWORD: password để kết nối vào database.

5.3. Mô hình MVC

- MVC: viết tắt của Model – View – Controller:
  - Model: quản lý, lưu trữ và truy xuất dữ liệu từ cơ sở dữ liệu theo lệnh của controller.
  - View: hiển thị dữ liệu cho người dùng dựa trên hành động của người dùng.
  - Controller: nhận lệnh từ người dùng, gửi lệnh đến cho Model để tương tác với dữ liệu, truyền lệnh đến View để cập nhập giao diện hiển thị.
- Quy trình hoạt động:
  - Request từ phía người dùng sẽ qua Route, dữ liệu được gửi đến Controller để xử lý, cần dữ liệu sẽ lấy từ Model lên hoặc cập nhật dữ liệu xuống Model, kết quả gửi ra View cho người sử dụng.

5.3. Mô hình MVC

- Mô hình MVC trong Laravel:



5.4. Router

- Route thực hiện chức năng định tuyến, đường dẫn cho các HTTP request được gửi đến đúng nơi muốn đến.
- Khi xử lý một request của người dùng, bước đầu tiên là phân tích format của URL gửi tới, mỗi một format của URL tương ứng với một route.
- Các routes được định nghĩa trong routes/web.php.
- Cấu trúc của một router: Route::METHOD('URL', ACTION);

```
Route::get('/', function () {
    return view('welcome');
});

// Xuất câu chào
Route::get('/route', function () {
    return "Hello Laravel";
});
```

5.4. Router

- Các loại route:
  - Route::get: nhận resquest với phương thức GET (index, create, show, edit).
  - Route::post: nhận resquest với phương thức POST (store).
  - Route::put: nhận resquest với phương thức PUT (update).
  - Route::delete: nhận resquest với phương thức DELETE (destroy).
  - Route::match: kết hợp nhiều phương phức như POST, GET, PUT...
  - Route::any: nhận tất cả các phương thức.
  - Route::group: tạo ra các nhóm route.
  - Route::controller: gọi đến controller tương ứng mà chúng ta tự định.
  - Route::resource: sử dụng với resource controller.

5.4. Router

- Resource controllers: tối ưu hóa code cho router
  - Các action được xử lý bởi resource controller:

Phương thức	URL	Hành động	Tên route
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy
GET	/photos	index	photos.index

5.4. Router

- Resource controllers: tối ưu hóa code cho router
  - khai báo route cho controller:

```
// Khai báo route cho controller
Route::resource('photos', 'PhotoController');

// Khai báo cho nhiều resource controller cùng 1 lúc
Route::resources([
    'photos' => 'PhotoController',
    'posts' => 'PostController'
]);

// Khai báo khi chỉ dùng một số action nhất định
Route::resource('photos', 'PhotoController')->only([
    'index', 'show'
]);
```

```
<!--Giả method (do trong html không có các method PUT, PATCH, DELETE)-->
<form action="/foo/bar" method="POST">
    @method('PUT')
</form>
```

5.4. Router

- Trả về chuỗi, view:

```
// Trả về chuỗi, truyền tham số
Route::get('/route/{name}', function ($str) {
    return "Hello: " . $str;
});

// Trả về view
Route::get('/hello', function () {
    return view('hello');
});

// Truyền tham số với tham số không bắt buộc
Route::get('/hello/{name}/{year?}', function ($name, $year = null) {
    if ($year == null) {
        echo ('Hello: ' . $name);
    } else {
        echo ('Hello: ' . $name . ', Year: ' . $year);
    }
});
```

5.4. Router

- Truyền tham số cho view:

```
// Trả về view, truyền tham số dùng mảng hoặc compact
Route::get('/hello_info/{name}/{year}', function($name, $year){
    return view('hello_info', array('name'=>$name,'year'=>$year));
    // hoặc:
    return view('hello_info', compact('name', 'year'));
});

// Truyền tham số sang view (1 tham số) dùng with
Route::get('/hello/{name}/{year?}', function ($name, $year = null) {
    $str = '';
    if ($year == null) {
        $str = 'Hello: ' . $name;
    } else {

        $str = 'Hello: ' . $name . ', Year: ' . $year;
    }
    return view('hello')->with('str', $str);
    // hoặc
    return view('hello1')->withStr($str);
});
```

5.4. Router

- Hiển thị giá trị biến trong view:

```
<!-- Đối với file *.php và *.blade.php template -->
<h1>Hello: <?= $name?></h1>
<h1>Year: <?php echo $year?></h1>

<!-- Đối với file *.blade.php template -->
<h1>{{$str}}</h1>

<!-- Đối với file *.blade.php template, có dịch mã HTML -->
<h1>{!!$str!!}</h1>
```

5.4. Router

```
// Namespace
Route::group(['namespace' => 'Admin'], function () {
    Route::get('/', function () {
    });

    Route::get('posts', function () {
    });
});

// Truy cập vào phương thức index của Controller WelcomeController
Route::get('/', 'WelcomeController@index');

// Prefix
Route::get('admin/posts', 'PostController@index');
Route::get('admin/posts/create', 'PostController@create');
Route::post('admin/posts/store', 'PostController@store');
Route::get('admin/posts/{id}/edit', 'PostController@edit');
```

5.4. Router

```
Route::group(['prefix' => 'admin'], function () {
    Route::get('posts', 'PostController@index');
    Route::get('posts/create', 'PostController@create');
    Route::post('posts/store', 'PostController@store');
    Route::get('posts/{id}/edit', 'PostController@edit');
});

// Ràng buộc tham số truyền vào
Route::get('/hello/{name}/{year}', function ($name, $year) {
    return 'Hello: ' . $name . ', Year: ' . $year;
})->where(['name' => '[a-z]+'], ['year' => '[0-9]+']);

//Tạo route với nhiều method http
Route::match(['get','post'],'/user',function(){
    return view('user');
});

////Tạo route với mọi method HTTP
Route::any('/user',function(){
    return view('user');
});
```

5.5. View

- View được sử dụng để sắp xếp hoặc trình bày dữ liệu trước khi trả về cho trình duyệt.
- View bắt buộc phải được nằm trong thư mục resources/views và phải có đuôi là .php hoặc .blade.php (nếu như muốn sử dụng blade template).
- Trong view có thể sử dụng tất cả các ngôn ngữ trong file PHP hỗ trợ như: HTML, CSS, JS...
- Gọi view: view('duong_dan', 'du_lieu_can_truyen_vao')
  - VD: Gọi view login trong thư mục admin: view('admin.login')

```
Route::get('duong_dan', function () {
    return view('ten_view', 'data_neu_co');
});
```

5.5. View

```php
// Trong Router
// Truyền tham số cho view
Route::get('hello_user/{user}', function ($user) {
    return view('hello_user', ['user' => $user]);
});

// Truyền tham số cho view
Route::get('admin_hello_user/{user}', function ($user) {
    return view('admin.hello_user', ['user' => $user]);
});

// Truyền tham số dạng mảng
Route::get('admin_hello_user1/{user}', function ($user) {
    return view('admin.hello_user1', array('user' => $user));
});
```

```php
// Trong Controler:
// Gọi view
public function index()
{
    return view('view', 'data_neu_co');
}
```

5.5. View

- Kiểm tra sự tồn tại của view:

```
// Kiểm tra sự tồn tại của view:
Route::get('contact', function () {
    if (view()->exists('fontend.contact')) {
        return view('fontend.contact');
    } else {
        return 'Trang liên hệ đang bị lỗi, bạn vui lòng quay lại sau';
    }
});
```

- Chia sẻ dữ liệu cho tất cả các view:

```
// Trong app/Providers/AppServiceProvider.php:
public function boot()
{
    view()->share('key', 'value');
}
```

5.5. View

- Template: là 1 bố cục chung cho tất cả các trang có sử dụng lại những thành phần giống nhau mà không phải viết lại toàn bộ. Trên mỗi trang, chỉ cần thay đổi ở một số nơi được chỉ định trên trang từ template.
- Blade template:
  - Blade template là templating engine trong Laravel, có nhiều tính năng giúp tạo view, trình bày dữ liệu dễ dàng.
  - Trong các view dạng blade có thể code PHP như bình thường.
  - Các blade views được biên dịch và được cache cho đến khi có thay đổi (chạy nhanh).
  - Các file view blade được đặt tên là *.blade.php và lưu trong folder resources/views.

5.5. View

- Blade template:
  - Kế thừa: cho phép sử dụng lại template tại trang khác
    - Cú pháp: @include('_example', ['name' = $name])

```
// File _example.blade.php trong thư mục view/admin
<p><?php echo $name; ?></p>
```

```
// File example.blade.php
<div>
    @include('admin/_example', ['name'=>'noname'])
</div>
```

5.5. View

- Blade template:
  - Layout: thiết kế layout cũng là một thành phần quan trọng để có thể tận dụng tối đa sức mạnh của blade
  - Các lệnh thường dùng trong để thiết kế layout của blade view:
    - @parent: kế thừa nội dung ở blade cha
    - @section: định nghĩa bắt đầu 1 section
    - @endsection: kết thúc 1 section
    - @yield: hiển thị nội dung
    - @show: kết thúc section và hiện section
    - @extends: dùng trong view blade con, để khai báo nó kế thừa blade cha nào
    - @include: chèn view con, giúp chèn những trang con để lồng vào template

5.5. View

- Blade template:
  - Ví dụ
    - Tạo trang master.blade.php trong thư mục frontend

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Laravel: @yield('title')</title>
</head>
<body>
    @section('slidebar')
    Phần chính trong slidebar
    @show

    <div style="color: red">
        @yield('content')
    </div>
</body>
```

5.5. View

- Blade template:
  - Ví dụ
    - Tạo trang example.blade.php

```
@extends('frontend.master')

@section('title','Ví dụ về blade template')

@section('slidebar')
    @parent
    <p>Phần phụ của slidebar</p>
@endsection

@section('content')
    <p>Phần nội dung chính</p>
@endsection
```

Phần chính trong slidebar

Phần phụ của slidebar

Phần nội dung chính

5.5. View

- Blade template:
  - Hiển thị biến: {{ $variable_name }}
  - Thực thi hàm: {{ function_name() }}
  - Kiểm tra dữ liệu:
    - {{ isset($name) ? $name : 'Hello world!' }}
    - {{ $name or 'Default' }}
  - Điều kiện:
    - @if - @else - @endif
    - @if - @elseif - @else - @endif
  - Lặp:
    - @for - @endfor
    - @foreach - @endforeach
    - @forelse - @empty - @endforelse
    - @while - @endwhile
  - @continue, @break

5.5. View

- Blade template:
  - $loop->index/iteration/remaining: chỉ số vòng lặp hiện tại...
  - $loop->count: tổng số lần lặp
  - $loop->first/last: true nếu là vòng lặp đầu tiên/cuối cùng

```
@for ($i = 0; $i < 10; $i++)
    <p>The current value is {{ $i }}</p>
@endfor

@php $users = array('user1'=>'A', 'user2'=>'B'); @endphp
@foreach ($users as $user)
    @if ($loop->first) Đây là vòng lặp đầu tiên @endif
    <p>This is user {{ $user }}</p>
@endforeach

@php $isTrue = true; @endphp
@while ($isTrue)
<p>endless loop</p>
@endwhile
```

5.6. Controller

- Định nghĩa tất cả logic xử lý request ở file routes.php.
- Controller phải được đặt trong đường dẫn App\Http\controllers
- Tên của controller phải giống với tên class trong file controller đó.
- Class controller phải extends (kế thừa) từ Controller
- Tạo Controller:
  - Cách 1: tạo thủ công
  - Cách 2: tạo bằng lệnh
    - php artisan make:controller nameController [--resource] (sử dụng --resource để tạo controller có sẵn các phương thức mặc định)

5.6. Controller

– Truyền tham số sang view từ controller:

```
public function aboutme(){
    $name = 'My name is A';
    $age = '18';
    return view("pages.aboutme")->with([
        'name' => $name,
        'age' => $age
    ]);
}

public function aboutme1(){
    $name = 'My name is Hieu';
    $age = '24';
    $data = [];
    $data['name'] = $name;
    $data['age'] = $age;
    return view("pages.aboutme", $data);
}
```

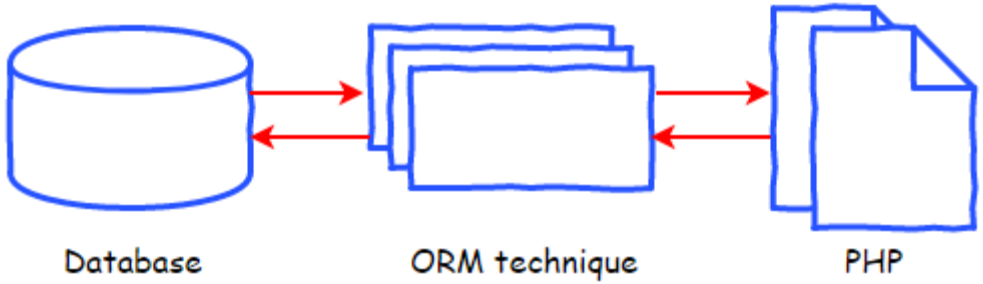
5.6. Controller

```php
<?php

namespace App\Http\Controllers;
class HomeController extends Controller
{
    public function index()
    {
        echo "Đây là phương thức index trong HomeController";
    }

    public function hello($name, $age)
    {
        echo "Hello:" . $name . ", Year: " . $age;
    }
}
```

```php
// Gọi đến Controller
Route::get('controller', 'HomeController@index');

// Gọi đến Controller và truyền tham số
Route::get('controller/{name}/{year}', 'HomeController@hello');
```

5.7. Model

- Model chứa các logic nghiệp vụ và các thao tác với cơ sở dữ liệu.
- ORM (Object Relational Mapping): là một kĩ thuật lập trình dùng để chuyển đổi dữ liệu giữa một hệ thống không hướng đối tượng như database sang một hệ thống hướng đối tượng như lập trình hướng đối tượng trong PHP.
- Eloquent Model:
  - Là thư viện ORM được cài đặt kèm với Laravel, cung cấp các thuộc tính, phương thức giúp tương tác với database một cách đơn giản, dễ triển khai.
  - Mỗi bảng dữ liệu đều được mô tả thành một Model trong Laravel và Eloquent ORM ánh xạ các dữ liệu này thành các đối tượng Active Record tương ứng với Model đó.



5.7. Model

- Điều kiện của Model:
  - Đặt ở trong thư mục App/
  - Tên class bên trong file trùng với tên file
  - Class vừa tạo phải kế thừa Model gốc của Laravel
- Tạo Model:
  - Cách 1: tạo thủ công
  - Cách 2: tạo bằng lệnh
    - php artisan make:model model_Name [--migration]
- Các hành động trên model (tương ứng với các hành động trên database - CRUD):
  - Create: tạo ra một Model, tương ứng với việc tạo ra một record dữ liệu trong bảng
  - Read: truy vấn dữ liệu từ database và map lên Model
  - Update: chỉnh sửa một record
  - Delete: xóa một record

5.7. Model

- Sử dụng Model:

```
class subject extends Model
{
    protected $table = 'subject'; // bảng cần map
    public $primaryKey = 'subject_id'; // khóa chính
    public $incresement = true;// mặc định
    public $timestaps = true;// mặc định
}
```

```
use App\subject;

class subjectController extends Controller
{
    public function index()
    {
        $listSub = subject::all();
        return view('quanlyhoc_db.subject', ['listSub' => $listSub]);
    }
}
```

5.7. Model

- Sử dụng Laravel Model (Eloquent ORM):
  - all(): lấy toàn bộ dữ liệu
  - find(): lấy một dòng dữ liệu
  - where(): lấy dữ liệu theo điều kiện
  - select(): chọn cột dữ liệu
  - count(): đếm số bản ghi
  - save(): thêm/sửa mới dữ liệu
  - update(): cập nhật dữ liệu
  - delete(): xóa dữ liệu
  - destroy(): xóa nhiều dữ liệu

5.7. Model

- Ví dụ:

```
//Lấy dữ liệu
$listSubject = subject::all();
$listSubject = subject::where('subject_id',1)->get();
$listSubject = subject::where('subject_amount','<',4)->get();
$listSubject = subject::where('subject_name','C++')-
>where('subject_amount','<',4)->get();
$listSubject = subject::select('subject_name','subject_amount')->get();

//Đếm tổng số bản ghi
$totalSubject = subject::all()->count();

//Chèn dữ liệu vào bảng
$sb=new subject();
$sb->subject_name='Java';
$sb->subject_amount='4';
$sb->save();
```

5.7. Model

- Ví dụ:

```
//Sửa dữ liệu
$sb = subject::where('subject_id', 1)->first();
$sb = subject::find(1);
$sb->subject_name = 'Python';
$sb->save();
//Hoặc
subject::where('subject_id',2)->update(['subject_name'=>'Java']);

//Xóa dữ liệu
$sb=subject::find(27);
$sb->delete();
//Hoặc
subject::where('subject_id','>',20)->delete();
//Hoặc
subject::destroy(3); subject::destroy(4, 5);
subject::destroy([4, 5, 6]); subject::destroy(array(7,8));
```

5.8. Làm việc với Database

- Laravel hiện đang hỗ trợ 4 loại database phổ biến hiện nay: MySQL (MariaDB), Postgres, SQLite, SQL Server.
- Truy vấn đến CSDL trong Laravel có 3 cách:
  - Sử dụng SQL thuần với các phương thức:
    - select, insert, update, delete
    - statement: chạy bất kỳ câu lệnh SQL nào (create, drop...)
  - Sử dụng Query Builder
  - Sử dụng Laravel Model (Eloquent ORM)

5.8. Làm việc với Database

- Thiết lập kết nối database
  - file .env: dùng để chạy localhost

```
DB_CONNECTION=mysql
DB_HOST=localhost
DB_PORT=3306
DB_DATABASE=quanlyhoc_db
DB_USERNAME=root
DB_PASSWORD=
```

  - config/database.php: dùng để chạy thực tế

```
'driver' => 'mysql',
'url' => env('DATABASE_URL'),
'host' => env('DB_HOST', '127.0.0.1'),
'port' => env('DB_PORT', '3306'),
'database' => env('DB_DATABASE', 'forge'),
'username' => env('DB_USERNAME', 'forge'),
'password' => env('DB_PASSWORD', ''),
```

5.8. Làm việc với Database

- Ví dụ: tạo trang web hiển thị danh sách sinh viên trong bảng sinh_vien (sử dụng SQL thuần)
  - Tạo file SinhVienController:

```
class SinhVienController extends Controller
{
    public function index()
    {
        $listSV = DB::select('select * from sinh_vien');
        // truyền tham số
        $listSV = DB::select('select * from sinh_vien where makhoa = ?',
['01']);
        // hoặc
        $listSV = DB::select('select * from sinh_vien where makhoa = :ma
khoa', ['makhoa'=>'01']);
        return view('quanlyhoc_db.SinhVien', ['listSV' => $listSV]);
    }
}
```

```
Route::get('sinhvien', 'SinhVienController@index');
```

5.8. Làm việc với Database

- Ví dụ:
  - Tạo file SinhVien.blade.php:

```
<table border="1">
    <tr>
        <th>Mã SV</th>
        <th>Mã khoa</th>
        <th>Lớp</th>
        <th>Họ tên</th>
    </tr>
    @foreach($listSV as $sv)
    <tr>
        <td>{{$sv->MaSV}}</td>
        <td>{{$sv->MaKhoa}}</td>
        <td>{{$sv->LopBC}}</td>
        <td>{{$sv->HoTen}}</td>
    </tr>
    @endforeach
</table>
```

Mã SV	Mã khoa	Lớp	Họ tên
01	01	57K	Nguyen thi luong
03	01	58K1	Nguyễn Văn Nam
04	01	58K1	Hồ Thị Mai

5.8. Làm việc với Database

- Sử dụng Query Builder:

```
//Lấy sinh viên
$listSV = DB::table('sinh_vien')->get();
//Join bảng
$listSV = DB::table('sinh_vien')-
>join('khoa_dao_tao', 'sinh_vien.MaKhoa', '=', 'khoa_dao_tao.MaKhoa')->get();
// //Tìm theo điều kiện
$listSV = DB::table('sinh_vien')->where('LopBC', '58K1')->get();
$listSV = DB::table('sinh_vien')->where('HoTen','like', '%n%')->get();
$listSV = DB::table('sinh_vien')->where([['LopBC','58K1'],['MaKhoa','01']])-
>get();
//Lấy một cột
$listSV = DB::table('sinh_vien')->where('LopBC', '58K1')->value('HoTen');
// //Lấy nhiều cột
$listSV = DB::table('sinh_vien')->pluck('HoTen', 'LopBC');
$listSV = DB::table('sinh_vien')->select('HoTen', 'LopBC')->get();
// //Lấy giá trị tổng hợp
$listSV = DB::table('sinh_vien')->count();
// //Raw expression
$listSV = DB::table('sinh_vien')->select(DB::raw('count(*) as sv_count'))-
>where('MaKhoa','<>','01')->groupBy('MaKhoa')->get();
```

5.8. Làm việc với Database

- Sử dụng Query Builder:

```
//Sắp xếp kết quả trả về
$listSV = DB::table('sinh_vien')->orderBy('HoTen', 'desc')->get();
//Group và Having
$listSV = DB::table('sinh_vien')->groupBy('LopBC')->having('LopBC', '01')->get();
//Insert bản ghi
DB::table('sinh_vien')->insert([
    ['MaSV' => '05', 'MaKhoa' => '01', 'LopBC' => '58k2', 'HoTen' => 'Hồ Thị Hiền'],
    ['MaSV' => '06', 'MaKhoa' => '01', 'LopBC' => '58k4', 'HoTen' => 'Trần Đình An']
]);
//Insert bản ghi và lấy ra id
$idSV = DB::table('sinh_vien')-
>insertGetId(['MaSV' => '05', 'MaKhoa' => '01', 'LopBC' => '58k2', 'HoTen' => 'Hồ Th
ị Hiền']);
//Update dữ liệu
DB::table('sinh_vien')->where('MaSV','05')->update(['LopBC'=>'58K2']);
//Delete dữ liệu
DB::table('sinh_vien')->where('MaSV','05')->delete();
//Xóa toàn bộ bản ghi và reset id
DB::table('sinh_vien')->truncate();
```

5.8. Làm việc với Database

- Giao dịch (transaction): là một nhóm các hành động có thứ tự trên database nhưng lại được người dùng xem như là một đơn vị thao tác duy nhất.
  - Ví dụ: một giao dịch thanh toán trực tuyến (trừ tiền tài khoản khách hàng, cộng tiền tài khoản nhà cung cấp, giảm số lượng hàng trong kho, tạo mới đơn hàng, lưu log...)
- Nếu một hành động lỗi, sẽ dẫn đến giao dịch lỗi, hệ thống sẽ thực hiện các lệnh sao cho quay về trạng thái ban đầu.

```
DB::transaction(function () {
    DB::table('users')->update(['votes' => 1]);
    DB::table('posts')->delete();
}, 5);
```

5.8. Làm việc với Database

- Schema Builder:
  - Là một class trong bộ Facades của Laravel giúp làm việc với những cơ sở dữ liệu mà Laravel hỗ trợ với các hàm được định nghĩa sẵn.
  - Thường được kết hợp với Migrations để xây dựng cấu trúc database.
  - Điều kiện để sử dụng:
    - Có kết nối với database
    - Sử dụng namespace Illuminate\Support\Facades\Schema

5.8. Làm việc với Database

- Schema Builder: Các lệnh thường dùng
  - Tạo bảng: Schema::table('tablename', function ($table) { //code })
  - Đổi tên bảng: Schema::rename($from, $to)
  - Xóa bảng:
    - Schema::drop('tableName')
    - Schema::dropIfExists('tableName')
  - Thêm cột: $table->typeData('columnName')
  - Đổi tên cột: $table->renameColumn('from', 'to')
  - Xóa cột:
    - $table->dropColumn('columnName')
    - $table->dropColumn(['columnName1','columnName2','columnName3'])

5.8. Làm việc với Database

- Schema Builder: Các lệnh thường dùng
  - Tạo index: $table->index('columnName')
    - Các kiểu index: index(), primary(), unique()
  - Xóa index:
    - $table->dropPrimary('users_id_primary');
    - $table->dropUnique('users_email_unique');
    - $table->dropIndex('geo_state_index');
  - Thêm khóa ngoại:
    - $table->integer('columnNameA')->unsigned();
    - $table->foreign('columnNameA')->references('columnNameB')->on('tableName2');
  - Xóa khóa ngoại
    - $table->dropForeign('columnName');

5.8. Làm việc với Database

- Migration:
  - Migration giống như một hệ thống quản lý phiên bản (version control) dành cho database.
  - Migration cho phép định nghĩa các bảng trong database, định nghĩa nội dung các bảng, cập nhật thay đổi các bảng đó hoàn toàn bằng PHP.
  - Migrations giúp cho việc quản lý database trở nên dễ dàng hơn.
  - Điều kiện để chạy migration:
    - Có kết nối đến database
    - Migration nằm trong thư mục \App\database\migrations

5.8. Làm việc với Database

- Migration:
  - Các phương thức:
    - Function up(): tạo bảng, thêm, bớt, thay đổi bảng...
    - Function down(): phục hồi lại trạng thái ban đầu tại thời điểm function up() chưa được thực thi
  - Thực thi phương thức:
    - Phương thức up(): php artisan migrate [--path=path_of_file_migration]
    - Phương thức down(): php artisan migrate:rollback/reset
    - Phương thức down() → up(): php artisan migrate:refresh
    - Tạo bảng migration: php artisan migrate:install
    - Trạng thái các file migration đã chạy: php artisan migrate:status
  - Tạo migration:
    - Tạo bảng: php artisan make:migration migration_name [--create=TableName]
    - Sửa bảng: php artisan make:migration migration_name --table=TableName

5.8. Làm việc với Database

- Migration:
  - Ví dụ: php artisan make:migration create_teacher --create=Teacher

```
class CreateTeacher extends Migration
{
    /*** Run the migrations.
     * @return void */
    public function up()
    {
        Schema::create('Teacher', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }
    /*** Reverse the migrations.
     * @return void */
    public function down()
    {
        Schema::dropIfExists('Teacher');
    }
}
```

5.8. Làm việc với Database

- Migration:

```
Schema::create('sanpham', function (Blueprint $table) {
    $table->increments('id');
    $table->string('tensp',30)->unique();
    $table->float('gia',8,2);
    $table->string('urlhinh',100);
});

Schema::table('teacher',function($table){
    // Cho phép null
    $table->string('column_name')->nullable();
    // Đổi tên cột
    $table->renameColumn('old_column','new_column');
    // Xóa cột
    $table->dropColumn('column_name');
    // Tạo foreign key constraints
    $table->integer('foreign_id')->unsigned();
    $table->foreign('foreign_id')->references('main_id')->on('users');
    // Xóa foreign key cóntraints
    $table->dropForeign('foreign_id');
});
```

5.8. Làm việc với Database

- Migration:
  - Schema:
    - Kiểm tra tồn tại bảng không: Schema::hasTable('table_name‘)
    - Kiểm tra tồn tại cột trong bảng không: Schema::hasColumn('table_name','column_name')
    - Đổi tên bảng: Schema::rename('table_name')
    - Xóa bảng: Schema::drop('table_name')
    - Xóa bảng nếu tồn tại: Schema::dropIfExists('table_name')

5.8. Làm việc với Database

- Seeding (seed):
  - Là một class chứa code để tạo ra các dummy data (dữ liệu mẫu) cho database trong quá trình xây dựng ứng dụng.
  - Có thể dùng dòng lệnh để nhập hàng loạt dữ liệu mẫu tự động
  - Các file seeder được đặt trong thư mục /database/seeds
  - Có thể viết code của Query Builder trong seeder
  - Các thao tác:
    - Tạo file: php artisan make:seeder file_name
    - Thực thi file: php artisan db:seed --class=file_name

```
class subject extends Seeder
{
     * @return void */
    public function run()
    {
        //
    }
}
```

5.8. Làm việc với Database

- Seeding (seed): Một số Faker tạo dữ liệu mẫu

Mục	Faker
Title, Name	$fake->name
Description	$fake->sentence
Content	$fake->sentence(500)
Label	$fake->randomLetter
Email	$fake->unique->email
Tel	$fake->phoneNumber
Number	$fake->numerify($string = '###')
City	$fake->city
Country	$fake->city
Category	$fake->numberBetween($min = 1, $max = 2)
Public	$fake->boolean()
Date	$fake->date("Y-m-d H:i:s")

5.8. Làm việc với Database

- Seeding (seed):
  - Ví dụ: insert dữ liệu vào bảng subject:

```php
public function run()
{
    // Insert dữ liệu bằng tay
    DB::table('subject')->insert([
        ['subject_name' => 'C', 'subject_amount' => 4],
        ['subject_name' => 'C++', 'subject_amount' => 3],
        ['subject_name' => 'C#', 'subject_amount' => 3],
    ]);

    // Insert dữ liệu ramdom
    for ($i = 0; $i <= 5; $i++) {
        DB::table('subject')->insert([
            'subject_name' => Str::random(15),
            'subject_amount' => mt_rand(1, 5),
            'created_at' => Carbon::now()
        ]);
    }
}
```

5.9. Collection

- Collection là một class tích hợp sẵn các phương thức xử lý dữ liệu nhằm làm giảm thiểu tối đa thời gian cho các lập trình viên
- Đặc điểm:
  - Có thể thay đổi kích thước (số lượng các phần tử) và sử dụng bộ nhớ động
  - Cung cấp các cấu trúc dữ liệu hữu dụng giúp lập trình nhanh hơn.
  - Hỗ trợ các phần tử không đồng nhất về kiểu
  - Có tính mở rộng cao, có tính thích ứng
  - Có các phương thức được xây dựng sẵn như sắp xếp, tìm kiếm...
- Khai báo: use Illuminate\Support\Collection
- Khởi tạo (với [] là mảng giá trị truyền vào):
  - $collection = collect([])
  - $collection = Collection::make([])

5.9. Collection

- In giá trị:
  - prinf_r($collection)
  - prinf_r($collection->toArray())
- Các collection hay dùng:
  - All: lấy tất cả giá trị
  - Filter: lọc dữ liệu
  - Expect: loại bỏ các giá trị không cần thiết
  - Count/avg/sum/...: đếm/trung bình/tính tổng/...
  - Chumk: tách mảng thành mảng con
  - First: lấy phần tử đầu tiên
  - Get: lấy giá trị các phần tử
  - SortBy/SortByDesc: sắp xếp tăng dần/giảm dần
  - Take: giới hạn số lượng trả về
  - ...

5.10. HTTP Request

- Laravel Request cung cấp dữ liệu về các yêu cầu HTTP cũng như cho phép thao tác với các yêu cầu này
- Khai báo request trong Controller:
  - use Illuminate\Http\Request
- Các phương thức:
  - path(): thông tin đường dẫn yêu cầu
  - is(): kiểm tra đường dẫn có khớp với mẫu không
  - url(): đường dẫn yêu cầu đầy đủ của url không có query string
  - fullurl(): đường dẫn yêu cầu đầy đủ
  - method()/isMethod(): phương thức yêu cầu/kiểm tra phương thức
  - ip(): địa chỉ ip của người dùng
  - server(): thông tin liên quan đến máy chủ
  - header(): thông tin header của request
  - input('name'): lấy thông tin nhập liệu

5.10. HTTP Request

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

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

$url = $request->url();

$url = $request->fullUrl();

$method = $request->method();
if ($request->isMethod('post')) {
    echo 'POST request';
} else {
    echo 'GET request';
}
```

5.10. HTTP Request

- Ví dụ: tạo trang thêm mới dữ liệu cho bảng khoa_dao_tao
  - Route:

```
Route::get('KhoaDaoTao', 'KhoaDaoTaoController@ShowKhoaDaoTao');
Route::post('KhoaDaoTao', 'KhoaDaoTaoController@InsertKhoaDaoTao');
```

  - Controller KhoaDaoTaoController:

```
public function InsertKhoaDaoTao(Request $request)
{
    $maKhoa=$request->input('MaKhoa');
    $tenKhoa=$request->input('TenKhoa');
    $dienThoai=$request->input('DienThoai');

    DB::table('khoa_dao_tao')-
>insert(['MaKhoa'=>$maKhoa,'TenKhoa'=>$tenKhoa,'DienThoai'=>$dienThoai]);
    return view('quanlyhoc_db.KhoaDaoTao')-
>with(['success'=>'Them moi thanh cong']);
}
```

5.10. HTTP Request

- Ví dụ: file KhoaDaoTao.blade.php

```
<form method="POST" action="/KhoaDaoTao">
    <input type="hidden" name="_token" value="<?php echo csrf_token() ?>">
    <table>
        <tr>
            <td>Mã khoa:</td>
            <td><input type="text" name='MaKhoa'></td>
        </tr>
        <tr>
            <td>Tên khoa:</td>
            <td><input type="text" name='TenKhoa'></td>
        </tr>
        <tr>
            <td>Điện thoại:</td>
            <td><input type="text" name='DienThoai'></td>
        </tr>
        <tr>
            <td colspan="2" align="center"><input type="submit" value="Gui">
        </tr>
    </table>
    <?php if (isset($success)) {
        echo $success;
    } ?>
</form>
```

5.11. Validation

- Validation là chức năng dùng để thẩm định sự chính xác của dữ liệu đầu vào nhằm đưa ra các quy tắc và kiểm tra xem dữ liệu có đúng với các quy tắc đó không
- Sử dụng type-hint của Form Request trong phương thức của Controller để validate trước khi phương thức trong Controller được gọi
- Nếu dữ liệu không thỏa mãn điều kiện kiểm tra, một response chuyển hướng được sinh ra và gửi ngược trở lại cho người dùng cùng với các thông báo lỗi
- Nếu request ở dạng AJAX, một HTTP response với mã trạng thái 422 sẽ được trả về cùng với dữ liệu dạng JSON chứa các lỗi kiểm tra
- Tạo Validation: php artisan make:request request_name

5.11. Validation

- Các yêu cầu khi dùng validation:
  - File validation nằm trong thư mục app/Http/Requests
  - Khai báo validation trong Controller:
    - use App\Http\Requests\validation_name
  - Tham số request trong Controller là dạng class request
- Các phương thức trong class validation:
  - rules(): khai báo các quy tắc kiểm tra dữ liệu, trả về 1 array các quy tắc
  - authorize(): cho phép/cấm dùng class này, trả về true (cho phép), false (cấm)
  - message(): định nghĩa các chuỗi thông báo lỗi thân thiện (khai báo thêm)
  - attributes(): định nghĩa tên thân thiện cho các field (khai báo thêm)

5.11. Validation

- Ví dụ: sử dụng validation đối với bảng khoa_dao_tao:
  - Controller KhoaDaoTaoController:

```
public function InsertKhoaDaoTao(CheckKhoaDaoTao $request)
{
    $input=$request->only(['MaKhoa','TenKhoa','DienThoai']);
    DB::table('khoa_dao_tao')->insert($input);
    return view('quanlyhoc_db.KhoaDaoTao')-
>with(['success'=>'Them moi thanh cong']);
}
```

  - Class validation CheckKhoaDaoTao.php:

```
public function rules()
{
    return [
        'MaKhoa' => 'required|min:2|max:10',
        'TenKhoa' => 'required|min:5|max:20',
        'DienThoai' => 'required|integer'
    ];
}
```

5.11. Validation

- Ví dụ: sử dụng validation đối với bảng khoa_dao_tao:
  - Class validation CheckKhoaDaoTao.php:

```
public function messages()
{
    return [
        'required' => ':attribute chưa nhập',
        'min' => ':attribute quá ngắn',
        'max' => ':attribute quá dài',
        'integer'  => ':attribute phải là số'
    ];
}

public function attributes(){
    return [
        'MaKhoa' => 'Mã khoa',
        'TenKhoa' => 'Tên khoa',
        'DienThoai' => 'Điện thoại'
    ];
    }
```

5.11. Validation

- Ví dụ: sử dụng validation đối với bảng khoa_dao_tao:
  - View KhoaDaoTao.blade.php
    - Dùng biến $errors:

```
@if($errors->any())
<ul>
    @foreach($errors->all() as $error)
    <li>{{$error}}</li>
    @endforeach
</ul>
@endif
```

    - Hoặc dùng hàm @error để check lỗi cho từng field và hiển thị với biến $message tại vị trí của field:

```
<tr>
    <td>Mã khoa:</td>
    <td><input type="text" name='MaKhoa' value="{{ old('MaKhoa') }}">
        @error('MaKhoa') {{$message}} @enderror
    </td>
</tr>
```

5.11. Validation

Ma khoa:

Ten khoa: CNTT

Dien thoai: Phone

Thêm mới

Ma khoa: Mã khoa chưa nhập

Ten khoa: Tên khoa quá ngắn

Dien thoai: Điện thoại phải là số

Thêm mới

- Mã khoa chưa nhập
- Tên khoa quá ngắn
- Điện thoại phải là số

5.11. Validation

- Tạo rule kiểm tra mới:
  - Tạo rule: php artisan make:rule rule_name
  - Rule mới sẽ được đặt trong thư mục App\Rules
  - Các phương thức trong rule:
    - passes(): viết code để định nghĩa dữ liệu như thế nào là pass, trả về true/false
    - message(): thông báo khi dữ liệu không pass, trả về chuỗi
  - Khai báo rule mới trong validation:
    - Use App\Rules\rule_name
  - Sử dụng:

5.11. Validation

- Ví dụ: tạo và áp dụng rule kiểm tra chữ hoa cho tên khoa
  - File CheckUpper.php:

```
public function passes($attribute, $value)
{
    return strtoupper($value) == $value;
}

public function message()
{
    return ':attribute phải là chữ hoa';
}
```

  - Controller KhoaDaoTao.php:

```
return [
    'MaKhoa' => 'required|min:2|max:10',
    'TenKhoa' => ['required','min:5','max:20',new CheckUpper],
    'DienThoai' => 'required|integer'
];
```

5.12. Authentication

- Authentication là một hành động nhằm thiết lập hoặc chứng thực một cái gì đó (hoặc một người nào đó) đáng tin cậy
- Authentication trong Laravel giúp tạo nên chức năng login, logout, register… một cách dễ dàng
- Khai báo các thông số authentication trong file config/auth.php
- Sử dụng model User (app\user.php)
- Triển khai authentication:
  - Cài đặt gói laravel/ui: composer require laravel/ui
  - Chạy lệnh: php artisan ui vue --auth

5.12. Authentication

- Các file mới, cấu hình mới sẽ được thêm vào:
  - Tạo các view trong folder views\auth: login.blade.php, register.blade.php…
  - Tạo file views\layouts\app.blade.php: là 1 file layout cơ bản với các class css dựa trên bootstrap
  - Tạo các đường route xử lý authentication trong file route/web.php
  - Tạo file Http\Controllers\HomeController.php
  - Tạo các controller trong folder Http\Controllers\Auth như RegisterController, LoginController, ForgotPasswordController, ResetPasswordController.
- Giao diện trang chủ sẽ được bổ sung thêm chức năng: login, register

📖 Câu hỏi thảo luận