**Giới thiệu**

Phần này được soạn cho các học viên có kiến thức căn bản về Php, mô hình MVC và csdl.

Các bài học, bài tập được thiết kế tù thấp đến cao và có tính kế thừa. Các học viên nên làm hết các bài tập cơ bản buổi trước để có kiến thức cho các buổi học sau.

Các phần thực hành hướng tới việc xây dựng một project hòan chỉnh sử dụng Laravel, giúp các học viên có các kiến thức thực tế sau khi kết thúc khóa học.

Đồ án bài tập: xây dựng một website bán sách (bookstore) với 2 ngôn ngữ: Việt - English. Có các chức năng cơ bản như:

User: Có nhiều chức năng cho người sử dụng:

* Xem, tìm kiếm sản phẩm, phân trang sản phẩm,
* Tin tức
* Quản lý tài khoản: đăng ký tài khoản (có thể sử dụng API các mạng xã hội), đăng nhập, chỉnh sửa tài khoản.
* Mua hàng, quản lý đơn hàng,…
* Chuyển đổi ngôn ngữ (nếu website có nhiều ngôn ngữ).
* Bình luận, đánh giá sản phẩm.

Admin: Có chức năng quản lý:

* Sản phẩm: Hiển thị danh sách, tìm kiếm, thêm, sửa xóa. Phần này kết hợp với các công cụ, công nghệ như: Ajax, datatable, bootstrap, ckEditor, …
* Tin tức
* Đơn hàng
* Khách hàng.
* Chăm sóc khách hàng: gửi email khuyến mãi tới danh sách khách hàng.
* Thống kê: Theo nhiều tiêu chí, sử dụng giao diện đồ họa.

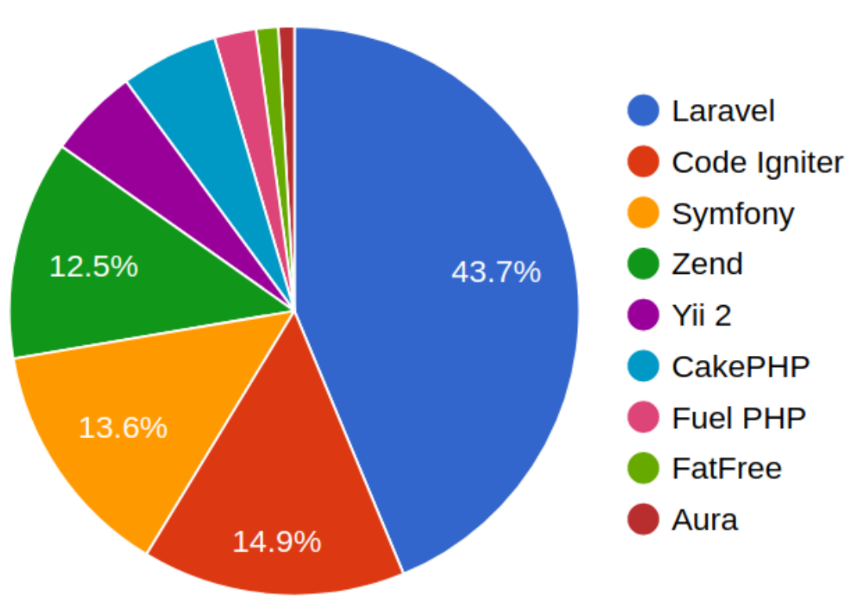
Thông tin cung cấp cho học viên

* Database: file bookstore.sql + data liên quan
* Một số template mẫu

Bài 1: Giới thiệu cài đặt Laravel và các công cụ liên quan

1. **Mục tiêu**
   * Hiểu, cài đặt sử dụng các công cụ để triển khai một project Laravel.
   * Cài đặt Laravel.
   * Cấu trúc project Laravel.
   * Hiện thực kết quả.
2. **Lý thuyết**
3. Laravel là gì? Tại sao sử dụng Laravel.

* Laravel là một framework mã nguồn mở php được phát triển nhanh và được sử dụng nhiều hiện nay trong xây dựng website.



Theo: <https://ozvid.com/blog/46/reasons-why-laravel>

1. Các tool cần thiết khi lập trình web Laravel

* Composer:
  + một công cụ Dependency Management trong PHP, công cụ quản lý các thư viện mà project Php sử dụng. Cho phép khai báo các thư viện mà dự án của bạn sử dụng, composer sẽ tự động tải code của các thư viện. Nó tạo ra các file cần thiết vào project, và cập nhật các thư viện khi có phiên bản mới…
  + Khai báo các thư viện mà dự án sử dụng. Quản lý tập trung các thư viện đang sử dụng cho project và cả phiên bản của chúng dễ dàng qua file composer.json.
  + Tìm các phiên bản của package có thể cài đặt và cần thiết cho dự án, sau đó cài đặt chúng vào dự án.
  + Một phần mềm tương tự cho node.js là NPM (Node package manager) là một công cụ tạo và quản lý các thư viện lập trình Javascript cho Node.js.
* Wamp (hoặc các ứng dụng tương ứng thay thế như xampp, appsev,..): ứng dụng các phần mềm webserver
* IDE: Visual studio code hoặc Sublime text.

1. Cài đặt Laravel sau khi đã cài đặt composer

* Tới thư mục cần tạo project bằng dòng lệnh (wamp/www -> localhost). Có thể sử dụng windows exploder mở thư mục /Bấm phím Shift + click phải chuột/ open PowerShell windows here.
* Chọn một trong 3 cách sau để tạo một project tên pro01:
  + 1. Gõ lệnh: composer create-project larevel/laravel **pro1.** Lệnh này sẽ tải ứng dụng Laravel phù hợp với php đã định nghĩa trong composer. Nếu muốn tải version Laravel phù hợp, sử dụng lệnh:

composer create-project laravel/laravel ProName "Version"

Ví dụ: composer create-project laravel/laravel appLaravel7 "7.\*"

* + 1. Sử dụng 2 lệnh:
       1. composer global require Laravel/installer . Sau lệnh này, kết quả được lưu trong C:/Users/…/AppData/Roaming/Composer. Có thể xem vị trí này bằng lệnh: composer global about
       2. laravel new **pro1**
    2. Download sourcecode trực tiếp từ máy chủ quản lý github.

**Tip:**

* Khi cài đặt, các thư viện được lưu trong thư mục vendor và danh sách các thư viện này nằm trong file composer.json. Khi phân phối, ta có thể không cần gửi thư mục này. Khi nhận sourcecode, từ thư mục, chạy lệnh: composer install để composer tải thư mục vendor theo yêu cầu về.
* Để xem version của Laravel, có thể: xem trong file composer.json hoặc trong vendor\laravel\framework\src\Illuminate\Foundation\Application.php hoặc bằng câu lệnh của công cụ Artisan: php artisan -V

1. Chạy thử website trên localhost:

* Mở wampserver
* Mở localhost-> chạy <http://localhost/>pro1/public
* Trong đó: pro1 là thư mục project vừa tạo.

1. Các thư mục quan trọng của Laravel

* App: chứa các code cấu hình, controllers, model và một số thư viện của ứng dụng.
* Boostrap: chứa file app.php làm việc như một bootstrap của ứng dụng và thư mục cache dùng để chứa các file bộ nhớ config, route, services... cho việc tối ưu hiệu năng.
* Config: chứa các file cấu hình của project như cấu hình về database, session, mail,…
* Database: chứa các thành phần hỗ trợ làm việc với CSDL
* Public: chứa file index.php, file này đảm nhận vai trò như một đích đến của các request và autoload các lớp. Ngoài ra nó còn chứa các tài nguyên mà trình duyệt (browser) có thể truy cập như JS, CSS, hình ảnh, video...
* Resources: chứa các tài nguyên chưa được biên dịch như view, LESS, SASS hoặc Javascript. Các file sau khi được biên dịch sẽ chuyển sang thư mục storage.
* Routes: Thư mục routes chứa các tuyến đường (route) đã định nghĩa của ứng dụng. Mặc định có các file: api.php, web.php, channels.php và console.php được kết nối với Laravel.
  + - File web.php: chứa những route chứa request từ trình duyệt, chịu ảnh hưởng từ session, cookie, CSRF (tính năng bảo mật trong Laravel). hầu như các route sẽ nằm trong file này.
      * Ví dụ: khi truy cập trang chủ (/) ứng dụng trả về view có tên welcome trong thư mục views.

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

return view('welcome');

});

* + - File api.php: chứa các route có chức năng như là RESTful API.
    - File console.php: nơi đây bạn có thể định nghĩa các Clouser bằng các lệnh console.
    - File channels.php: sử dụng khi ứng dụng sử dụng thời gian thực (real-time), file này hỗ trợ cho ứng dụng của bạn có thể tương tác các sự kiện giữa phía người dùng (client-side) và phía hệ thống (server-side).
* Storage: chứa các file blade template đã được phiên dịch (complied), các file session, file cache và một số file khác được tạo bởi framework. Thư mục này gồm app, framework và logs.
  + - Thư mục app dùng để lưu trữ bất kỳ file nào do ứng dụng của mình tạo ra. Thư mục storages/app/public có thể dùng để lưu trữ các file do người dùng (user) đăng tải, chẳng hạn như ảnh đại diện (avatar) có thể truy cập công khai.
    - framework dùng để lưu trữ các file mà framework tạo ra để hỗ trợ trong việc chạy ứng dụng (các file được biên dịch).
    - Logs sẽ chứa các file log gồm có các lỗi trong quá trình code (errror log).
* Tests: chứa các class thử nghiệm bằng commander
* Vendor: chứa bộ mã nguồn của laravel và các thành phần đi kèm laravel, cũng như các gói (packages) sau này sẽ thêm vào laravel.

1. Cấu hình Laravel: cần nhiều cấu hình: Cấu hình chung, gửi email, database, upload,…

* File .env:
* Thư mục config
* File composer.json

1. Các bước publish website đã có lên hosting.

* Upload sourcecode lên hosting bằng phần mềm FTP hoặc file .zip
* Đăng nhập vào phần quản lý của hosting (cpanel) và tạo csdl, username và mật khẩu để sử dụng csdl này. Sau đó sử dung phpmyadmin để import csdl từ file .sql.
* Sửa lại file cấu hình cho phù hợp csdl vừa tạo (trong file .env hoặc config/database.php).

1. Chạy thử Lararavel với Database mysql

Laravel sử dụng PDO để kết nối đến CSDL và có thể kết nối đến nhiều loại CSDL khác nhau. Laravel có nhiều thư viện hỗ trợ quản lý, sử dụng database nên thao tác với CSDL trong Laravel rất nhanh.

* Tạo CSDL trong mysql (hoặc MariaDB)
* Cấu hình trong file .env
* Mở file routes/web.php và chỉnh sửa code trong route trang chủ như sau:

Route::get('/', function () {

    $data = DB::table('book')->get();

    dd($data);

  //  return view('welcome');

});

* Chạy kiểm tra lại kết quả trang trang chủ website.

Lệnh DD($var): sử dụng để debug (cùng với các lệnh print\_r hay var\_dump)

1. **Thực hành cài đặt công cụ**
   * + - 1. Cài đặt công cụ: wampserver - Kiểm tra kết quả cài đặt

* Download wamp từ: <https://www.wampserver.com/en/>
* Cài đặt theo hướng dẫn.
* Mở wamp, nhập vào localhost trong trình duyệt web. Xác định đường dẫn documentroot trên máy (thường là c:\wamp\www).
* Sử dụng các tool trong trang chủ wamp server (trong hình) để xem các thông tin cài đặt trên wamp, quản lý database và tạo host ảo trỏ tới thư mục public.
* Tạo database bookstore (sử dụng phpMyAdmin) và csdl được cung cấp



* + - * 1. Cài đặt và sử dụng composer
* Download: https://getcomposer.org/
* Cài đặt: chú ý chọn version php đang sử dụng (c:\wamp\bin\php\php ..\php.exe)
* Cấu hình
* Kiểm tra kết quả cài đặt composer: trong màn hình command -> chạy composer – version
  + - * 1. Cài đặt, kiểm tra node.js (tùy chọn). được sử dụng để chuyển đổi các file CSS pre-processors sang các file css thông thường.
        2. Cài đặt Laravel: sử dụng command
* Mở màn hình command và di chuyển tới thư mục www (tương ứng với <http://localhost>). Trong máy windows, mở thư mục www bằng windows explorer, Giữ phím shift + right click, chọn: Open Powershell window here
* Gõ vào dòng lệnh: composer **create-project Laravel/laravel proName**

Trong đó: proName: tên project (thư mục) muốn tạo.

* Chạy thử kết quả: <http://localhost/proName/public>
* Tạo host ảo (ví dụ proName.local) trỏ tới www/proName/public. Restart DSN rồi chạy thử: <http://proName.local>
* Kiểm tra phiên bản Laravel đang sử dụng: Mở file: vendor/laravel/framework/src/illuminate/Foundation/ Application.php, tìm hằng số **const VERSION**. const VERSION = **'7.22.4'**;
  + - * 1. Cài đặt database bookstore\_vn bằng phpMyAdmin (file source ookstore\_vn.sql).

1. **Thực hành thao tác trên Laravel mới cài đặt**

* Cấu hình liên quan tới CSDL và host

Mở file .env, thay thế thông tin cho CSDL và host. Sau đó chạy thử kết quả.

APP\_DEBUG=true

APP\_URL=http://proName.local

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=bookstore\_vn

DB\_USERNAME=root

DB\_PASSWORD=

Mở file config/database.php, tới dòng **'database' => env('DB\_DATABASE', 'forge').**

Hàm env(Hằng\_số\_trong \_.env, giá\_trị\_mặc\_định): hàm env sẽ đọc hằng số trong file .env, nếu có, trả về giá trị này, nếu không có sẽ trả về database mặc định 'forge'. Có thể thay thế nội dung và chạy thử kết quả.

* Route và view.

Xem nội dung file routes/web.php

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

return view('welcome');

});

Xem nội dung file Views/welcome.blade.php

* 1. Thêm các route sau trong routes/web.php

route::get('/san-pham', function(){

return 'San Pham';

});

route::get('/san-pham2', function()

{

return view(**'sanpham2'**, ['data'=>'Trang san pham']);

});

route::get('/san-pham3', function()

{

return view(**'sanpham3'**, ['data'=>**DB::select('select \* from sach')]** );

});

route::get('chi-tiet/{id}', function($id){

return view('chi-tiet')->with(**'id'**, $id);

})->where(['id'=>'[0-9]+']);

* 1. Thêm các view:
* Resources/views/**sanpham2.blade.php**, có nội dung

<?php echo **$data;** ?>

* Resources/views/sanpham3.blade.php, nội dung

<?php print\_r**($data**); //DD($data); ?>

* Resources/views/chi-tiet.blade.php, nội dung

Trang chi tiet cua san pham co id (number) là: {{$id}}

* 1. Chạy thử kết quả và cho nhận xét
* htttp://proName.local/san-pham
* htttp://proName.local/san-pham2
* htttp://proName.local/san-pham3
* htttp://proName.local/chi-tiet/10
* htttp://proName.local/chi-tiet/th

1. Thực hành với ứng dụng Laravel hoàn chỉnh. (tin tức)

* Unzip source code
* Tạo csdl
* Chỉnh sửa config: trong .env, trong thư mục config
* Chạy thử kết quả.
* Tạo virtual host trỏ tới public
* Chạy thử kết quả.

1. Upload Laravel tintuc lên hosting

* Xóa cache trong Laravel
* Upload sourcecode
* Tạo csdl trên host
* Cấu hình
* Chạy thử

**Tip**: Công cụ Artisantrong Laravel: là một công cụ dòng lệnh được tích hợp sẵn trong các dự án sử dụng Laravel, nó cung cấp rất nhiều các chức năng trợ giúp việc xây dự án, giảm thời gian viết code cũng như tự động hóa một số công việc. Các việc hay sử dụng với Artisan:

* Vận hành dự án: Hỗ trợ các công việc liên quan đến vận hành dự án như tối ưu hóa, chuyển ứng dụng sang chế độ bảo trì, chạy các công việc ngầm theo kiểu hàng đợi (queue job), tạo và thay đổi sử dụng bộ đệm dữ liệu.
* Quản lý CSDL: Các công việc xử lý cơ sở dữ liệu như migrate, đưa dữ liệu vào database, tạo dữ liệu kiểm thử.
* Tạo ra các Model, Controller, Event...
* Bảo mật: cài đặt xác thực người dùng, tạo key mã hóa và các việc liên quan đến sử dụng OAuth2.
* Sử dụng câu lệnh php artisan list để xem danh sách các câu lệnh hỗ trợ bởi Artisan.

Danh sách lệnh Artisan: php artisan list

Laravel Framework 8.19.0

Usage:

command [options] [arguments]

Options:

-h, --help Display help for the given command. When no command is given display help for the list command

-q, --quiet Do not output any message

-V, --version Display this application version

--ansi Force ANSI output

--no-ansi Disable ANSI output

-n, --no-interaction Do not ask any interactive question

--env[=ENV] The environment the command should run under

-v|vv|vvv, --verbose Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

Available commands:

clear-compiled Remove the compiled class file

db Start a new database CLI (Command Line Interface) session

down Put the application into maintenance / demo mode

env Display the current framework environment

help Displays help for a command

inspire Display an inspiring quote

**list** Lists commands

**migrate** Run the database migrations

optimize Cache the framework bootstrap files

serve Serve the application on the PHP development server

test Run the application tests

tinker Interact with your application

**up** Bring the application out of maintenance mode

auth

auth:clear-resets Flush expired password reset tokens

cache

**cache:clear** Flush the application cache

cache:forget Remove an item from the cache

cache:table Create a migration for the cache database table

config

config:cache Create a cache file for faster configuration loading

**config:clear** Remove the configuration cache file

db

db:seed Seed the database with records

db:wipe Drop all tables, views, and types

event

event:cache Discover and cache the application's events and listeners

event:clear Clear all cached events and listeners

event:generate Generate the missing events and listeners based on registration

event:list List the application's events and listeners

key

**key:generate** Set the application key

make

make:cast Create a new custom Eloquent cast class

make:channel Create a new channel class

make:command Create a new Artisan command

make:component Create a new view component class

**make:controller** Create a new controller class

make:event Create a new event class

make:exception Create a new custom exception class

make:factory Create a new model factory

make:job Create a new job class

make:listener Create a new event listener class

**make:mail Create a new email class**

**make:middleware Create a new middleware class**

**make:migration Create a new migration file**

**make:model** **Create a new Eloquent model class**

**make:notification Create a new notification class**

make:observer Create a new observer class

make:policy Create a new policy class

make:provider Create a new service provider class

make:request Create a new form request class

make:resource Create a new resource

make:rule Create a new validation rule

**make:seeder Create a new seeder class**

make:test Create a new test class

migrate

migrate:fresh Drop all tables and re-run all migrations

migrate:install Create the migration repository

**migrate:refresh Reset and re-run all migrations**

**migrate:reset Rollback all database migrations**

migrate:rollback Rollback the last database migration

migrate:status Show the status of each migration

notifications

notifications:table Create a migration for the notifications table

optimize

optimize:clear Remove the cached bootstrap files

package

package:discover Rebuild the cached package manifest

queue

queue:batches-table Create a migration for the batches database table

queue:clear Delete all of the jobs from the specified queue

queue:failed List all of the failed queue jobs

queue:failed-table Create a migration for the failed queue jobs database table

queue:flush Flush all of the failed queue jobs

queue:forget Delete a failed queue job

queue:listen Listen to a given queue

queue:restart Restart queue worker daemons after their current job

queue:retry Retry a failed queue job

queue:retry-batch Retry the failed jobs for a batch

queue:table Create a migration for the queue jobs database table

queue:work Start processing jobs on the queue as a daemon

route

route:cache Create a route cache file for faster route registration

route:clear Remove the route cache file

**route:list List all registered routes**

sail

sail:install Install Laravel Sail's default Docker Compose file

sail:publish Publish the Laravel Sail Docker files

schedule

schedule:list List the scheduled commands

schedule:run Run the scheduled commands

schedule:test Run a scheduled command

schedule:work Start the schedule worker

schema

schema:dump Dump the given database schema

session

session:table Create a migration for the session database table

storage

storage:link Create the symbolic links configured for the application

stub

stub:publish Publish all stubs that are available for customization

vendor

vendor:publish Publish any publishable assets from vendor packages

view

view:cache Compile all of the application's Blade templates

view:clear Clear all compiled view files

# Bài 2 Định tuyến Route

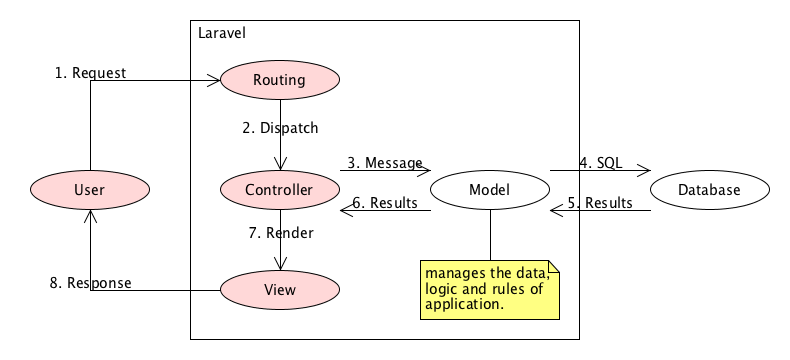
## Tài liệu: https://laravel.com/docs/8.x/routing

## Mục tiêu

* Sử dụng route cho ứng dụng, các loại route
* Truyền tham số trong route

## Lý thuyết

* Mô hình luồng xử lý theo MVC trong Laravel:



* Route: route nó có vai trò định hướng cho các reques trong Laravel. Khi index.php nhận được request từ người dùng, nó sẽ đưa request này cho route, từ route sẽ chỉ dẫn tiếp cho request này đi đến đâu (controller, action,…) hoặc cũng có thể trả lời ngay tại route. Routes được định nghĩa trong các file ở thư mục: routes.
  + Routes/web.php: Các route định nghĩa trên web. Hầu hết ứng dụng web sử dụng file này.
  + Routes/api.php
  + Routes/channels.php
  + Routes/console.php

|  |  |
| --- | --- |
| Route | |
| url | Công việc |
| / | Trang chu |
| Book/{id} | Nội dung sách có mã {id} |

Thông tin sách mã 1

Tên sách: php Laravel

Giá: 80000

http://mylaravel.com/book/1

* Cấu trúc route

Route::phương\_thức(đường\_dẫn, tham\_số);

Phương thức: get, post, put, delete, …

Đường dẫn: url thực thi

Tham\_số: các hàm, controller,.. xử lý request url.

**Chú ý**: Laravel 8 không tự thêm namespace các controller trong route. Để sử dụng cần:

* Load namespace và sử dụng theo Laravel 8 trong routes/web.php

use App\Http\Controllers\Home;

…

route::get('cat', [Home::class, 'f1']);

* Sử dụng path đầy đủ

Route::get('home2', **'App\Http\Controllers**\Home@f1');

## **Thực hành**

Bài 1: Tạo và chạy thử các route đơn giản sau

Route::get('/home', function (){ return "View home"; });

Route::get('/san-pham', function (){ return "View Sản phẩm"; });

Route::get('/san-pham/chi-tiet/{id}', function ($id){ return $id; });

Route::get('/san-pham/danh-sach/{page?}', function ($page=1){ return $page; });

Bài 2. Route và view. Trong thư mục resourcse/views, tạo các file php sau (nội dung tùy ý).

Views/home.php

Views/sanpham.blade.php

Views/chitietsanpham.blade.php

Views/timkiem.blade.php

Chỉnh sửa lại các route trong bài 1

Route::get('/home', function ()

{

return View('home');

});

Route::get('/san-pham', function ()

{

$data = DB::table('sach')->get();

/\*

DD($data);

\*/

return View('sanpham', ['data'=>$data]);

/\*return View('sanpham')->with('data'=>$data);

return View('sanpham', compact($data));

\*/

});

Route::get('/san-pham/chi-tiet/{id}', function ($id)

{

$data = DB::table('sach')->where(['masach'=>$id])->first();

return View('chitietsanpham', compact($data));

});

Bài 3. Route group: Tạo các route group cho loại sách phần user

Route::prefix('loai-sach')->group(function ()

{

Route::get('', function (){ return "Danh mục loại sách"; });

Route::get('/chi-tiet/{id}', function ($id){ return $id; });

Route::get('/{page?}', function ($page=1){ return $page; });

});

Bài 4. Tạo các route group cho quản trị để quản trị nội dung.

* Mở của sổ dòng lệnh trong project vừa tạo
* Chạy lệnh sau để tạo controller HomeController và loaiConttroller trong thư mục Admin:

**php artisan make:controller Admin/HomeController**

**php artisan make:controller Admin/LoaiController**

* Mở các controller Admin/HomeController, Admin/LoaiController và thêm vào các hàm tương ứng sau:

|  |  |  |
| --- | --- | --- |
|  |  | URL |
| HomeController | function index(){} | Bookstore.local/admin |
| LoaiController | function danhsach(){} | Bookstore.local/admin/loai/danhsach |
| function sua(){} | Bookstore.local/admin/loai/sua/th |
| function luusua(){} | Bookstore.local/admin/loai/luusua |
| function them(){} | Bookstore.local/admin/loai/themmoi |
|  |  | … |

Route::group(['prefix' => 'admin'], function ()

{

Route::get('home', 'Admin\HomeController@index');

//Loại sách

Route::group(['prefix' => 'loai'], function ()

{

Route::get('/', 'Admin\LoaiController@danhsach');

Route::get('sua/{id}', 'Admin\LoaiController@sua');

Route::post('luusua/{id}', 'Admin\LoaiController@luusua');

Route::get('xoa/{id}', 'Admin\LoaiController@xoa');

Route::get('them', 'Admin\LoaiController@theml');

Route::post('luuthem', 'Admin\LoaiController@luuthem');

Route::post('ketquatimkiem', 'Admin\LoaiController@ketquatimkiem');

});

//san pham sach

Route::group(['prefix' => 'sanpham'], function ()

{

Route::get('/', 'Admin\sanphamController@dstin');

Route::get('them', 'Admin\ sanphamController @themtin');

Route::post('luuthem', 'Admin\ sanphamController @laydulieuthem');

Route::get('chitiet/{id}', 'Admin\ sanphamController @chitiettin');

Route::get('ketquatimkiem', 'Admin\ sanphamController @ketquatimkiem')->name('search');

Route::get('xoa/{id}', 'Admin\ sanphamController @xoa');

Route::get('sua/{id}', 'Admin\ sanphamController @sua');

Route::post('luusua/{id}', 'Admin\ sanphamController @luusua');

});

# Bài 3. View trong Laravel

* + - 1. Mục tiêu
* Hiểu về view
* Hiểu template Blade Engine
* Gọi và truyền tham số cho view
* Dữ liệu toàn cục cho view
* Hiển thị kết quả trên view
* Sử dụng layout trong view
* Triển khai đưa template html thành view Laravel
  + - 1. Lý thuyết (<https://laravel.com/docs/7.x/views>)
  1. **view**: thành phần chứa HTML và tách rời khỏi các thành phần khác trong ứng dụng MVC. View được lưu trữ trong resources/views. Một view đơn giản

<!-- View stored in resources/views/greeting.blade.php -->

<html>

<body>

<h1>Hello, {{ $name }}</h1>

</body>

</html>

* 1. **Gọi view**: Khi được lưu trong resources/views/greeting.blade.php, có thể gọi tới view này và truyền data cho nó thông qua routes/web.php hoặc trong các controller.

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

return view('greeting', ['name' => 'James']);

});

* Gọi view: View có thể được gọi từ route hoặc controller theo tên file. Nếu các file đặt trong các thư mục con, sẽ được ngăn cách bằng dấu chấm.

Ví dụ: gọi views/hello.php

Route::get('/demo1', function () {

return view(**'hello'**, [**'name'** => 'James']);

});

Ví dụ: gọi views/user/hello.php

Route::get('/demo1', function () {

return view(**'user.hello'**, [**'name'** => 'James']);

});

* 1. **Truyền tham số vào view**. Khi gọi view, có thể truyền giá trị vào view thông qua mảng kết hợp, hàm with hoặc sử dụng hàm compact như sau:

return view(**'hello'**, [**'name'** => 'James']);

hoặc:

return view(**'hello')->with(**[**'name'** => 'James']);

Khi đó, các phần tử của mảng trở thành biến trong view. Cũng có thể truyền bằng cách sử dụng compact. Ví dụ sau, trong view name thành tên biến và có giá trị bằng $name

$name = 'James';

return view('welcome', compact('name'));

|  |  |
| --- | --- |
| Routes/web.php | Resource/views/hello.php |
| Route::get('/demo1', function () {  return view(**'hello'**, [**'name'** => 'James']);  }); | <?php echo $**name**;?> |
| Route::get('/demo1', function () {  return view(**'hello')**->with([**'name'** => 'James']);  }); | <?php echo $**name**;?> |
| route::get('url3/{id}', function($id){  return View('tam3',compact('id'));  }); | Product id: <?php echo $**id** ?> |

* 1. **Chia sẻ dữ liệu trong view**: Để chia sẻ data tới tất cả các view, có thể đặt dữ liệu trong hàm root của class AppServiceProvider trong file: **app\Providers\AppServiceProvider.php.** Vis duj sau, site và loạitin là 2 biến toàn cục, được sử dụng trong tất cả các view.

<?php

namespace App\Providers;

use Illuminate\Support\ServiceProvider;

use DB;

class AppServiceProvider extends ServiceProvider

{

public function boot()

{

View()->share(**'site'**, 'News');

View()->share(**'loaitin'**, DB::table('loaitin')->where(['Trangthai'=>1])->get());

/\*

View()->share(['site'=>'News', 'loaitin'=>array() ] );

\*/

}

//….

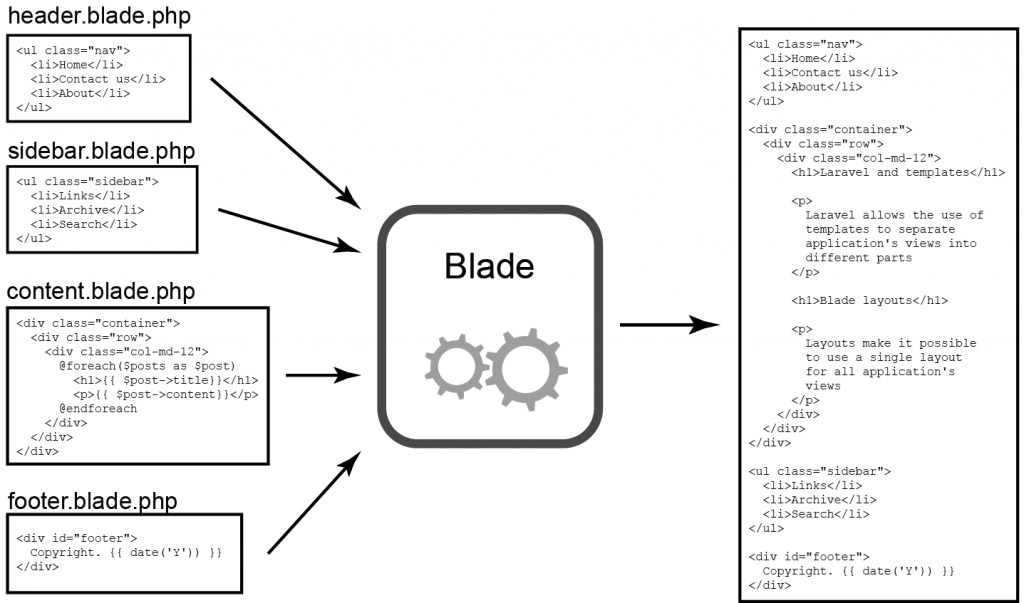
}

* 1. **Blade engine**: Blade là một templating engine được cung cấp bởi Laravel. Tất cả các Blade View sẽ được biên dịch thành mã PHP thuần và được lưu vào bộ đệm cho đến khi chúng được sửa đổi.

Các file Blade View có phần mở rộng là .blade.php và lưu trong thư mục resources/views. Nếu trong views có các view thường (.php) và blade (.blade.php) có cùng tên, view blade được chọn trước.

Sử dụng blade template để kế thừa bố cục và kế thừa các phần trong trang và thay đổi cách xuất dữ liệu, tránh đóng mở mã php nhiều lần.

Một file .blade.php không khác gì 1 file giao diện .html thông thường ngoại trừ nó có thêm các directive - (chỉ dẫn). Một directive luôn bắt đầu bằng kí tự "@" ví dụ: @section, @yield, @show ... Khi các file .blade được compiled, engine của Laravel sẽ dựa vào các directive này rồi biên dịch ra file giao diện .html.



* 1. **Layout trong Blade engine**: Cho phép xây dựng layout để sử dụng chung cho các view khác nhau. Ta sử dụng các directive @yield, @section để định nghĩa và đưa nội dung vào layout. Trang layout này còn gọi là master Page.

Ví dụ: Định nghĩa một layout có tên master.blade.php. layout này chứa các từ chỉ định: @yield và section. Các phần này định nghĩa một vùng dữ liệu sẽ được các view kế thừa và đổ dữ liệu vào.

<!-- Stored in resources/views/layouts/**master**.blade.php -->

<html>

<head>

<title>App Name - @**yield**(**'title'**)</title>

</head>

<body>

@**section**(**'sidebar'**)

This is the master sidebar.

@**show**

<div class="container">

@**yield**(**'content'**)

</div>

</body>

</html>

Sử dụng layout blade trong view. **Demo.blade.php**

@extends('layouts.**master'**)

@section('title', 'Page Title')

@section('sidebar')

@parent

<p>This is appended to the master sidebar.</p>

@stop

@section('content')

<p>This is my body content.</p>

@stop

* 1. **Các directive sử dụng trong blade engine cho layout.**
* @extends đặt ở đầu trang để xác định layout là trang nào. (**master**.blade.php).
* Các phần định nghĩa section cho layout được đặt giữa @section(NAME) và @stop. Khi biên dịch sẽ đặt phần section này vào vị trí của thẻ @section trong file master.blade.php dựa vào tên section.
* @section('key', 'value'): sẽ đặt giá trị value vào vị trí của thẻ @yield('key') trong file layout.
* Khác biệt giữa section và yield trong file master
* Yield: nội dung của view con sẽ ghi đè lên vị trí trong layout. Sử dụng khi muốn viết đè toàn bộ nội dung từ view con
* Section: có thể ghi đè hoặc giữ lại nội dung của phần layout – nếu trong view con có sử dụng @parent. Sử dụng khi muốn giữ lại nội dung của layout.
* Cũng có thể sử dụng lệnh @include(path) để load 1 view con vào master
  1. **Sử dụng blade template**
* Hiển thị nội dung của biến {{ $var }}, {!! $var !!} hoặc <?php echo $var ?>. (Khác biệt {{ $var }} và {!! $var !!} là gì? )
* Comment: {-- nội dung commment --}.
* Gán giá trị và thay đổi nội dung biến:

@php

$x=3;

@endphp

* Các lệnh – directive điều khiển

|  |  |  |
| --- | --- | --- |
| Lệnh | Cú pháp | Ví dụ |
| If | @if (bieu\_thuc\_dieu\_kien)    @endif | @if ($id<0)  ID khong hop le  @endif |
| If – else | @if (bieu\_thuc\_dieu\_kien)    @else  @endif | @if ($id<0)  ID khong hop le  @else  ID hop le  @endif |
| @if (bieu\_thuc\_dieu\_kien1)    @elseif (bieu\_thuc\_dieu\_kien2)    @else    @endif | @if ($id<0)  ID khong hop le  @elseif ($id<10)  ID nho hon 10  @else  ID lop hon hoac bang 10  @endif |
| switch | @switch($i)  @case(1)  First case...  @break  @case(2)  Second case...  @break  @default  Default case...  @endswitch | @switch($i)  @case(1)  First case...  @break  @case(2)  Second case...  @break  @default  Default case...  @endswitch |
| For | @for(BThuc1; BThuc2; BThuc3)  @endfor | @for($i=1; $i<10; $i++)  <div>Phan tu {{$i}}</div>  @endfor |
| foreach | @foreach($array as $k=>$v)  @endforeach | @foreach($loaitin as $k=>$v)  <div>{{$v->Ten\_loaitin}}</div>  @endforeach |
| Forelse | @forelse ($array as $k=>$v)    @empty    @endforelse | @forelse ($users as $user)  <li>{{ $user->name }}</li>  @empty  <p>No users</p>  @endforelse |
| While | @while(bieu\_thuc\_dk)  @endwhile | @while($id>0)  <div>{{$id}}</div>  @php  $id--;  @endphp  @endwhile |
| Continue and break |  | @foreach ($users as $user)  @if ($user->type == 1)  @continue  @endif  <li>{{ $user->name }}</li>  @if ($user->number == 5)  @break  @endif  @endforeach |

* Một số directive hay sử dụng trong blade engine

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | @includeFirst(['view1, view2, ..., viewN'], data) | @if(view()->exists('view1'))  @include('view1')  @else  @include('view2')  @endif |
|  | @includeIf('view', ['key' => 'data']) |  |
|  | @includeWhen() |  |
|  | @include($flag, 'view', ['some' => 'data']) |  |
|  | @auth  // The user is authenticated.  @endauth | @if(auth()->user())  // The user is authenticated.  @endif |
|  | @guest  // The user is not authenticated.  @endguest | @if(auth()->guest())  // The user is not authenticated.  @endif |
|  | {{@csrf}} | {{ csrf\_field() }} |
|  |  |  |

# **Bài 4 Thao tác với Cơ sở dữ liệu**

1. Mục tiêu

* Các cách truy xuất database trong laravel
* So sánh các cách truy xuất CSDL
* Sử dụng

1. Lý thuyết
2. Cách thức truy xuất CSDL: Laravel hiện nay đang sử dụng 2 kiểu truy vấn phổ biến với database là Eloquent ORM và Query Builder.

* Query Builder: Query Builder cung cấp 1 giao diện thuận tiện và dễ dàng tạo và chạy những truy vấn từ database. Nó có thể được sử dụng để thực thi hầu hết những thao tác về database trong ứng dụng và làm việc với tất cả những database được hỗ trợ.
* Eloquent ORM (Object Relational Mapping) cung cấp ActiveRecord để làm việc với database. Mỗi bảng của database sẽ được ánh xạ qua thành một class ‘Model’, và model này được sử dụng để tương tác với bảng và object instance sẽ tương ứng với các record trong các bảng dữ liệu đó.

1. So sánh: Các ứng dụng, có thể sử dụng Query Builder hoăc Eloquent ORM tùy vào người sử dụng và các tính năng của project. Có thể so sánh các tính năng của 2 thành phần trong bảng.

|  |  |  |
| --- | --- | --- |
|  | Query builder | Eloquent ORM |
| Dễ sử dụng | Sử dụng tốt cho các truy vấn đơn giản và phức tạp.  Với các truy vấn phức tạp, có thể sử dụng DB::raw (các sql thuần) hoặc QueryBuilder. | Dễ sử dụng hơn trong việc truy xuất, thay đổi cơ sở dữ liệu, cú pháp ngắn gọn, đơn giản |
| Hiệu suất | QueryBuilder có hiệu suất truy vấn dữ liệu nhanh hơn | Chậm hơn vì cần thêm một lớp trong ứng dụng và yêu cầu nhiều truy vấn SQL. |
| Linh hoạt | Sử dụng DB::raw nếu có thay đổi hệ quản trị CSDL -> có thể phải sửa lại code | Không ảnh hưởng khi thay đổi hệ quản trị CSDL |

1. Sử dụng Query Builder.

Query Builder là class DB chứa các phương thức, thuộc tính, cho phép dễ dàng truy xuất database. Các Query Builder sử dụng PDO và được chuyển đổi sang các câu truy vấn thông thường để truy xuất CSDL. Để có thể xem câu truy vấn, đặt Query Builder sau DB::**enableQueryLog**(). Để lấy câu truy vấn vừa thực thi: $query = DB::**getQueryLog**(), kết quả trả về mảng chứa truy vấn và các tham số.

|  |  |
| --- | --- |
| Lệnh | Câu truy vấn tương đương |
| $users = DB::table('users')->get(); | Select \* from users |
| $users = DB::table('users')->get('name');  $data = DB::table('sach')->get(['masach', 'tensach']);  $data = DB::table('sach')->select('masach', 'tensach')->get(); | Select name from users  Select masach, tensach from sach |
| $data = DB::table('sach')  ->select('masach', 'tensach')  ->where('tensach', 'like', "%php%")  ->get(); | Select masach, tensach from sach where tensach like ‘%php%’ |
| $data = DB::table('sach')  ->select('masach', 'tensach', 'gia')  ->where('tensach', 'like', "%a%")  ->orderby('gia', 'desc')  ->get(); | Select masach, tensach from sach where tensach like ‘%php%’  Order by gia desc |
| $data = DB::table('sach')  ->select('masach', 'tensach', 'gia')  ->where('tensach', 'like', "%a%")  ->orderby('gia', 'desc')  ->offset(3)  ->limit(2)  ->get(); |  |
| DB::table('sach')  ->join('loai', 'sach.maloai', '=', 'loai.maloai')  ->join('nhaxb', 'sach.manxb', '=', 'nhaxb.manxb')  ->select('sach.\*', 'tenloai', 'tennxb')  ->get(); | Select sach.\*, tenloai, tennxb  From sach join loai on sach.maloai=loai.maloai  Join nhaxb on sach.manxb=nhaxb.manxb |
| $users = DB::table('users')->where('id', 1)->first(); | Select \* from users where id=1 limit 0,1 |
| $name = DB::table('users')->where('id', 1)->value('name'); | Select name from users where id=1  Return $data[0][‘name’] |
| addUser = DB::table('users')->insert(  ['email' => 'test@gmail.com']  );  DB::table('users')->insert([  ['email' => 'taylor@example.com', 'votes' => 0],  ['email' => 'dayle@example.com', 'votes' => 0],  ]); | Insert into users(email) values('test@gmail.com') |
| editUser = DB::table('users')->where('id', 1)->update(['name' => 'nameTest']); | Update users set name='nameTest' where id=1 |
| deleteUser = DB::table('users')->where('id', '=', '1')->delete(); | Delete users where id=1 |
| DB::table('users')  ->updateOrInsert(  ['email' => 'john@example.com', 'name' => 'John'],  ['votes' => '2']  ); |  |
| $users = DB::table('users')->count();  $price = DB::table('orders')->max('price'); | Select Count(\*) from user |
| $data = DB::table('sach')  ->select(DB::raw('count(\*) as dem, maloai'))  ->where('maloai', '<>', '1')  ->groupBy('maloai')  ->get(); |  |

1. Sử dụng Eloquent ORM

* Laravel ánh xạ mỗi table thành một class model. Các Model lưu trong App. Tên model mô tả một dòng dữ liệu (activeRecord).
* Một model (table) được kế thừa từ class model trong laravel.
* Để tạo một model, có thể tạo bằng tay hoặc sử dụng tool artisan.

Php artisan make:model modelName --option

Option: resource

Ví dụ:

*php artisan make:model nhaxb*

*php artisan make:model loai –resource*

* Lưu ý: Mỗi model biểu diễn một dòng dữ liệu trong table và có các thuộc tính:
* tableName: default className + s.
* primaryKey: default là id
* primaryType: default: int
* autoincreament: default true
* public $timestamps=true;
* protected $dateFormat = 'd-m-Y';
* const CREATED\_AT = 'creation\_date';
* const UPDATED\_AT = 'last\_update';

Nếu các table trong db có các thông số khác trên, khai báo thêm các thuộc tính trong model.

Ví dụ:

class loai extends Model

{

//

protected $table='loai';//default loais

protected $primaryKey ='maloai';//default id

protected $keyType='String';//default int

protected $increatming =false;//default true

public $timestamps = false;

protected $dateFormat = 'd-m-Y';

const CREATED\_AT = 'creation\_date';

const UPDATED\_AT = 'last\_update';

}

* Sử dụng model
  + Sử dụng các phương thức: tên\_model::tên\_phương\_thức() để select, delete, update
  + instance của model để thêm mới.

Ví dụ trong Routes/web.php

Route::get(‘orm1’, function()

{

$data = App\loai::all();//select \* from loai

foreach($data as $r)

echo $r->tenloai;

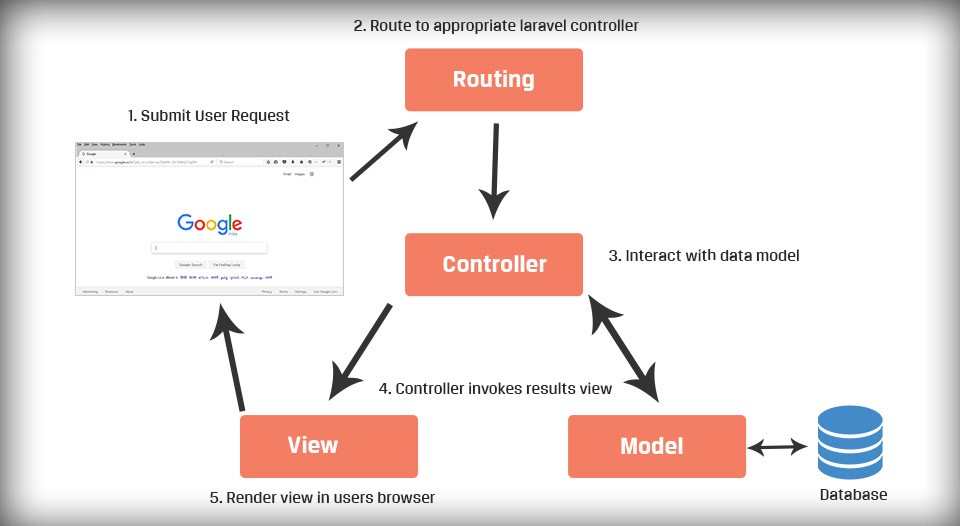
});

|  |  |
| --- | --- |
| Lệnh | Mô tả - câu truy vấn tương đương |
| $users = User::all(); |  |
| $users = User::find(1); |  |
| $data = App\sach::where(['maloai'=>$maloai])  ->orderBy('gia', 'asc')  ->orderBy('tensach', 'desc')  ->get(); |  |
| data = App\sach::addSelect(['masach', 'tensach'])  ->where(['maloai'=>'th'])  ->get(); |  |
| $data = App\sach::addSelect(['masach', 'tensach'])  ->where('tensach', 'like', '%p%')  ->first(); |  |
| $name = User::where('id', 1)->value('name'); |  |
| try {  $sach = sach::findOrFail(1);  //hoặc  $sach = sach::where(banchay, true)->firstOrFail();  } catch (ModelNotFoundException $e) {  echo $e->getMessage();  } |  |
|  |  |
| $user = new User;  $user->email = 'test@gmail.com';  $user->save(); |  |
| $user = User::find(1);  $user->name ='nameTest';  $user->save(); |  |
| // nếu là 1 câu truy vấn  $user = User::find(1);  $user->delete();  //hoặc nếu không phải 1 câu truy vấn  $user->destroy(); |  |
| $count = User::where('name', '%a%')->count();  $max = Order::where('name', '%a%')->max('price'); |  |

1. Bài tập:

# Bài 5 Controller - Model

* + - 1. Mục tiêu
* Hiểu MVC
* Tạo và sử dụng Controller
* Tạo và sử dụng Model
  + - 1. Lý thuyết
         1. Mô hình MVC



* + - * 1. Công cụ Artisan trong Laravel:
* Artisan là một công cụ trong Laravel. Nó cung cấp một số lệnh hữu ích mà có thể hỗ trợ bạn trong khi xây dựng ứng dụng dễ dàng hơn. Sử dụng artisan, mở cửa sổ dòng lệnh trong thư mục cài đặt và nhâp: php artisan **lệnh**
* Để xem danh sách tất cả các lệnh Artisan có sẵn, có thể sử dụng lệnh list.

Php artisan list

* Một số lệnh hay sử dụng của artisan

// composer create-project --prefer-dist laravel/laravel ten-thu-muc '5.5.\*'

php artisan serve

php artisan route:list

php artisan make:controller UserController

php artisan make:controller UserController --resource

php artisan make:model ModelName

php artisan make:model ModelName –resource

…..

* + - * 1. Controller trong Laravel

Mục đích: Thay vì định nghĩa tất cả logic xử lý request ở file routes.php, có thể muốn quản lý việc này bằng cách sử dụng các lớp Controller. Các Controller có thể nhóm các request HTTP có logic liên quan vào cùng một lớp. Các Controller được chứa tại thư mục app/Http/Controllers.

Tạo controller thủ công: trong thư mục App\https\controllers, tạo file *nameController*.php, với *nameController* là class controller cần tạo. class này extends từ controller class của Laravel. Ví dụ: class sendMail extends Controller. Cũng có thể nhóm các controller trong các thư mục con.

Tạo bằng artisan. Cho phép tạo controller nhanh chóng bằng dòng lệnh.

*Php artisan make:controller controllerName*

*Php artisan make:controller controllerName* *--resource*

*Php artisan make:controller path/controllername*

Tham số --resource sẽ tạo thêm các function tương ứng với các thao tác CRUD.

Ví dụ

php artisan make:controller abcController –resource

tạo file controllers/abcController.php

<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class abcController extends Controller

{

//list resource

public function index(){}

/\*\*

\* Show the form for creating a new resource.

\*/

public function create(){}

/\*\*

\* Store a newly created resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function store(Request $request){}

/\*\*

\* Display the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function show($id){}

/\*\*

\* Show the form for editing the specified resource.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function edit($id){}

/\*\*

\* Update the specified resource in storage.

\*

\* @param \Illuminate\Http\Request $request

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function update(Request $request, $id)

{}

/\*\*

\* Remove the specified resource from storage.

\*

\* @param int $id

\* @return \Illuminate\Http\Response

\*/

public function destroy($id)

{

//

}

}

* + - * 1. Sử dụng controller:

Gọi controller trong routes/web.php

Xác định một route đến một action của controller như sau:

Route::get('user/{id}', 'UserController@show');

Truyền tham số vào controller

* + - * 1. Model
      1. Bài tập

# Bài 6. Xây dựng giỏ hàng

## Mục tiêu

## Lý thuyết

**Cài đặt**

Có nhiều package hỗ trợ xây dựng giỏ hàng trong Laravel. Các package hỗ trợ Laravel 7: olimortimer, hardevine, …Ta chọn cài đặt một package thích hợp.

* Cài đặt hardevine: https://packagist.org/packages/hardevine/shoppingcart

composer require hardevine/shoppingcart

Quá trình này sẽ:

* + Download thu viện về vendor
  + Cập nhật file composer.js
  + Cập nhật các khai báo thư viện class Cart
* Cài đặt olimortimer: : <https://github.com/olimortimer/LaravelShoppingcart#usage>

composer require olimortimer/laravelshoppingcart

* Thay đổi trong file composer.json
* Load thư viện về vender (Vendor/olimortimer)

**Sử dụng:**

Các cart hỗ trợ các phương thức cho phép dễ dàng thao tác trên giỏ hàng: thêm, xóa 1 sản phẩm, cập nhật số lượng, xóa toàn bộ giỏ hàng. Giỏ hàng sử dụng session để lưu giũa thông tin.

* Thêm 1 sản phẩm vào giỏ hàng: Cart::add,sẽ thêm một sản phẩm vào giỏ hàng, được lưu dạng session. Phương thức add sẽ thêm một sản phẩm gồm các thông tin theo thứ tự: id, name, qty, price , options=array().

Cart::add('293ad', 'Product 1', 1, 9.99);

Cart::add('293ad', 'Product 1', 1, 9.99, ['size' => 'large']);

Cũng có thể truyền vào bằng một mảng kết hợp, với chỉ số mảng như đã nêu: id, name, qty, price, options

Cart::add(['id' => '293ad', 'name' => 'Product 1', 'qty' => 1, 'price' => 9.99, 'options' => ['size' => 'large']]);

Cart::add() sẽ return về sản phẩm vừa thêm

Có thể thêm nhiều sản phẩm vào giỏ hàng, phương thức trả về mảng các item được thêm

Cart::add([

['id' => '293ad', 'name' => 'Product 1', 'qty' => 1, 'price' => 10.00],

['id' => '4832k', 'name' => 'Product 2', 'qty' => 1, 'price' => 10.00, 'options' => ['size' => 'large']]

]);

* Cập nhật giỏ hàng: Cart::update().

Cho phép cập nhật số lượng trong giỏ hàng. Mỗi sản phẩm khi thêm vào, được quản lý bới một chỉ số là rowId.

Ví dụ $rowId = 'da39a3ee5e6b4b0d3255bfef95601890afd80709'; $qty=2;

Hàm update($rowId, $qty) sẽ cập nhật sản phẩm $rowId có số lượng là 2.

* Cart::remove(): xóa phần tử trong giỏ hàng.

Để xóa một phần tử trong giỏ hàng, sử dụng $rowId.

$rowId = 'da39a3ee5e6b4b0d3255bfef95601890afd80709';

Cart::remove($rowId);

* Cart::get(): lấy item (sản phẩm) trong giỏ hàng khi biết rowId.

$rowId = 'da39a3ee5e6b4b0d3255bfef95601890afd80709';

Cart::get($rowId);

* Cart::content(); lấy tất cả các items trong giỏ hàng. Kết quả trả về một list và có thể dung vòng lặp duyệt qua.

Cart::destroy(); Xóa hết tất cả items trong giỏ hàng.

* Một số phương thức khác

Cart::total(); Tính nhanh tổng giá trị các items (giá \* số lượng)

Cart::count(); tổng số items trong giỏ hàng

Cart::count();

## Bài tập

Routes/web.php

Route::group([‘prefix’=>’cart’], function(){

//hien thi gio hang

*Route::get(‘/’, function(){*

*Dd(Cart::content());*

*});*

*Route::get(‘add/{id}’, function($id){*

*//$sp = [‘id’=1, ‘name’=>’sp1’, ‘qty’=>1, ‘price’=>10];*

*$data= DB::table(‘sach’)->select(‘masach’, ‘tensach’, ‘gia’, ‘hinh’)->where(‘masach’, $id)->first();*

*Dd($data);*

*Cart::add($sp);*

*});*

});

Domain/cart

# Bài 7. Gửi nhận dữ liệu client-server

## Mục tiêu:

* Đối tượng request
* Truy xuất
* Csrf

## Lý thuyết

* Đối tượng request
* Nhận data
* Sử dụng CSRF
  + Trong form
  + Trong ajax

## Bài tập

* Tạo route cho form đăng ký/đăng nhập
* Quản lý sách

Gửi email trong Laravel 7

Sử dụng gmail: ví dụ: <https://www.youtube.com/watch?v=jp92IB3CRQs&feature=youtu.be>

<https://www.youtube.com/watch?v=g0WTOjxZID4>

https://www.tutorialspoint.com/laravel/laravel\_sending\_email.htm

Cấu hình trong file .env

Sửa thư viện (nếu lỗi)

vendor\swiftmailer\swiftmailer\lib\classes\Swift\TransportStreamBuffer.php on line 263

Thêm 2 dòng sau:

$options['ssl']['verify\_peer'] = FALSE;

$options['ssl']['verify\_peer\_name'] = FALSE;

Code gui email trong web.php

 $data=['ten'=>'Trần Văn Hùng'];

    $data['khuyenmai']=DB::table('sach')->where('khuyenmai', '>', 0)->get();

    Mail::send('mail', $data, function ($m){

        $m->from('hung.seu@gmail.com','Tran Van hung SEU');

        $m->to('hung.tranvan@stu.edu.vn', 'Hung STU');

        // $m->attach('path/to/attachment.txt');

        // $m->embed('path/to/attachment.jpg');

        $m->subject('Quà tặng, sản phẩm khuyến mãi từ Aptech');

Gửi email

1. Cấu hình
2. Chuẩn bị email-> setting trong email
3. Tạo roure
   1. Viết hàm gửi email
   2. Tạo controller gửi email
4. Tạo mailable: trong Laravel, một email được gửi bởi ứng dụng được xem là một "mailable" class. Những class này lưu trong app/Mai. Để tạo một mailable class, sử dụng artisan.

php artisan make:mail OrderShipped

php artisan make:mail Khuyenmai

Trong mailable, ta có thể thiết lập các thông tin cho email: to, subject, view, attach file,..

Các cấu hình cần lưu ý:

From: cấu hình trong function built hoặc .env

'from' => ['address' => 'example@example.com', 'name' => 'App Name'],

'reply\_to' => ['address' => 'example@example.com', 'name' => 'App Name'],

View:

return $this->view('emails.orders.shipped');

# Session – cookie

Nội dung:

Session

* Giới thiệu session
* Cấu hình
* Điều kiện sử dụng Session Driver
* Cách sử dụng session

Cookie

* Giới thiệu
* Sử dụng

Session:

* Khi cần lưu trữ các thông tin như thông tin người dùng đăng nhập
* Thông tin khi khách hàng đặt hàng (giỏ hàng)
* Thông báo tức thì đến người sử dụng

Chú ý: thông tin lưu trữ trong session sẽ bị mất khi thoát khỏi website. Để có thể lưu trữ lâu hơn, có thể thay thế bằng cách lưu trữ trong database.

Session trong laravel:

* Laravel cung cấp nhiều trình điều khiển khác nhau cho session như: file, cookie, memcached, Redis, database.
* Mặc định: session lưu trữ trong file: vì nhẹ
* Cấu hình lưu trữ session trong file: config/session.php

Cấu hình session: Laravel tích hợp sẵn một số session driver:

* File: session sẽ lưu trữ tại storage/framework/sessions
* Cookie: session sẽ lưu có bảo mật, mã hóa bằng cookie
* Database: lưu trữ trong csdl. Để sử dụng, cần tạo một table session:

Schema::create('sessions', function ($table) {

$table->string('id')->primary();

$table->foreignId('user\_id')->nullable()->index();

$table->string('ip\_address', 45)->nullable();

$table->text('user\_agent')->nullable();

$table->text('payload');

$table->integer('last\_activity')->index();

});

* Memcached/redis: 2 cách thức lưu trữ sử dụng cho các website cần truy xuất với tốc độ cao.