Prettifying PHP: More expressive: To find out when a file was last modified, use Storage::lastModified($path) instead of filemtime(realpath($path)). To delete a file, use Storage::delete($path) instead of unlink($path), which is the plain old PHP equivalent.

Models: Models represent resources in your application. models can also refer to data from other data sources, such as an XML file, or the response from a web service or API.

Controller or Route: Controllers or routes: Controllers, at their simplest, take a request, do something, and then send an appropriate response. Controllers are where the actual processing of data goes, whether that is retrieving data from a database, or handling a form submission, and saving data back to a database. Although you are not forced to adhere to any rules when it comes to creating controller classes in Laravel, it does offer you two sane approaches: RESTful controllers and resource controllers. A RESTful controller allows you to define your own actions and what HTTP methods they should respond to. Resource controllers are based around an entity and allow you to perform common operations on that entity, based on the HTTP method used. Another option is to bypass using controller classes altogether and instead write your logic in your routes, by way of anonymous functions

View or Template: Views are responsible for displaying the response returned from a controller in a suitable format, usually as an HTML web page. They can be conveniently built by using the Blade template language or by simply using standard PHP. The file extension of the view, either .blade.php or simply .php, determines whether or not Laravel treats your view as a Blade template or not.

The Public/directory: is meant to act as the document root; in other words, the directory in which your web server starts looking after every incoming request. Once URL rewriting is properly set up, every request that does not match an existing file or directory hits the /public/index.php file

Form requests: Laravel 5 introduces form request classes. These classes can be injected into your controller actions. They take the current request, and on it, you can perform data validation and sanitizing and even user authorization (that is, check if the currently-logged in user can perform the requested action). This streamlines validation, meaning you have to do very little, if any, data validation in your controller actions

Homestead: tự tìm hiểu thêm

Handling HTTP exceptions:

Eloquent models: The $fillable array defines the list of fields that Laravel can fill by mass assignment, which is a convenient way to assign attributes to a model, The properties of an Eloquent model can be retrieved with the -> operator: $cat->name. The same goes for the properties of the related models, which are accessible with: $cat->breed->name. Behind the scenes, Eloquent will perform the necessary SQL joins.

Route-model binding: Route-model binding is the method of automatically transforming a route parameter to a model instance, so we don't have to manually retrieve the model. Since this is such a common pattern, Laravel provides you with a way to automatically bind a model to a route and, therefore, make your code shorter and more expressive. To bind the $cat variable to the Cat model, open app/Providers/ RouteServiceProvider.php. Modify the boot() method so that it looks like this: public function boot(Router $router) { parent::boot($router); $router->model('cat', 'Furbook\Cat'); } This allows you to shorten your route and pass a Cat object to it instead: Route::get('cats/{cat}', function(Furbook\Cat $cat) { return view('cats.show')->with('cat', $cat); });

Resource controllers: Laravel greatly simplifies the creation of REST APIs with resource controllers. Since they adhere to conventions, there is only a limited defined set of actions that can be performed from the controller. In fact, all the routes we created earlier can be rewritten as follows: Route::resource('cat', 'CatController');

Eloquent ORM: $ php artisan app:model Cat-Our model class extends the base Eloquent Model class, which contains all of the goodness we're going to use over the course of this chapter. The first thing you should do after creating a model is define the database table it maps to. In our case, the database table will be called cats: class Cat extends Model { protected $table = 'cats'; }--If you have a table with lots of columns, then it will become tiresome to assign each property manually like this. To this end, Eloquent allows you to fill models by passing an associative array with values, and the keys representing the column names. You can fill a model while either creating or updating it: $data = [ 'name' => 'Garfield', 'birth\_date' => '1978-06-19', 'breed\_id' => 1, ]; $cat->create($data); However, this will throw a MassAssignmentException error.

Mass assignment: class Cat extends Model { protected $table = 'cats'; protected $fillable = [ 'name', www.it-ebooks.info Eloquent ORM [ 52 ] 'birth\_date', 'breed\_id', ]; }

Deleting data: There are two ways of deleting records. If you have a model instance that you have fetched from the database, then you can call the delete method on it: $cat = Cat::find(1); $cat->delete(); Alternatively, you can call the destroy method, specifying the IDs of the records you want to delete, without having to fetch those records first: Cat::destroy(1); Cat::destroy(1, 2, 3, 4, 5);-- By default, Eloquent will hard-delete records from your database. This means, once it's deleted, it's gone forever. If you need to retain deleted data (that is, for auditing), then you can use soft deletes. When deleting a model, the record is kept in the database but instead a deleted\_at timestamp is set, and any records with this timestamp set will not be included when querying your database. www.it-ebooks.info Chapter 4 [ 53 ] Soft deletes can be easily added to your Eloquent model. All you need to do is include the trait: use Illuminate\Database\Eloquent\SoftDeletes; class Cat extends Model { use SoftDeletes; protected $dates = ['deleted\_at']; }---If you find you need to include deleted records when querying your database (for example, in an administration area), then you can use the withTrashed query scope. Query scopes are just methods you can use in chaining: $cats = Cat::withTrashed()->get(); This will mix deleted records with non-deleted records. If you find you need to retrieve only deleted records, then you can use the onlyTrashed query scope: $cats = Cat::onlyTrashed()->get(); If you find you need to "un-delete" a record, then the SoftDeletes trait provides you with a new restore method to undo this: $cat->restore(); Finally, if you find you really need to delete a record from your database, you can use the forceDelete method. As the name implies, once you delete a record with this method, it's truly gone. $cat->forceDelete();

Query scopes: <https://laravel.com/docs/5.5/eloquent> dung để tạo những dàng buộc trong những câu lệnh query.

Model events: <https://laravel.com/docs/5.5/eloquent#events>

event listeners: <https://allaravel.com/laravel-tutorials/quan-ly-su-kien-trong-ung-dung-voi-laravel-event/>

Testing: <https://viblo.asia/p/unit-test-trong-laravel-phan-1-DzVkpgryenW--https://viblo.asia/p/tim-hieu-ve-unit-test-phan-2-to-chuc-thu-muc-bang-testsuite-YmjeoomVeqa>

-LIFECYCLE:tat ca requests se di vao file public/index.php.index loadd the composer generate autoload definition from boostrap/app.php

-Tinker: php artisan tinker. Roi query du lieu de test binh thuong

Route: the routes/web.php file defines routes that are for your web interface. These routes are assigned the web middleware group, which provides features like session state and CSRF protection.

-tao rang buoc cho route: thi vao provider/RouteServiceProvider: viet regular expression tai function boot.

Route Model Binding

Query builder : muon su dung phai use Illuminate\Support\Facades\DB;

-pluck(‘column’): lay tat ca gia tri cua 1 cot trong cac ban ghi

-chunk(): lay ve mot khoi luong ban ghi nhat dinh

Eloquent: can tao model.va moi model se tuong ung voi 1 bang.

Eloquent Polymorphic Relationship

-la một bảng lien kết với 2 bảng. vd. Bảng Comment sẽ lien kết với 2 bảng là Post và Video.thì bảng comment này sẽ chứa bình luận của Post và video.cso 2 column đặc biệt là table\_type: (thuong la ten của bảng post hoạc video), 2 là (id của post cụ thể nào đó hoặc video nào đó): https://www.youtube.com/watch?v=Rhby8SBEYh8&index=12&list=PLe30vg\_FG4OQz1yZq0z19ZuWD\_C3MZbA4

**Accessors**: là những function dùng khi bạn muốn lấy dữ liệu từ database ra xử lý database trước khi đổ ra view. Định nghĩa ham nay trong Model.và thường sử dụng với các biến trong view. <https://www.youtube.com/watch?v=UgDXkLZ7IPg&list=PLe30vg_FG4OQz1yZq0z19ZuWD_C3MZbA4&index=16>

<https://laravel.com/docs/5.6/eloquent-mutators#accessors-and-mutators>

Mutator: ngược lại với accesstor là modifile dữ liệu trước khi lưu vào trong database. <https://www.youtube.com/watch?v=fCD3a5uFjsU&list=PLe30vg_FG4OQz1yZq0z19ZuWD_C3MZbA4&index=17>

**Gate và policy**: <https://viblo.asia/p/laravel-va-nhung-dieu-can-biet-XQZGxolmvwA>

Ví dụ ứng dụng với phân quyền người dùng.chỉ admin mới có quyền update,create bài đăng. Thì những rằng buộc này sẽ dc định nghĩa bằng gate.tương tự với các quyền khác. <https://www.youtube.com/watch?v=CB4Q-oR9w1g&index=33&list=PLe30vg_FG4OTELVqQgHaMaq2oELjpSWy_>

**Queues**: là sử dụng hàng đợi để thực hiện tuần tự và tự động các sự kiện được khai báo.

<https://allaravel.com/laravel-tutorials/laravel-queue-xu-ly-cong-viec-kieu-hang-doi/>

**Cronjob**: chức năng là bộ lệnh đặt lịch rồi tự động thực hiện của website: <https://viblo.asia/p/cronjob-don-gian-trong-laravel-54-gGJ59XrJlX2>

<https://www.youtube.com/watch?v=jlCWLlLYEDc&t=267s>

**Magic Function**: thực chất là những hàm tự động trong PHP như \_\_construct, \_\_disconstruct. <https://viblo.asia/p/php-magic-methods-qzaGzLzdkyO>

**Trait:** có thể hiểu như là 1 class,là nơi tập hợp các phương thức có thể sử dụng tại các class khác.giống như abstract class. Chúng ta cũng không thể khởi tạo đối tượng từ trait. <https://viblo.asia/p/traits-in-php-and-laravel-l0rvmxzQGyqA>

**Hàm Lambda**: là một hàm ẩn danh.không có tên. Và thường được được gán cho 1 biến nào đó. Vì không có tên nên không thể sử dụng như những hàm bình thường khác. <https://viblo.asia/p/lambda-va-closures-trong-php-DXOGRZZnGdZ>

**Closure:** tương tự như hàm Lambda. Nhưng có thể sử dụng biến ngoài phạm vi của hàm. <https://viblo.asia/p/lambda-va-closures-trong-php-DXOGRZZnGdZ>

**Service Container**: <https://viblo.asia/p/service-container-trong-laravel-53-Ljy5VbB95ra>

<https://viblo.asia/p/laravel-beauty-tim-hieu-ve-service-container-3KbvZ1wLGmWB>

**Facades:** <https://viblo.asia/p/laravel-beauty-tim-hieu-ve-facade-znVGLYLbvZOe>

**Repository pattern**: <https://dinhquochan.com/article/repository-pattern-trong-laravel/>

**Dependency injection:** https://viblo.asia/p/dependency-injection-hoat-dong-the-nao-trong-laravel-3Q75wD3JKWb