Laravel Cheat Sheet

#laravel #php

Lam Hoang Jun 9, 2020 · 7 min read

Full version Laravel Cheat Sheet

Setup Laravel Environment

Chose running the Bitnami WAMP stack instead of using Homestead

1. Download Bitnami 5.6 WAMP stack
2. uncomment C:\Bitnami\wampstack-5.6.30-1\apache2\conf\bitnami\bitnami-apps-prefix.conf and uncomment Include "C:/Bitnami/wampstack-5.6.30-1/frameworks/laravel/conf/httpd-prefix.conf"
3. Restart the servers
4. add C:\Bitnami\wampstack-5.6.30-1\frameworks\laravel to your windows path
5. Run the bitnami bash shell by executing 'use_wampstack.bat' from the bitnami application root folder.
6. test your laravel installation by typing php artisan serve [--port=8000] . Will serve by default on port 8000.

Add the following to your path so you can use the laravel.bat
C:\Users\motero\AppData\Roaming\Composer\vendor\bin

Get the laravel installer using composer

```
composer global require "laravel/installer"
```

Once installed you can use the
laravel new command to create a fresh laravel installation in the directory you specify.

Creating a new project

```
Create a project: composer create-project myprojectname
```

If you get any errors when running php artisan serve
or any artisan commands then run:

```
composer update --no-scripts
```

You may also need to regenerate the ciphers for the project:

```
php artisan key:generate
```

from your project folder.

Alternative is Laravel Homestead, which comes with Nginx, PHP 5.6, MySQL, Redis, Memcached, and Beanstalk.

https://laravel.com/docs/4.2/homestead

Routes are in /config/web.php

views are in resources/views/myview.blade.php

Working with Databases

Consider using MariaDB versus MySQL

```
mysql -uroot -p
create database databasename;
use databasename;

show tables;
```

Setup your database connection information in the .env file.

From command line (inside the project folder):

```
php artisan migrate
```

```
php artisan make: These are your file generators
```

Prepare the Migration

```
// creates the migration stub and the database table as well.
// Creates a migration stub under the /project/database/migrations folder

artisan make:model create_database_name --create=tasks
```

modify database columns

You do this by modifying the file at /database/migrations/create_database_name

Migrate the database

```
php artisan migrate
```

```
php artisan help make:model [Artisan Help Feature]
```

when you make a mistake, delete the item from the database stub file, and refresh your migration.

```
php artisan migrate:refresh // rolls back and reruns the migrations
```

Fetching database records

in web.php routes

```
$tasks = DB::table('tasks')->get(); //Laravel Query Builder

from the view get the content from the database:
{{ $task->body }}

dd($id) // die and dump [get more info on this

  $task = DB::table('tasks')->find($id);
Route::get('/tasks/{task}', function ($id) {
```

Hyperlinking to dynamic ids

```
<li><a href="/tasks/{{ $task->id}}">{{ $task->body }}</a></li>
```

- If you ever return a database result from a view, Laravel returns it as JSON.

Creating a new database table

Make a new Comment model, and make migration

```
php artisan make:model Comment -m
```

Edit the migrations file to add any additional fields.

If one to many relatioship the many takes the id of the one.

```
$table->integer('post_id');
```

Migrate the database

```
php artisan migrate
```

Add many relationship to Post model

```
class Post extends Model
{
    public function comments()
    {
      return $this->hasMany(Comment::class);
    }
}
```

Eloquent

A model is a representation of something in your system. It could be a noun.
It is Laravel's active record implementation. simplifies your interaction.

This creates /app/task.php

Errors:

The only supported ciphers are AES-128-CBC and AES-256-CBC with the correct key lengths.

```
php artisan key:generate
```

error related to not having a .env file. Renamed .env.example to .env then run

```
php artisan key:generate
```

You may need to add the composer.bat / composer.phar files to your PATH. The location is here: C:\ProgramData\ComposerSetup\bin

```
php artisan make:model Task -m // make model and a migration file
```

```
composer dump-autoload // in case you get an error message when creating the model and mig
```

Blade Templating Language

Blade is laravel's templating language.

```
php artisan make:model task
```

Create a model with singular noun. php artisan make:model MyModel

Tinker

```
php artisan tinker // boots up Laravel's Shell
```

It allows you to replace the php directive with @

Using Controllers

The middleman. Receive a request and will ask a task or post model to get the data and then passes it to the view.

Controller Deletgates.
Model fetches data
View HTML

```
Route::get('/tasks', 'TasksController@index' ); //@index is the index method
```

Controllers are stored in /app/Http/Controllers

use a make controller generator to generate your controllers

```
php artisan make:controller TasksController
```

Variables and wildcards in route controllers should match.

registering routes ###

Routes are registered in /routes/web.php

```
Route::get('/', 'PostsController@index');
Route::get('/posts/{post}', 'PostsController@show'); // Corresponds to /posts/show.blade.p
```

controller lives in /app/http/controllers In this case postscontroller.php

Layouts

in resources/views

create layout.blade.php

```
@yield('content')
```

Assets

found in /resources/assets

/sass folder

use a build tool called **webpack** using Laravel's **Mix** configuration tool. It grabs the file in the resources/assets folder and compiles it for use in the /public folder.

They are node dependencies. You need to have Node and npm installed on your machine.

from your application folder run:

```
npm install
```

Pulls in packages found in your **package.json** file.

You can run the files based on your package.json scripts section.

```
npm run dev
```

Order of Things

IMPORTANT

// controller => PostsController
// php artisan make:controller PostsController

// eloquent Model => Post
// php artisan make:model Post

// migration => create_posts_table
// php artisan make:migration create_posts_table --create=posts
// You an do them all indivdiually as above, or you can have Laravel
// create the model and the migration when you create the controller.

// php artisan make:model Post -mc

// after running the above command go into the migrations file in
// /database/migrations and add the columns you need

```
            $table->increments('id'); // default
            $table->string('title'); // added this one
            $table->timestamps(); // default
```

[Add information on the types of columns that you can create and examples for them]

Migrate the database:

```
php artisan migrate
```

go to your controller and add the required methods to link it back to the view, or get data from the model, etc..

```
class PostsController extends Controller
{
    //add methods
    public function index()
    {
      return view('posts.index')
    }

}
```

Create your view file.
/resources/views/posts/index.blade.php

View can then extend the layout file:

```
@extends('layout')

@section('content')
<p>This is my content section, this is only a test.</p>
@endsection
```

```
test your creation: php artisan serve
```

Folder Structure

created a /layouts folder

Have these partials in the folder
master.blade.php
nav.blade.php
sidebar.blade.php
footer.blade.php

/posts/
index.blade.php {this is where the posts live}

index extends the master layout with

```
@extends('layouts.master')
@section('content') // starts all of the content pieces.
```

master.blade.php

links to the partials via:
@include ('layouts.nav')
@yeld('content')
@include ('footer.nav')
@include ('layouts.footer')

CSS and JS files can go in the /public folder and are referenced like this: /css/styles.css.

Working with Forms

```
<form method="POST" action="/posts">

GET /posts - view all of the posts
GET /posts/create - create a post
GET /posts/{id}/edit - edit a specific post
GET /posts/{id} - displays a single post specified by the ID in the URL
PATCH /posts - send to database
DELETE /posts/{id}
```

```
// @store is the method in the PostController
  Route::post('/posts', 'PostsController@store');
```

```
//creates a resourceful controller, a boilerplate with all of the types of responses.
artisan make:controller TasksController -r
```

CSRF - Cross Site Request Forgery

Laravel automatically generates a CSRF token for each active user session managed by the application. This token is used to verify that the authenticated users is the one actually making teh requests to the application.

Anytime you define an HTML form, you should include a hidden CSRF token field in the form so that the CSRF protection middleware can validate the request.

```
{{ csrf_field() }}
```

Begin you class paths with \ so that you are at the root, and not inside the namespace (applies to controller pages)

or set use App\Post; at the top, and then create new Post;

Example of store method in PostController

```
public function store()
    {
      // dd(request()->all());
      // create a new post using the request database
      $post = new \App\Post;
```

```
        $post->title=request('title');
        $post->body=request('body');

        // save it to the database
        $post->save();

        // and redirect to the home page
        return redirect('/');
      }
```

```
An alternative to the above is to simply use:
```

```
    Post::create(request(['title', 'body']));
```

and it creates the object, and saves the request items to the database.

Form Validation

- At the minimum use HTML 5 form validation

```
<input type="text" required>
```

- Setup server side validation in controller (PostController)

```
$this->validate(request(), [
        'title' => 'required',
        'body' => 'required'

      ]);
```

and in the view spit out the errors which are added to the $errors variable. $errors is available globally to all views.

```
@if(count($errors))
<div class="form-group">
  <div class="alert alert-danger">
    <ul>
       @foreach($errors->all() as $error)
         <li>{{ $error }}
         </li>
```

```
        @endforeach

    </ul>

  </div>
</div>
@endif
```

Autentication

Using Laravel's Built in authentication using Scaffolding.

```
php artisan make:auth
```

This will create a number of things:

1. in /routes/web.php a new route:

```
Auth::routes();
```

1. Migration files, which need to be migrated.
2. Point to the database you want to use. If SQLite, change your .env file and create a database.sqlite file in the /databases folder.

NOTE: in windows you need to add the php/extension directory to your path.

~~~
```composer require doctrine/dbal```

 is a prerequisite to using SQLite.

You also have to uncomment this line in the php.ini file:


```extension=php_pdo_sqlite.dll```



## Artisan ##
~~~

~~~
## Blade ##
## Composer ##

## Eloquent ##

##Carbon##

## Github Primer ##
Assume that you have a GitHub account, and that you have created a new repo.

1. Create local directory
2. Initialize Git

 ```git init```


3. Check status

 ```git status ```


4. Add files

 ```git add .```


5. Commit your files

 ```git commit -m "My commit message" ```


6. Add remote host

 ```git remote add origin https://github.com/username/myrepo.git```


7. Check remote host info

 ```git remote -v```


8. Push to remote repo

 ```git push origin master ```
~~~

NOTE: if you wish to override the remote changes with your local use

```git push --force ```

Pulling from remote repo

1.

 ```git pull origin master ```

Linux Notes

View processes

 ```top```

 or

 ```ps aux```

Kill processes

 ```kill <pid>```

Start Apache

 ``` sudo apachectl [start | stop | restart]```

```apachectl configtest ```

~~~~
```httpd -v ```



### Installing MySQL on Mac using Homebrew ###


```brew install mysql```



login to MySQL

 ```mysql -uroot```



configuration

 ```/usr/local/etc/opt/phpmyadin.config/inc.php```





```brew services start mysql```





```brew install phpmydmin```


NOTE: you may need to create a symbolic link for PHPMyAdmin

### Installing PHP with Homebrew ###


```brew install php71 | php56 ```


NOTE: you need to change the default PHP path in the Mac OS.
~~~~

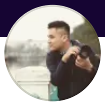
~~~
```export PATH=/usr/local/php5/bin:$PATH```



#### Making composer executable ####


```mv composer.phar /usr/local/php/bin/composer```



### Changing PATH ###


``` export PATH=/usr/local/php5/bin:$PATH```




```php --version```




### ATOM Setup ###

c:\ProgramData\ComposerSetup\bin\composer.phar
C:\Bitnami\wampstack-5.6.30-1\php\php.exe

1. Create route
2. Create controller
~~~

php artisan make:controller ControllerName

1. add required method to controller class

Discussion (0)

Code of Conduct • Report abuse



Lam Hoang

Hello

JOINED

Feb 22, 2020

More from Lam Hoang

Laravel Cheat Sheet - Form

#laravel #php

Laravel Cheat Sheet - Resource Controllers

#laravel #php

Laravel Cheat Sheet - Models Eloquent

#laravel #php