Laravel and Heroku

Study Case: Creating a Content Management System

Nugroho Satriyanto
Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl Ganesha no 10, Bandung 40132
13514038@std.stei.itb.ac.id

***Abstract*—Laravel, the most used php framework together with heroku, a cloud platform as a service, creating a CMS based website. CMS offers convenience for writing and adjust contents, laravel offers convenience for building the application, while heroku offers convenience for deploying application. With these three together, building website will be a lot easier.**

Keywords—CMS, platform, framework, migration

I. BACKGROUND

Content management system is the most used platform for creating a website. It allows for multiple users to work in a collaborative environment [1]. It is widely used because it supports even person who does not understand programming to edit the pages easily. Even for a programmer, it will make the content editing and structuring a lot easier.

Now, there is so many projects that implements the content management system. Because of that, the author write the topic of this report to create the content management system. For a personal purpose, the author decided to make a blog for its study case.

II. INTRODUCTION

A. Laravel

Laravel is the most used framework for PHP [2]. It claims for more beautiful code when it implemented and it makes the code is much more readable for another person. Laravel has so much documentation that helps the developer to understand the framework.

B. Heroku

Heroku is a cloud platform that lets companies build, deliver, monitor and scale apps [3]. Heroku allows the developer to focus to the apps he works for, without caring how to deploy it. As long as the developer knows how to use git, the developer can deploys the apps as easy as pushing into a git remote such as gitlab or github.

Not only facilitates the user for a deployment, heroku even provide a free PostgreSQL database to support the application. The heroku free database support up to 10,000 rows and 20 multiple connections which is more than enough to create a simple content management system like a personal blog that will the author create.

III. CONTENT

For creating a CMS platform there is several basic aspects that must be implemented. First is the homepage with multiple contents displayed. Second is the detailed page for a post that displays all the content of a post. Third is the admin dashboard that make the editing of content easier. Other than those basic aspects, there is several additional aspects that can be implemented, such as searching and filtering posts.

The first thing before creating CMS is first defining the database structure. For creating a simple CMS, at least we need two tables, which is users and posts table. The users table save all author that can make change to the website contents. While the posts table save all the posts both which is published and drafted posts.

Laravel has its own way to describe the database structure, it's called database migration. Migration allows developer to easily make query using laravels query builder and mapping the relational database to a php objects. Migration also allows developer to easily collaborate without using sql exports and imports. Developer just run simple commands such as "php artisan migrate" to get the latest database structure or "php artisan migrate:rollback" to back to the previous state of database structure.

The laravel posts table migration will looks like this

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->increments('id');
        $table->timestamps('created_on');
        $table->string('title', 40);
        $table->string('subtitle', 60);
        $table->text('content');
        $table->string('creator', 15);
        $table->string('category', 15);
    });
}
```

Figure 1 Post Database Migration

While the laravel users table migration will looks like.

```php
public function up()
{
    Schema::create('users', function (Blueprint $table) {
        $table->increments('id');
        $table->string('name');
        $table->string('email')->unique();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
    });
}
```

Figure 2 Users Database Migration

With laravel, the homepage, searching and filtering is easier to be implemented. It only needs a view to implement three functions. The author only create one view named home.blade.php which placed in resources/view directory.

```php
@section('content')
@foreach ($posts as $post)
<div class="post-preview">
    <a href="/post/{{$post->id}}">
        <h2 class="post-title">
            {{ $post->title }}
        </h2>
        <h3 class="post-subtitle">
            {{ $post->subtitle }}
        </h3>
    </a>
    <p class="post-meta">Posted by <a href="#">{{ $post->creator }}</a> at
        {{ $post->created_at }} on <a href="">{{ $post->category }}</a></p>
</div>
<hr/>
@endforeach
```

Figure 3 Home Blade

From that view, we only need to pass a different posts variable to make that three functionalities. This is what I do for the controller.

```php
public function allPost(){
    $posts = DB::table('posts')->select('id','created_at','title','subtitle','creator','category')->latest()->get()
    return \View::make('home', compact("posts"));
}
```

Figure 4 Post Controller

```php
public function search(){
    $posts = DB::table('posts')->where('title','like','%'.Input::get('search').'%')->
                orwhere('subtitle','like','%'.Input::get('search').'%')->
                orwhere('content','ilike','%'.Input::get('search').'%')->
                latest()->get();
    return \View::make('home', compact("posts"));
}
```

```php
public function filterCategory($category){
    $posts = DB::table('posts')->
                select('id','created_at','title','subtitle','creator',
                'category')->where('category',$category)->
                latest()->get();
    return \View::make('home', compact("posts"));
}

public function filterAuthor($author){
    $posts = DB::table('posts')->
                select('id','created_at','title','subtitle','creator',
                'category')->where('creator',$author)->
                latest()->get();
    return \View::make('home', compact("posts"));
}
```

For the post detail author add the route to /post/{id} where the id is unique identifier between posts. Because of the route directory is different from the base route directory, it must have a different blade view. For the controller I return all of the columns that the table have to be displayed.

```php
public function viewPost($id){
    $posts = DB::table('posts')->where('id',$id)->get();
    $post = $posts[0];
    $codes = DB::table('postcode')->select('code')->where('id_post',$id)->get();
    return \View::make('post', compact("post","codes"));
}
```

Figure 5 View Post

Where codes that is passed into view means the programming language the post has. The author used it in order to include a necessary syntax highlighter javascript for the code.

For creating a login and authentication system in laravel is very simple, just running a simple command, “php artisan make:auth” and laravel will scaffold a basic authentication system which is already good enough.

The last step for this project will be deploying the application to heroku. With heroku, deploying application is much easier. Just execute command “git init” to initialize a git for the directory, committing the changes and then push it with “git push heroku master”.

Every time the changes is pushed to heroku master branch, the remote automatically selecting the correct configurations based on the programming language and framework its use. The developer can literally do nothing about deploying the application because heroku handles everything. After the push command is finished, the website is ready and can be accessed.

IV. CONCLUSION

From author’s experience from using laravel, compared with creating a vanilla web without framework, laravel:

1. Have more learning curve if developer hasn’t experienced using framework.
2. Easier to read because of the MVC design pattern.
3. Less code to implement multiple view with same structure using layout.
4. Easier to collaborate for database because of laravel’s

database migration.
5. More secure because of csrf field, query builder, and routing.

Meanwhile compared to the usual deployment using a server, heroku:

1. Much easier, just push it and it's automatically deployed.
2. Free
3. Has a lot of programming language support.

From this paper, author has deployed a blog that created in a way that written in this paper. It can be accessed through http://scarycode.herokuapp.com/.

Figure 6 Home Page

ACKNOWLEDGMENT

Gratitude for the author to God that for His permission that the author was able to complete this paper. Author wants to say thank you to Mr Rinaldi Munir, Mrs Ayu Purwarianti, and Mrs Dessi Puji Lestari as Socio Informatics and Professionalism lecturers for their guidance, and the laravel and heroku forerunners that because of them this paper can be completed.

REFERENCES

[1] http://searchcontentmanagement.techtarget.com/definition/content-management-system-CMS accessed at May 4th 2.20 PM.

[2] http://www.hongkiat.com/blog/best-php-frameworks/ accessed at May 4th 4.00 PM.

[3] https://www.heroku.com/what accessed at May 5th 8.00 AM.

STATEMENT

With this, I state that this paper that I write is my own writing, not a summary nor a translated version of another paper or plagiarism.

Bandung, May 5th 2017

Nugroho Satriyanto