Laravel or How To Get Lost In Documentation

Internet Applications, ID1354

Linus Berg Linus@Fenix.me.uk

November 26, 2018

1 Introduction

The task of this seminar was to move the initially unstructured PHP files to a MVC structure, such as the Laravel framework (my choice).

On top of that the security should be improved to further accomodate a production server / code, passwords should be hashed et cetera, along with other security concerns.

2 Literature Study

The majority of this seminar was spent trying to learn Laravel, all of the learning was done on their official documentation.
Laravel Docs

3 Method

The first objective was to select which framework to use, the framework chosen was Laravel for the overwhelming amount of support / documentation available, it is the most widespread PHP framework currently, so it seemed like a good choice.

Once the PHP framework was chosen the conversion of the code started, firstly by setting up the routing, then the controllers, and if needed models. Each step included reading the documentation, for example, during the route creation, I went back and forth between implementation and reading the documentation for routes.

For each route that was setup the according view was also created using the Laravel Blade templating system. In each view that required AJAX request, the corresponding Ajax requests were also implemented in the routes and setup in the controller. It resulted

in a lot of back and forth between different files, however it turned out good enough in the end.

Because templates were already utilised in seminar one, it was merely a matter of changing them from the Jinja templating engine syntax to the Laravel one, which was extremely easy.

The security concerns were almost an afterthought as Laravel handles a lot of it for you by default.

Tool	Choice
Editor	*Vim*
Version Control	*Git - Github*
Web Server	*nginx*
Database	*PostgreSQL 10.5-1*
PHP	*7.2.10-1*
Framework	*Laravel 5.7*
UML Editor	*Dia*

4 Result

Github

4.1 Framework & code

When porting the code to Laravel, the framework makes several abstraction layers of code, for example, routing is a simple call to *Route::Get()*, this leads to significantly less code in the final result.

As such, the final class diagram is below, with each of the routing calls to a controller, the routes */comment/post/* and */comment/delete* are authenticated by the **auth** middleware. The class diagram is quite small due to the abstractions that laravel makes.

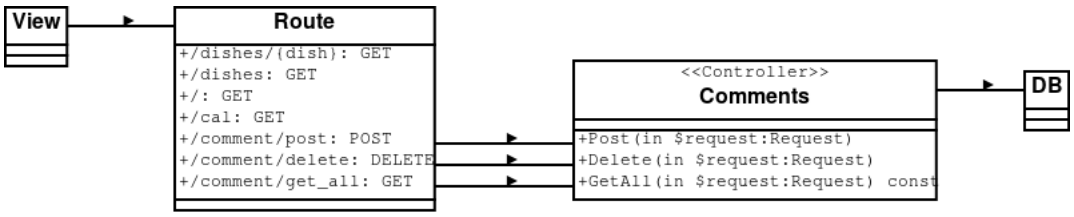

Figure 1: Laravel class diagram.

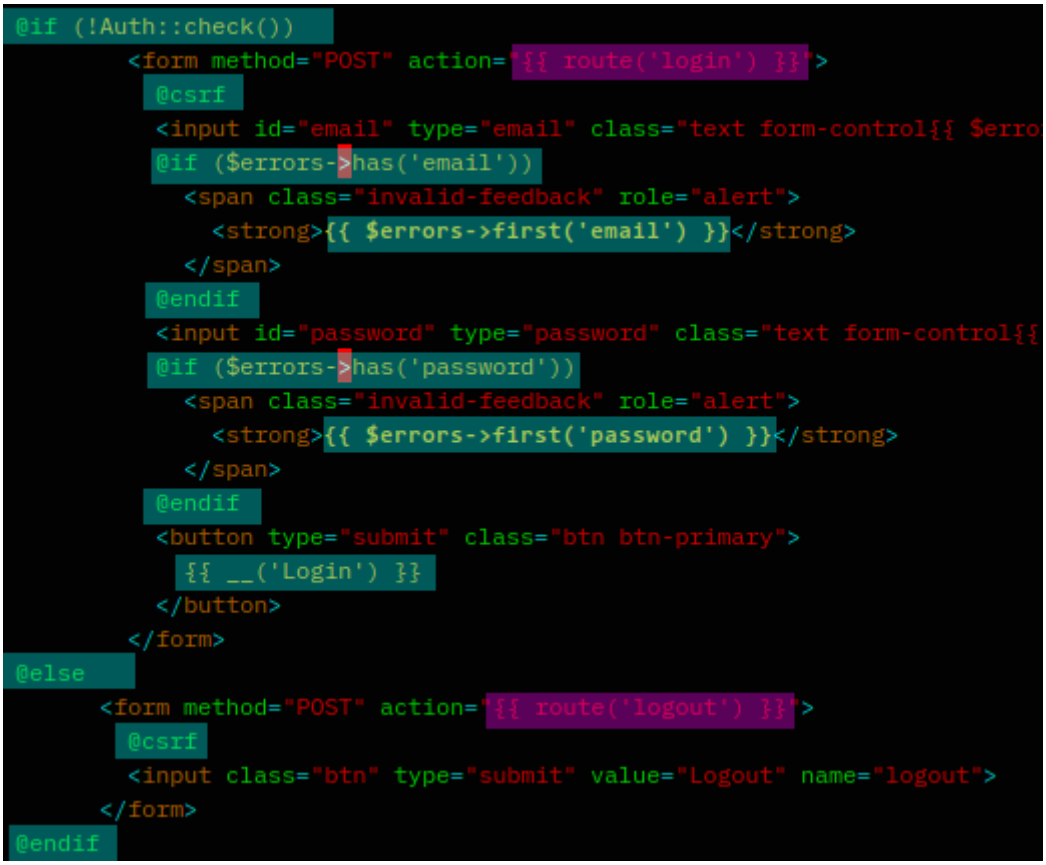

Figure 2: Laravel blade template.

In **??** a Laravel blade example is shown where the Laravel commands are highlighted. Laravel handles all authentication needs and therefor a lot of code does not need to be written to deal with the user logging in / commenting.

```php
Route::get('/', function () {
  return view('index');
})->name('home');

Route::get('/dishes', function () {
  return view('dishes');
})->name('dishes');

Route::get('/dishes/{dish}', function ($dish) {
  return view('display', ["DISH_ID"=>$dish]);
})->name('details');

Route::get('/cal', function () {
  return view('cal');
})->name('cal');

Route::delete('/comment/delete', "Comments@Delete")->middleware('auth');
Route::post('/comment/post', "Comments@Post")->middleware('auth');
Route::get('/comment/get_all', "Comments@GetAll");

Auth::routes();
```

Figure 3: Laravel routing.

The routing in laravel is extremely simple, in **??** the entirety of the routing used for the recipes website is shown, as described earlier, the *delete* and *post* are authenticated using the auth module from Laravel. Each of the */comment/* API is called by using Ajax in the webpage, which made it a lot easier to port the code from the standalone structure to the Laravel framework. Other than that, most of the functionality of Laravel is abstracted into simpler methods and not exposed to the user.

```php
namespace App\Http\Controllers;

use Illuminate\Support\Facades\Auth;
use Illuminate\Http\Request;
use DB;

class Comments extends Controller {
  public function Post(Request $request) {
    DB::table('comments')->insert(['user' => Auth::user()->email,
                                   'recipe' => $request->dish_id,
                                   'comment' => $request->txt]);
    return (Auth::Check() ? '1' : '0');
  }

  public function GetAll(Request $request) {
    $result = DB::table('comments')->where('recipe', $request->dish_id)->get();
    return $result->toJson();
  }

  public function Delete(Request $request) {
    $where = [
      ['id', $request->cmt_id],
      ['user', Auth::User()->email],
    ];
    DB::table('comments')->where($where)->delete();
  }
}
```

Figure 4: Comments controller.

The controller shown in **??** is the only functionality (other than routes) that Laravel utilises, that was written specifically for this site, the simple returns could be further improved to contain more information. Each method simply manipulates or gathers data from the database layer, and it utilises the Laravel query builder, unforunately this functionality is also abstracted.

4.2 Security

- File System Security ✓
- Input filtering ✓
- Database security ✓
- Password Encryption ✓
- XSS ✓
- Impersonation ✓
- HTTPS ✗

Above are the security concerns that should be considered/implemented, the implemented ones are marked with a checkmark. As stated earlier the security concerns were mostly an after thought as Laravel handles many of them, for example, Laravel handles, *Input filtering, Password encryption, and Impersionation*, however, it also handles a lot more in the background that the user is not exposed to. Impersonation is handled via CSRF tokens which are also sent in the header of each Ajax request.

I utilise Arch Linux on my personal laptop and CentOS with SELINUX on my rackserver at home, so I am no stranger to setting up the user permissions in the file system, the nginx user was always set up to minimize the amount of access it has. The database was correctly setup as well with the user, and all remote access was promptly removed.

As for the remaining two check marks (*XSS & HTTPS*), XSS should not be possible due to the input filtering occuring by Laravel. I did not setup HTTPS on my local test server, however obviously these days HTTPS should be used by any website to secure the connection between user and server. I do however run HTTPS on my personal website Https://TheMountain.XYZ as an example of how I utilise HTTPS.

5 Discussion

During this seminar I chose Laravel because of the simplicty of the framework, however as stated many times in the text, it has a lot of abstraction layers, and that leads to less understanding of making a PHP MVC structure, however, it is rare these days to develop your own framework and not modify an existing one.

I used a database in the previous seminars and therefor I used one in this one as well since it integrates very easily into Laravel, but it also lead to optional task 2 being solved without even trying.