OULUN AMMATTIKORKEAKOULU

Quang Hung Nguyen

BUILDING A WEB APPLICATION WITH LARAVEL 5

BUILDING A WEB APPLICATION WITH LARAVEL 5

Quang Hung Nguyen
Bachelor thesis
Autumn 2015
Business Information Technology
Oulu University of Applied Sciences

CONTENTS

1 Introduction

In modern IT industry, it is essential for web developers to know at least one battle-proven framework. Laravel is one of the most successful PHP framework in 2015, based on annual framework popularity survey conducted by SitePoint (SitePoint, The Best PHP Framework for 2015: SitePoint Survey Results, cited, 25.10.2015).

There are several advantages and benefits of using web framework in general and Laravel in particular. Framework is a product of collective intelligence, comprising many robust libraries and convenient tools from other developers. They help to reduce most of the repetitive tasks and complex tasks in simple interface, which means developers can write less and do more with highest quality in a certain amount of time. Therefore, using a reliable framework also help to lower the development cost.

Furthermore, using a robust web framework also helps to strengthen the security of the application. It does not require developers have a deep knowledge about security. There are many forms and types of cyber attacks, fortunately, Laravel as well as most of web frameworks support several features to prevent basic security vulnerabilities such as SQL injection, cross-site request forgery (CRF) and cross-site scripting (XSS).

Moreover, Laravel also provides a well-structured skeleton for building big projects. Basically, Laravel employ the famous MVC architectural pattern, which helps organizing code better. MVC pattern separate the business logic (models), the control coordination (controllers) and the presentation of data (views) into 3 different layers. In other words, the heart of MVC pattern is the idea of "Separation of concern" *(Adam Freeman, Pro Design Patterns in Swift, page 529)*. The framework also supports modular architect which enables developers separate code into independent workable modules.

The main outcome of this thesis is to build a movie review website which allow users give ratings for their favorite movies, browsing movies by genres and searching for a movie. In order to give review or rating, guest must open an account and login to the system. In the backend side, administrators can also create and manage new genres, movies, reviews and users.

The motivation behind this project is the need for a movie reviews website that help people get a glance of upcoming movies or un-watched movies. In fact, more and more movies are released in theaters, approximately 500 movies each year. In 2013 and 2014, there were 659 and 707 movies were released respectively in North America (Motion Picture Association of America, Theatrical Market Statistics 2014, cited 30.11.2015). It is hard for movie lovers to choose their suitable titles because of fast pace of life.

There are several famous movie review websites on the internet, however there is no equivalent website in Vietnamese. Thus, I want to make a similar site for Vietnamese netizens, but the website user interface will be implemented in multiple languages. Additionally, the collected movie data and reviews from users can be used for further study: building a movies recommendation system. The suggestion system will need a relatively large enough amount of ratings data from users in order to make suggestions. Therefore, it is necessary to create review website to collect enough data first.

As mentioned previously, using a framework (Laravel 5.1 in particular) helps to improve the quality and the development cost in overall. In consequence, Laravel 5.1 is a good choice to use to build a movie review website. It is also a good opportunity for me to upgrade my knowledge from Laravel 4 to Laravel 5.1 and apply it in a practical project. Laravel 4.x is released for a long time and now it is replaced by Laravel 5.x. There are no much differences between Laravel 5.0, 5.1 and 5.2, therefore it is fast to upgrade the application to the next Laravel minor version. Moreover, in this document, I assume that readers have some basic understanding about web development as well as Laravel 4. Therefore, I will only cover several common Laravel features and focus mainly on new features in Laravel 5 in this thesis.

2 Laravel

Laravel is a clean and well-structured web framework for PHP development. According to its author, Laravel is “The PHP framework for Web artisans*” (Laravel official website, homepage, cited 25.10.2015).*

Over years, PHP gained the bad reputation because of many badly written websites. PHP is an easy-to-learn language, and it was designed mainly for small web sites. The name “PHP” originally came from the term “Personal home page” (PHP official website, History of PHP, cited 25.10.2015). However, in the past couple of years, PHP has been evolved to bring many modern new features such as namespace, traits and *a de facto* package manager: Composer.

“Laravel decided to hop on the shoulder of giants and embrace these pre-existing mature components.” (Martin Bean, Laravel 5 Essentials, page 3)

Laravel is also built on top of Symfony HTTP foundation, which is a solid, flexible and testable HTTP foundation for PHP application. Besides Laravel, latest version of Drupal, phpBB and other open sources projects also use Symfony HTTP foundation as a essential part of theirs core (Projects using Symfony). Moreover, Laravel also leverages other Symfony components and several various popular PHP libraries such as SwiftMailer, Carbon, Doctrine. etc. (Packagist: The PHP Package Repository, The Laravel Framework, cited 20.11.2015)

Basically, Laravel is a fully MVC-compliant framework (Maks Surguy, History of Laravel PHP framework, Eloquence emerging, cited 21.04.2015). It comprises all necessary components for a MVC web framework and highly configurable. The framework come along with Eloquent, a simple ORM and blade templating language that helps developers build website in shorter time and less effort

.

2.1 Structure of a Laravel 5 application

Basically, all user-defined code live in app directory (Figure 1). It is the heart of Laravel application where all HTTP requests are handled. In app folder, each type of application layer is stored in separate sub folders. Previously in Laravel 4, the app folder keeps all application's logic, framework code as well as configuration files. However, in Laravel 5, the app folder is trimmed down and mainly stores application logic classes including model, controllers, commands, middleware etc. By default, the app directory is auto loaded by Composer using PSR-4 auto loading standard. It worth to know that most of class types in app folder can be created via artisan commands *(Laravel official website, Artisan Console, cited 20.11.2015)*.

In addition, other type resources such as blade templates, LESS or SASS files, CoffeeScript and language files are now stored outside the resources folder instead of the app folder. Tests and migration folder are also moved out of app folder. However, vendor and bootstrap and public folder still remain the same place as in Laravel 4.

```
.
├── app # Your Laravel application
│   ├── Commands
│   ├── Console
│   ├── Events
│   ├── Exceptions
│   ├── Handlers
│   ├── Http
│   ├── Providers
├── artisan # Artisan command-line utility
├── bootstrap
│   ├── app.php
│   ├── autoload.php
│   └── cache
├── composer.json
├── config
├── database
│   ├── migrations # Database migration classes
│   └── seeds # Database seeder classes
├── package.json
├── phpspec.yml
├── phpunit.xml # PHPUnit configuration file
├── public
│   ├── assets
│   ├── favicon.ico
│   ├── index.php # Entry point to run the application
│   ├── packages
│   ├── robots.txt
├── resources
│   ├── assets
│   ├── lang
│   └── views # Contain blade template
├── server.php # A lightweight local development server
├── storage
│   ├── app # App storage, like file uploads etc
│   ├── framework # Framework storage (cache)
│   └── logs # Application-generated logs
├── tests # Contain all test cases
└── vendor # All third-party libraries installed by Composer
```
Figure 1 Laravel application structure

2.2 Model-View-Controller and Middleware

Model-View-Controller is one of the most popular architectural design pattern for web application (Figure 2).The concept was originally invented by Trygve Reenskaug in 1970s as part of a Smalltalk system being developed at Xerox PARC before the born of World Wide Web and the era of personal computer (c2.com, Model View Controller History, cited 25.10.2015). It is worth noting that that Laravel is a fully MVC-compliant framework (Maks Surguy, History of Laravel PHP framework, Eloquence emerging, cited 21.04.2015). Therefore, it is important for Laravel developer to understand this pattern thoroughly.

The most essential idea is to separate application/business logic from the user interface. It provides a basic structure to organize your code in order to improve the maintainability. In other words, instead of mixing logic PHP code into the same file with HTML markups, they are split into three separate layers. Moreover, it is not only improving the source code readability but also help to separate the work between front end and backend developers.

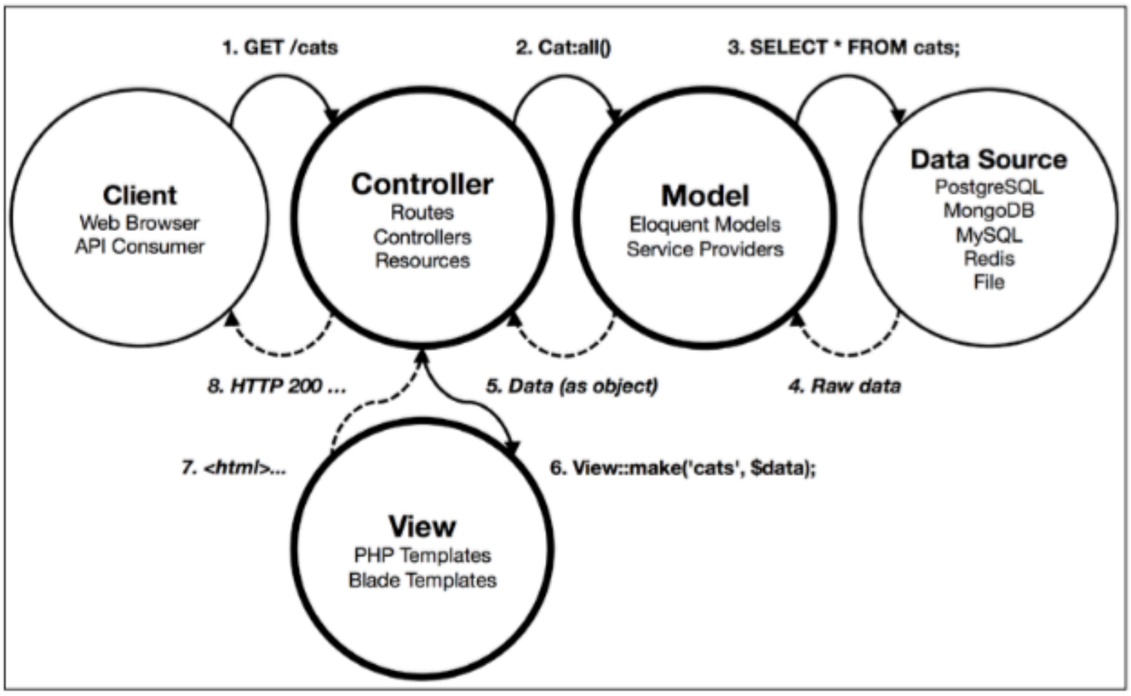


Figure 2 MVC pattern diagram

Model is an abstraction layer, which presents data in an application. In general, it usually corresponds to the records in DBMS database or any type of storage engine such as NoSQL or Microsoft Access. In Laravel 5, a Model usually extends from Eloquent master class and presenting a table in database. (Martin Bean, Laravel 5 Essentials, page 8)

View is the layer that present data in suitable and in desire format for end client, and usually presented as a HTML web page. It is the final visual presentation of data in a web application that Controller has received based on the change from Model. Laravel 5 come along with Blade template language, which allow developers create reusable and maintainable layouts and template components. (Martin Bean, Laravel 5 Essentials, page 9)

Basically, controller is the layer that handles requests and return appropriate responses to the correct View. Therefore, Controller acts as a coordinator between View and Model. In other words, it is the logical layer of the application. For example, controller can receive form submission from the View and perform validation, writing valid data to a database, and redirecting users to other routes. (Martin Bean, Laravel 5 Essentials, page 9)

Laravel 5 come with new concept that supplement to Controller: Middleware. The Figure 3 explains the relations between middlewares and application as well as client requests and responses. It is a convenient mechanism to control and security check all incoming requests and responses with proper header (Laravel official website, HTTP Middleware page, cited 03.01.2016)

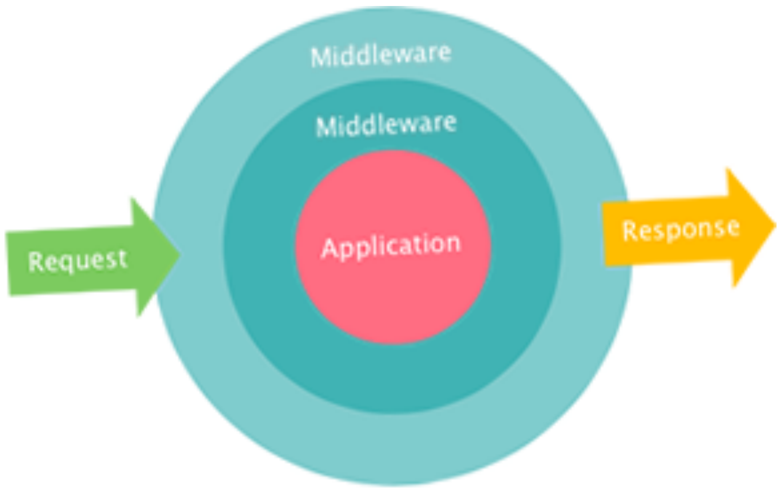


Figure 3 Middleware illustration

2.3 Contracts and Repository pattern

Basically, repository pattern and contracts in Laravel 5 is used to make application more flexible and easier to test than fat controller. It is also a way to organize the source code better. The main idea is instead of your code is tightly coupled to a concrete implementation, you can easily write an alternative implementation of any given contracts.

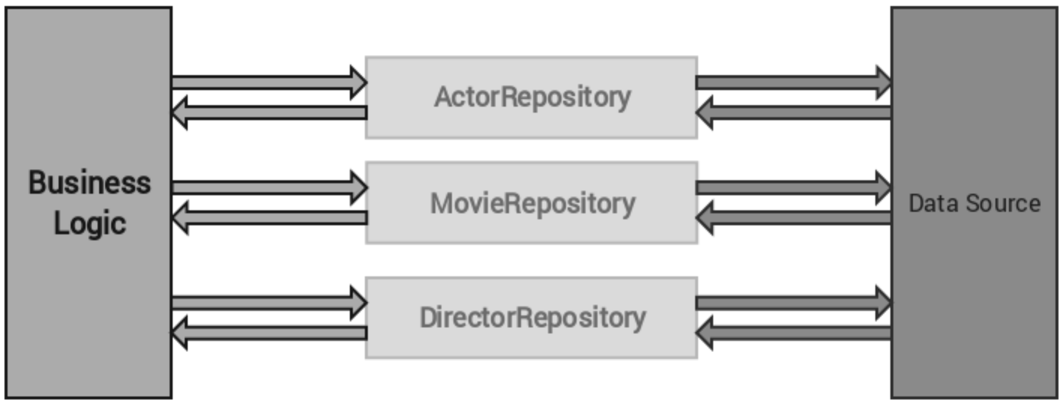


Figure 4 Repository layer

Figure 5 illustrates a particular tightly coupled function, and it is hard to maintain and extend because of two reasons: Firstly, it is tied to App\Models\Movie class. Particularly it is a class extends from Eloquent. If developer want to switch to another data storage such as Mongo DB, he/she need to refactor the whole application source code. Secondly, fat controller is hard to test, because developer has to simulate request data from client browser, and normally controller return a html response, which is hard to validate the raw result.

```php
public function doUpsert(Request $request, $id = null)
{
  $form = $this->form(\App\Forms\MovieForm::class);

  if (!$form->isValid()) {
      return redirect()->back()->withErrors($form->getErrors())->withInput();
    }

  $movie = (empty($id)) ? new Movie : Movie::find($id);
  $movie->fill($request->all());
  $movie->user()->associate(Auth::user());
  $movie->save();

  return redirect()->route('admin.movies');
}
```

Figure 5. A sample tightly coupled function in controller

Firstly, within fat controller function, all business logics and database interactions are written in each function and they are not reusable in others. Secondly, when the storage engine need to be changed, the whole code base have to be refactored. It takes more resources (e.g: time and money) and contains higher probability risk for hidden bugs. It also violate the clean code principles: don't repeat yourself and single responsibility principle.

When repository pattern is used, it is much easier to change the logic and easier to write unit test. The Figure 6 illustrates an example of loose-coupled controller function where all actual database create/update code is separated from business logic code.

```php
public function doUpsert(Request $request, $id = null)
{
    $form = $this->form(\App\Forms\MovieForm::class);

    if (!$form->isValid()) {
        return redirect()->back()->withErrors($form->getErrors())->withInput();
    }

    if (empty($id)) {
        $this->movie->create(Auth::user()->id, $request->all(), $request);
    } else {
        $this->movie->update($id, Auth::user()->id, $request->all(), $request);
    }

    return redirect()->route('admin.movies');
}
```

Figure 6. An example of loosely coupled function

The Figure 7 illustrates a particular contract. It creates a common contract of all repositories which technically an PHP interface. Contracts contains all required methods that the concrete repository classes need to implement.

```php
<?php namespace App\Contracts;

interface RepositoryInterface {

    public function find($id);

    public function create($user_id, array $data, $request);

    public function update($id, $user_id, array $data, $request);

    public function delete($id);

    public function all();

    public function get();
}
```
Figure 7 An example of contract

The interface can easily bind to the concrete implementation as demonstrated in Figure 8, and it helps developers easily change the implementation later. Moreover, each repository interface and its concrete implementation have to be registered in `AppServiceProvider`. It is the static way to bind the contract with its implementation.

```php
<?php namespace App\Providers;

use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider {

    public function register()
    {
        $this->app->bind(
            'Illuminate\Contracts\Auth\Registrar',
            'App\Services\Registrar'
        );

        $this->app->bind(
            'App\Contracts\MovieRepositoryInterface',
            'App\Repositories\MovieRepository'
        );
    }
}
```
Figure 8 Contract and concrete implementation binding

Particularly, MovieRepository which currently use Eloquent and MySQL as shown in Figure 9. The mapping can be easily changed to MovieMongoDBRepository that connect to MongoDB in the future or in complex situations. Laravel supports contextual bindings for the situation where two classes use the same interface, but they require different implementations. Developers can also bind the interface with the desired concrete class based on certain context (Laravel official website, Service Container, cited 31.05.2016).

```php
<?php namespace App\Repositories;

use App\Models\Movie;
use App\Contracts\MovieRepositoryInterface;

class MovieRepository implements MovieRepositoryInterface {

    public function __construct(\App\Models\Movie $movie) {
        $this->movie = $movie;
    }

    public function find($id) {
        return $this->movie->find($id);
    }

    public function create($user_id, array $data, $request) {
        $movie = new Movie;
        return $this->upsertMovie($movie, $user_id, $data, $request);
    }

    public function update($id, $user_id, array $data, $request) {
        $movie = Movie::find($id);
        return $this->upsertMovie($movie, $user_id, $data, $request);
    }

    private function upsertMovie($movie, $user_id, $data, $request)
    {
        $movie->fill($data);
        $movie->user()->associate($user_id);

        $image_ext = $request->file('image')->getClientOriginalExtension();

        if (!empty($data['image'])) {
            $movie->image = str_replace('/', '_', $data['image'] . str_random(12)) . '.' . $image_ext;
        }

        if (!empty($data['genre_id'])) {
            $movie->genre()->associate((int)$data['genre_id']);
        }

        $request->file('image')->move(
            base_path() . '/public/images/movies/', $movie->image
        );

        return $movie->save();
    }

    public function delete($id) {
        return;
    }

    public function all()
    {
        return Movie::orderBy('year', 'desc')->get();
    }

    public function news()
    {
        return Movie::where('year', '>=', 2000)->orderBy('year', 'desc')->limit(5)->get();
    }

    public function get() {
        return Movie::get();
    }
}
```
Figure 9 Concrete implementation of movie repository

2.4 Blade template

In web development, templating engine/language is a template preprocessor to combine all view elements to form the final finished web page. The main purpose is to separated application logic to its presentations (Cristian Darie, Beginning PHP and PostgreSQL E-Commerce: From Novice to Professional, page 22). Laravel 5 support Blade templating language, it is an essential tool to create and manage hierarchical (inheritance and sections) layouts (Laravel official website, Blade Templates, cited 01.12.2015). With Blade template language, developers can easily create common master view as well as the particular content of each child view. Each individual view can have different and unique markups, all sub views are inherited common sections (e.g.: view header, footer and menu) from master, and only page-specific content must be defined in each sub view.

Blade templating language contains a small set of directives and control structures. In order to use it we need to know the basics of Blade. Fundamentally, blade templating language will be parsed into PHP code at the end. The Figure 10 illustrates how blade templating statements are finally parsed into PHP code.

Blade	Standard PHP syntax
`{!! $var !!}`	`<?php echo $var;?>`
`{{ $var }}`	`<?php echo htmlentities($var); ?>`
`@if ($cond) ... @endif`	`<?php if ($cond): ?>...<?php endif; ?>`
`@foreach($values as $value)...@endforeach`	`<?php foreach($values as $value):?>...<?php endforeach;?>`
`@while(true)...@endwhile`	`<?php while(true):?>...<?php endwhile;?>`

Figure 10 Blade syntax and equivalent parsed PHP code

Blade templating language also supports directives to create master layout with multiple inherit sections. The Figure 11 explains several important directives in Blade and its application.

Blade syntax	Application
`@section…@end`	Define a section in template
`@extends(‘template_name’)`	Extend from an exists template
`@parent`	Inherit from parent template
`@yield('content')`	Define a section in parent template that can be replaced in the child templates.

Figure 11 Blade templating language directives

2.5 Package manager: Composer

As previously described, Laravel is built on top of several third-party libraries. Instead of including all these external dependencies in its source code, Laravel uses Composer to manage its dependencies. Composer is a PHP tool, which allows you declare all required libraries in your project (in composer.json) and it will manage (install/update) it for you. Composer is inspired by other popular dependency mangers in other languages, such as Ruby’s Bundler or npm in Node.js. It has quickly become de-facto dependency manager in PHP.

Developer can easily create new Laravel project via composer. Composer will check if your environment settings meets Laravel 5 requirements or not.

```
composer create-project laravel/laravel --prefer-dist
```

Composer will show warnings if your working environment does not meet the requirements. For example:

```
- This package requires php >=5.5.9 but your PHP version (5.4.30) does not satisfy
that requirement.
```

(Composer website, Getting started page, cited 25.10.2015)

2.6 Artisan

There are several common tasks that developers need to do repetitively during the development process. Those common tasks contain a lot of boilerplate code which can be constructed automatically. Therefore, Laravel provide `artisan` - a built-in command line interface. It is a collection of useful command utilities to help developer quickly create the skeleton code or doing some administration tasks. *(Laravel official website, Artisan Console, cited 20.11.2015)*

Artisan helps developers build and manipulate database schema (data definition) via migration files. We can also swiftly create Controller, Model, Middleware and Event classes via this interface without copy-and-paste boilerplate codes.

Moreover, besides the existing utilities, developers can make their own commands and execute them via artisan. For instance, we can create a command to backup our database every 15 minutes and send it to backup server. Queue jobs can be also managed via this command interface.

2.7 Setup working environment and workflow

In order to set up and create the first Laravel application, developers need to set-up the proper environment for Laravel 5. Traditionally, developers install each component of LAMP stack separately. However, it is not good for working on collaborative project. Therefore, Vagrant is introduced to overcome that issue.

The Laravel framework has a few system requirements:

- PHP >= 5.5.9
- Mcrypt PHP Extension
- OpenSSL PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension

As of PHP 5.5, some OS distributions may require you to manually install the PHP JSON extension. When using Ubuntu, this can be done via `apt-get install php5-json`. *(Laravel official website, Installation page, cited 25.10.2015)*

Laravel comes with officially development environment: Homestead. It is an official pre-packaged Vagrant box that includes all basic necessary software for Laravel development including the Nginx web server, PHP 7.0, MySQL, Postgres, Redis, Memcached, Node. Homestead is built on top of vagrant, and both of them run cross-platform, disposable and easy to maintain. If something went wrong, the whole development environment can be destroyed and re-created in few minutes.

Traditionally, developers have various choices to build their local web development environment, ranging from all-in-one server stack (e.g: XAMPP, Zend Server), via package manager like homebrew, apt and yum to built and install each individual component from source code.

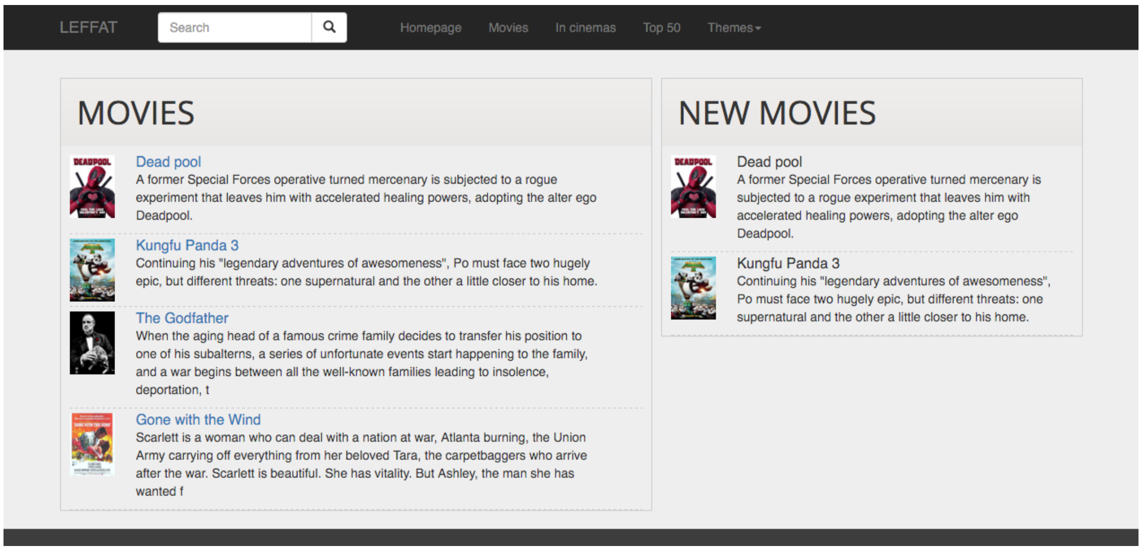
The problem is you can have multiple projects running various PHP, Redis, Node.js version. Moreover, developers in one team can have different local server setups. There are many different combination of package dependencies and software version for each local server. However, it is very important to make sure your local and testing environment match with production server. Therefore, there is possibility for a local test passed code but cannot run on production server due to different OS or software version.

Virtualization is an answer for this problem, with homestead and vagrant developers can setup their own working environment to match other developers, and more importantly with production server. With vagrant, every team member who is involved in a collaborative project can easily share a unified working environment. Moreover, it is fast and simple when the whole team need to upgrade all development OS and tools.

3 Building application with Laravel

3.1 The movie review application

In this chapter, the actual movie review application will be built with Laravel in order to demonstrate corresponding Laravel 5 features. Basically, users are able to browse through list of movies as shown in Figure 12. Moreover, they can go further in to movie details page and leave reviews as well as rating for that movies.



MOVIES

Dead pool
A former Special Forces operative turned mercenary is subjected to a rogue experiment that leaves him with accelerated healing powers, adopting the alter ego Deadpool.

Kungfu Panda 3
Continuing his "legendary adventures of awesomeness", Po must face two hugely epic, but different threats: one supernatural and the other a little closer to his home.

The Godfather
When the aging head of a famous crime family decides to transfer his position to one of his subalterns, a series of unfortunate events start happening to the family, and a war begins between all the well-known families leading to insolence, deportation, t

Gone with the Wind
Scarlett is a woman who can deal with a nation at war, Atlanta burning, the Union Army carrying off everything from her beloved Tara, the carpetbaggers who arrive after the war. Scarlett is beautiful. She has vitality. But Ashley, the man she has wanted f

NEW MOVIES

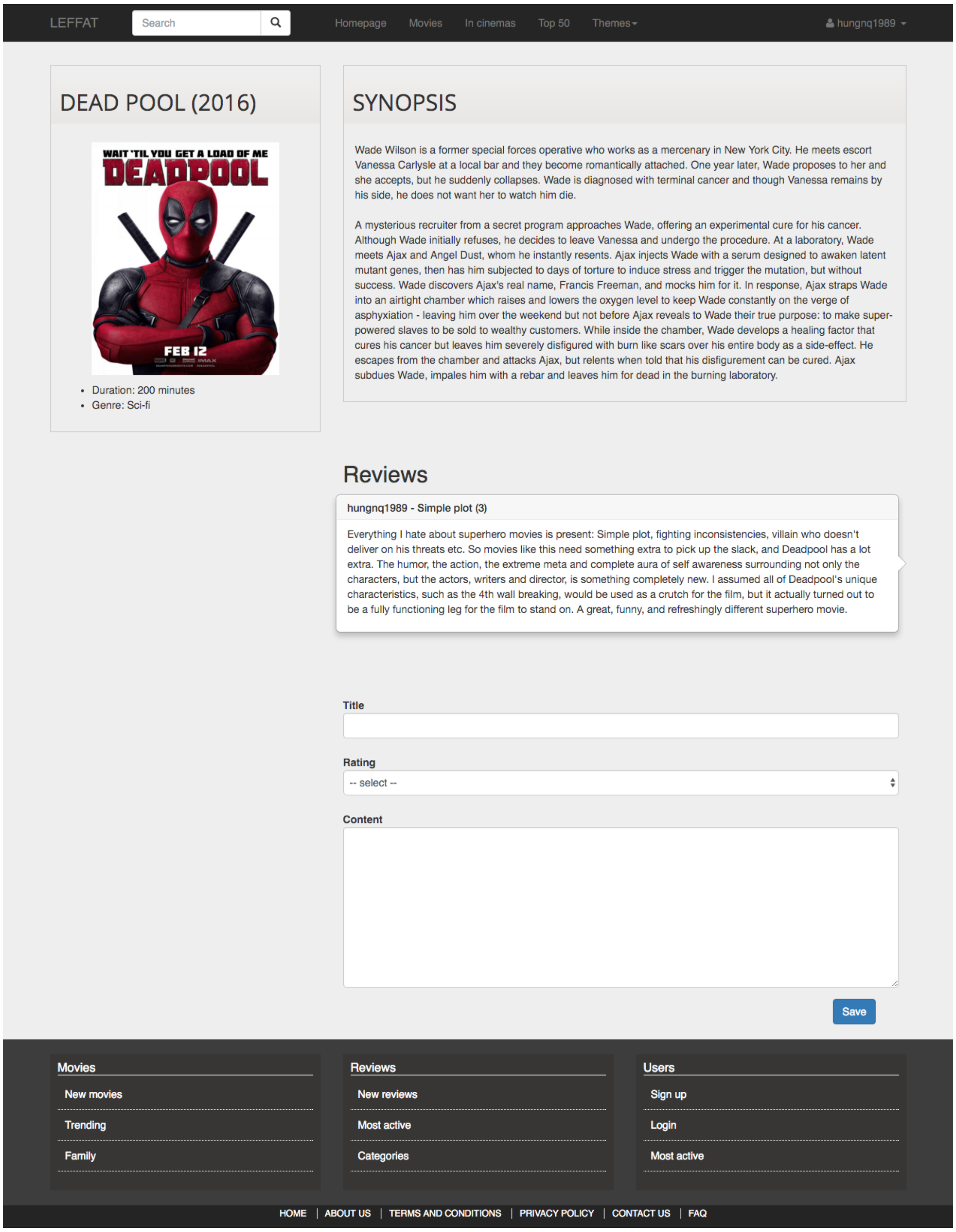
Dead pool
A former Special Forces operative turned mercenary is subjected to a rogue experiment that leaves him with accelerated healing powers, adopting the alter ego Deadpool.

Kungfu Panda 3
Continuing his "legendary adventures of awesomeness", Po must face two hugely epic, but different threats: one supernatural and the other a little closer to his home.

Figure 12 All movie list page design

The Figure 13 illustrates the appearance of a particular movie detail page. The page shows all information about that movie such as title, synopsis, published year and duration. In addition, it also displays all users' reviews and a form for registered users who want to leave a review and rating for that movie.

The UI interfaces also reflects real life situation, where different Laravel features and techniques are employed to create web application. Although, not all movie review application features are shown in this chapter, the selected features and techniques in the two illustrations are enough for building all other features.



DEAD POOL (2016)

- Duration: 200 minutes
- Genre: Sci-fi

SYNOPSIS

Wade Wilson is a former special forces operative who works as a mercenary in New York City. He meets escort Vanessa Carlyle at a local bar and they become romantically attached. One year later, Wade proposes to her and she accepts, but he suddenly collapses. Wade is diagnosed with terminal cancer and though Vanessa remains by his side, he does not want her to watch him die.

A mysterious recruiter from a secret program approaches Wade, offering an experimental cure for his cancer. Although Wade initially refuses, he decides to leave Vanessa and undergo the procedure. At a laboratory, Wade meets Ajax and Angel Dust, whom he instantly resents. Ajax injects Wade with a serum designed to awaken latent mutant genes, then has him subjected to days of torture to induce stress and trigger the mutation, but without success. Wade discovers Ajax's real name, Francis Freeman, and mocks him for it. In response, Ajax straps Wade into an airtight chamber which raises and lowers the oxygen level to keep Wade constantly on the verge of asphyxiation - leaving him over the weekend but not before Ajax reveals to Wade their true purpose: to make super-powered slaves to be sold to wealthy customers. While inside the chamber, Wade develops a healing factor that cures his cancer but leaves him severely disfigured with burn like scars over his entire body as a side-effect. He escapes from the chamber and attacks Ajax, but relents when told that his disfigurement can be cured. Ajax subdues Wade, impales him with a rebar and leaves him for dead in the burning laboratory.

Reviews

hungnq1989 - Simple plot (3)

Everything I hate about superhero movies is present: Simple plot, fighting inconsistencies, villain who doesn't deliver on his threats etc. So movies like this need something extra to pick up the slack, and Deadpool has a lot extra. The humor, the action, the extreme meta and complete aura of self awareness surrounding not only the characters, but the actors, writers and director, is something completely new. I assumed all of Deadpool's unique characteristics, such as the 4th wall breaking, would be used as a crutch for the film, but it actually turned out to be a fully functioning leg for the film to stand on. A great, funny, and refreshingly different superhero movie.

Title

Rating

-- select --

Content

Figure 13 Movie details page

3.2 Creating and configuring new application

It is very easy to install Laravel and create new application with composer. After Laravel is installed, developers can create new Laravel application in terminal with this simple command: `laravel new.` For example, “laravel new movies” would create a new “movies” directory with the framework and all dependencies installed.

```
laravel new movies
```

After the new application is created, environment file has to be changed in order to meet development environment otherwise Laravel cannot connect to database and other services. Developers can file all configuration files in config directory, but most of them read values from .env file. Generally, all sensitive data such as database or email credentials and basic application setting are set in .env file. Normally, this environment file is not pushed to version control server (e.g: git or svn) due to security reasons. This is a example of an environment file (.env):

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=oOeFlqzBxJQwJymSHcbUY0eYIHcSW4CX

AUTH_DRIVER=ELOQUENT
AUTH_MODEL=USER
AUTH_TABLE=USERS
SESSION_DRIVER=file

DB_HOST=localhost
DB_DATABASE=movie_review
DB_USERNAME=root
DB_PASSWORD=

CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync

MAIL_DRIVER=smtp
MAIL_HOST=mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
```

3.3 Migration files and Eloquent models creation

As previously discussed, models are the presentation layer of data in our application. They are usually mapped with tables in database. Laravel 5 is equipped with Eloquent ORM, a simple ActiveRecord implementation, that helps developers interact with database tables more effectively by making reusable and concise code *(Laravel official website, Eloquent: Getting Started, cited, 05.12.2015)*. Moreover, in order to support database schema version management, Laravel 5 also packaged with Migration tool. It is like version control of your database, enabling all team members can create, modify and share the database schema (Laravel official website, Database: Migrations, cited, 05.12.2015).

Normally, the first and most essential step is to create database schema (migrations) based on the ERD diagram. Afterward, the Eloquent models need to be defined to match with the database design. These models help developers interact with certain tables in database effectively by converting tabular data to objects. It also helps developers reuse and handle SQL query easier by supporting relationship mapping and query builder (Laravel official website, Eloquent: Relationships, cited 30.05.2016).

The Figure 14 illustrates an example of a migration which is created by Laravel. The user table definition is written in the up method. When the command “php artisan migrate” is run, the code inside up method will be executed to create a new table named “users” in database. Noted that user migration file is always created by default in Laravel 5 installation, it is not required to run any command to create this migration file. However, with user-defined tables, developers need to create corresponding migration file manually or using “php artisan” utility.

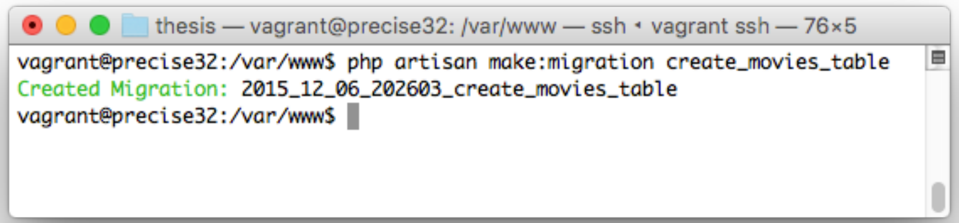
```php
<?php
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateUsersTable extends Migration {

    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function(Blueprint $table)
        {
            $table->increments('id');
            $table->string('name');
            $table->string('email')->unique();
            $table->string('password', 60);
            $table->rememberToken();
            $table->timestamps();
        });
    }
}
```

Figure 14 A sample migration file

Figure 8 demonstrates how a user-defined migration file is created using artisan utility. It is a good practice that to use the artisan tool instead of create it manually, because the utility also adds date time to the file name that keeps our migrations file in chronological order. The command will create a new backbone migration file. Developers only need to add table definition code inside "up" method.


thesis — vagrant@precise32: /var/www — ssh ◂ vagrant ssh — 76×5

```
vagrant@precise32:/var/www$ php artisan make:migration create_movies_table
Created Migration: 2015_12_06_202603_create_movies_table
vagrant@precise32:/var/www$
```

Figure 15 Command to create a migration file for movies table

```php
<?php
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateMoviesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('movies', function (Blueprint $table) {
            $table->increments('id');
            $table->integer('user_id')->unsigned()->index();
            $table->string('title');
            $table->string('synopsis');
            $table->integer('year');
            $table->integer('duration');
            $table->foreign('user_id')
                ->references('id')
                ->on('users')
                ->onDelete('cascade');
            $table->timestamps();
            $table->softDeletes();
        });
    }
}
```

Figure 16 User-defined migration file with foreign key declaration

The Figure 9 illustrates how a table with primary key and foreign key is defined. It also explains how developer can easily add timestamp field (creation time and update time) to the table. Moreover, Laravel also support soft delete feature that help we logically delete an entry in database by marking it as deleted. It is a convenient feature that allow developers resurrect data when it is accidentally removed by users.

After migration files and tables in database were created, Eloquent models need to be built based on the created tables. The Figure 17 illustrates a particular eloquent model with all mapped attributes as well as its relationships. It also explains how developers can create custom attribute for a model. The custom attribute must follow the naming convention "`getXXXAttribute`", and it can be automatically accessed in other places as an attribute of the model.

```
<?php namespace App\Models;

use Illuminate\Database\Eloquent\Model;


class Movie extends Model
{
    /**
     * The database table used by the model.
     *
     * @var string
     */
    protected $table = 'movies';

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = ['title', 'image', 'year', 'duration', 'synopsis'];


    public function getImagePathAttribute()
    {
        return '/images/movies/' . $this->image;
    }

    public function getShortSynopsisAttribute()
    {
        return str_limit($this->synopsis, 225);
    }

    public function user()
    {
        return $this->belongsTo('App\User');
    }

    public function genre()
    {
        return $this->belongsTo('App\Models\Genre');
    }
}
```

Figure 17 Movie model

3.4 Middleware and controller

Authentication and authorization is one of the most important part of any IT system involves users. In this paper, authentication and authorization is consider as one entity which is inseparable. Laravel 5 provides several tools for developers to easily and securely implement this feature. As mentioned earlier in chapter 2, middleware enable developer control and security check all incoming requests and handle them properly. Sepecificly, guest and auth middleware are used to create authentication and authorization mechanism.

In this movie review system, guests are allowed to register and sign in to leave comments and reviews. Moreover, registered users can not access to administrator control panel, which mean certain type of user can only access certain zone in the system. Therefore, this feature needs at least two essential parts: registration, authentication and authorization part.

Although PHP don't have any standard in authenticate users, Laravel 5 is facilitated with an ready-to-use package for authentication including controllers, routes and views. The main authentication controller is placed at `app/Http/Controllers/Auth/AuthController.php`. The controller use AuthenticatesAndRegistersUsers trait which contains all neccessary methods to handle essential actions such as login, logout.

There are two main type of users in this system: normal users and administrators. Therefore, two middleware classes need to be implemented: `Authenticate` and `AdminAuthenticate`. Basically an administrator is a normal user with admin role, and a normal user is an authenticated guest.

Laravel 5 does not natively support to handle roles and permissions in the system. However, there is a package to support that purpose: “bican/roles”. Before using “bican/roles”, it must be installed via composer as following commands:

```
$ composer require "bican/roles:2.1.*"
$ composer update
```

Figure 18 Install bican/roles package using composer

Besides that, admin role also need to be created, and admin users must be assigned admin role. Firstly, admin role must be generated via a command as in Figure 19. Note that to keep it simple, only the most essential parts of the commands are shown. Moreover, each role in the system need to created separately depends on the requirements. For example, moderator or premium user can be created along with admin and guest role.

```php
<?php
try {
   $adminRole = Role::create([
        'name' => 'Admin',
        'slug' => 'admin',
        'description' => 'Admin role', // optional
        'level' => 1, // optional, set to 1 by default
    ]);
    $this->info('Admin role is created');
} catch(\Exception $ex){
    $this->error($ex->getMessage());
}
```

Figure 19 Create admin role in the system

To simplify the process, instead of creating GUI for making new admin, a make-admin command is made for this purpose. The User model also need to be modified to support authorization. Specifically, `Authenticatable, HasRoleAndPermission, CanResetPassword` trait must be used in the model as illustrated in Figure 20.

```php
<?php
class User extends Model implements AuthenticatableContract,
CanResetPasswordContract {

    use Authenticatable, CanResetPassword, HasRoleAndPermission;

    protected $table = 'users';

    protected $fillable = ['name', 'email', 'password'];

    protected $hidden = ['password', 'remember_token'];
}
```

Figure 20 User model with authentication's traits

```
$email = $this->argument('email');
$user  = User::where('email', '=', $email)->first();
$adminRole = Role::where('name', '=' , 'admin')->first();
$user->attachRole($adminRole);
```

Figure 21 Create a new admin user with admin role

After creating admin role, the previously created admin user must be attach with the new role. The Figure 21 demonstrates how a user can be associated with a role. From now on, there are two types of users in the system: registered users and administrators.

```
<?php
namespace App\Http\Controllers\Auth;
use App\User;
use Validator;
use App\Http\Controllers\Controller;
use Illuminate\Foundation\Auth\ThrottlesLogins;
use Illuminate\Foundation\Auth\AuthenticatesAndRegistersUsers;

class AuthController extends Controller
{
    use AuthenticatesAndRegistersUsers, ThrottlesLogins;

    public function __construct()
    {
        $this->middleware('guest', ['except' => 'getLogout']);
    }

    protected function validator(array $data)
    {
        return Validator::make($data, [
            'name' => 'required|max:255',
            'email' => 'required|email|max:255|unique:users',
            'password' => 'required|confirmed|min:6',
        ]);
    }

    protected function create(array $data)
    {
        return User::create([
            'name' => $data['name'],
            'email' => $data['email'],
            'password' => bcrypt($data['password']),
        ]);
    }
}
```

Figure 22 Authentication controller

The Figure 12 illustrates the general layout design of movie list page. The page comprises two main list: general movie list on the left and newest movie list on the right. Movie poster, movie title and short synopsis are shown in each list. When users click on the movie title, it will redirect them to movie detail page.

The controller for movie list page is pretty simple, it only fetches two movie lists and pass them to the view. Basically, the movie lists are only different in sorting order. Data retrieving and sorting is quite easy with Eloquent as illustrated in Figure 22, one movie collection is sorted by title and the other is sorted by year.

```
// Movie controller
public function movies()
{
    $movies     = $this->movie->all();
    $new_movies = $this->movie->news();

    return view('front.movies',[
        'movies' => $movies,
        'news'   => $new_movies
    ]);
}
//Movie repository
public function all()
{
    return Movie::orderBy('title', asc)->get();
}

public function news()
{
    return Movie::where('year', '>=', 2000)->orderBy('year', 'desc')->limit(5)->get();
}
```

Figure 23 Movie controller and related repository code

As shown in the Figure 13, it describes the appearance of movie detail page. Beside all movie details information such as title, published year, poster, synopsis etc. the page also includes movie review form for registered user and the list of users' reviews.

Before writing main business logic code in controller, all necessary routes must be defined. The Figure 24 illustrates how routes are registered. Basically the desired paths have to be mapped with corresponding controller action. The routes can also be aliased with shorter name for re-used with route helper later. When the responsible action is changed, developers only need to change the “uses” part without refactory all places using alias name of the route.

```
Route::get('/movie/{id}', [
    'as'   => 'front.movie',
    'uses' => '\App\Http\Controllers\Front\Movie@movie'
]);

Route::post('/movie/doReview/{id}', [
    'as'   => 'admin.movie.doReview',
    'uses' => '\App\Http\Controllers\Front\Movie@doReview'
]);
```

Figure 24 Routes for movie detail page and movie review function

The Figure 25 explains how the review form is created with LaravelFormBuilder package. In this form, there are three elements: a textbox for review title, a drop box is initialized with values from 1 to 5 and a submit button.

```
<?php

namespace App\Forms;

use Kris\LaravelFormBuilder\Form;

class ReviewForm extends Form
{
    public function buildForm()
    {
        $this->add('title', 'text', [
            'rules' => 'required|min:5',
        ])->add('rating', 'select', [
            'choices' => [1, 2, 3, 4, 5],
            'empty_value' => '-- select --'
        ])->add('content', 'textarea', [
            'rules' => 'required|min:4',
        ])->add('save', 'submit', [
            'template' => 'elements.form.button',
            'attr' => ['class'=>'btn btn-primary pull-right'],
            'label' => trans('common.save')
        ]);
    }
}
```

Figure 25 Review form code

The Figure 26 llustrate how movie controller can displays the movie detail page as well as handle the review submission from that page. The movie detail page only process input data to display information of related movie and the review form in the bottom of that page. The `doReview` action in the movie controller is responsible for validating and saving (via movie repository class) review to database. After inserting new review to the database, it will redirect the user back to the movie detail page. However, if the submitted data is invalid, the controller will redirect users back to the previous page with extra error messages.

```php
<?php
    // Movie controller
    public function movie(FormBuilder $formBuilder, $id)
    {
        $movie   = $this->movie->find($id);
        $reviews = $this->review->all();

        $form = $formBuilder->create(\App\Forms\ReviewForm::class, [
            'method' => 'POST',
            'class' => 'form-horizontal',
            'url' => route('admin.movie.doReview', ['id' => $id])
        ]);

        return view('front.movie',[
            'movie' => $movie,
            'reviews'=> $reviews,
            'form'  => $form
        ]);
    }

    public function doReview(Request $request, $movie_id)
    {
        if (empty(Auth::user()->id)) {
            return redirect()->back();
        }

        $form = $this->form(\App\Forms\ReviewForm::class);

        if (!$form->isValid()) {
            return redirect()->back()->withErrors($form->getErrors())-
>withInput();
        }

        $this->review->create(Auth::user()->id, $movie_id, $request->all());

        return redirect()->back();
    }
```

Figure 26 Code handles movie details page routes

4 Conclusion

Laravel 5 is an ideal choice for PHP developers that want to create fast prototype as well as large scale project later. The learning curve is also not so steep, because its documentation is very comprehensive and clear. The Laravel community is also big and web developers can easily find answers for theirs questions and resources on the internet.

Laravel 5 is a full stack and rich feature web frameworks, it has everything that web developers may need. From development environment, database migration, MVC structure etc. It is worth to mention that Laravel equipped with `artisan` which is a very handy toolset. The framework also come with its own ORM (Eloquent), a beautiful and simple ActiveRecord implementation. Moreover, Laravel 5 is very easy to test, comparing to Laravel 4. Unfortunately, testing is not demonstrated in this thesis.

Furthermore, it is very to control dependencies and install new packages with composer. Composer will find and install the given packages in composer.json including all dependencies. Besides Laravel itself, there are several good packages that developers can install via composer. They help developers build modern web application faster web application with less effort such as form builder, image processing, or mail sender packages.

The project goal is to demonstrate how to use Laravel 5 in practice, therefore in this thesis all features are simplified. There are still a lot of rooms for improvements in term of functionality and usability. However, Laravel shown that it is a promising framework to study and to spend time more in further study projects as well as working projects. In Laravel version, not only the overall structure is changed but also many other essential features were added such as middleware, contracts and route cache. The testability of the framework is also improved. Laravel now enables developers write controller tests easier. (Laravel official website, Release notes, cited 13.05.2016)

With PHP 7, the overall performance of PHP is improved dramatically. Currently, Laravel 5.2 is the latest version and the project itself was written in PHP 5.x but it is compatible with PHP 7. This is a good thing for all who use PHP in production. Furthermore, Laravel project is very promising in the future with the next major version. In summary, Laravel is outstanding PHP framework and it helps developers easily practice clean code and TDD as well as contribute to the software industry.

REFERENCES

Adam Freeman. 2015. Pro Design Patterns in Swift. USA: Springer Science

Cunningham & Cunningham, Inc. Model View Controller History. Cited 20.10.2015, http://c2.com/cgi/wiki?ModelViewControllerHistory

Cristian Darie. 2006. Beginning PHP and PostgreSQL E-Commerce: From Novice to Professional (Beginning, from Novice to Professional). United States: Apress

History of PHP. Cited 25.10.2015, http://fi2.php.net/manual/en/history.php.php

Laravel - The PHP framework for web artisans. Cited 25.10.2015, http://laravel.com/

Laravel official website. Installation. Cited 25.10.2015, http://laravel.com/docs/5.1

Laravel official website. Artisan Console. Cited 20.11.2015, http://laravel.com/docs/5.1/artisan

Laravel official website. Blade. Cited 01.12.2015, http://laravel.com/docs/5.1/blade

Laravel official website. Eloquent: Getting Started. Cited 05.12.2015, http://laravel.com/docs/5.1/eloquent

Laravel official website. Eloquent: Relationships. Cited 30.05.2016, https://laravel.com/docs/5.1/eloquent-relationships

The Laravel Framework. Packagist: The PHP package repository. Cited 25.10.2015, https://packagist.org/packages/laravel/framework

Laravel official website. Database: Migrations. Cited 05.12.2015, http://laravel.com/docs/5.1/migrations

Laravel official website. HTTP Middleware. Cited 03.01.2016, https://laravel.com/docs/5.1/middleware

Laravel official website. Service Container. Cited 31.05.2016,
https://laravel.com/docs/master/container#contextual-binding

Martin Bean. 2015. Laravel 5 Essentials. United Kingdom: Packt Publishing

Projects using Symfony. Cited 25.10.2015, http://symfony.com/projects

Motion Picture Association of America website, Theatrical Market Statistics 2014. Cited 30.11.2015,
http://www.mpaa.org/wp-content/uploads/2015/03/MPAA-Theatrical-Market-Statistics-2014.pdf

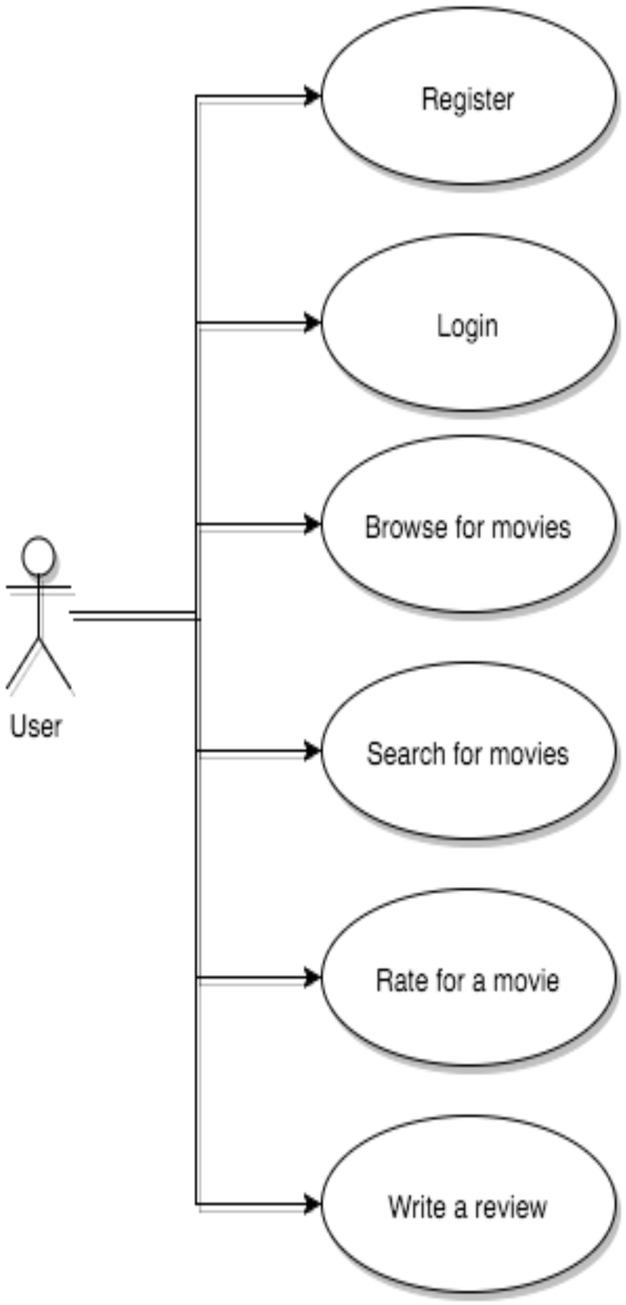
The Best PHP Framework for 2015: SitePoint Survey Results. Cited 25.10.2015,
http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/

Wikipedia. Laravel. Cited 20.11.2015, https://en.wikipedia.org/wiki/Laravel

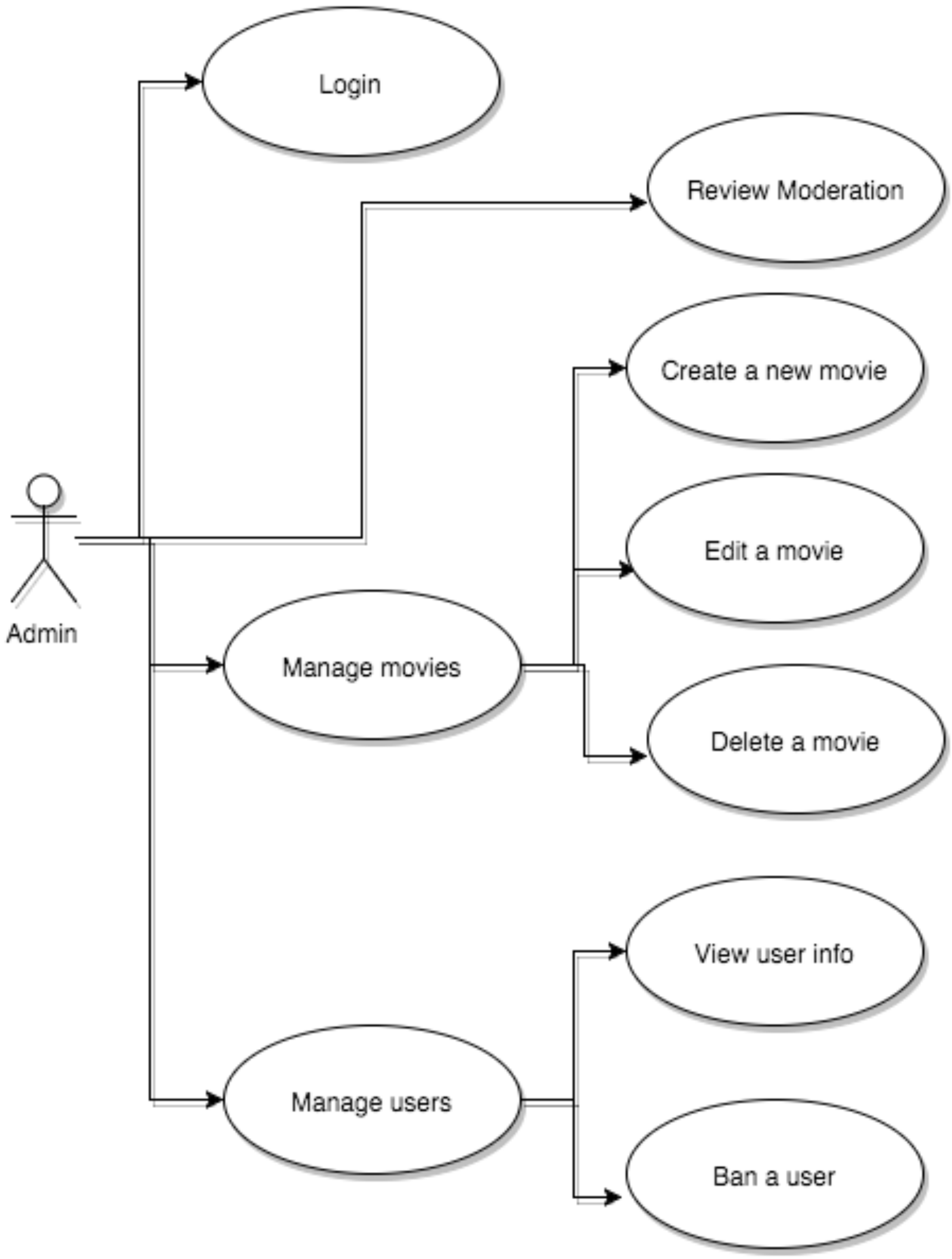
APPENDICES

Use cases diagram

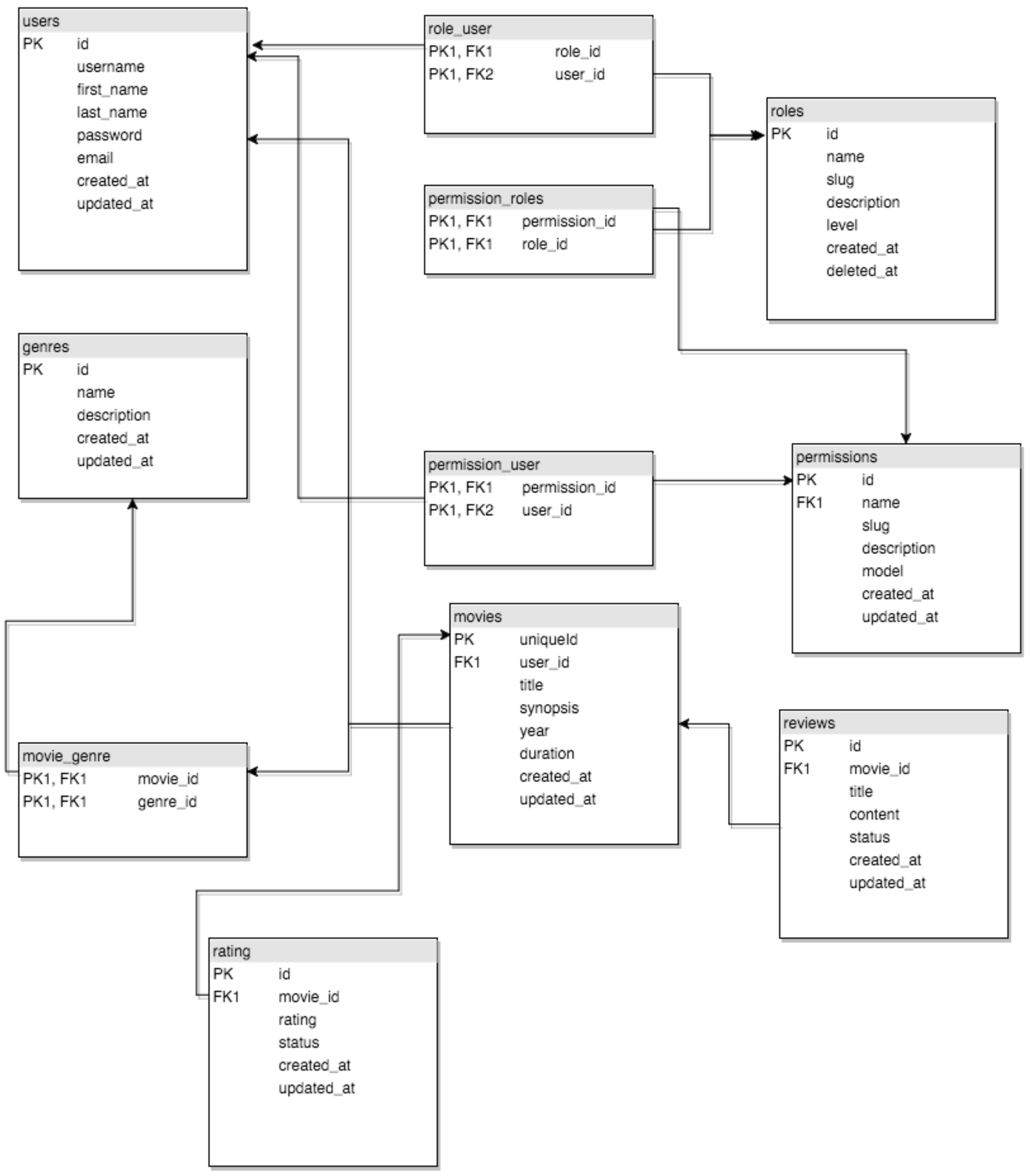
- User



- Admin


Login
Review Moderation
Create a new movie
Edit a movie
Admin
Manage movies
Delete a movie
View user info
Manage users
Ban a user

Entity-relationship diagram


users
PK id
username
first_name
last_name
password
email
created_at
updated_at
role_user
PK1, FK1 role_id
PK1, FK2 user_id
roles
PK id
name
slug
description
level
created_at
deleted_at
permission_roles
PK1, FK1 permission_id
PK1, FK1 role_id
genres
PK id
name
description
created_at
updated_at
permission_user
PK1, FK1 permission_id
PK1, FK2 user_id
permissions
PK id
FK1 name
slug
description
model
created_at
updated_at
movies
PK uniqueId
FK1 user_id
title
synopsis
year
duration
created_at
updated_at
reviews
PK id
FK1 movie_id
title
content
status
created_at
updated_at
movie_genre
PK1, FK1 movie_id
PK1, FK1 genre_id
rating
PK id
FK1 movie_id
rating
status
created_at
updated_at