

Bachelor's thesis

Information Technology

2019

Md Sakib Hossain

# WEB APPLICATION DEVELOPMENT WITH LARAVEL FRAMEWORK



BACHELOR'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology

2019 | 37

Md Sakib Hossain

## WEB APPLICATION DEVELOPMENT WITH LARAVEL FRAMEWORK

The primary objective of this thesis was to implement a movie streaming application with the Laravel framework and cover the installation and configuration of Laravel in a cloud environment.

This thesis covers essential features for creating a web application from Laravel framework such as routing, middleware, package management, migration; moreover, it focuses on developing a secure authentication system to restrict unwanted access to media assets and user role management for application-specific asset management.

The finalized application operates in the cloud environment with database management tool; further, the application consists of a browsable list of movies, movie streaming page, admin panel for application-specific asset management.

KEYWORDS:

PHP, Laravel, MVC

# CONTENT

<b>LIST OF ABBREVIATIONS (OR) SYMBOLS</b>	<b>5</b>
<b>1 INTRODUCTION</b>	<b>6</b>
<b>2 LARAVEL FRAMEWORK AS A WEB DEVELOPMENT TOOL</b>	<b>7</b>
2.1 Implementation of MVC design pattern in Laravel framework	7
2.2 Laravel router for URL mapping	8
2.3 Middleware	8
2.4 Package management with composer	9
2.5 Artisan	10
<b>3 INSTALLATION AND CONFIGURATION OF LARAVEL FRAMEWORK</b>	<b>11</b>
3.1 System requirements of Laravel framework	11
3.2 The integrated development environment and hosting support by Cloud9	11
3.3 The installation process of Laravel framework in Cloud9 IDE	12
3.4 Database creation with graphical user interface and database configuration	14
3.5 Laravel application architecture	16
<b>4 BUILDING THE APPLICATION WITH LARAVEL FRAMEWORK</b>	<b>19</b>
4.1 The goal	19
4.2 Database table creation with migration feature of Laravel	22
4.3 Mapping the URL endpoints of the application with Laravel routes	24
4.4 User registration and role management with authentication and middleware	24
4.4.1 Enabling basic authentication of Laravel in the application	24
4.4.2 User role management	26
4.5 Controller and request	30
4.5.1 Movie upload form validation with request feature of Laravel	30
4.5.2 CRUD operations for movies with controller	31
<b>5 CONCLUSION</b>	<b>33</b>
<b>REFERENCES</b>	<b>34</b>

## FIGURES

- Figure 1. Simplest form of a Laravel route.
- Figure 2. Available routing methods for six HTTP verbs.
- Figure 3. Composer repository for PHP packages.
- Figure 4. Artisan command to create a crud skeleton in the controller.
- Figure 5. Workspace creation form in Cloud9.
- Figure 6. Available prebuilt configurations in Cloud9 environment.
- Figure 7. Command to install Laravel through the composer.
- Figure 8. Successful installation of Laravel in Cloud9.
- Figure 9. Successful creation of MySQL database in Cloud9.
- Figure 10. Successful installation of PHPmyadmin in Cloud9.
- Figure 11. Database credentials set up in environment file.
- Figure 12. Folder structure of freshly installed Laravel framework.
- Figure 13. The homepage of movie streaming site.
- Figure 14. Streaming page for the individual movie.
- Figure 15. Form to add a movie on the site.
- Figure 16. Page to update or delete movies.
- Figure 17. Database tables for the movie streaming application.
- Figure 18. Migration file for movies.
- Figure 19. Registration page of the application.
- Figure 20. Migration file for roles table.
- Figure 21. Implementation of many to many relations between users and roles table.
- Figure 22. Code to assign roles to application users.
- Figure 23. Many to many relationships between User and Roles eloquent model.
- Figure 24. Auth controller method to create the user with a role.
- Figure 25. Middleware to restrict HTTP request in “admin” only routes.
- Figure 26. Restricted routes for admin users.
- Figure 27. Form submit request validation.
- Figure 28. Controller methods to display, store and create movies.
- Figure 29. Controller methods to update, delete and listing of movies.

## TABLES

- Table 1. Description of the folder structure of a Laravel application.
- Table 2. The URL endpoints of the application.

## LIST OF ABBREVIATIONS (OR) SYMBOLS

API	Application programming interface
CRUD	Create, read, update, delete
CSS	Cascading style sheets
HTML	Hypertext markup language
HTTP	Hypertext transfer protocol
LAMP	Linux Apache Mysql Php
MVC	Model view controller
PHP	Hypertext preprocessor
SQL	Structured query language
URI	Uniform resource identifier

# 1 INTRODUCTION

The traditional methodology of web developing is not enough for the market's demands nowadays. Therefore, there are frameworks that make high quality large-scale project development convenient and faster. For example, a framework usually has built-in solutions for common tasks such as user authentication, built-in user interface, predetermined security. By using frameworks, programmers can focus on the actual code that is required for a project so that they do not need to write repetitive code. Frameworks like Angular, React, Laravel, provide developers with boilerplate code which reduces the development time and improves the quality of code. However, some frameworks are more suitable than others for different development projects. As an example, the full-stack development environment of the Laravel framework is more suitable for the development of the movie streaming application than other frameworks as the application needs both the frontend and backend support to be operable. In addition to that, the built-in user authentication, available online documentation covering the full-stack development process are the main reasons to choose the Laravel framework for the development of the movie streaming application for the thesis over other full-stack frameworks.

The main objective of this thesis is to analyze the main features of the Laravel application and develop a movie streaming application with the framework. The development of the application implements the features analyzed in the thesis. The aim of the thesis determines the division of the thesis in the following parts:

Chapter 1 describes the main objectives and provides information about the chapters of the thesis

Chapter 2 describes the theoretical background, covering the main features, underlying software design principle and folder structure of the Laravel framework.

Chapter 3 describes the process of installation and configuration of the Laravel framework in the cloud environment. It also covers the installation of database, application architecture and database management tool named 'phpmyadmin'.

Chapter 4 describes the development process of the movie streaming application.

Chapter 5 summarizes the result.

## 2 LARAVEL FRAMEWORK AS A WEB DEVELOPMENT TOOL

### 2.1 Implementation of MVC design pattern in Laravel framework

Laravel, a PHP (Hypertext preprocessor) framework, implements the MVC (Model View Controller) design pattern to separate the application logic from the presentation layer. According to the MVC concept, Laravel separates the application logic into three separate components (Saunier 2014, 13). The responsibilities of these three components in the Laravel framework are as follows:

- Responsibilities of the Model in the Laravel framework

The Model is a class which represents a database table and the relationship between other models in the application. It is possible to define SQL (Structured query language) relationships like “one to many”, “one to one”, “many to many” inside a model class (Saunier 2014, 14).

- Responsibilities of the View in the Laravel framework

The View visualizes the data received from the model through controller or route. For a simple application, application logic to provide data to view component is used inside routes instead of controllers (Saunier 2014, 14).

- Responsibilities of the Controller in the Laravel framework

There are two types of controller named “standard” and “resource” controller to handle incoming HTTP (Hypertext transfer protocol) requests and send the appropriate response. The standard controller handles incoming request from the view component. On the other hand, the resource controller handles API (Application programming interface) centric HTTP requests (Saunier 2014, 14).

## 2.2 Laravel router for URL mapping

Laravel has a built-in URL rewrite engine which can make a URL short or search engine friendly. The simplest Laravel route accepts a URI (Uniform resource identifier) and closure as shown in Figure 1 (Stauffer 2016, 25) .

```
Route::get('foo', function () {  
    return 'Hello World';  
});
```

Figure 1. Simplest form of a Laravel route (Stauffer 2016, 25).

The Router allows the developer to create a route which responds to an HTTP verb. The Laravel router can process six different kinds of HTTP verbs named get, post, put, patch, delete, options (HTTP Routing 2019). The implementation of these HTTP verbs in the simplest form is shown in Figure 2.

```
Route::get($uri, $callback);  
Route::post($uri, $callback);  
Route::put($uri, $callback);  
Route::patch($uri, $callback);  
Route::delete($uri, $callback);  
Route::options($uri, $callback);
```

Figure 2. Available routing methods for six HTTP verbs (HTTP Routing, 2019).

Routes are assignable with middleware (discussed in Section 2.3) which has features such as CSRF protection. HTML (Hypertext markup language) forms which point to post, put or delete routes should have a CSRF protection field. (Gilmore 2015, 147:176)

## 2.3 Middleware

The Laravel specific feature named middleware filters the HTTP request and response of an application. The HTTP requests are filtered before they reach application logic through middleware. As an example, middleware can restrict unauthenticated HTTP requests for accessing media files of Laravel application. On the other hand, the HTTP



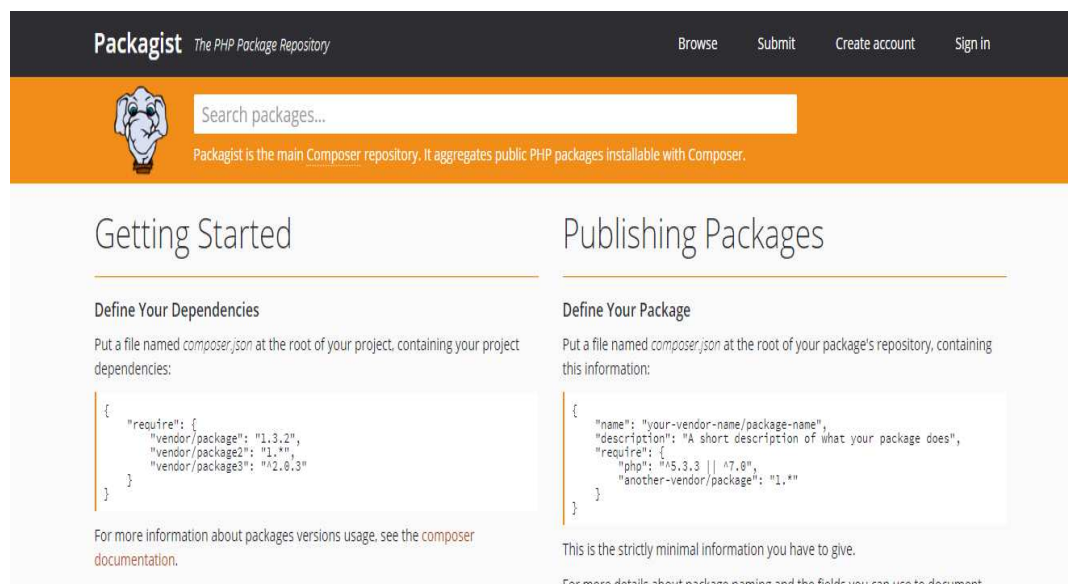
response is also filtered by middleware. As an example, a view counter of a media file can be programmed based on the middleware response (Rees 2014, 176).

The middleware which filters HTTP requests is named 'BeforeMiddleware'. On the other hand, the middleware which is named 'AfterMiddleware' filters HTTP responses (HTTP Middleware, 2019).

## 2.4 Package management with Composer

The Composer manages the dependency of PHP projects. It can manage the dependency up to the nth level. For example, if in a project a library is dependent on three other libraries then composer will install all three libraries for it (Rees 2016, 2-4).

The convenience of the Laravel framework is composer readiness. The framework itself is a mixture of packages maintained by the composer. The Laravel framework itself can be installed and updated through the Composer (Gilmore 2015, 16).



The screenshot shows the Packagist website interface. At the top, there is a navigation bar with the Packagist logo and the tagline "The PHP Package Repository". Navigation links include "Browse", "Submit", "Create account", and "Sign in". Below the navigation bar is a search bar with the placeholder text "Search packages...". A sub-header states: "Packagist is the main Composer repository. It aggregates public PHP packages installable with Composer."

The main content area is divided into two columns:

- Getting Started**
  - Define Your Dependencies**

Put a file named `composer.json` at the root of your project, containing your project dependencies:

```
{
  "require": {
    "vendor/package": "1.3.2",
    "vendor/package2": "1.*",
    "vendor/package3": "2.0.3"
  }
}
```

For more information about packages versions usage, see the [composer documentation](#).
- Publishing Packages**
  - Define Your Package**

Put a file named `composer.json` at the root of your package's repository, containing this information:

```
{
  "name": "your-vendor-name/package-name",
  "description": "A short description of what your package does",
  "require": {
    "php": "5.3.3 || ^7.0",
    "another-vendor/package": "1.*"
  }
}
```

This is the strictly minimal information you have to give.

For more details about package naming and the fields you can use to document

Figure 3: Composer repository for PHP packages (The php package repository, 2019).

The central repository of the composer is called "Packagist". This repository is accessible from a browser ("<https://packagist.org/>") or from the composer. One can search for packages through the website (Figure 4). Each package contains statistical information of popularity, user reviews. For this reason, one can determine the usefulness, feasibility of a package before installation (Rees 2016, 27).

## 2.5 Artisan

Artisan is a command line tool of Laravel framework to automate common and repetitive tasks. This automation saves development time and reduces the complexity of a Laravel project. For example, Artisan has built-in commands for user authentication, database migration file generation, and a frontend asset minifier (Saunier 2014, 73).

Artisan commands are compatible with Laravel specific features. Features such as an event, middleware and policy are generated through the Artisan command line environment (Artisan console, 2019). However, Artisan commands not only generate Laravel specific features but are also extendable for project specific needs such as transferring pending mails to the recipient is obtainable through extending Artisan commands. (Gilmore 2015, 15).

```
sakibwebworm:~/workspace/5.2.* $ php artisan make:controller MovieController --resource  
Controller created successfully.  
sakibwebworm:~/workspace/5.2.* $ █
```

Figure 4: Artisan command to create a CRUD (Create, read, update, delete) skeleton in the controller.

Along with the generation of files, Artisan also produces skeleton code and manages database through schema files (Saunier, 77-78). The Artisan command creates database through schema files while it can also drop database to fix errors. On the other hand, Artisan commands can generate boilerplate for CRUD functionality. For example, this code in figure 4 will create a CRUD boilerplate in the controller named "MovieController" (Laravel Artisan, 2017).

## **3 INSTALLATION AND CONFIGURATION OF LARAVEL FRAMEWORK**

### 3.1 System requirements of Laravel framework

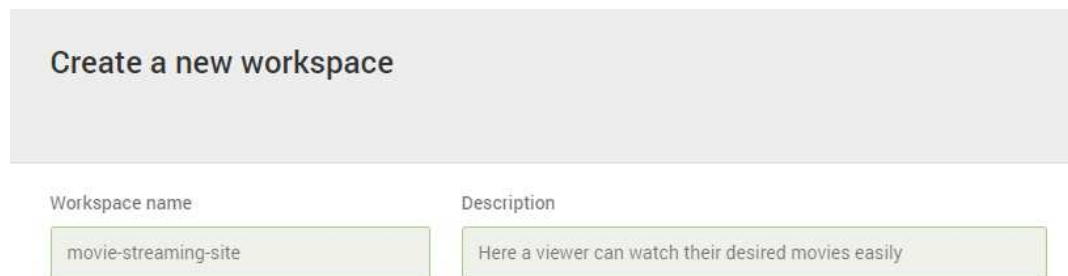
The system requirements of a specific Laravel framework should be satisfied to install the framework in a server. These system requirements are indifferent to both local and online server for a specific framework. However, system requirements vary with the version of a framework. For this thesis, the 5.2 version of Laravel framework has been used. The system requirements for Laravel version 5.2 are listed below.

- PHP=>5.5.9
- Openssl PHP extension
- Pdo PHP extension
- Mbstring PHP extension
- Tokenizer PHP extension (Installation, 2019)

### 3.2 The integrated development environment and hosting support by Cloudnine platform

In this work, the Cloud9 platform has been used to setup an integrated development environment and hosting the application. Cloud9 supports major programming languages, such as PHP: the language behind the Laravel framework. Additionally, Cloud9 provides prebuilt server configurations for programming languages to reduce the time for setting up a project. For this thesis work, the prebuilt configuration of a LAMP (Linux Apache MySQL PHP) server has been utilized from Cloud9. On the other hand, Cloud9 also serves as an integrated development environment for application development. This IDE of cloud9 is accessible within a browser which creates the opportunity to develop and debug a web application faster than the local environment.

### 3.3 The installation process of the Laravel framework in Cloud9 IDE



**Create a new workspace**

Workspace name	Description
movie-streaming-site	Here a viewer can watch their desired movies easily

Figure 5: Workspace creation form in Cloud9.

In Cloud9, the first step to install the Laravel framework is to create a new workspace. The workspace should contain the name and description of the project as shown in figure 5. After filling up the name and description for the workspace, Cloud9 provides a new page to select prebuilt configurations (figure 6). Among these prebuilt solutions, LAMP server can be chosen to install Laravel. However, the selection of a configuration is not necessary to create the workspace as there is an option to create configurations manually after creating the workspace.

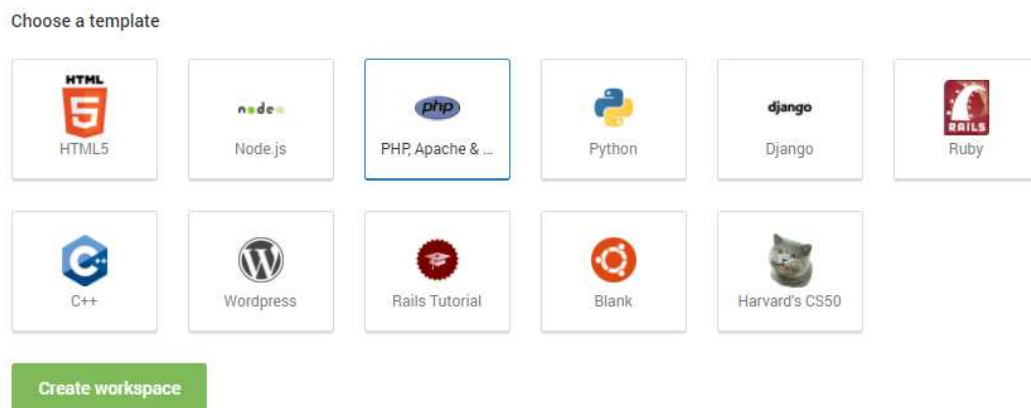


Figure 6: Available prebuilt configurations in Cloud9 environment.

A workspace in Cloud9 with LAMP configuration meet the server requirements to install Laravel framework. In this work, prebuilt LAMP configuration was chosen to shorten the installation process of Laravel. From the workspace command line, one can install the

Laravel framework by writing a single line of command (Figure 7).

```
composer create-project --prefer-dist laravel/laravel blog "5.2.*"
```

Figure 7. Command to install Laravel through the composer.



Figure 8. Successful installation of Laravel in Cloud9.

The next step to run a Laravel application is to generate an application key. The application key is generated through the command line by typing the command “php artisan key: generate” (Figure 8). After execution of the command, Laravel application becomes viewable in the browser.

### 3.4 Database creation with a graphical user interface and database configuration for Laravel application

In Cloud9 IDE workspace, a MySQL database is created with a single line of command in the workspace terminal. The command “\$ mysql-ctl start” will install a MySQL database in the application. Also, the command will generate the username and database name for the MySQL database (Figure 9).



```
bash - "sakibweb" x Immediate x +
sakibwebworm:~/workspace $ mysql-ctl start
Installing MySQL
* Stopping MySQL database server mysqld
...done.
* Starting MySQL database server mysqld
...done.
* Checking for tables which need an upgrade, are corrupt or were
not closed cleanly.

MySQL 5.5 database added. Please make note of these credentials:

    Root User: sakibwebworm
    Database Name: c9

* Starting MySQL database server mysqld
...done.
```

Figure 9. Successful creation of MySQL database in Cloud9.

After installation of MySQL database, a graphical user interface named “Phpmyadmin” can be installed to aid the development process. In Cloud9 IDE, the command “Phpmyadmin-ctl install” installs the PHPMyadmin interface. The interface (Figure 10) provides common database operations such as the creation of databases, tables and running SQL queries within a web browser.

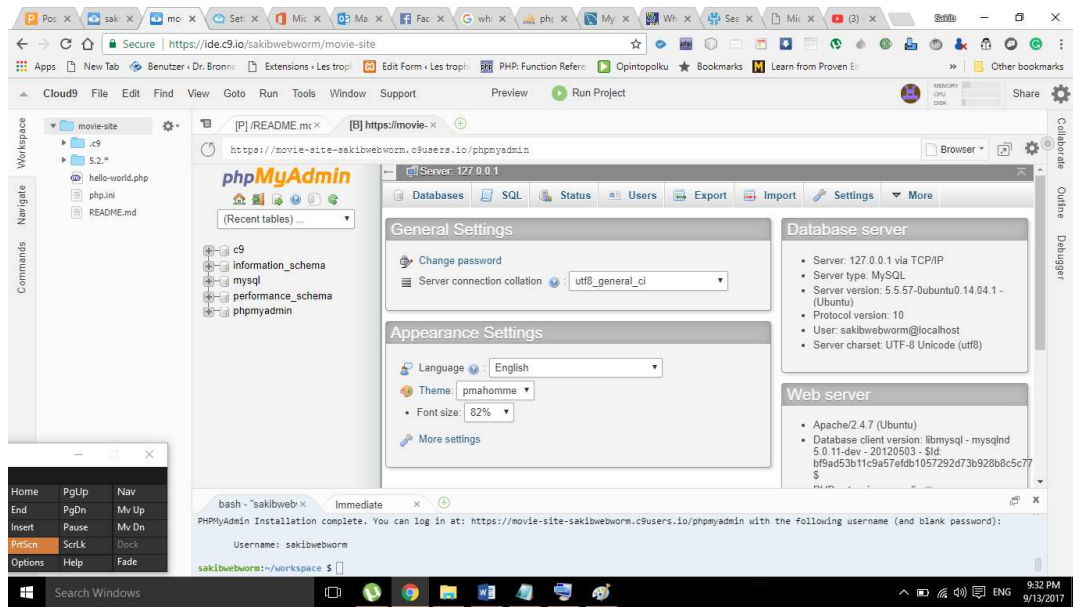


Figure 10. Successful installation of PHPmyadmin in Cloud9.

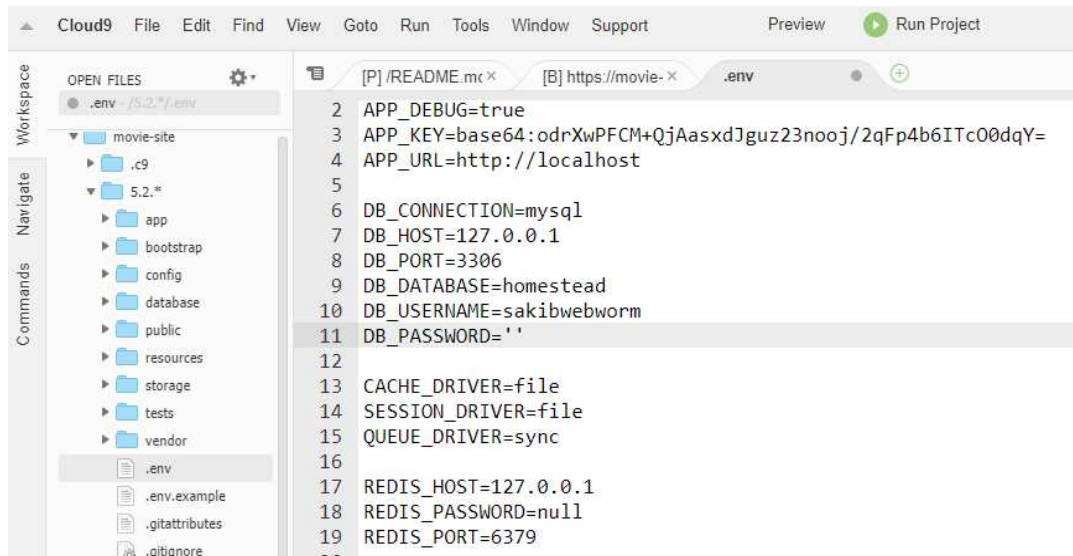


Figure 11. Database credentials set up in environment file.

Environment file of a Laravel application configures the connection to a database system. The configuration file should contain the appropriate username and password to connect to a database (Figure 11).

### 3.5 Laravel application architecture

The structure of a Laravel application provides a starting point for a small or large project. There is a possibility to organize the application structure according to the need. Laravel has almost no restriction on the location if classes can be autoloaded by the composer. (Laravel Structure, 2019)

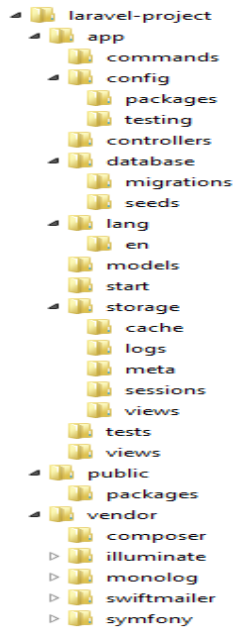


Figure 12. Folder structure of freshly installed Laravel framework.

A freshly installed Laravel framework contains the folder structure according to Figure 12. In the root directory of a Laravel application, there are several folders and files. The responsibilities of these folders are explained in Table 1.



Table 1. Description of the folder structure of a Laravel application.

App	Contains core code of the application
Bootstrap	Contains few files to bootstrap autoloading configuration and a directory named cache to preserve framework generated files
Config	Contains all configuration file for the application
Database	Contains database migration and seeds file
public	Contains the assests (CSS (Cascading style sheets), javascript files) for the application
Resources	Contains the view folder and preprocessor files (such as sass, coffescript) for the application
Storage	Contains the Laravel template engine generator, file based sessions, caches and framework generated files.
Logs	Contains application log file
Tests	Contains automated testing tools
Vendor	Contains composer dependencies

Table 1 describes the folder structure of a freshly installed Laravel application. The app, database, config, public, tests, resources folders are used by developers while an application is in the development phase. On the other hand, bootstrap, logs, and vendor folders are used by composer and framework.

## 4 BUILDING THE APPLICATION WITH LARAVEL FRAMEWORK

### 4.1 The goal and visualization of the application

The features of the application will be built using Laravel components. The components of Laravel will be used for the application are blade, migration, authentication, middleware, route, controller and request. The features of the application are listed below.

1. User registration system
2. User role system
3. Movie management system and
4. CRUD operations on movies.

Visually, the application is divided into four individual pages. The functionality of these pages is listed below.

1. “Homepage” page



Figure 13. The homepage of movie streaming site.

The “Homepage” (Figure 13) contains a list of posters of movies. Each individual poster contains a unique URL, which leads a visitor to stream his/her choice of movie.

## 2. "Streaming" page

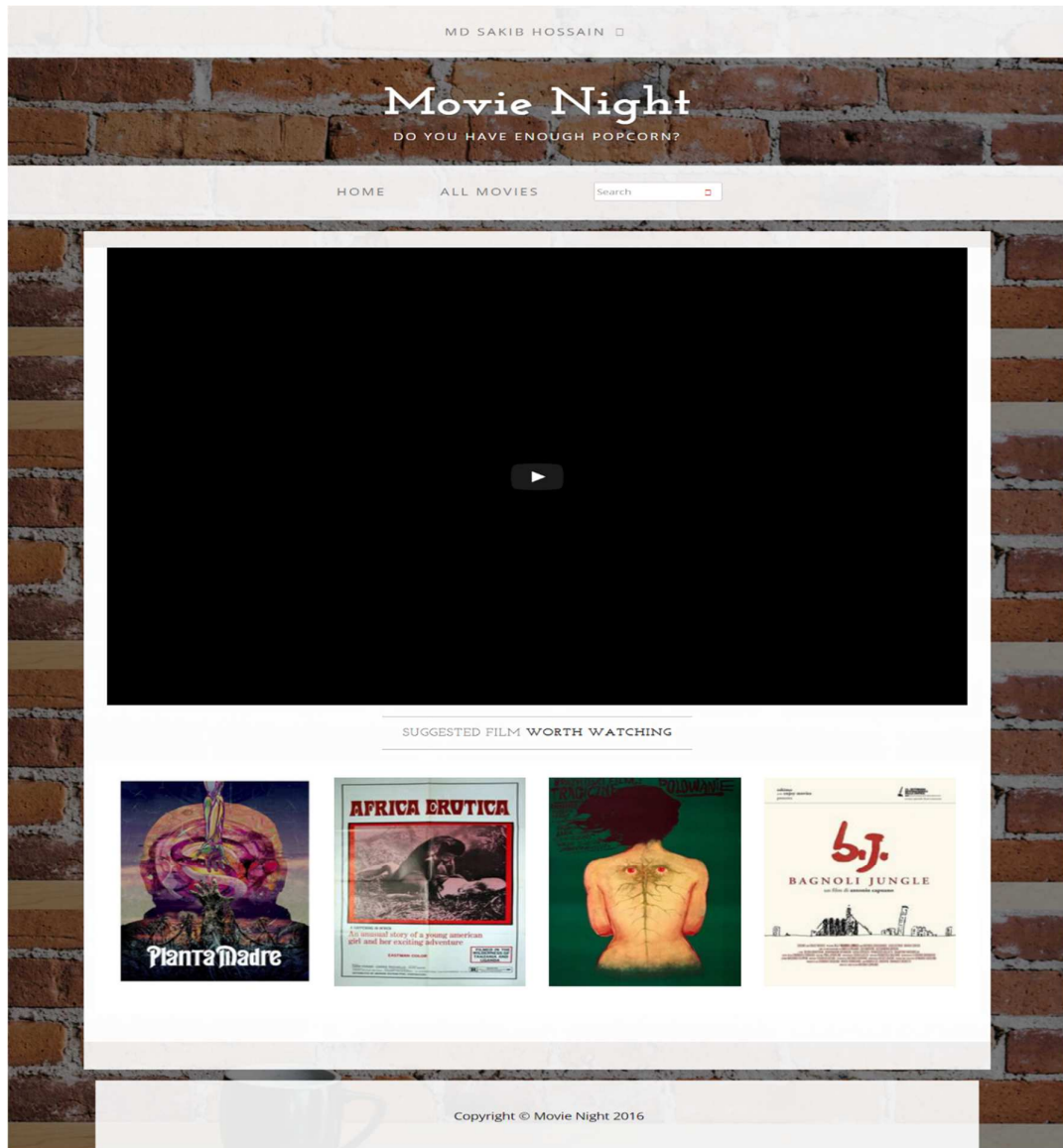


Figure 14. Streaming page for the individual movie.

The "Streaming" page contains a video player and a list of suggested movies (Figure 14).

## 3. "Movie upload" page

Movie Night Md Sakib Hossain | Logout

**Dashboard**

- User
- Slider
- Movie

## Add movie here

original\_title

overview

Poster  No file chosen

video

Figure 15. Form to add a movie on the site.

“Movie upload” page (Figure 15) contains a form to upload movies. This page is restricted to admin users of the application.

#### 4. “Update/delete” page

Movie Night Md Sakib Hossain | Logout

**Dashboard**

- User
- Slider
- Movie

## List of all the movies(Update/delete)

Bordered Table

Title	Created	Update	Delete
Pianta Madre	Jun 14, 2017	<input type="button" value="Update"/>	<input type="button" value="Submit"/>
Jungle	Jun 14, 2017	<input type="button" value="Update"/>	<input type="button" value="Submit"/>
Jungle Erotic	Jun 14, 2017	<input type="button" value="Update"/>	<input type="button" value="Submit"/>
Selva Trágica	Jun 14, 2017	<input type="button" value="Update"/>	<input type="button" value="Submit"/>
Bagnoli Jungle	Jun 14, 2017	<input type="button" value="Update"/>	<input type="button" value="Submit"/>
A Dog's Purpose	Jun 14, 2017	<input type="button" value="Update"/>	<input type="button" value="Submit"/>
台北·叢林	Jun 14, 2017	<input type="button" value="Update"/>	<input type="button" value="Submit"/>
Bandidas	Jun 14, 2017	<input type="button" value="Update"/>	<input type="button" value="Submit"/>

Figure 16. Page to update or delete movies.

“Update/delete” page (Figure 16) contains a list of movies with corresponding buttons to update information of a movie which is already stored or to delete a movie from the

database. On the movie upload page (Figure 15), only the admin users can update or delete a movie.

#### 4.2 Database tables creation with migration feature of Laravel

According to the goal of the project, the database structure of the application is a combination of prebuilt migrations files shipped with Laravel along with custom made migrations. The prebuilt migrations of Laravel are “users”, “password\_resets” and “migrations”. Laravel utilizes these three migration files to provide a basic authentication and user registration functionality out of the box. On the other hand, the application also needs three additional migration for movie management, browsable list of movies and user role management. These three additional migration files are “movies”, “roles” and “role\_user”.

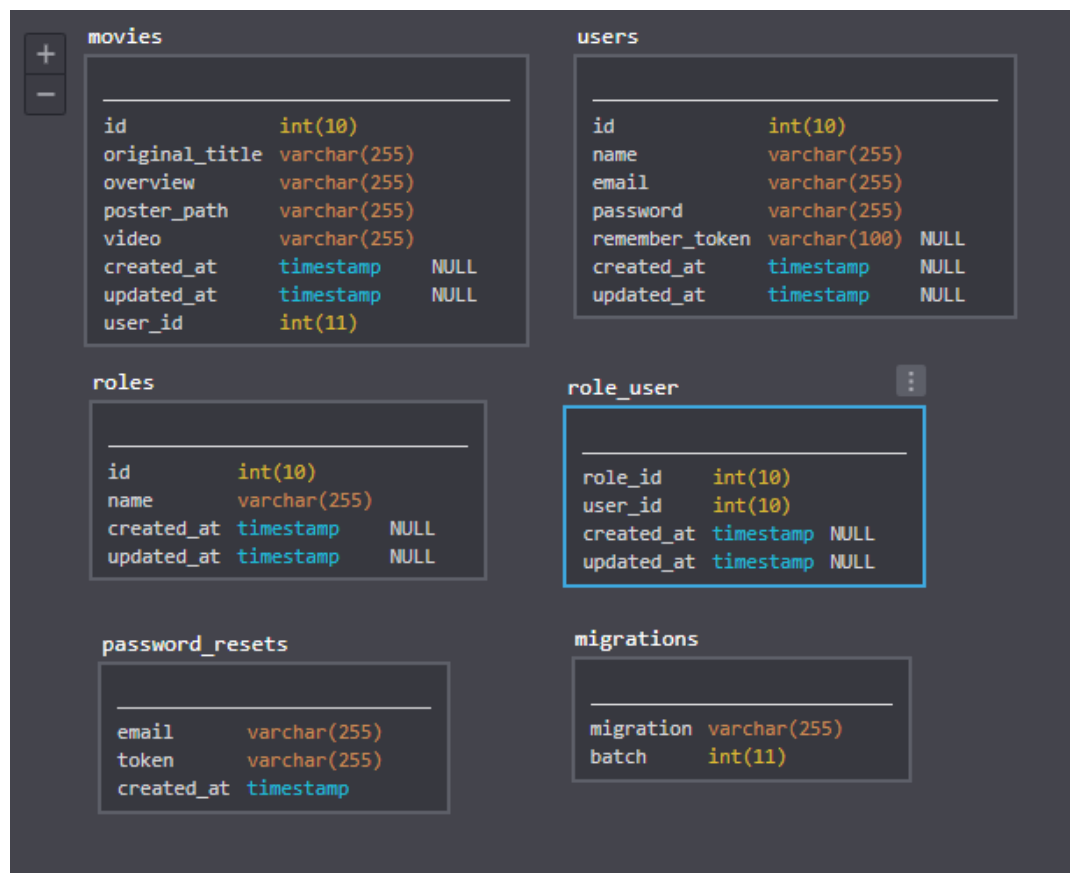


Table Name	Column Name	Data Type	Nullable
movies	id	int(10)	
	original_title	varchar(255)	
	overview	varchar(255)	
	poster_path	varchar(255)	
	video	varchar(255)	
	created_at	timestamp	NULL
	updated_at	timestamp	NULL
	user_id	int(11)	
users	id	int(10)	
	name	varchar(255)	
	email	varchar(255)	
	password	varchar(255)	
	remember_token	varchar(100)	NULL
	created_at	timestamp	NULL
roles	id	int(10)	
	name	varchar(255)	
	created_at	timestamp	NULL
	updated_at	timestamp	NULL
password_resets	email	varchar(255)	
	token	varchar(255)	
	created_at	timestamp	
migrations	migration	varchar(255)	
	batch	int(11)	
role_user	role_id	int(10)	
	user_id	int(10)	
	created_at	timestamp	NULL
	updated_at	timestamp	NULL

Figure 17. Database tables for the movie streaming application.

The attributes of an individual table must be written inside a migration file. Tables with attributes for this application (Illustrated in Figure 17) was generated through Artisan command. The Artisan command which generates migration file is “php artisan make:migration”. However, the command is only used to generate custom made migration file for a table, as the prebuilt migration files are already shipped with Laravel.

```
<?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->string('original_title');
            $table->string('overview');
            $table->string('poster_path');
            $table->string('video');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::drop('movies');
    }
}
```

Figure 18. Migration file for movies.

Artisan command creates the database through the “up” method of a migration file. The up method should contain the attributes of the tables. Figure 18 illustrates the attributes of “movies” table as a visual sample of a migration file. The command “php artisan migrate” runs all the “up” method of migration files, and thus create the application.

### 4.3 Mapping the URL endpoints of the application with Laravel routes

After database creation, the URL endpoints for the application need to be defined. The URL endpoint of the application is shown in the table below.

Table 2. The URL endpoints of the application.

Method	Route	Description
GET	/	Index page
GET	/movies/{id}	Movie streaming page
POST	/admin/addmovie	Movie Upload page
POST	/admin/movies/{id}/edit	Movie edit page
DELETE	/admin/movies/{id}/delete	Route to delete a movie
GET	/login	Login page
POST	/register	Register page

Table 2 shows the HTTP routes for the web application. The routes except login and register would be linked with an individual controller for development of CRUD system for movie management. The login and register route of the application is generated through Laravel authentication system.

### 4.4 User registration and role management with authentication, middleware feature of Laravel

#### 4.4.1 Enabling basic authentication of Laravel in the application

As mentioned earlier in Section 4.2, Laravel ships with the basic user authentication system. This authentication system has user registration and login functionality. Also, the prebuilt authentication system provides password recovery option for a registered user.



The screenshot shows a web application header with 'Movie Night' on the left and 'Login Register' on the right. Below the header is a 'Register' form. The form contains five input fields: 'Name', 'Code', 'E-Mail Address', 'Password', and 'Confirm Password'. A blue 'Register' button with a user icon is positioned below the 'Confirm Password' field.

Figure 19. Registration page of the application.

Along with authentication system, Laravel also generates view files for login, register (Figure 19) and password page. These pages are extensible to fit the requirement of a project. In this project, code input box (Figure 19) was used in the registration form to identify the role of a user.

#### 4.4.2 User role management

The movie streaming application has two types of the user role. The first type of users is a normal user who can only visit a browsable list of movies and movie streams. On the other hand, the second type of user is admin user who can perform CRUD tasks on movies. However, an admin user also inherits the functionality of a normal user.

The implementation of user roles needs a “many to many” relationship between “users” and “roles” table. This relationship will let a user have multiple roles while an individual role can belong to multiple users. As an example, admin users of the movie streaming application have both admin and normal user’s role.

```

class CreateRolesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('roles', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name')->unique();
            $table->timestamps();
        });
        //insert roles in table if the table is empty
        DB::table('roles')->insert(
            array(
                ['id' => 1,
                 'name' => 'users'],
                ['id' => 2,
                 'name' => 'admin']
            )
        );
    }
}

```

Figure 20. Migration file for roles table.

The two user roles named “admin” and “users” are manually inserted through the migration file. The migration file for ‘roles’ table holds the insertion command for these two roles (Figure 20).

```

class CreateRoleUserPivotTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('role_user', function (Blueprint $table) {
            $table->integer('role_id')->unsigned()->index();
            $table->foreign('role_id')->references('id')->on('roles')->onDelete('cascade');
            $table->integer('user_id')->unsigned()->index();
            $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
            $table->primary(['role_id', 'user_id']);
            $table->timestamps();
        });
    }
}

```

Figure 21. Implementation of many to many relations between users and roles table.

The “role\_user” shown in Figure 21 implements the “many to many” relationship between “users” and “roles” table. The table stores the unique id of a user and the associated role.

```

/**
 * Assign admin role if the code is correct otherwise assign user role
 */

public function assignroles($code)
{
    if ($code=='ckpp345s') {
        $this->roles()->attach(2);
    } else {
        $this->roles()->attach(1);
    }
}
}

```

Figure 22. Code to assign roles to application users.

```

/**
 * Relationship of user and role
 *
 * @return $this->belongsToMany('App\Role')->withTimestamps()
 */

public function roles()
{
    return $this->belongsToMany('App\Role')->withTimestamps();
}

```

Figure 23. Many to many relationships between User and Roles eloquent model.

After the creation of the database, the “many to many” relationship and custom code for role assignment must be declared in the “user” model. The method “roles” as illustrated in Figure 23 creates a many to many relationships between eloquent model “User” and “Role”. On the other hand, the method “assignroles” (Figure 22) is programmed to assign a numeric value to the method “roles” depending on the conditional statement. For the

project, in the registration form, admin role is assigned if a user submitted field named “code” matches with this string “ckpp345s”.

```
protected function create(array $data)
{
    //$user=\App\User;
    $user = User::create(
        [
            'name' => $data['name'],
            'email' => $data['email'],
            'password' => bcrypt($data['password']),
        ]
    );
    $user->assignroles($data['code']);
    return $user;
}}
```

Figure 24. AUTH controller method to create the user with a role.

After the declaration of the relationship of “Role” in “User” model, the next step to run the “assignroles” function in “User” model (Figure 22) in the “registerController” of Laravel.

The method “create” in “AuthController” file handles user registration. It is possible to use Eloquent model-specific methods any controller. In the figure, the method “assignroles” of model “User” assign roles to registered users.

Router and middleware, these two features have been discussed in Sections 2.2 and 2.3, respectively. In the project, based on the user role, such as the “user” role will be restricted to access to certain routes. However, it is necessary to set up middleware to gain this kind of restriction.

```

class AdminMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request http request
     * handle the request
     * @param \Closure $next move forward
     *
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        if (Auth::check() && Auth::user()->hasRole('users')) {
            return redirect('home');
        }

        return $next($request);
    }
}

```

Figure 25. Middleware to restrict HTTP request in “admin” only routes.

Figure 25 illustrates the middleware to handle HTTP request based on the user’s role. If the user has role “admin”, it handles the request. Otherwise, it redirects the user to the front page of the application.

```

Route::group(array('prefix' => 'admin', 'middleware' => 'auth'), function () {
Route::group(
    ['middleware' => 'App\Http\Middleware\AdminMiddleware'], function () {
        Route::get('add_movie', 'MoviesController@create');
        Route::post('store_movie', 'MoviesController@store');
        Route::get('movies/{id}/edit', 'MoviesController@edit');
        Route::put('movies/{id}/update', 'MoviesController@update');
    }
}

```

Figure 26. Restricted routes for admin users.

Figure 26 illustrates the routes which can only be accessed by admin users. In Laravel, middleware can be assigned to single or group of routes. In this project, the path to admin routes has been restricted by using middleware “auth” and “admin”, respectively.

## 4.5 Controller and request

### 4.5.1 Movie upload form validation with Request feature of Laravel

The request feature of Laravel is used to restrict unexpected access and data validation in the “movies” table of the application. The “authorize” method in Figure 27 restricts normal users to upload movies. On the other hand, the “rules” method (Figure 27) validate the submitted data by “admin” users.

```
<?php
namespace App\Http\Requests;
use App\Http\Requests\Request;
class CreateMovieRequest extends Request
{
    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            //
            'original_title'=>'required',
            'poster_path'=>'required',
            'overview'=>'required',
            'video'=>'required'
        ];
    }
}
```

Figure 27. Form submit request validation.

#### 4.5.2 CRUD operations for movies with controller

```
<?php
namespace App\Http\Controllers;
use Request;
use App\Movie;
use App\Http\Requests\CreateMovieRequest;
use Auth;
class MoviesController extends Controller
{
    public function index()
    {
        $movie = new Movie();
        $movies = $movie->showMovie(4);
        return view('layouts.all_movies', compact('movies'));
    }

    public function create()
    {
        //return the view to manually add movie
        return view('admin.add_movie');
    }

    public function store(CreateMovieRequest $request)
    {
        $requests = Movie::saveImage(Request::all());
        //object
        $movie = new Movie($requests);
        //session
        \Session::flash('flash_message', 'Your Movie Have Been added!');
        //create the movie
        Auth::user()->movies()->save($movie);
        //redirect to the same page we were before
        return redirect('admin/add_movie');
    }

    public function show($id)
    {
        //find the movie by id and compact it for the video page

        $movie = Movie::findOrFail($id);
        return view('layouts.single_movie', compact('movie'));
    }

    public function edit($id)
    {
        //edit movie
    }
}
```

Figure 28. Controller methods to display, store and create movies.

```

    $movie = Movie::findOrFail($id);
    return view('admin.update_movie', compact('movie'));
}

public function update(CreateMovieRequest $request, $id)
{
    //update movie
    $movie = Movie::findOrFail($id);
    $movie->update($request->all());
    \Session::flash('flash_message', 'Your Movie Have Been Updated!');
    return redirect('admin/all_movies');
}

public function destroy($id)
{
    //delete movie
    $movie = Movie::findOrFail($id);
    $movie->delete();
    \Session::flash('flash_message', 'Your Movie Have Been Deleted!');
    return redirect('admin/all_movies');
}

public function allmovies()
{
    //request all movies from db
    $movies = Movie::all();
    return view('admin.all_movies', compact('movies'));
}
}

```

Figure 29. Controller methods to update, delete and listing of movies.

Figures 28 and 29 illustrate the methods responsible for handling CRUD operations on movies. Each of these methods run only when the appropriate route is requested by a user.



## 5 CONCLUSION

The primary goal of this thesis was to cover the essential features of the Laravel framework to create a web application in the cloud environment. The goal was achieved through literature research, practical implementation of theoretical knowledge of relational database gained from coursework and with the use of cloud technologies. The result produced a functional video streaming application which contains the essential features of a basic streaming application. During the process of creating the application, the convention of the Laravel framework was strictly maintained and proper documentation method was followed. On the other hand, the limitation of reliable literature for the Laravel framework was faced during the time of this work. The previous experience on Laravel framework gained from an internship played a vital role to overcome this obstacle. In addition, the result of thesis also relied on the skills acquired from the web developmental courses and programming courses from the study.

Finally, the source code for the project has been published in the developer community. This work can be extended further in terms of asset and email management

## REFERENCES

Gilmore W. J (2015). Easy Laravel 5 (1<sup>st</sup> edition). Victoria: Lean Publishing.

Laravel documentation, December 21, 2015. Application Structure [accessed March, 2019]. Available at: <https://laravel.com/docs/5.2/structure>

Laravel documentation, December 21, 2015. Artisan Console [accessed March, 2019]. Available at: <https://laravel.com/docs/5.2/artisan>

Laravel documentation, December 21, 2015. HTTP Middleware [accessed March, 2019]. Available at: <https://laravel.com/docs/5.2/middleware>

Laravel documentation, December 21, 2015. HTTP Routing [accessed March, 2019]. Available at: <https://laravel.com/docs/5.2/routing>

Laravel documentation, December 21, 2015. Installation [accessed March, 2019]. Available at: <https://laravel.com/docs/5.2/installation>

Packagist, The PHP Package Repository. [accessed March, 2019]. Available at: <https://laravel.com/docs/5.2/installation>

Rees, D (2014). Laravel: Code Bright (2<sup>nd</sup> edition). Victoria: Lean Publishing.

Rees, D (2016). PHP: Composer (1<sup>st</sup> edition). Victoria: Lean Publishing.

Saunier, R (2014). Getting Started with Laravel 4 (1<sup>st</sup> edition). Birmingham: Packt Publishing.

Stauffer, M (2016). Laravel: Up and Running (1<sup>st</sup> edition). Boston: O'Reilly.