Cloud Computing Project Report

# Introduction

The purpose of this project is to allow a user to search for a particular movie and displaying to the user the movie content. There are many movie sites available that I use quite often when looking for information about a movie. Sites such as IMDB and rotten tomatoes are the most common. I wanted to replicate some basic functionality of these sites by providing something that specifically focuses one particular task such as a movie search function. The content provided by my search function consists of basic movie information about the film such as cast, directors, ratings and displaying trailer for the movie.

I used Rotten Tomatoes API to get movie information as it seemed at the time to be well documented and meet my basic requirements. Rotten Tomatoes required me to register for an API key which would be required in order for accessing and making requests. Later during development I discovered that the RT API purposely restricted images to small thumbnails and some synopsis can’t be access due to copyrights. I found a solution by using TMDB API to get links to movies posters. TMDB API required me to register for an API key in order for accessing and making requests.

To cover trailer requirements I used an API called TheMovieClips which I found on a website called Mashape. TheMovieClips API was created to help developers to easily embed movie trailers into websites which was exactly what I was looking for. In order for me to access and make requests I needed to setup an application in Mashape and subscribe to the API. I also used another API as backup in case a trailer was not returned called Trailer Addict which easily allowed me to embed trailers from Trailer Addict with variables under my control.

I was unfamiliar with PHP and rather than do everything from scratch I decided to research different frameworks to see if they would help. After examining a few other frameworks I decided to create my application using the Laravel 4. Laravel is a free, open source PHP web application framework, designed for the development of model–view–controller (MVC) web applications. To use Laravel 4 I installed composer which is a tool for dependency management in PHP. It allows you to declare the dependent libraries your project needs and it will install them in your project for you. The documentation Laravel provide on their website is quite clear and easy to follow which meant I could have a project working locally within minutes.

Benefits:

* Convention over configuration – minimal initial configuration and setup.
* Ready out of the box - install and start using Laravel in just few minutes.
* Clear organization of all parts of the application.
* Well documented and tutorials free from Lara casts.

For information regarding deployment to the server see Deployment strategy.

# Technical Overview

**Interface Design**

The websites layout was designed as a single page application consisting of two views. The first view is a search form that allows the user to search for a movie while the second view will display movie information.

A master design was create using bootstrap 3.2 and Laravels blade templating engine. Blade is a simple, yet powerful templating engine provided with Laravel. Unlike controller layouts, Blade is driven by template inheritance and sections. All Blade templates should use the .blade.php extension.

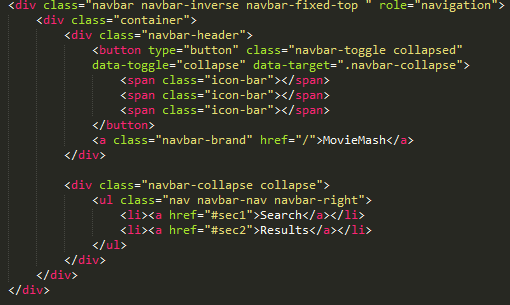
The master page includes a top navigation and footer bar which I have create separately in partial views that can be load when needed. It might not be required for such a small application such as this but it helps to keep things from being too messy.

All required external scripts and CSS files have been included in the master page using blade syntax. The search view will be loaded into the content section and wrapped in a bootstrap container. The movie result view will be added to the results section of the page using Ajax once the user starts a search.

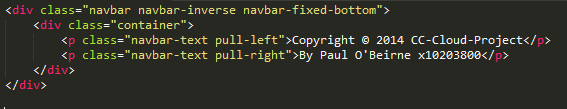
master.blade.php



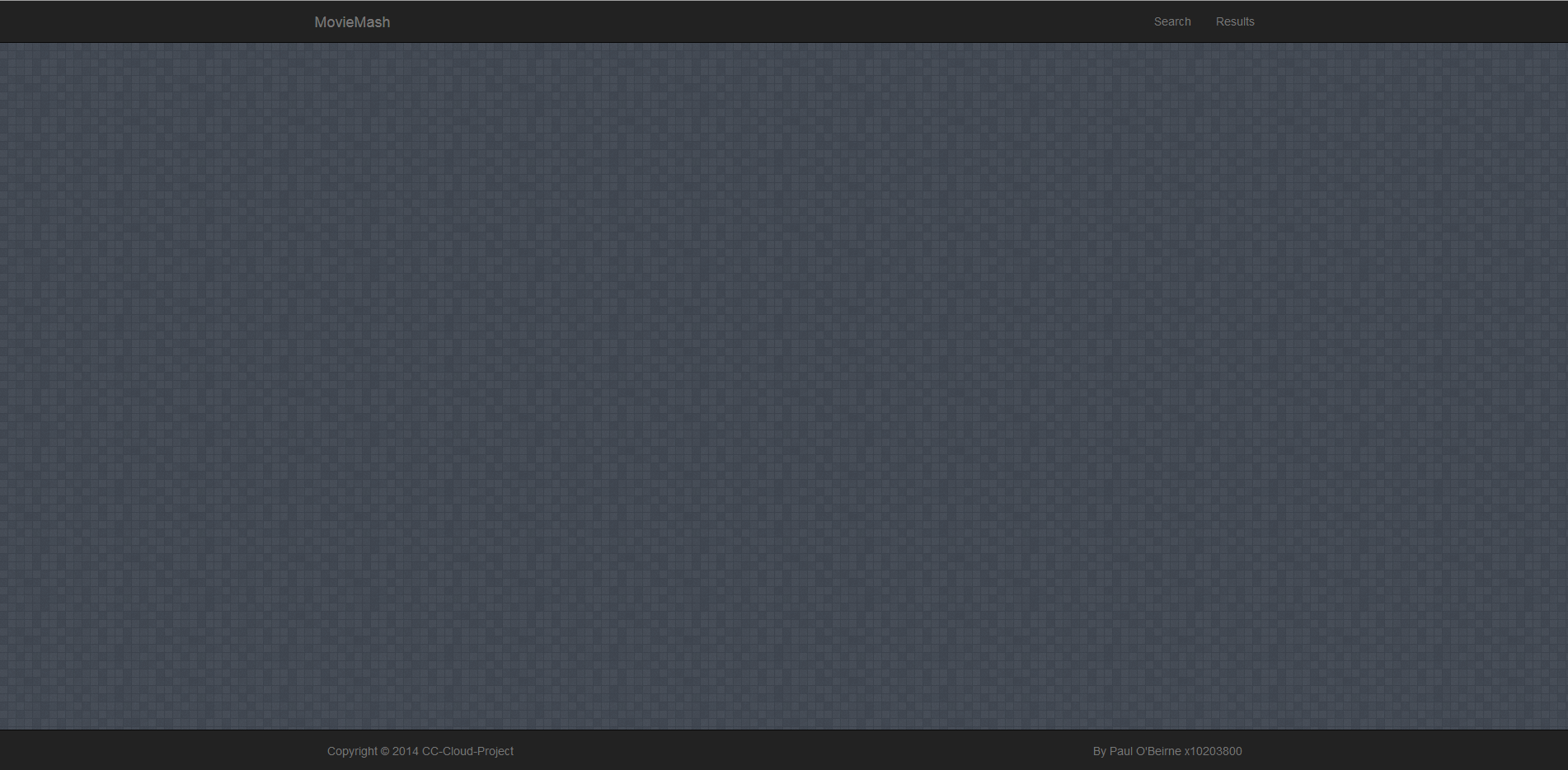
navigation.blade.php



footer.blade.php



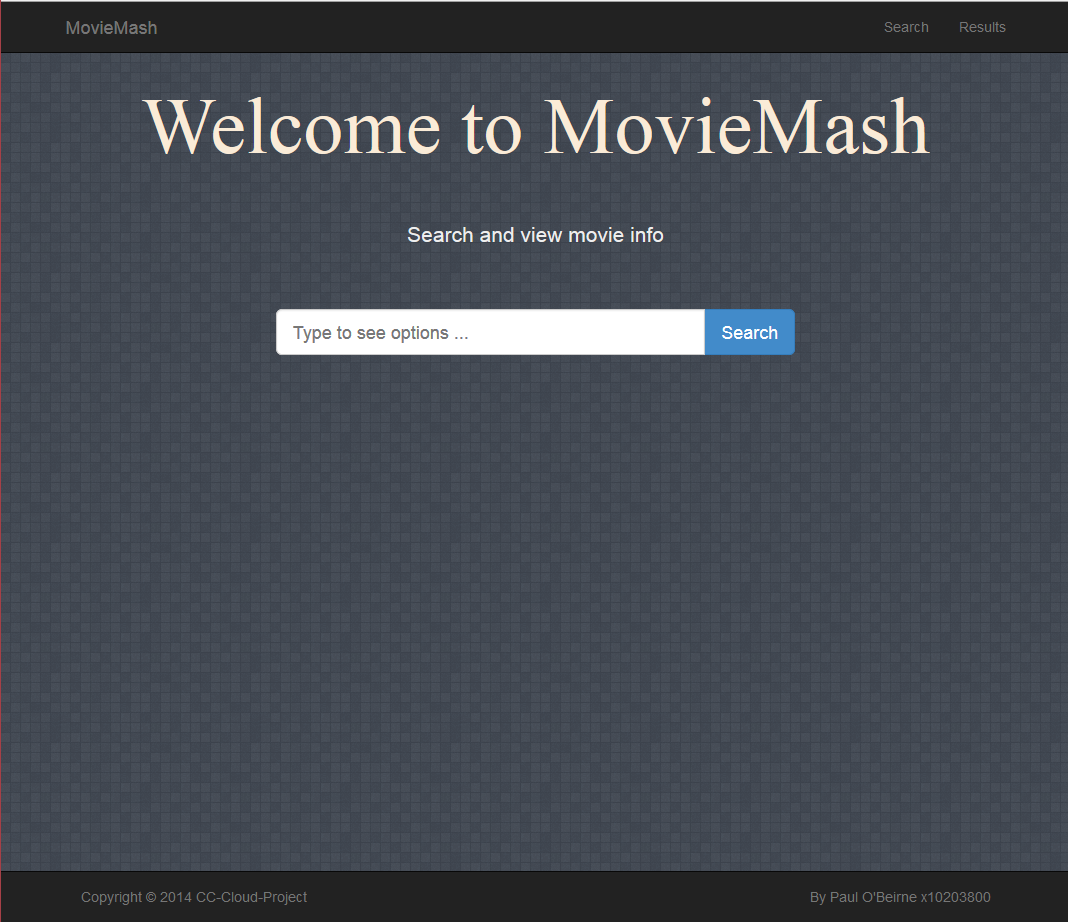
Rendered view extending master



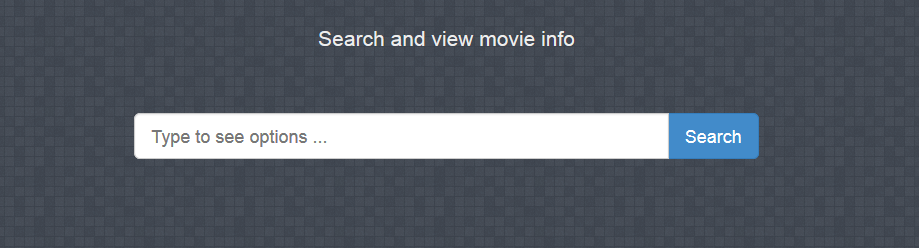
## Search View Interface Design

The search page is used to get selected user input. An autocomplete search field allows the user to type a movie name while showing possible matches. The user then must select from one of these items in order to perform the search function correctly. Once an item is selected it will populate some hidden fields with values. The user can then successfully click the search button to view results.

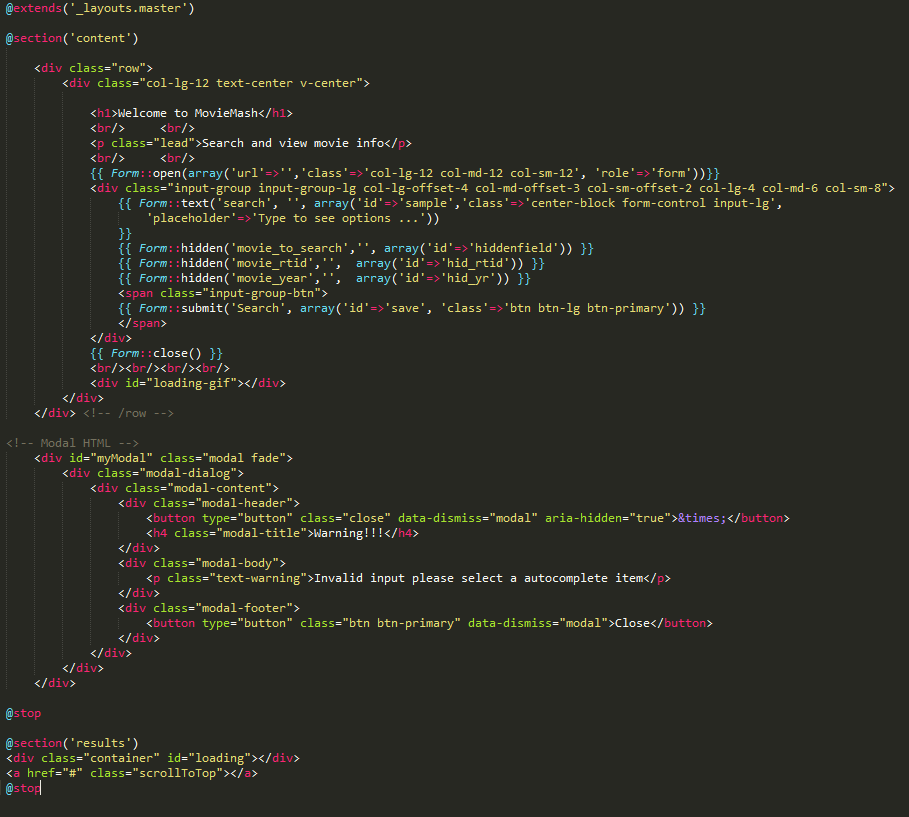
Search.blade.php



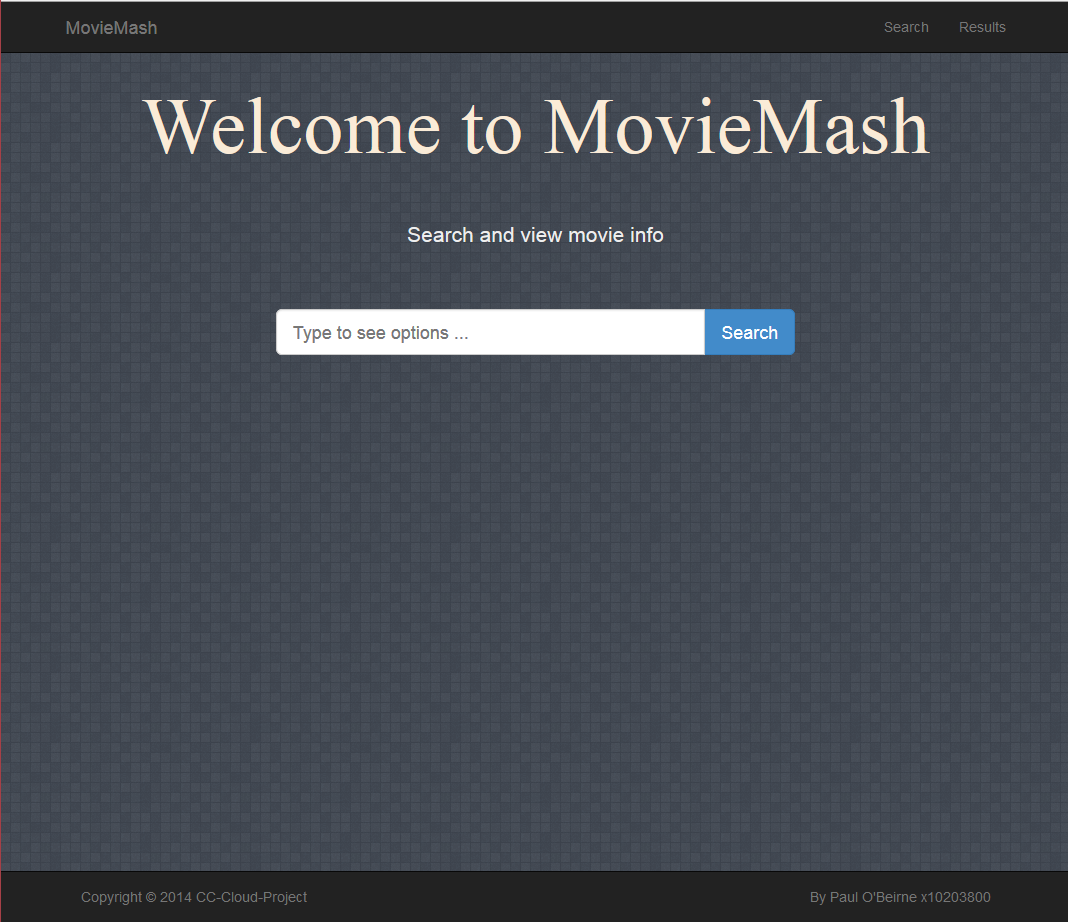
Search Form



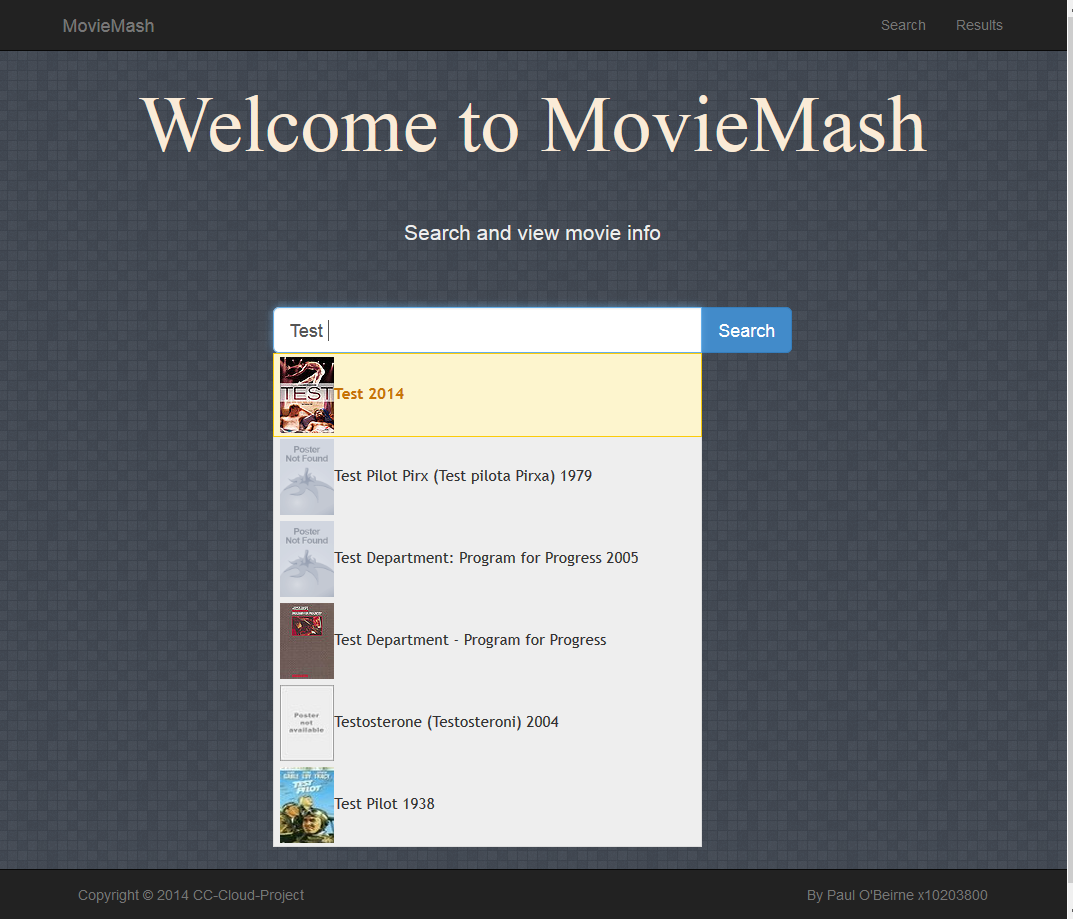
search.blade.php code



Initial load:



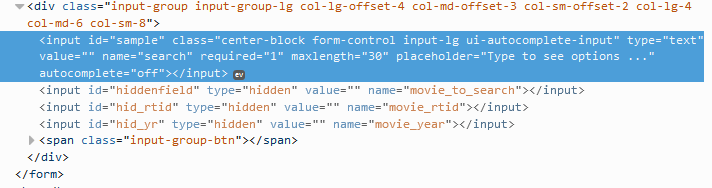
User starts typing:



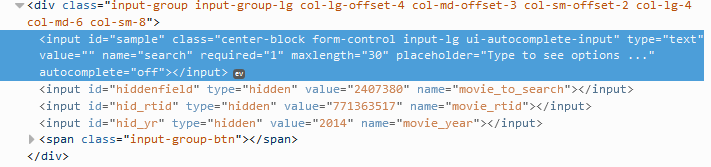
User makes selection:



Form hidden values before selection:



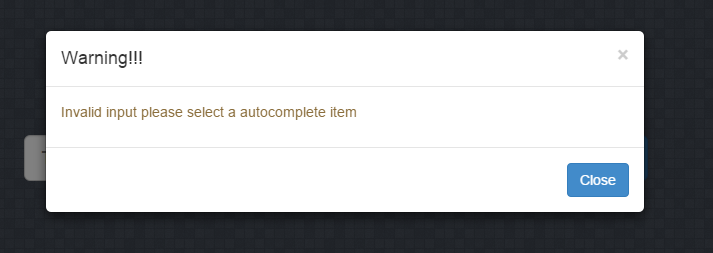
Form hidden values after selection:





A loading gif will appear if the submitting has been successful and the results will be shown and the user will be directed to them see Results Interface bellow.

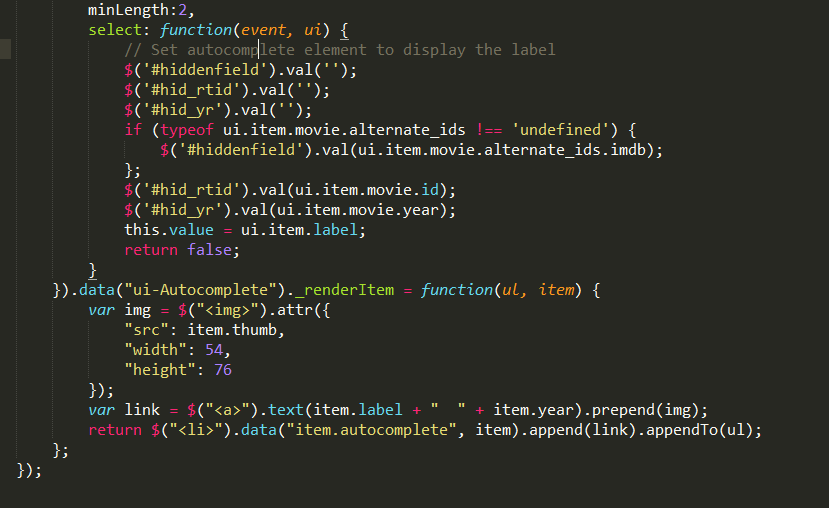
Invalid user input notification, form will be reset.



Autocomplete JQuery:

The auto complete is used to search Rotten Tomatoes API for possible movie matches. The parameters passed to the URL are the users input into the search filed as (q), an API key and a limit on how many results should be returned. The results are mapped and formatted for the user to select one. The select function clears previously set hidden filed options and repopulates them with the new values if the exist. The movie title is displayed in the search box and the user can now proceed to successfully click search.





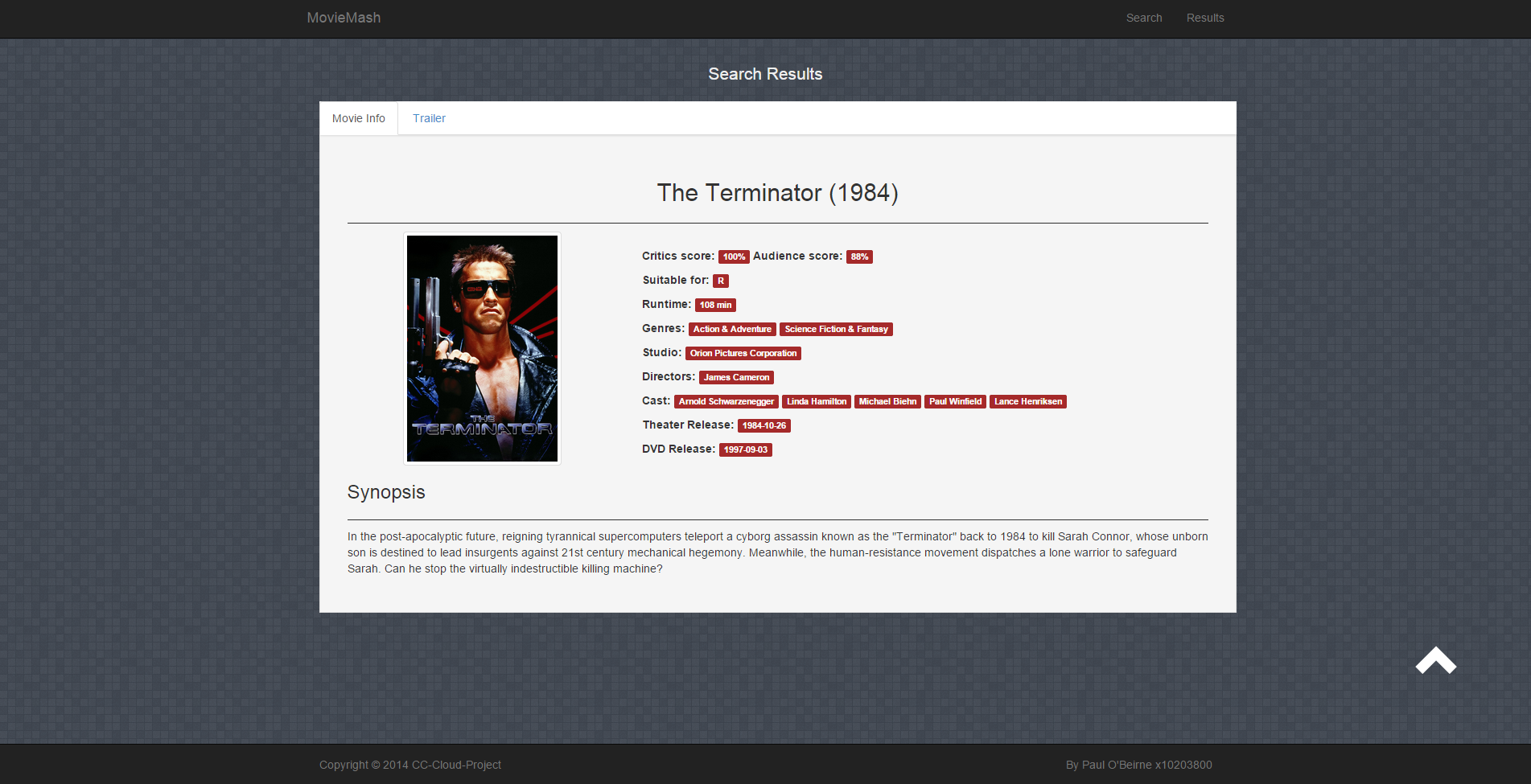
Movie result view Loading JQuery

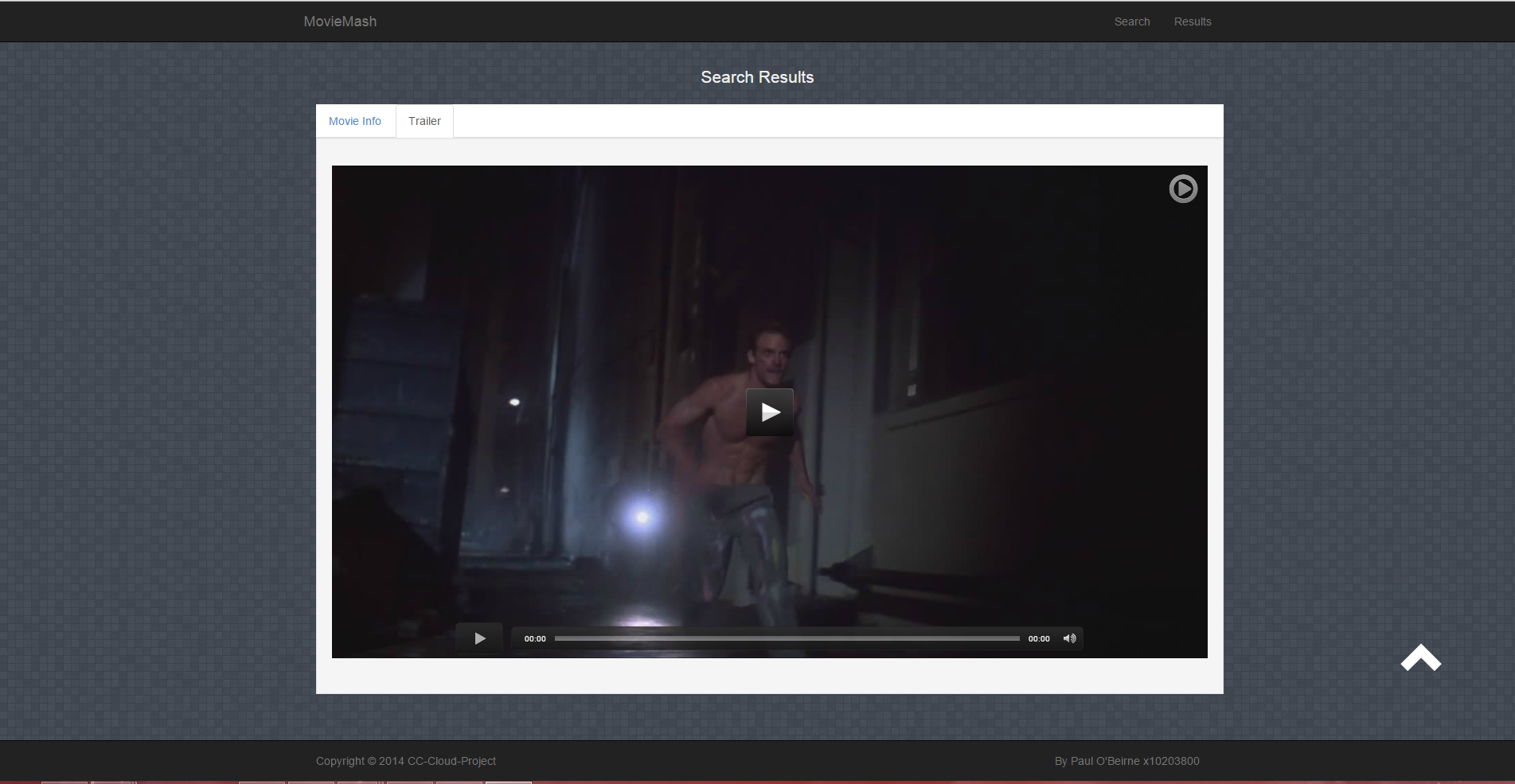


## Movie Interface Design

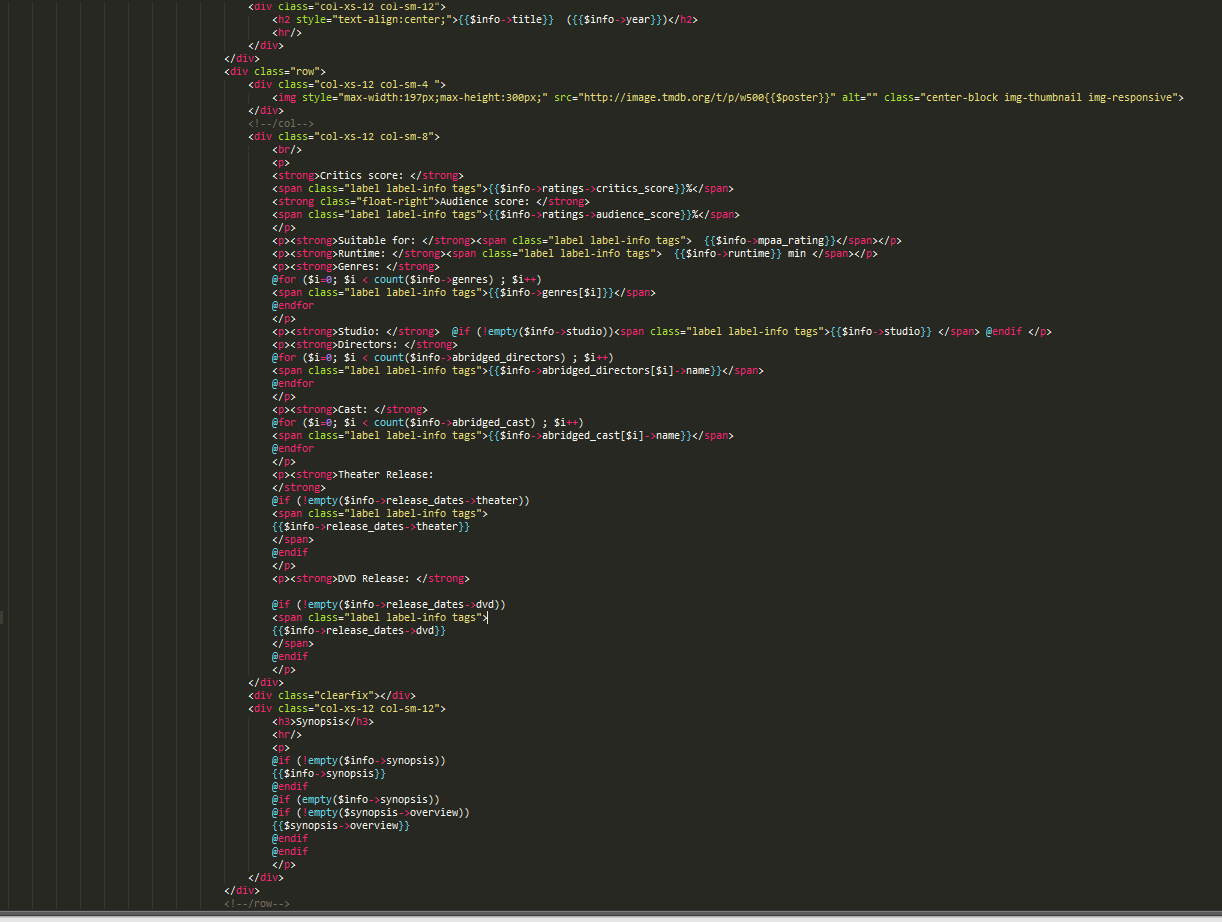
The movie results view will be used to display movie information to the user such as the movies cast, directors, runtime, rating and synopsis. The view is loaded through based on the parameters where passed from the search form to the controllers. The controllers will do some business logic and try to decide which API method calls to make. The movie view is a tabbed panel designed using bootstrap 3.2, the data that is loaded into the view before its rendered uses the blade templating engine as a place holder for the data. The tabbed panel consists of two sections one displaying movie information the other displaying a movie trailer.

movie.blade.php





movie.blade.php Tab1



movie.blade.php Tab2



## Data Sources

To handle most of the API requests I install a HTTP wrapper library which is a set of lightweight HTTP libraries that offer:

* It supports form parameters, file uploads and custom body entities
* Supports Basic Authentication natively
* Customizable timeout
* Customizable default headers for every request (DRY)
* Automatic JSON parsing into a native object for JSON responses

To install in my Laravel project I used Composer

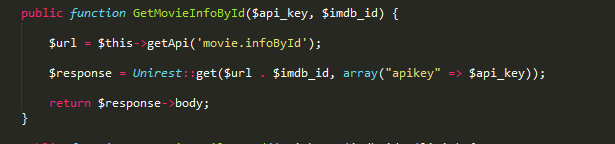
* Add dependencies in composer.json file
* Run composer update to download and install the required dependencies

Composer.json file:



After composer update/install is run the package will be visible under the vendor folder

Example usage:



I created a class MovieAPI to handle all the different APIs in one place. If the application was bigger I would have separated them into each of their own classes but I don’t think it was required for this application. In the MovieAPI I created an array to hold all the different end points I would be using.

End point Array



Example referencing Array



## Data Source 1

The Rotten Tomatoes API is Restful web service that was designed to be easy to explore and use. Developers can use the API to access the Rotten Tomatoes' database of movie data by using the Lists, Movies, or DVDs sections of the API.

|  |  |
| --- | --- |
| Method: | GetMovieInfoById |
| Description: | This method is used to get information on a movie by using an IMDB Id number. |
| End point: | http://api.rottentomatoes.com/api/public/v1.0/movie\_alias.json? |
| Parameters: | api\_key, imdb\_id |
| Response: | Json |
| Implementation: | |

|  |  |
| --- | --- |
| Method: | GetMovieInfoByRtId |
| Description: | This method is used to get information on a movie using a Rotten Tomatoes movie Id number. It was needed as not all of the movies contain a IMDB Id. |
| End point: | http://api.rottentomatoes.com/api/public/v1.0/movies/ |
| Parameters: | api\_key, rt\_id |
| Response: | Json |
| MovieAPI Implementation: | |

## Data Source 2

The TMDB API is a resource for any developers that want to integrate movie, TV show and cast data along with posters or movie fan art. Themoviedb.org is a free and community edited database.

|  |  |
| --- | --- |
| Method: | GetMoviePosterById |
| Description: | This method is used to get the poster of a movie by using an IMDB Id. Required as Rotten Tomatoes restricted media content. |
| End point: | http://api.themoviedb.org/3/find/tt |
| Parameters: | api\_key imdb\_id |
| Response: | Jsonp |
| Implementation: | |

|  |  |
| --- | --- |
| Method: | GetMoviePosterByTitle |
| Description: | This method is used to get the poster of a movie by querying both the title and year of release. It was needed as not all of the movies contain a IMDB Id. |
| End point: | http://api.themoviedb.org/3/search/movie? |
| Parameters: | api\_key, title ,year |
| Response: | Jsonp |
| Implementation: | |

|  |  |
| --- | --- |
| Method: | GetMovieSynopsisById |
| Description: | This method is used to get a movies synopsis using an IMDB movie Id. It was needed as not all of the movies contain a synopsis due to ownership rights. |
| End point: | http://api.themoviedb.org/3/movie/ |
| Parameters: | api\_key, imdb\_id |
| Response: | Jsonp |
| Implementation: | |

|  |  |
| --- | --- |
| Method: | GetMovieSynopsisByTitle |
| Description: | This method is used to get a movies synopsis by querying both the title and year of release. It was needed as not all of the movies contain a IMDB Id. |
| End point: | http://api.themoviedb.org/3/movie/ |
| Parameters: | api\_key, title, year |
| Response: | Jsonp |
| Implementation: | |

## Data Source 3

The MovieClip API was created to help developers to easily embed trailers or get movies information’s into websites, applications, etc. This is the first version of our public API and it has xml and Json feeds.

|  |  |
| --- | --- |
| Method: | GetMovieTrailerById |
| Description: | This method is used to get a movie trailer based using a IMDB Id. |
| End point: | https://themovieclips.p.mashape.com/trailers? |
| Parameters: | api\_key, imdb\_id, limit |
| Response: | Json |
| Implementation: | |

|  |  |
| --- | --- |
| Method: | GetMovieTrailerByTitle |
| Description: | This method is used to get a movies trailer by querying the title. It was needed as not all of the movies contain an IMDB Id. |
| End point: | https://themovieclips.p.mashape.com/trailers? |
| Parameters: | api\_key, title, limit |
| Response: | Json |
| Implementation: | |

## Data Source 4

TrailerAddict offers movie trailers in high-definition. Users have the ability to embed trailers in HD anywhere. TrailerAddict lets users find trailers by film, genre, actor, actress and more. The TrailerAddict API gives developers the ability to embed trailers with variables such as width, offset, trailer count, and trailers by which film and more. The API uses Restful calls and responses are formatted in XML.

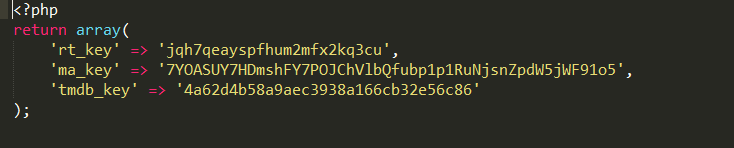
|  |  |
| --- | --- |
| Method: | GetMovieTrailerByIdBck |
| Description: | This method is used as a backup against the movieclips api not returning anything. Gets a movie trailer based using a IMDB Id. |
| End point: | http://api.traileraddict.com/ |
| Parameters: | imdb\_id, limit |
| Response: | XML |
| Implementation: | |

|  |  |
| --- | --- |
| Method: | GetMovieTrailerByTitleBck |
| Description: | This method is used as a backup against the movieclips api not returning anything. Gets a movie trailer based using a title. |
| End point: | http://api.traileraddict.com/ |
| Parameters: | title, limit |
| Response: | XML |
| Implementation: | |

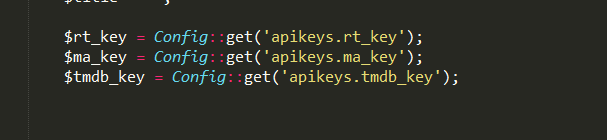
# Home Controller

The home controller was holds business logic which will decide the required MovieApi methods based on the submitted parameters from the search form. I also created a config class to store the different API keys in one place. This config class can be used by the controller to pass API keys to the MovieAPI class.

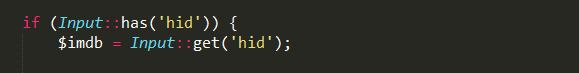
apikeys.php



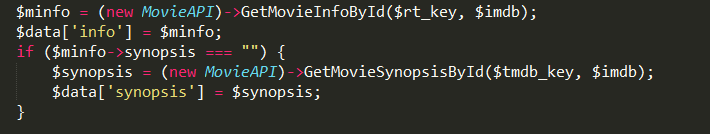
Example retrieving a key



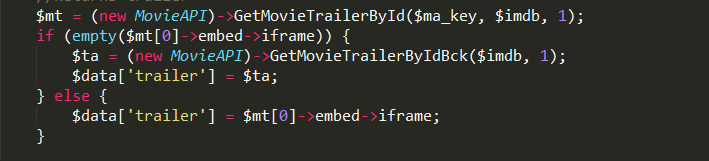
The first condition requires an IMDB Id to exist which will then start making request passing the IMDB ID as a base parameter along with any required API key.



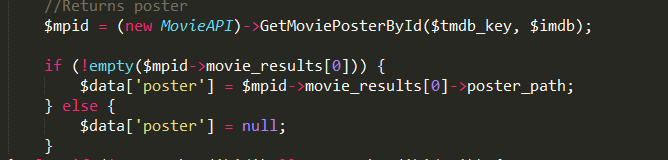
Returns movie info and checks if the returned data has a synopsis, if not it will make a request to another data source to retrieve one.



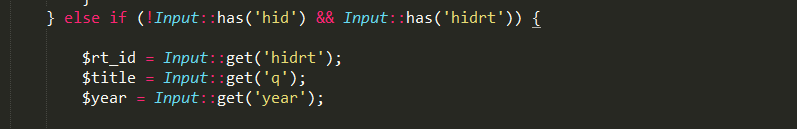
Returns trailer info and checks if the returned data has a trailer, if not it will make a request to another data source to retrieve one.



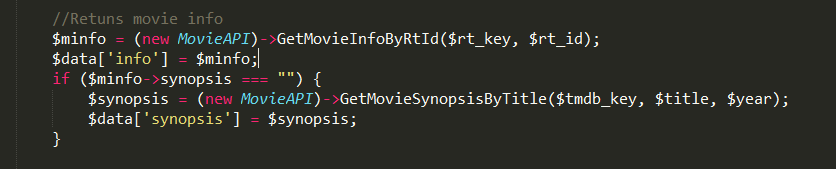
Returns poster info and checks if the returned data has a poster, if not it will make return a null value.



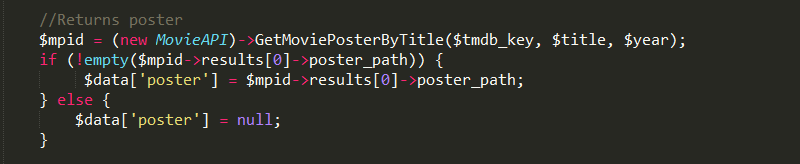
If the first condition is not met a Rotten Tomatoes Id will need to be available in order to continue. It will then start using a different set of parameters and API methods to find the correct information.



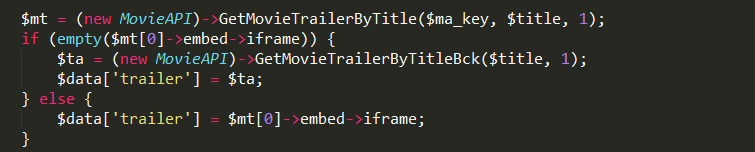
Returns movie info and checks if the returned data has a synopsis, if not it will make a request to another data source to retrieve one.



Returns poster info and checks if the returned data has a poster, if not it will make return a null value.



Returns trailer info and checks if the returned data has a trailer, if not it will make a request to another data source to retrieve one.



# Deployment strategy

## Platform One

Application Link: <http://still-retreat-1205.herokuapp.com/>

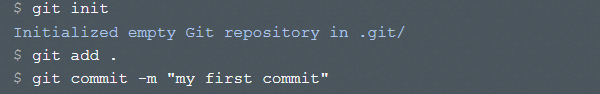
I chose to use Heroku as my first platform as it supported Laravel 4 framwork .Heroku is a cloud-based development platform as a service (PaaS) provider. The Heroku platform supports development in Ruby on Rails, Java, Node.js, Python, Scala and Clojure.

Originally, Heroku supported only Ruby on Rails. In 2010, the company reported that it supported a million Ruby developers writing and running apps directly on Amazon Web Services (AWS). Heroku maintains the development environment and integrates other cloud services. The company partnered with Facebook to allow developers with Heroku accounts to create and deploy Web apps quickly and easily through a Facebook interface.

Deploying with Git

Git is a powerful decentralized revision control system, and is the means for deploying apps to Heroku. Heroku apps expect the app directory structure at the root of the repository. If your app is inside a subdirectory in your repository, it won’t run when pushed to Heroku.

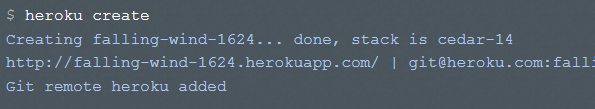
Before you can push an app to Heroku, you’ll need to initialize a local Git repo and commit your files to it. For example, if you have an app in a directory, “myappfolder”, then create a new repo for it:



This is a local repository, now residing inside the .git directory. Nothing has been sent anywhere yet; you’ll need to create a remote and do a push to deploy your code to Heroku.

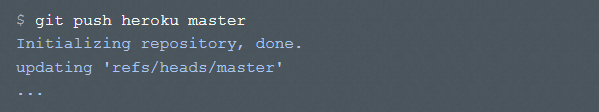
Creating a Heroku remote repository

[Git remotes](http://git-scm.com/book/en/Git-Basics-Working-with-Remotes) are references to remote repositories. You can have any number of these, but for now we’ll focus on just the remote to Heroku. The heroku create command creates a new application on Heroku – along with a git remote that must be used to receive your application source.



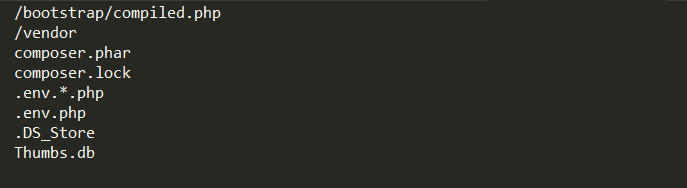
Deploying code

Your Heroku app starts with a blank repository – it has no branches and no code. So the first time you deploy, you’ll need to specify a remote branch to push to. You can do your first push:

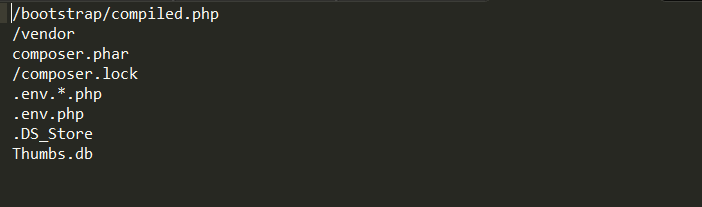


Addition Configuration for Laravel Git initially is set to ignore the composer.lock file which is required when deploying to a remote repositories. I had to make sure .gitignore wasn’t ignoring the file so I modified the .gitignore file.

.gitignore before



.gitignore after



## Once the changes have been make I had to add the composer.lock file to the git repository and then I was able to proceed in remotely deploying to a Heroku git repository.

## Platform Two

Application Link: <http://cc-cloud-project.eu1.frbit.net/>

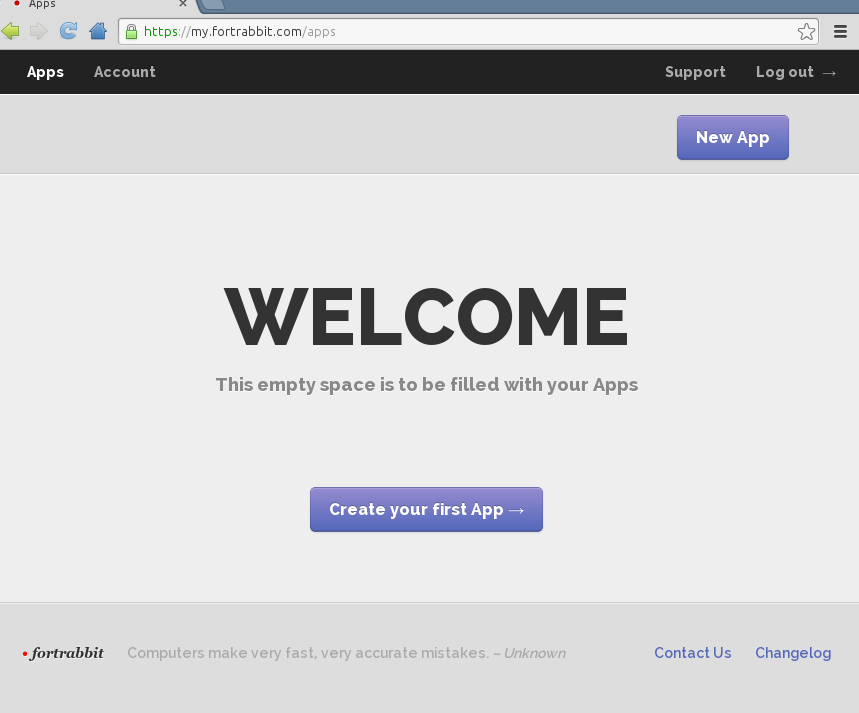
I chose to use Fortrabbit as my second platform as it supported Laravel 4 framwork.Fortrabbit is a cloud hosting platform dedicated to PHP applications. It offers versatile deployment options (Git, SSH, SFTP) and native Composer integration. Fortrabbit is a PaaS company who say the hosting business is broken, and too much about price dumping and too little about quality. In order to fix the problem, they made a PHP-exclusive business that caters to developers’ most common needs. Their hosting plans are accessibly-tiered for everyone, from hobbyists (free plans) to enterprises (high-availability upgrades are available at the click of a button).

1: Sign up

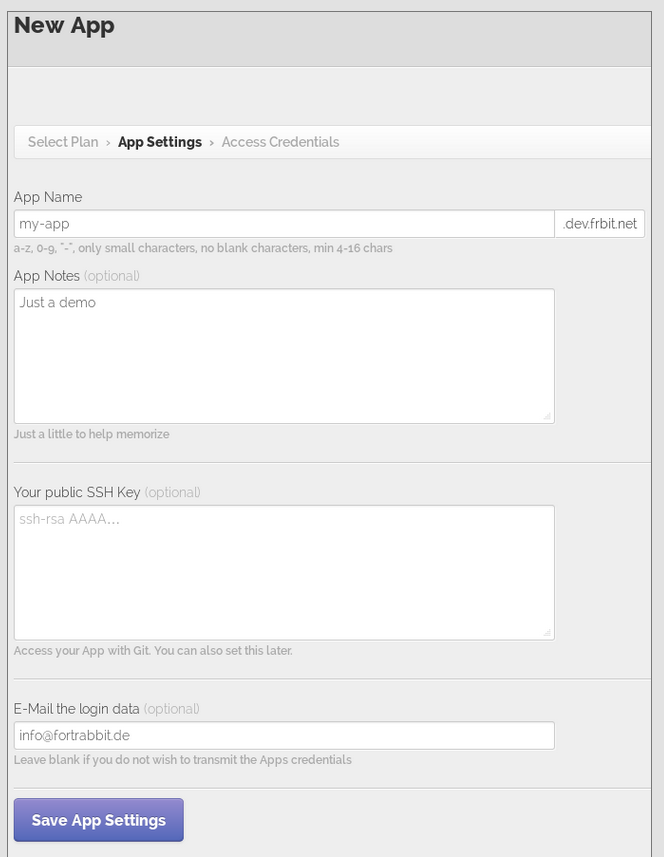
Go to https://my.fortrabbit.com/register and sign up. To create your account we just ask you for: your full name, email address and a secure password of your choice. Next you'll receive an email confirmation. Now you can login to the dashboard. (It took 5 days for access to a fee development account to be created)

2: Create your first App

Once you're logged in, pick any shiny purple button you'd like and get started :)

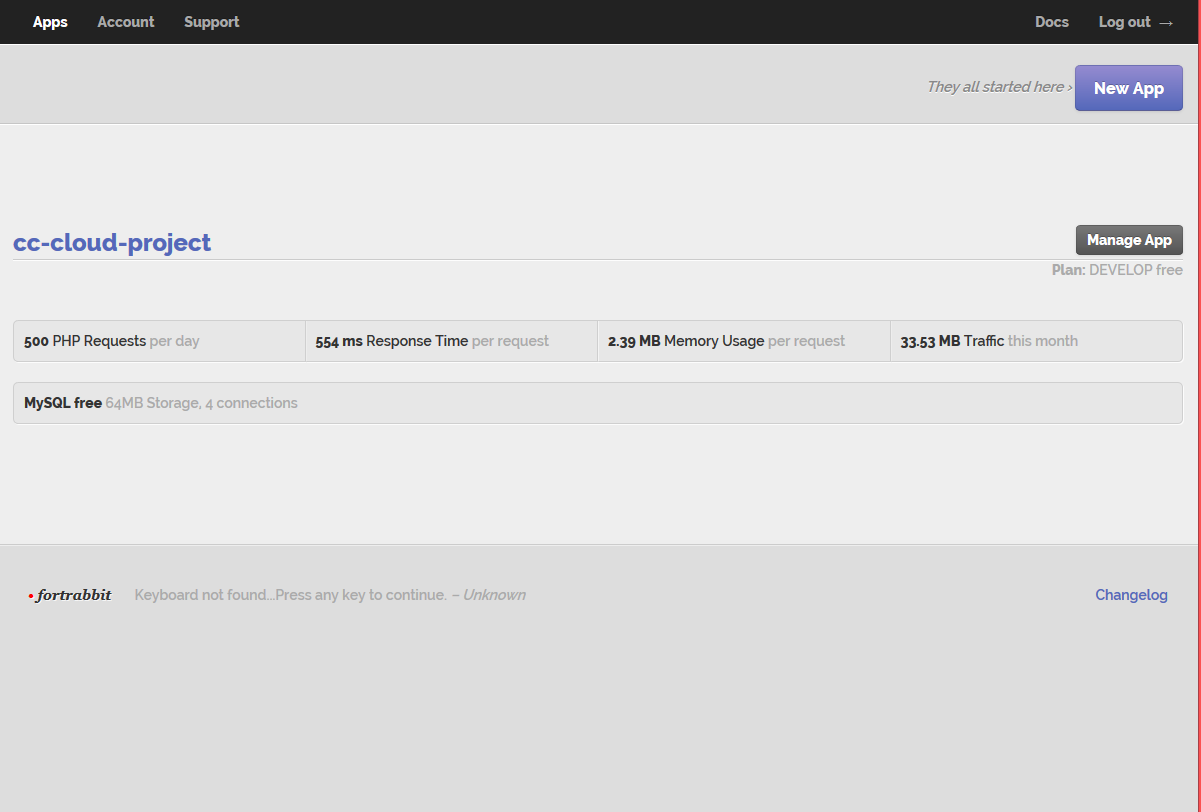


The first-time App creation will choose a free plan by default and ask you for an App Name. It gives you the option to write a short comment (what are you using the App for) and to send you the credentials by e-mail.

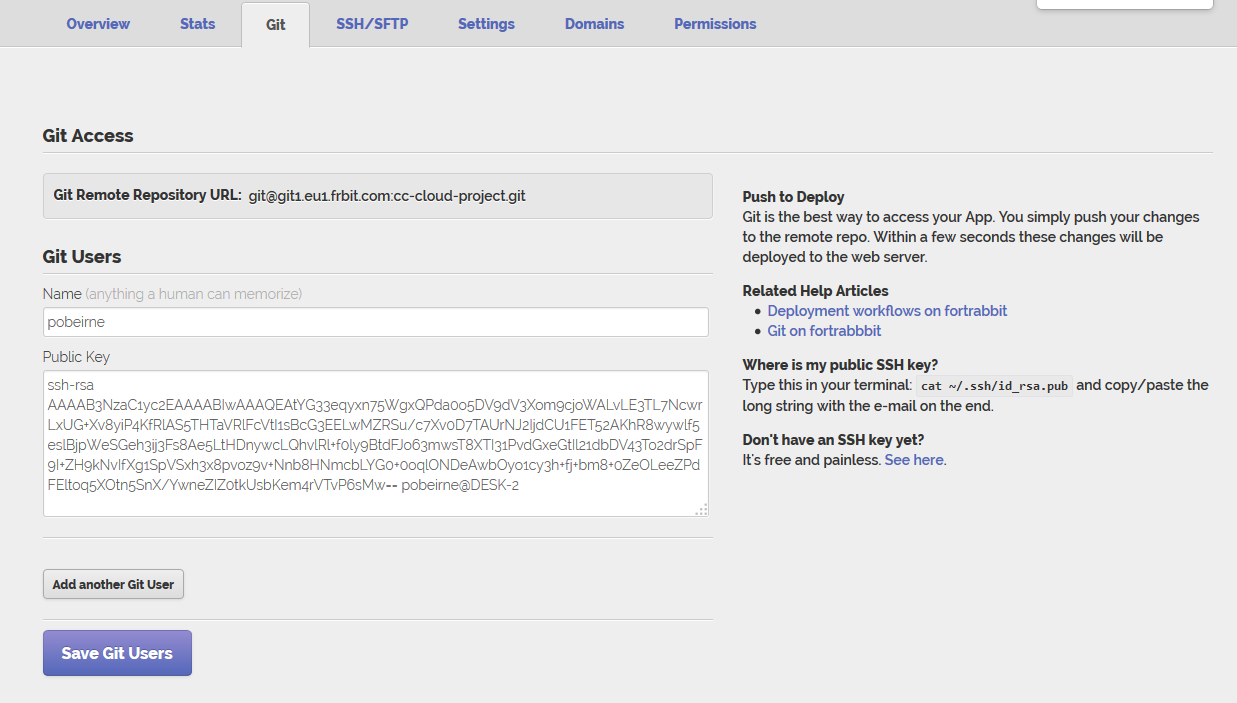


Adding public key to Fortrabbit

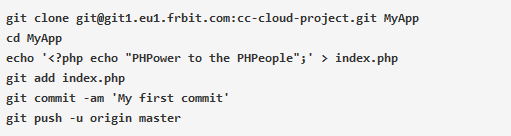
In order to be able to git push to Fortrabbit (the way to deploy your app to Fortrabbit) we need to add the public key part of the SSH key pair to our application. This can be done in the manage view of your application via the application overview. Check for SSH keys open up your Git Bash and type: cat ~/.ssh/id\_rsa.pub



Navigate to the **git** tab and enter a name for the git user, I picked **pobeirne** and paste the public key part into the big text area.

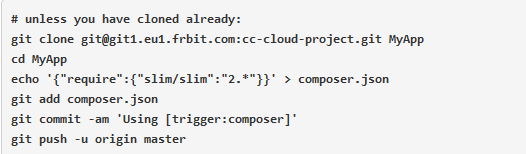


### How to deploy with Git



Once I cloned the git repository to a folder I copied the contents from my Heroku applications folder. I used Git to add all files to the local git and ran these commands (composer.phar, composer dump, composer update) to make sure all the dependencies and composer.lock files where correctly configured. I then made sure the composer.lock file was added and not being ignored by git before pushing to the remote repository.

Use Composer hook when first deploying



Add everything, make a commit with the additional [trigger:composer] keyword in the message and push. You need to wait a bit, because composer will install quite a lot of packages.

References

*Composer Download,* Available at: [*https://getcomposer.org/*](https://getcomposer.org/)

(Accessed: 1st November 2014).

*Git Download,* Available at: [*http://git-scm.com/*](http://git-scm.com/%20)

(Accessed: 1st November 2014).

*Laravel,* Available at: [*http://laravel.com/*](http://laravel.com/)

(Accessed: 1st November 2014).

*Heroku Documentation,* Available at: [*https://devcenter.heroku.com*](https://devcenter.heroku.com)

(Accessed: 1st November 2014).

*Heroku,* Available at: <https://www.heroku.com/>

(Accessed: 1st November 2014).

Fortrabbit *Documentation*, Available at: [http://fortrabbit.com/docs/getting-started](http://fortrabbit.com/docs/getting-started%20) (Accessed: 20th November 2014).

Rotten Tomatoes API, Available at: <http://developer.rottentomatoes.com/docs>

(Accessed: 1st November 2014).

MovieClips API, Available at: <https://www.mashape.com/razor/themovieclips>

(Accessed: 20th November 2014).

TMDB API, Available at: <https://www.themoviedb.org/documentation/api>

(Accessed: 20th November 2014).

Trailer Addict API, Available at: <http://www.traileraddict.com/trailerapi>

(Accessed: 1st November 2014).